Provable Security in the Real World: New Attacks and Analyses

Joanne Adams-Woodage

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
School of Mathematics and Information Security
Royal Holloway, University of London

2019

Declaration

These doctoral studies were conducted under the supervision of Professor Kenny Paterson.

The work presented in this thesis is the result of original research that I conducted, in collaboration with others, whilst enrolled in the School of Mathematics and Information Security as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Author Name: Joanne Adams-Woodage

Date: September, 2019.

For Sam.

“Do a Maradona, son. Do a Maradona.”

— Mike Bassett, England Manager

Abstract

The provable security paradigm constructs rigorous, reductionist proofs that a cryptographic scheme achieves a security goal. In this thesis, we illustrate the power and versatility of this approach by presenting a practice-oriented provable security treatment of three real world problems. We use these as a jumping-off point to construct novel security models, uncover new attacks, analyse deployed solutions, and develop improved ones.

First we consider message franking, a technique to enable verifiable abuse reporting in encrypted messaging applications. We demonstrate an attack breaking Facebook's attachment franking protocol. The attack is facilitated by the lack of fast and secure franking schemes. We address this gap by building the most efficient franking scheme to date, while also showing that secure franking schemes cannot match the efficiency of the fastest authenticated encryption schemes.

Our next two problems take root in the NIST SP 800-90A standard, notorious for its prior inclusion of the backdoored Dual-EC-DRBG. First, we investigate the security of the three pseudorandom number generators (PRNGs) with input that remain in the standard. We prove (with a caveat) the robustness [60] of HASH-DRBG and HMAC-DRBG in the ROM. Regarding the caveat, we show that if an optional input is omitted then HMAC-DRBG is not forward secure. We conclude with a more informal and practice-oriented exploration of flexibility in permitted implementation choices.

Finally, inspired by Dual-EC-DRBG, we present the first provable security treatment of backdoored PRNGs (BPRNGs). We resolve an open problem from [57] by demonstrating that forward secure PRGs succumb to stronger forms of backdooring than previously envisioned. Building on this, we modify a robust PRNG from [60] to embed a weak form of backdoor. We present a new security model for BPRNGs which targets a significantly stronger backdooring notion, and construct and analyse a robust BPRNG that provably achieves it.

Acknowledgements

The deepest thanks to my supervisor Kenny Paterson, without whom none of this would have been possible. From introducing me to provable security, to reading my thesis drafts so carefully that not a single misplaced punctuation mark went unremarked, Kenny has been the most brilliant and supportive supervisor at every step of the way. Inspiring to work with, always on hand when I was in need of advice or guidance, and just generally fun to be around, I've learned so much from Kenny and feel so lucky to have had him as a supervisor.

A huge thank you to Tom Ristenpart for a brilliant research visit to Cornell Tech and for our subsequent collaborations. Tom is so full of good ideas and incredibly exciting to work with, and taught me so much that went into this thesis, from how to approach a problem right down to the proper way to typeset an em dash (three hyphens!).

A massive thank you to Dan Shumow for an excellent internship at Microsoft Research. Dan is so great to work with, as well as being a hugely supportive mentor and exceedingly interesting person, all of which made for a great summer — who knew that wading through NIST standards could be so much fun! Many thanks to Brian LaMacchia and the rest of the team at MSR for the opportunity to intern with them and for making me so welcome.

I have been very lucky to work with many brilliant collaborators throughout my PhD. Thank you to: Jean Paul Degabriele, Jacob Schuldt, Rahul Chatterjee, Yevgeniy Dodis, Ari Juels, Anusha Chowdhury, Yuval Pnueli, Paul Grubbs, Mihir Bellare, and Douglas Stebila; I learned so much from all of you. Special thanks to Jean Paul for giving me a lot of his time and help at the start of my PhD; to Douglas for hosting me for a great visit to Waterloo; and to Paul for being a good pal during our research visits to each other's sides of The Pond. Many thanks to Bertram Poettering and Bart Preneel for agreeing to be my thesis examiners and for their insightful feedback.

I am very grateful to the EPSRC and the CDT at Royal Holloway for funding my PhD and facilitating many opportunities to travel and engage with industry. The CDT has been a brilliant place to do a PhD; thanks to all the staff and students for making it such a good place to be. Special thanks to my partner in crime (and prime) Jake for all the cups of tea; and to Thyla for being a great source of support and for introducing me to Ye Olde Swiss Cottage.

Huge thanks to all my wonderful friends: Emily and Poppy, for twenty plus amazing years of friendship and for still being up for going to Tru every now and then; and to Catherine, Esmé, Kirsty, Louisa, Louise, Sophia, Sophie, and Tash for making our time in Manchester far too much fun to go to too many lectures, and for being such precious friends ever since.

I am very lucky to have such an amazingly close and supportive family. The biggest thanks to Mum and Dad for all their love and support, and for always being there to help whether it's a major crisis or just that I've got stuck in a parking space again. Special thanks to Dad for proof-reading every word of this thesis and to Mum for always being on hand with kind words and delicious food whenever things got too much. A special *spasiba* to my lovely little sister Lydia for being my best pal always and for finding all the best pubs, and to my wonderful aunts and uncles Sally, Sharon, Stephen, and Tess.

Finally, a million thank you's to my wonderful Sam, the love of my life, my best friend, and the sunshine in all of my days. Thank you for always being there to cheer me on and cheer me up; I couldn't have got through my PhD (or life in general, really) without you by my side. Most of all, thank you for making life so much *fun*: I love you so much and I can't wait for everything that lies ahead.

Contents

Chapter 1

Introduction

Contents

1.1 Motivation

For as long as there has been communication, there has been a need to communicate in secret. Cryptography — the art of crafting schemes for secret communication (and as we shall see, much more) — has existed since antiquity. For much of that time, its use was largely confined to military, government, and intelligence applications. However, with the advent of the internet, cryptography has become part of the fabric of our everyday lives. Each time we make an online purchase, use a messaging application, or connect to the near 80% of the internet now encrypted with TLS [2], we participate in a cryptographic protocol. Even more remarkably, the integration of cryptography into these applications is so seamless that the average user may never be aware of the complex cryptographic machinery underpinning the security of the task at hand. The 'invisibility' of cryptography in applications is highly desirable: it not only improves user experience but crucially removes the human factor — notoriously the weakest link in the information security chain. This does however place a great deal of trust in the 'real world cryptography' that touches our lives every day. With such high stakes, cryptographers have a responsibility to ensure that deployed cryptographic schemes are as secure as possible.

Provable security. The increased interest in cryptography that has accompanied its widespread adoption has rapidly accelerated the development of the field, transforming it from an art into a science. While once the security of a scheme was inferred from the absence of known attacks, we now expect a far higher level of assurance that a scheme should be trusted.

A particularly powerful lens through which to scrutinise the security of a cryptographic scheme is the *provable security paradigm.* Beginning with the ground breaking work of Goldwasser and Micali [72], a provable security analysis defines precisely what security means in a given context and then rigorously proves (or disproves) that a scheme achieves it.

The beauty of this approach is that, in a single stroke, we can rule out the possibility of *any* attack strategy permitted by the security definition succeeding. Of course, such a security proof should not be construed as an absolute guarantee that the scheme cannot be broken — the proof will typically be conditioned on certain assumptions, and moreover says nothing about attacks that lie outside the security model. Nonetheless, provable security has revolutionised cryptography, elevating security arguments from intuition to formal reasoning and providing a rigorous framework within which to unpick why a scheme withstands or succumbs to a certain class of attack.

Practice-oriented provable security [13, 132], a branch of cryptographic research that applies the precise analysis of provable security to real world problems, has had a resounding real work impact. Particular success stories originating from this line of work include the now ubiquitous HMAC [15] and HKDF [96] schemes, the formalism of authenticated encryption (in a long line of work, beginning with [19, 24, 87]), and key contributions to the development of TLS 1.3 [82, 98, 120]. In this thesis, we take inspiration from this legacy of influencing cryptographic standards, developing widely deployed protocols, and informing best practice, and present a practice-oriented provable security treatment of a variety of novel real world problems.

1.2 Contributions

In this thesis, we contribute to the body of work on practice-oriented provable security by analysing a number of real world problems within the provable security paradigm. The starting points for our chapters are, respectively, a protocol available to a billion users each day (Facebook's encrypted messenger, Chapter 3), and a widely used — but controversial — cryptographic standard (NIST SP 800-90A, Chapters 4 and 5). We use these real world applications as a springboard to define new security models, analyse deployed schemes, and propose new solutions. We will consider both high-level protocols and fundamental primitives, and in each setting pay particular attention to uncovering and modelling new attacks. We discuss our contributions in more detail below.

1.2.1 Thesis Structure

In this section we highlight our main contributions, and give an overview of the chapters in this thesis and the real world problems by which they are motivated.

Background and preliminaries. In Chapter 2, we provide the necessary preliminaries for the work to follow. We begin with an overview of the provable security paradigm. We describe the evolution of this line of work, introduce key techniques, and discuss a number of limitations and common criticisms of the approach. We then define a number of cryptographic primitives and concepts that will be called on throughout the subsequent chapters.

With this groundwork in place, we proceed to analyse a number of real world problems within the framework of practice-oriented provable security.

End-to-end encryption in Facebook messenger. For our first technical chapter, we take as inspiration a real world protocol available to billions of users each day: end-to-end encrypted messaging in Facebook Messenger. Messages exchanged in end-to-end encrypted applications are readable only by the sender and receiver: no intermediaries — including the service provider — can learn anything about the message contents. While end-to-end encrypted messaging is certainly desirable in terms of privacy, it would appear to be at odds with the verifiable reporting of abuse, since the service provider has no way of determining if a reported message was *actually* sent. In 2016, Facebook suggested a way to navigate this tension in the form of *message franking*, which aims to provide cryptographically verifiable abuse reporting in Facebook Messenger. In their system, the ciphertext is accompanied by a cryptographic proof of the message contents. In 2017, Grubbs, Lu, and Ristenpart [73] (GLR) captured the goals of message franking via a new primitive called compactly committing authenticated encryption with associated data (ccAEAD), and proved that Facebook's approach to message franking for short messages is secure. However, all known secure ccAEAD schemes are slow compared to regular authenticated encryption with associated data (AEAD) schemes. Because of this, Facebook handles large files such as attachments differently.

In **Chapter 3**, we investigate how to build ccAEAD schemes that are secure and *fast*. To motivate the need for fast message franking, in the first part of the chapter we present an attack on Facebook's attachment franking scheme which allows an attacker to prevent messages from being reported as abusive. In essence, this vulnerability stems from a lack of secure ccAEAD schemes which are fast enough to be used with long messages. Because of this, Facebook use an ad hoc scheme which includes the AEAD scheme GCM [107] as a component. As we shall see, the attack is facilitated by the fact that GCM lacks a security property called *robustness* [67] which requires that it is infeasible to construct ciphertexts that decrypt correctly under different keys, coupled with a server side bug. As far as we are aware, this is the first real world attack exploiting the non-robustness of an encryption scheme. As such, the attack serves as an interesting cautionary tale of how a security property that may seem to be more of theoretical interest than practical concern can lead to a real world vulnerability when in interaction with other components in a complex protocol.

Since the use of fast but non-committing encryption underlies the attack, a natural question — which we explore in the second part of the chapter — is whether it is possible to build secure ccAEAD schemes which match the efficiency of the fastest AEAD schemes such as GCM or OCB [99, 134, 137]. Such schemes are single-pass, and require only one block cipher application per block of message data processed. To answer this question, we first show that one can isolate the core technical challenge of building secure ccAEAD in a new primitive called *encryptment*. We demonstrate a connection between encryptment and collision resistant hash functions, which inspires both our negative and positive results on fast ccAEAD. This ability to exploit connections between areas of cryptography that might not immediately appear linked is a useful benefit of the provable security technique of formalising precisely what we would like a cryptographic primitive to achieve.

For the negative result, we first show that a broad class of encryptment schemes are also collision resistant (CR) hash functions. This allows us to use known impossibility results on high-rate collision resistant hashing [37,140] to rule out secure high-rate ccAEAD schemes. For a positive result, we use the classic Merkle-Damgård [52,109,110] construction of a CR hash function as a basis to build a secure encryptment scheme. Our construction is single-pass, and for certain instantiations incurs no additional overhead over simply hashing the message data. We then show how to lift encryptment to fully-fledged ccAEAD via simple and efficient transforms. Put together, this yields the first single-pass ccAEAD scheme.

Pseudorandom number generators with input. Having analysed a higher level cryptographic application, we devote the next two chapters to a vital lower-level cryptographic primitive. Pseudorandom number generators with input (PRNGs) convert a short high entropy seed into much larger quantities of pseudorandom bits, and offer strong security guarantees when given continual access to an entropy source. (The 'with input' here refers to a PRNG's ability to draw entropy samples from the source to update its state.) A secure PRNG is critically relied on by the vast majority of cryptographic applications, and so these primitives represent a natural subject for any analysis of real-world cryptography. Indeed, with much currently deployed cryptography being effectively 'unbreakable' when correctly implemented, exploiting a weakness in the underlying PRNG emerges as a highly attractive target for an attacker.

NIST SP 800-90A. Motivated by this, **Chapters 4** and **5** present a provable security treatment of two distinct flavours of PRNG attack. The inspiration for both chapters ultimately lies in the NIST Special Publication 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators (NIST SP 800-90A) [10]. The standard has had a troubled history due to the inclusion of the infamously backdoored Dual-EC-DRBG. Perhaps because of the focus on the Dual-EC, the other algorithms standardised in the document — HASH-DRBG, HMAC-DRBG, and CTR-DRBG — have received surprisingly little attention and analysis. Indeed, prior to our work, the procedures by which these PRNGs generate their initial state and subsequently refresh that state with entropy have never been analysed.

In **Chapter 4**, we address this gap in analysis by providing an in-depth investigation into the security of the remaining NIST SP 800-90A DRBGs, with a focus on HASH-DRBG and HMAC-DRBG. We pay particular attention to flexibilities in the specification of these algorithms, which are frequently abstracted away in previous analysis, and uncover a mixed bag of positive and less positive results.

The first part of the chapter presents a formal provable security analysis of the *robustness (for PRNGs)*[1] of HASH-DRBG and HMAC-DRBG. Introduced by Dodis et al. [60], robustness is the 'gold-standard' for PRNG security and encapsulates the security properties of backtracking and prediction resistance which the standard claims are achieved by each of the NIST DRBGs. As a (somewhat surprising) negative result, we demonstrate an attack showing that implementations of HMAC-DRBG which omit optional strings of additional input in output generation requests are not backtracking resistant, directly contradicting claims in the standard. We first discovered this attack while attempting to *prove* the backtracking resistance of HMAC-DRBG, which highlights the power of the provable security paradigm to uncover subtle flaws in cryptographic specifications. As positive results, we prove that HASH-DRBG and HMAC-DRBG (called with additional input) are robust in the random oracle model (ROM). The first result is fully general, while the latter is with respect to a class of entropy sources which includes those approved by the standard.

In the second part of the chapter, we counter these formal and (largely) positive results with a more informal discussion of flexibilities in the standard. We argue that when the NIST DRBGs are used to produce many blocks of output per request — a desirable implementation choice in terms of efficiency, and permitted by the standard — then the usual security models may overlook important attack vectors against these algorithms. We propose an informal security model in which we suppose an attacker compromises part of the state of the DRBG — for example through a side-channel attack — *during* an output generation request. Reconsidered within this framework, we find that each of the constructions admits vulnerabilities which allow an attacker to recover unseen output. These potential attacks do not contradict the positive robustness results given earlier in the chapter. Rather, they serve as a reminder that a security proof is only a guarantee with respect to the security model in question, and re-analysing security in the face of an attacker who lies outside the scope of that model may yield strikingly different results. We find a further flaw in a certain variant of CTR-DRBG which allows an attacker who compromises the state to also recover strings of additional input — which may contain secrets — previously fed to the DRBG. While our attacks are theoretical in nature, we follow this up with an analysis of the open-source OpenSSL and mbed TLS CTR-DRBG im-

[1]Robustness for PRNGs is a distinct notion from that of robustness for AEAD schemes, although both capture a scheme's ability to withstand certain kinds of misuse.

plementations, which shows that the implementation decisions we highlight as potentially problematic are taken by implementers in the real world.

Chapter 4 considers the type of attacker that was traditionally the focus in cryptographic literature prior to 2013. This attacker, presented with a cryptographic application, tries his best to break it. In contrast, **Chapter 5** considers a more powerful and insidious adversary than had been previously envisaged, who instead tries to compromise the process by which a cryptographic application is *designed*. Such an adversary — who we call 'Big Brother' — was thrust into the cryptographic consciousness in 2013 when the Snowden leaks confirmed long-held suspicions [153] that the Dual-EC-DRBG contained a backdoor inserted by the NSA [122]. It immediately became apparent that an entirely new class of attack models was urgently required in order to understand the potential damage incurred by this new threat and what can be done to protect against it.

Backdoored PRNGs. In **Chapter 5** we present a provable security treatment of backdoored pseudorandom number generators, thereby investigating a threat which the Dual-EC debacle demonstrated to be all too 'real world'. Given the aforementioned ubiquity of PRNGs in cryptographic implementations these constitute the ideal target for maximising the spread and impact of a backdoor — indeed, this was likely why NIST SP 800-90A was targeted for subversion — and so are a natural primitive to study in the context of such attacks. The question at the heart of our study is — to what extent can a PRNG be backdoored *and* provably secure?

In 2015, Dodis et al. [57] presented the first formal treatment of backdoored deterministic PRGs (i.e., a PRNG which, once seeded, evolves deterministically). An open problem from [57] is whether a PRG which is *forward secure* (a property which requires the PRG to preserve security of past output in the event of a state compromise) can be backdoored in a way that allows Big Brother to recover past output values. The apparent tension between the two notions, coupled with the prior lack of constructions, raises the question of whether forward secure PRGs have inherent resistance to this form of backdooring. In the first part of the chapter we resolve this problem in the negative by presenting two constructions of forward secure backdoored PRGs (BPRGs) which allow Big Brother to recover the initial PRG state (and hence *all* subsequent output) given an arbitrary public output.

In the second part of the chapter, we expand the scope of our analysis to present the first investigation of backdoored PRNGs with input (BPRNGs). We require that BPRNGs are robust in the face of a non-backdoor attacker, and — unsurprisingly — treating these more complex primitives requires definitions and constructions which are substantially more involved. Recall that, unlike a deterministic PRG, a PRNG may refresh its state with fresh entropy. As such, a particular challenge is ensuring that the exploitability of a backdoor can persist through a high entropy refresh. We navigate this tension by constructing a robust BPRNG which allows Big Brother to recover past output values with probability roughly $\frac{1}{4}$ up to a bounded number of high entropy refreshes. The number of refreshes is proportional to the size of the BPRNG state, raising as an open problem the question of whether the large state of our construction is inherent. We conclude the chapter with a discussion of the intuition and challenges behind proving such a result, and point out a critical flaw in an argument to this effect made in [54], the published version of this chapter.

Conclusion. We close this thesis in Chapter 6 with some conclusions and reflections on the results of the preceding chapters.

1.3 Associated Publications

This work is based on a number of co-authored publications. All authors contributed to discussions about these works, and many contributed to the write-up of the corresponding paper. As such, it is at times challenging to attribute an idea or contribution to a particular author. I contributed to the production of all sections of these works unless indicated otherwise, and shall attempt to highlight my key technical contributions here. However, I emphasise that these frequently evolved in conjunction with valuable guidance and input from other co-authors.

- Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. *Fast message franking: From invisible salamanders to encryptment. In Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I, pages 155–186, 2018* [59].

  This work is presented in Chapter 3. I was involved in the development of all sections of the paper and the definitions and results contained, except for the section detailing the Facebook attack. In particular I wrote all proofs, except those for the compression function-based encryptment-to-ccAEAD transform (although I made a significant contribution to their development). I wrote the version of these proofs which appear here (Section 3.7.4), correcting a number of errors that appeared in the version of these in [59] and adding more precise analysis of certain bad events to tighten the bound. I developed the sections on the relations and separations between robust encryption and sr-BIND secure ccAEAD (Section 3.7.6), and between r-BIND and sr-BIND secure ccAEAD (Theorem 3.6). I additionally contributed the section analysing the otCTXT security of the HFC encryptment scheme (Theorem 3.5 and surrounding text) and the result showing how to build encryptment from ccAEAD (Theorem 3.7.5).

- Joanne Woodage, Daniel Shumow. *An analysis of NIST SP 800-90A. In Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II, pages 151–180, 2019* [167].

  This work is presented in Chapter 4. Much of this project was completed while interning with Dan Shumow at Microsoft Research Redmond. Dan had the idea to analyse the NIST SP 800-90A standard, and highlighted the weakness of CTR-DRBG in the event of a side channel attack as an area to look into. I extracted a specification of the DRBGs from the standard (Sections 4.2-4.4), devised the robustness in the ROM security model (Section 4.5) and wrote all the security analyses and proofs (Section 4.6 and Section 4.7). In conjunction with Dan, I worked on the side channel attack security model and corresponding analysis (Section 4.8). I contributed the additional attack against CTR-DRBG implemented without a derivation function (Section 4.8.6), and Dan added the section on real world implementation analysis (Section 4.9).

- Jean Paul Degabriele, Kenny Paterson, Jacob Schuldt, and Joanne Woodage. *Backdoors in pseudorandom number generators: Possibility and impossibility results. In Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I, pages 403–432, 2016* [54].

  This work is presented in Chapter 5. I worked on definitions and security models in collaboration with the other authors, and contributed to the development of constructions proposed during this process. In particular, I worked on the proofs for the various BPRG and BPRNG constructions and wrote the final version of all of these except that for Lemma 5.4. I also contributed the relations and separations between PRG backdooring models (Section 5.3.4), and the proof that our modified preserving and recovering security notions imply robustness (Theorem 2.2, the proof of which appears in [54] and is omitted here since it is easily derived from the more involved proof of Theorem 4.1 which extends the result for the NIST DRBGs). This chapter is a significant re-write of the original [54], both to bring the presentation in line with the other thesis chapters and to fill in sketched or missing details and rectify small errors in the original paper. In particular, the sketched security argument for the simple backdoored PRNG (shown here in Section 5.4) which was given in [54] overlooked a number of important subtleties; I contributed the proof that appears here, the definitions of enhanced PRG security used therein, and showed that the backdoored PRGs of Sections 5.3.5 and 5.3.6 are suitable to instantiate it. During the production of this thesis, I discovered a significant error in an impossibility result stated in [54]; I developed a counterexample to the claimed result and wrote Section 5.4.6 describing the problem.

Also published during this PhD, but not included in this thesis:

- Joanne Woodage, Rahul Chatterjee, Yevgeniy Dodis, Ari Juels, and Thomas Ristenpart. *A new distribution-sensitive secure sketch and popularity-proportional hashing. In Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III, pages 682–710, 2017* [166].

- Rahul Chatterjee, Joanne Woodage, Yuval Pnueli, Anusha Chowdhury, and Thomas Ristenpart. *The TypTop system: Personalized typo-tolerant password checking. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pages 329–346, 2017* [45].

Chapter 2

Background & Preliminaries

Contents

2.1 Preliminaries

This chapter introduces a number of fundamental concepts that we will call on throughout this thesis. We first introduce notation. We then give an overview of the provable security paradigm. Finally, we recall a number of useful definitions relating to computational primitives, statistical notions, and pseudorandom number generators.

2.1.1 Notation

Strings and numbers. For an alphabet Σ, we let Σ^* denote the set of all strings of symbols from that alphabet, and let Σ^n denote the set of all such strings of length n. For a string $x \in \Sigma^*$, we write $|x|$ to denote the length of x. The set of binary strings of length n is denoted by $\{0,1\}^n$. We let $\{0,1\}^*$ denote the set of all binary strings, and let $\{0,1\}^{\leq \alpha}$ for $\alpha \in \mathbb{N}$ denote the set of all binary strings of length at most α. We include the empty string ε in both sets. We write $\{0,1\}^{n \leq i \leq \overline{n}}$ to denote the set of binary strings of length between n and $\overline{n}$-bits inclusive. We use $\perp$ to represent the null symbol.

We write $x \oplus y$ to denote the exclusive-or (XOR) of bit strings $x, y \in \{0,1\}^*$. If $|x| \neq |y|$, then we define XOR to return the XOR of the shorter string and the truncation of the longer string to the length of the shorter string. We write $x \,\|\, y$ to denote the concatenation of two binary strings x and y. For notational ease, we sometimes write $(x, y) \in \{0,1\}^m \times \{0,1\}^n$ to denote the string $x \,\|\, y \in \{0,1\}^{m+n}$. We let $\mathrm{len}(x)$ denote the length of a finite length binary string $x \in \{0,1\}^*$ in bits. The algorithm Parse_d is used to partition a binary string $x \in \{0,1\}^*$ into d-bit blocks. Formally, we define Parse_d to be the algorithm which on input x outputs $(x_1, \ldots, x_\ell)$ such that $|x_i| = d$ for $1 \leq i \leq \ell - 1$ and $|x_\ell| = |x| \mod d$. For correctness, we require that $x = x_1 \,\|\, \ldots \,\|\, x_\ell$. Similarly, we define Trunc_β to be the algorithm which on input x outputs the β leftmost bits of x. We sometimes write $\mathrm{left}(x, \beta)$ to denote the leftmost β bits of string x, and $\mathrm{select}(x, \alpha, \beta)$ to denote the substring of x consisting of bits α to β inclusive, where we index from 1 unless otherwise stated[1].

We convert binary strings to integers, and vice versa, in the standard way. For an integer $j \in \mathbb{N}$, we write $(j)_c$ to represent j encoded as a c-bit binary string. For $j \in \mathbb{N}$, we let $[j]$ denote the set of integers from 1 to j inclusive, and $[j_1, j_2]$ the set of integers between j_1 and j_2 inclusive. We let $\mathbb{N}^{\leq \alpha}$ denote the set of natural numbers up to and including α, $\mathbb{N}^{\leq \alpha} = [1, \ldots, \alpha]$.

[1]While left() and select() can be defined in terms of Trunc(), we include them here as explicit functions to maintain consistency with the presentation of algorithms in NIST SP 800-90A (Chapter 4).

Distributions. The notation $x \leftarrow_{\$} \mathcal{X}$ denotes sampling an element uniformly at random from the set $\mathcal{X}$, and we let $x \leftarrow X$ denote sampling a point according to the distribution X. For a pair of distributions X, Z, we write $X|Z = z$ to denote X conditioned on $Z = z$. We write $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$.

Algorithms. For a deterministic algorithm A, we write $y \leftarrow A(x_1, \dots)$ to denote running A on inputs $x_1, \dots$ to produce output y. For a probabilistic algorithm A with coin space $\mathcal{C}$, we write $y \leftarrow_{\$} A(x_1, \dots)$ to denote choosing coins $c \leftarrow_{\$} \mathcal{C}$ and returning $y \leftarrow A(x_1, \dots; c)$, where $y \leftarrow A(x_1, \dots; c)$ denotes running A on the given inputs with coins c fixed to deterministically produce output y. We assume that an algorithm returns the error symbol $\perp$ if called on an input that lies outside its defined input space. Our proofs assume a RAM model of computation where most operations are unit cost. We sometimes use big-O notation $\mathcal{O}(\cdot)$ to hide small constants. When referring to run times in theorem statements and proofs, we sometimes write $T' \approx T$ to indicate that T' is equal to T plus a minor overhead (e.g., to simulate oracle queries) which can be derived from the proof. All logs are to base 2.

2.2 Provable Security

In this section, we give an overview of the provable security paradigm. Much of the discussion on provable security and its evolution is drawn from Katz and Lindell [86] and essays on practice-oriented provable security by Bellare [13] and Rogaway [132], to which the reader is referred for a more detailed discussion of the points raised.

Analysis by cryptanalysis. 'Security' is a concept that we all intuitively understand, and yet it can be surprisingly difficult to write down a sentence stating what security actually *is*. We have secure prisons and secure relationships, financial security and national security, secure foundations and a false sense of security, each of these representing a different variation on the same theme. What constitutes security in a given setting is highly context-dependent, and it is clear that security in one context does not necessarily imply security in another.

With these subtleties surrounding security in mind, imagine now that we have designed a cryptographic scheme and would like to know if it is 'secure'. Prior to the introduction of provable security in the 1980s, the approach to this would be to try and break, or *cryptanalyse*, the scheme. If a successful attack was discovered, the scheme would be tweaked to prevent it. Finally, when no more attacks could be found, the scheme would be declared secure.

There are a number of disadvantages to this approach. Firstly, any assurance derived from it may be dangerously misplaced, as it is precisely the set of attacks which had *not* been anticipated by the designer that we should be most concerned about. History bears out the fact that it can take a long time for an insecure scheme to be successfully crypt-analysed; for example, the Chor-Rivest knapsack-based encryption scheme [49] resisted cryptanalysis for a decade despite all other knapsack schemes being broken, before finally being demonstrated insecure by Vaudenay [162]. Moreover, this approach fails to pin down what security actually *means* in our context, and so we get no clear characterisation of the security properties our scheme is aiming for (let alone achieving). Rather than ruling out individual attacks, it would be far more satisfying to have a guarantee that our scheme withstands *any attack* in a given class. Such a guarantee is precisely what the provable security paradigm aims to provide.

Provable security. The provable security paradigm, beginning with the ground breaking 1982 paper of Goldwasser and Micali [72], represented a radical shift in assessing the security of a cryptosystem. Instead of relying on intuition to argue that a scheme is secure by dint of resisting all the attacks we can come up with, provable security draws on proof techniques from theoretical computer science to construct a rigorous argument that a scheme achieves a well-defined security property. As we shall see, calling such an argument a 'proof' can be seen as a misnomer since our argument will typically rely on unproven assumptions. However, by basing our analyses on a fairly small set of assumptions we can focus our cryptanalytic efforts on these rather than attempting to cryptanalyse each scheme individually. Each advance in analysing an assumption can then be extrapolated to increase our understanding of *all* the schemes that base their security upon it.

Principles of provable security. The three key principles upon which provable security is built are: **(1)** formal definitions; **(2)** precise assumptions; and **(3)** security proofs. We elaborate on each of these principles below, and lay out the steps by which a provable security analysis of a scheme is constructed.

(1) *Formal definitions.* Before we can possibly hope to prove anything about the security of our scheme, we must specify what we would like our scheme to achieve. As such, formal security definitions are at the heart of any provable security analysis. We express security models in the form of a *game* between a challenger and an attacker $\mathcal{A}$ which dictates the rules by which the attacker may interact with and / or compromise the scheme. The attacker is modelled as a randomised algorithm with a specified amount of *resources*, such as run time or oracle queries. In particular, if the attacker has no upper limit on their run time they are said to be *unbounded* and the security analysis is *information theoretic*; otherwise, the attacker is said to be *computationally bounded* and the analysis is *computational.*

(2) *Precise assumptions.* Our analysis will typically be with respect to a number of assumptions; that is to say, unproven statements — such as that a block cipher is computationally indistinguishable from a random permutation, or that finding a solution to a certain mathematical problem is hard. These assumptions are likely to remain unproven for the foreseeable future, since resolving them lies well beyond our current understanding of complexity theory. Basing our analysis on unproven statements is clearly not ideal, and may seem at odds with the rigour we hope to achieve. However by choosing assumptions that are well-studied and widely believed, we can, via a proof, relate our scheme (about which nothing is currently known) directly to problems that have been extensively researched and analysed. As such, we derive assurance in the security of our scheme from our assurance in the soundness of the assumptions.

(3) *Security proofs.* With definitions and assumptions in place, we will construct a formal proof that a scheme satisfies the given definition with respect to the assumptions. For computational results in particular, security proofs are characterised by cryptographic *reductions*; arguments that 'reduce' the security of a scheme to that of an assumption by demonstrating that any attacker $\mathcal{A}$ who can win the security game specified in **(1)** can be used to construct an attacker $\mathcal{A}'$ that violates one of the assumptions X specified in step **(2)**. Provided that no algorithm with $\mathcal{A}'$'s resources is believed to be able to violate assumption X, this implies, by contradiction, that no attacker $\mathcal{A}$ can reliably win the security game against the scheme.

Impact. The power of this approach is that in one fell swoop our argument excludes the possibility of *any* attack strategy within the scope of the security model succeeding — including those the designer of the scheme may never have anticipated. This remarkable shift from empirically arguing that specific attacks won't work to formally excluding well-defined classes of attack is the defining feature of the provable security paradigm. To get the most analytical bang for our buck, we will choose security models carefully so that the class of attacks covered is very broad. For a classic example of this, imagine a public key encryption (PKE) scheme for which an attacker who sees a ciphertext produced by the scheme learns *nothing* about the underlying message that they didn't know *a priori*. If we can prove a PKE scheme meets such a property, we no longer have to repeatedly cryptanalyse it to see if it prevents e.g., full plaintext recovery, learning the least significant bit of the message, or whether that message is equal to 'sell' or 'buy', since *all* these security goals (and millions more) are encapsulated by our definition. This incredibly strong security property — that at first sight might seem an impossibly high bar to meet — is called *semantic security*, and is exactly that targeted by Goldwasser and Micali [72]. By demonstrating a scheme that provably achieves it, they gave the first glimpse of the breakthroughs that this new formulation of cryptography would produce.

2.2.1 Practice-Oriented Provable Security

Theory and practice. While the advent of provable security transformed cryptography from an art into a science, at first few of the advances being made in conference papers seemed to be impacting real world systems. In fact, the way problems were treated in the provable security literature was somewhat at odds with how cryptographic schemes were being constructed in the real world. Cryptographers dreamed up fantastic problems that were of great theoretical interest but far removed from the nuts and bolts of real systems. Similarly, schemes were designed with the aim of proving their security under the most minimal assumptions possible, such as merely assuming the existence of one-way functions. While this produced remarkable feasibility results, the resulting schemes were often far too inefficient and convoluted for practical use. Valued for their versatility and efficiency, block ciphers formed the basis of most real world schemes and yet no assumption for modelling block cipher security existed; indeed, symmetric cryptography as a whole was generally overlooked.

Asymptotic vs. concrete security. There were also issues with how security results were framed. Drawing on complexity theory, early provable security took an asymptotic approach in which both scheme and attacker are parameterised by λ taken to be e.g., the size of an RSA modulus. A scheme is declared secure with respect to a definition if no attacker running in time polynomial in λ can achieve an advantage that is non-negligible in λ. The problem was that for many real world (and in particular symmetric) schemes all parameters are fixed by the object's specification, and there is no security parameter to play with to increase or decrease the level of security achieved. For example, while key size is the natural security parameter for a block cipher, AES can only take 128-bit, 192-bit, or 256-bit keys. More problematic still is that the block size for all these variants is 128-bits and so cannot be scaled with key size. This precludes security bounds that contain a term capturing e.g., the probability of blocks colliding from being negligible in the security parameter for larger key sizes. More generally, it can be tricky to work out what level of security is actually achieved for a particular instantiation of a scheme from an asymptotic bound. In practice, we would often like a *concrete* statement that says that an attacker running in time T will be unable to break a given instantiation of the scheme with probability greater than ϵ.

Bridging the gap. Out of a desire to bridge the gap between theory and practice, a new strand of provable security emerged that would come to be known as *practice-oriented provable security* [13, 132]. Spearheaded by Bellare and Rogaway, this line of work is characterised by an emphasis on analysing cryptography as it is used in the real world, with the intention of yielding practically useful results and influencing best practice.

This line of research is best defined by contrasting it with the first wave of provable security as described in the previous paragraph. Practice-oriented provable security takes cryptography as used in the real world as inspiration for problems to study. Schemes are designed with the aim of being practical and efficient, as opposed to being based on minimal assumptions, and impact is measured not only in conference publications, but also adoption and standardisation of schemes. Symmetric cryptography, in line with its abundance in real world systems, is a key target of analysis, and there is a willingness to accept well-defined heuristics — such as the random oracle model (see Section 2.3.3) — when this enables the analysis of highly practical schemes. Results are stated concretely, and cryptanalysis to find the best possible attack goes hand in hand with security proofs to establish how closely security bounds match the success of a concrete attack. (The gap between a bound and the best known attack is called the *tightness* of the bound. Ideally, we would like a matching attack for each bound, although this can be difficult to achieve.)

Practice-oriented provable security has had a resounding real-world impact, from producing the now-ubiquitous HMAC [15] to influencing the design of TLS 1.3 [82, 98, 120]. In keeping with our focus on real world problems, we take a practice-oriented approach throughout this thesis.

2.2.2 Provable Security in Perspective

While the introduction of provable security has been revolutionary, firmly cementing cryptography as a rigorous branch of science and producing spectacular theoretical and practical results, it is not a panacea. In particular, a positive provable security analysis certainly does not imply that every implementation of the scheme in question will be secure.

Limitations. It is important to keep in mind that a proof of security is only a guarantee with respect to the model and assumptions against which the scheme was analysed. The proof offers no protection if the assumption is later found to be unsound, or the scheme is subjected to attacks outside the model. When a formal security analysis is a factor in selecting a scheme for use in a real world application, care must be taken to ensure that the security analysis is sufficient to capture the threats the scheme may be subjected to, and that the assumptions are ones we trust. Moreover, threats such as side channel and fault injection attacks, and of course implementation errors, can result in an implementation of a provably secure scheme being completely insecure in practice. This does not contradict the security proof; rather, the model was not intended (and does not claim) to cover such threats. It is important to interpret a provable security analysis as just one thread in the design and implementation of a secure system, and there are many further steps that must be taken to translate a scheme that is secure on paper into a secure implementation.

Criticism and counterarguments. Provable security has come in for particular criticism from Koblitz and Menezes in their "Another Look at . . ." series of papers (see [92]), who argue that the scientific rigour provable security has brought to cryptography is a façade. They highlight examples of non-tight reductions that are useless in practice because they lead to wildly inflated and impractical parameter settings, and schemes falling victim to attacks outside the security model within which they were analysed. They also question its utility, arguing that the paradigm has lead to schemes being designed to facilitate a proof, leading to unnatural constructions [93].

These are certainly valid criticisms, and would be echoed to some extent by many whose work falls under the umbrella of provable security. However, using these as a basis to dismiss security proofs altogether is rather throwing the baby out with the bathwater. As discussed in the paragraph above, a provable security analysis does not claim anything with respect to attacks outside the model, but is very useful to give a precise formulation of the attacks that *are* covered. Practice-oriented provable security in particular has yielded a lot of practical and intuitive schemes that have enjoyed widespread adoption, and moreover, one can never be sure if a seemingly unnecessary design feature required to construct a proof might actually enable a subtle attack if removed. It is universally agreed that tight reductions are preferable to non-tight, as evidenced by the considerable amount of work that sets out to improve the tightness of previous bounds. However even when a proof is not tight enough to be useful in setting parameters, it is still of value to give assurance that the design is sound and to build understanding of why the construction works (and indeed, why the non-tight bound is hard to avoid).

Koblitz and Menezes also criticise security proofs as having a "credibility problem" [93] due to a lack of careful validation of proofs, giving as a high profile example an analysis of OAEP by Bellare and Rogaway [23] that was public for many years before Shoup [148] found a mistake in the proof that was significant enough to require either modifying the design or restricting the result in order to recover the claim. Koblitz and Menezes contrast this state of affairs to that in mathematics, in which proofs — including 200-page epics such as Andrew Wiles' proof of Fermat's Last Theorem [165] — are closely scrutinised and results only cautiously accepted until the proof has been validated by the community. (A small and easy-to-miss flaw was indeed found in Wiles' result, and a corrected proof was subsequently published by Taylor and Wiles the following year [156].)

It is certainly alarming how frequently errors are uncovered in publications (including those at the most prestigious conferences). Fortunately, most errors are small and easily fixed, but there are examples — such as the recent spate of attacks [81] that devastated OCB2 [134] — in which a well known (and in this case, standardised) scheme was much later found to be totally broken following a bug in the original security proof. It seems likely that the conference model of publication in cryptography — in which deadlines are rushed for, page limits too small to contain proofs, and reviewers highly overstretched — is a major contributory factor to this problem, inducing a tendency towards proofs that may be badly written or merely sketched, and papers accepted for publication without the proofs ever being closely reviewed.

We do not wish to denigrate the many fastidiously analysed results published every year, or the monumental efforts of reviewers faced with a stack of submissions. However, security proofs can be difficult and delicate (indeed, the flaw in the OCB2 proof is highly subtle), and if the field is to uphold its claimed rigour then a shift is needed to place greater value on the proofs after which it is named. On the author side, proofs should be treated as an integral part of a paper and a way to illuminate the ingenuity of a design, rather than an inconvenience to be endured or rushed off. We should endeavour to write proofs with care and sufficient detail to make verification as pain free as possible. That said, even best efforts are not infallible and so on the other side of the coin it would be beneficial to reduce the burden on reviewers to allow them more time to check the technical details of results. It is reassuring to see conferences moving to a journal model of publication in which submissions can be staggered throughout the year and submitted when they are ready. Looking forward, we hope that in the future machine checked proofs can relieve some of the human effort and error in writing and checking proofs.

Summary. While provable security has its shortcomings, to disregard it because of these is to overlook the many breakthroughs and innovations it has produced, and any return to assessing security solely in terms of resistance to cryptanalysis is a clear backwards step. By imposing order on the previously ad hoc nature of cryptographic design, provable security has given us a language with which to reason about what level of security a scheme achieves, and the richness of the field, spanning highly theoretical results to real world success stories, is a testament to its impact. Keeping its shortcomings in mind, constantly questioning and evolving our approaches to analyses, and working in tandem with cryptanalysis, can only make the field stronger.

2.2.3 Security Models and Proofs

Security models and formal proofs are defining features of the provable security paradigm, and we will present many throughout this thesis. This section provides an overview of these and introduces the game-playing methodology; the reader is referred to e.g., [25] for a fully detailed treatment of the subject.

Security games. We express security notions in the form of a *security game* G between a challenger and a randomised algorithm $\mathcal{A}$ that we call the *attacker*. The specification of the game dictates precisely what they attacker may learn, how they may interact with the scheme, what restrictions are in place, and so on. The challenger, typically implicit in games rather than written explicitly, sets up the experiment by e.g., generating keys, and then executes the game with the attacker according to the specification. During a game, an attacker $\mathcal{A}$ may be given access to one or more oracles $\mathcal{O}_1, \ldots, \mathcal{O}_k$; that is to say, black boxes to which $\mathcal{A}$ may submit inputs and receive back responses, and which we use to model different ways in which $\mathcal{A}$ may interact with the scheme. We indicate that $\mathcal{A}$ has access to such oracles by writing $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_k}$. We carefully quantify the resources, e.g., run time or number of oracle queries, available to $\mathcal{A}$. At some point the game terminates and returns a value; typically a bit $b' \in \{0,1\}$. We write $G \Rightarrow x$ and $\mathcal{A} \Rightarrow x$ to indicate that, respectively, the game G and attacker $\mathcal{A}$ returned x. The game G may use *flags* to indicate whether an event has occurred, or maintain sets or look-up tables to keep track of attacker queries. We assume that all flags are initialised to false, all sets are initialised to the empty set $\emptyset$, and that look-up tables have all entries initialised to $\perp$.

Advantage terms. To each game, we associate an *advantage term*, which quantifies how well $\mathcal{A}$ has performed in that game. This is typically framed as a comparison to some naive strategy such as simply guessing or not trying at all. Security games fall broadly into two classes called *unpredictability* and *indistinguishability* games. In an unpredictability game, the attacker is challenged to solve some problem instance for a scheme P, for example, successfully inverting a trapdoor permutation applied to a uniform domain point (see Section 5.2.2). The game returns true if the attacker returns the correct answer. For an unpredictability game G against a primitive P, we define the advantage of an attacker $\mathcal{A}$ to be

$$\mathbf{Adv}_P(\mathcal{A}) = \Pr\left[\, G_P^{\mathcal{A}} \Rightarrow 1 \,\right] ,$$

where $G_P^{\mathcal{A}}$ denotes attacker $\mathcal{A}$ playing game G against primitive P. The probability is over the coins of $\mathcal{A}$ and the game.

An indistinguishability game challenges an attacker to distinguish a scheme P from its idealised functionality, for example, distinguishing ciphertexts produced by an AEAD scheme from random bit strings of appropriate length (see Section 3.2.2). We begin by defining an experiment capturing the attacker's interaction with the scheme P. The game G then tosses a coin $b \leftarrow_{\$} \{0, 1\}$ to decide whether to run the attacker against this 'real' experiment, or an idealised version in which P is replaced with its idealised counterpart. The attacker's goal is to work out whether they are interacting with the real or ideal scheme; the game returns true if they guess correctly, and the attacker's advantage is defined to be how much better in this task they perform than simply guessing. More formally, for such a game G against a primitive P we define the advantage of an attacker $\mathcal{A}$ to be

$$\mathbf{Adv}_P(\mathcal{A}) = 2 \cdot \left| \Pr\left[\, G_P^{\mathcal{A}} \Rightarrow 1 \,\right] - \frac{1}{2} \right| ,$$

where again $G_P^{\mathcal{A}}$ denotes $\mathcal{A}$ playing game G against primitive P, and the probability is over the coins of $\mathcal{A}$ and the coins used by the game. We sometimes rewrite this advantage term in the form:

$$\mathbf{Adv}_P(\mathcal{A}) = \big|\Pr\left[\, \mathcal{A} \Rightarrow 1 \text{ in } G \mid b = 0 \,\right] - \Pr\left[\, \mathcal{A} \Rightarrow 1 \text{ in } G \mid b = 1 \,\right]\big| .$$

It is straightforward to verify that the two formulations are equivalent and we use them interchangeably.

2.2.4 Game Hopping Proofs

Our security proofs use a technique called *game hopping*. To analyse a game G against an attacker $\mathcal{A}$ and primitive P, a game hopping proof defines a finite series of games, $G_0, \ldots, G_n$ for some $n \geq 1$, where all games are implicitly parameterised by P and $\mathcal{A}$. Game G_0 is typically identical to $G_P^{\mathcal{A}}$, and for each $i \in [1, n]$, game G_i is a modification of G_{i-1}, where the modifications are chosen carefully so as to not alter the probability of $\mathcal{A}$ winning the game by too much. We will make a formal argument to bound the 'gap' between each pairs of games; more formally, to establish an upper bound on the term

$$\left|\Pr\left[\, G_i \Rightarrow 1 \,\right] - \Pr\left[\, G_{i-1} \Rightarrow 1 \,\right]\right| .$$

For example, this argument may take the form of a computational reduction to the security of an underlying primitive, a statistical argument bounding e.g., a collision probability, or a justification that the second game is simply a rewriting of the first (in which case the gap between the games is zero). We may then use the triangle inequality to combine the bounds on each game hop into a single expression:

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_n \Rightarrow 1]| \leq \sum_{i=0}^{n-1} |\Pr[G_i \Rightarrow 1] - \Pr[G_{i+1} \Rightarrow 1]| .$$

For a game hopping argument to be useful we must relate the series of games to the probability that we wish to bound. For an unpredictability game, we typically use game hops to move to a game in which the probability that $\mathcal{A}$ succeeds is easy to analyse, and preferably small. For an indistinguishability game, we will use our series of game hops to transition from the experiment against the real scheme to that against its ideal functionality, thereby bounding the gap between them via the series of transitions.

The Fundamental Lemma of Game Playing. We will frequently invoke a useful result called The Fundamental Lemma of Game Playing [25, 149]. Consider a pair of games $G^{\mathcal{A}}$ and $H^{\mathcal{A}}$ with associated adversary $\mathcal{A}$, the pseudocode for which includes a flag bad that will be set to true only if some event E occurs in either game. For example, the event E might be that $\mathcal{A}$ queries a certain point to an oracle, or that when randomly sampling keys we accidentally choose the same key twice. Suppose that $G^{\mathcal{A}}$ and $H^{\mathcal{A}}$ are identical *except* they differ in a conditional that is only executed if the flag bad is set. In this we case, we say that $G^{\mathcal{A}}$ and $H^{\mathcal{A}}$ are *identical-until-bad.* (A more rigorous definition of identical-until-bad games is given in [25]; however, as the authors note, when written in pseudocode it is usually obvious which games are identical-until-bad, and so the additional formalism is not really necessary.) We write 'bad $= 1$ in game $G^{\mathcal{A}}$' (resp. $H^{\mathcal{A}}$) to denote that the flag bad is set to true at the conclusion of game $G^{\mathcal{A}}$ (resp. $H^{\mathcal{A}}$).

The Fundamental Lemma of Game Playing [25, 149] says that an adversary's maximum advantage in distinguishing a pair of identical-until-bad games is upper bounded by the probability that bad is set in either game. This is formalised in the following lemma, which we recall here following [25]. Since distinguishing advantage can be defined in terms of both game output and adversary output, the lemma contains a statement for each case. Note that setting $I = G$ or $I = H$ in the lemma allows the probability of bad being set in either game to be used for the upper bound.

Lemma 2.1 (The Fundamental Lemma of Game Playing)**.** *Let $G^{\mathcal{A}}$, $H^{\mathcal{A}}$, and $I^{\mathcal{A}}$ be identical-until-bad games against an attacker $\mathcal{A}$. Then*

$$\left|\Pr\left[\,G^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,H^{\mathcal{A}} \Rightarrow 1\,\right]\right| \leq \Pr\left[\,\mathsf{bad} = 1 \text{ in } I^{\mathcal{A}}\,\right] \text{ ; } \textit{and}$$

$$\left|\Pr\left[\,\mathcal{A} \Rightarrow 1 \text{ in } G^{\mathcal{A}}\,\right] - \Pr\left[\,\mathcal{A} \Rightarrow 1 \text{ in } H^{\mathcal{A}}\,\right]\right| \leq \Pr\left[\,\mathsf{bad} = 1 \text{ in } I^{\mathcal{A}}\,\right] .$$

The H-coefficient technique. Patarin's H-coefficient technique [119] has proved to be a highly useful tool for analysing indistinguishability games. We recall the formulation of this method from [48] below, and refer the reader to that work for a full discussion of the approach.

The proof technique consider two experiments, which we denote Real and Ideal, and a deterministic and computationally unbounded adversary $\mathcal{A}$ who tries to distinguish the two. We define a *transcript* which captures $\mathcal{A}$'s view of the experiments and say that a transcript is *valid* if it could be produced by an execution of one of the experiments. We let T_0 and T_1 denote the distributions of valid transcripts corresponding to the real and ideal experiments respectively. We additionally define a set of Bad transcripts, and view all other valid transcripts as being Good. The following theorem then allows us to bound the advantage of $\mathcal{A}$ in distinguishing the real and ideal experiments:

Theorem 2.1. *Suppose that there exist $\delta, \epsilon \in [0, 1]$ such that for all transcripts $\tau \in \mathsf{Good}$ it holds that*

$$\Pr\left[\,\mathsf{T}_0 = \tau\,\right] / \Pr\left[\,\mathsf{T}_1 = \tau\,\right] \geq 1 - \epsilon \ ,$$

and moreover it holds that $\Pr\left[\,\mathsf{T}_1 \in \mathsf{bad}\,\right] \leq \delta$. Then:

$$\left|\Pr\left[\,\mathcal{A} \Rightarrow 1 \text{ in } \mathsf{Real}\,\right] - \Pr\left[\,\mathcal{A} \Rightarrow 1 \text{ in } \mathsf{Ideal}\,\right]\right| \leq \epsilon + \delta \ .$$

Looking ahead, we will call on Patarin's H-coefficient technique when analysing $\mathsf{HASH\text{-}DRBG}$ in Section 4.6.

This concludes our overview of provable security and the game playing technique. In the next section, we introduce a number of key cryptographic primitives.

2.3 Computational Building Blocks

Cryptographic *primitives* are low-level cryptographic objects that do not in themselves have a direct practical application but are the building blocks from which higher level schemes are built. Indeed in [13], Bellare memorably describes a cryptographer as “an engine for turning atomic primitives into protocols”. In this section, we define a number of the primitives that will be used in constructions throughout this thesis.

2.3.1 Pseudorandom Functions and Block Ciphers

A *pseudorandom function* is a keyed function that, to an attacker who is given oracle access to the function keyed with a secret random key, is indistinguishable from a truly random function with the same domain and range. More formally, let $\mathrm{Func}(\mathit{Dom}, \mathit{Rng})$ denote the set of all functions $F : \mathit{Dom} \to \mathit{Rng}$. The *pseudorandom function* (PRF) distinguishing advantage of an adversary $\mathcal{A}$ against a keyed function $\mathsf{F} : \mathcal{K} \times \mathit{Dom} \to \mathit{Rng}$, given q oracle queries, is defined as

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}, q) = \left| \Pr\left[\mathcal{A}^{\mathsf{F}(K,\cdot)} \Rightarrow 1 : \; K \xleftarrow{\$} \mathcal{K} \right] - \Pr\left[\mathcal{A}^{F(\cdot)} \Rightarrow 1 : F \xleftarrow{\$} \mathrm{Func}(\mathit{Dom}, \mathit{Rng}) \right] \right| ,$$

where recall that the superscript denotes a functionality that $\mathcal{A}$ is given oracle access to. We call $\mathcal{K}$ the key space of the scheme, and typically take $\mathcal{K} = \{0,1\}^{\kappa}$ for some $\kappa \in \mathbb{N}$.

In Chapter 3, we require a PRF variant in which the key constitutes the second input to the function (as opposed to its first input, as in the definition above). In this case, we define the PRF to be a function $\mathsf{F} : \mathit{Dom} \times \mathcal{K} \to \mathit{Rng}$. Security for PRFs which are keyed on their second input is defined entirely analogously to that for regular PRFs.

Lazy sampling. A useful proof technique that we frequently employ is *lazy sampling* a random function. Intuitively, this means that instead of choosing a random function $F \leftarrow_{\$} \mathrm{Func}(\mathit{Dom}, \mathit{Rng})$ and thereby fixing its specification at the start of an experiment, we instead fill in the function table gradually, setting the images of points under F as they are needed. In more detail, we initialise the function table of F by setting $F[X] = \perp$ for all $X \in \mathit{Dom}$. When we require the value of F on some domain point X, we check if $F[X] \neq \perp$ and return the stored value if so. If not, we choose $Y \leftarrow_{\$} \mathit{Rng}$, set $F[X] = Y$, and return Y. (In a proof, it may be more convenient to rewrite these steps to e.g., sample $Y \leftarrow\!\!\$\ \mathit{Rng}$ upfront and revert to the stored value if $F[X]$ has already been set. At such points, it is easy to check that the steps taken are equivalent to those described here.)

It is straightforward to verify that replacing a random function with a lazy sampled random function does not alter the distribution of the game in which the function is being used. Lazy sampling a function can be useful in proofs since it allows us to subsequently define games in which the function is modified such that e.g., outputs are sampled without replacement, or repeat queries are no longer answered consistently.

Block ciphers. A *block cipher* is a PRF which is additionally a permutation for each $K \in \mathcal{K}$. Block ciphers are often described as the 'work horse' of modern cryptography, since these simple primitives are a remarkably versatile building block in constructions of more complex cryptographic primitives. In later chapters we will see how collision-resistant hash functions, authenticated encryption, and pseudorandom number generators can be constructed from block ciphers.

Formally, a block cipher is a function $\mathrm{E} : \mathcal{K} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ such that $\mathrm{E}(K,\cdot)$ is a permutation over $\{0,1\}^{\ell}$ for each $K \in \mathcal{K}$. We let $\mathrm{D}(K,\cdot)$ denote the inverse of $\mathrm{E}(K,\cdot)$, and so $\mathrm{D}(K,\mathrm{E}(K,X)) = X$ for all $K \in \mathcal{K}$ and $X \in \{0,1\}^{\ell}$. We sometimes write $\mathrm{E}_K(\cdot)$ to denote computing $\mathrm{E}(K,\cdot)$. We require that block ciphers are good *pseudorandom permutations* (PRPs). PRP security is defined analogously to PRF security, except we now challenge the attacker to distinguish the primitive in question from a random permutation $\pi \leftarrow\!\!\$\ \mathrm{Perm}(\{0,1\}^{\ell})$, where $\mathrm{Perm}(\mathit{Dom})$ denotes the set of all permutations over domain *Dom*. Formally, the PRP distinguishing advantage of an adversary $\mathcal{A}$ against a block cipher $\mathrm{E} : \mathcal{K} \times \{0,1\}^{\ell} \to \{0,1\}^{\ell}$, given q oracle queries, is defined

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{E}}(\mathcal{A}, q) = \left| \Pr\left[\mathcal{A}^{\mathrm{E}(K,\cdot)} \Rightarrow 1 : K \xleftarrow{\$} \mathcal{K} \right] - \Pr\left[\mathcal{A}^{\pi(\cdot)} \Rightarrow 1 : \pi \xleftarrow{\$} \mathrm{Perm}(\{0,1\}^{\ell}) \right] \right| .$$

Tweakable block ciphers. Since a block cipher is deterministic, there are limits on how much data can be processed using a block cipher with a fixed key. At the same time, frequently re-keying a block cipher typically has a negative impact on efficiency. In order to increase the mileage we can get out of each block cipher key, Liskov et al. [102] introduced the notion of a *tweakable block cipher* (TBC). A TBC extends the syntax of a block cipher to have E take an extra non-secret input called a *tweak*. For a key $K \in \mathcal{K}$, we would like $\mathrm{E}(K, T, \cdot)$ to behave like an independent random permutation under each tweak $T \in \mathcal{T}$. Moreover, it should be less costly to change the tweak than it is the key.

Formally, a TBC is a function $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^\ell \to \{0,1\}^\ell$ such that $\widetilde{E}$ is a permutation for all $(K, T) \in \mathcal{K} \times \mathcal{T}$. We let $\widetilde{\mathrm{D}}(K, T, \cdot)$ denote the inverse of $\widetilde{E}(K, T, \cdot)$, and so $\widetilde{\mathrm{D}}(K, T, \widetilde{E}(K, T, X)) = X$ for all $(K, T) \in \mathcal{K} \times \mathcal{T}$ and $X \in \{0,1\}^\ell$. We sometimes write $\widetilde{E}_K^T(\cdot)$ to denote computing $\widetilde{E}(K, T, \cdot)$. The TBC distinguishing advantage of an adversary $\mathcal{A}$ against a TBC $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^\ell \to \{0,1\}^\ell$, given q oracle queries, is defined

$$\mathbf{Adv}^{\text{tbc-prp}}_{\widetilde{E}}(\mathcal{A}, q) = \left| \Pr\left[\mathcal{A}^{\widetilde{E}(K,\cdot,\cdot)} \Rightarrow 1 \ : \ K \leftarrow\!\!\$\, \mathcal{K} \right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1 \right] \right| ,$$

where the oracle $\$(\cdot, \cdot)$ chooses a random permutation $\pi_T \leftarrow\!\!\$\, \mathrm{Perm}(\{0,1\}^\ell)$ for each $T \in \mathcal{T}$, and on input (T, X) returns $\pi_T(X)$. We note that this definition allows the adversary $\mathcal{A}$ to control the choice of tweaks.

2.3.2 Hash Functions

A *hash function* $\mathsf{H} : \mathit{Dom} \to \{0,1\}^\ell$ is a deterministic function mapping inputs from a domain Dom (typically containing inputs of varying lengths) to strings of some fixed length ℓ. When a hash function satisfies certain security properties, they are highly useful cryptographic building blocks. In this section, we recall the notion of collision-resistant hash functions.

Collision-resistance. Let $\mathsf{H} : \mathit{Dom} \to \{0,1\}^\ell$ be a function on domain $\mathit{Dom} \subseteq \{0,1\}^*$. H is said to be *collision-resistant* (CR) if it is infeasible for an attacker to find distinct domain points $X \neq X' \in \mathit{Dom}$ which map to the same value under H. If analysis is with respect to an ideal primitive such as an ideal permutation (see Section 2.3.3) then $\mathcal{A}$ is given oracle access to this primitive also. More formally, the CR advantage of an adversary $\mathcal{A}$ against H is defined

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(\mathcal{A}) = \Pr\left[X \neq X' \wedge \mathsf{H}(X) = \mathsf{H}(X') \ : \ X, X' \leftarrow\!\!\$\, \mathcal{A} \right] ,$$

where we require that $\mathcal{A}$ outputs $X, X' \in \mathit{Dom}$. We will measure the efficiency of the attacker in terms of their resources; typically run time (when working in the standard model) or number of oracle queries (when in an idealised model).

The contradiction of unkeyed CR. Declaring a function $\mathsf{H} : \mathit{Dom} \to \{0,1\}^\ell$ for which the domain Dom contains more than 2^ℓ points 'collision-resistant' in the standard model is technically incorrect. Indeed, for any candidate hash function H we know by the pigeonhole principle that there must exist a pair of points $X, X' \in \mathit{Dom}$ for which $\mathsf{H}(X) = \mathsf{H}(X')$. Since H is unkeyed and does not utilise an idealised primitive (to be chosen at random in the CR game), we can immediately define a collision-finding attacker $\mathcal{A}$ to be the algorithm with X, X' hard-wired that simply outputs these values and achieves $\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(\mathcal{A}) = 1$. However, the existence of such an attacker does not necessarily imply that the function is insecure in practice, since writing down the code of $\mathcal{A}$ requires knowing a collision for H, and there are many real-world hash functions (e.g., SHA-256) for which it is widely believed that no one on earth knows such a collision at this point in time.

The way to escape this paradox, as formalised by Rogaway in [135], is to carefully construct theorem statements and proofs in which we reduce the security of a protocol P based on an unkeyed hash function H to the collision-resistance of H. We cannot say that for any efficient attacker $\mathcal{A}$ achieving a good advantage in game G against protocol P there exists a collision finding attacker $\mathcal{A}'$ for which $\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(\mathcal{A}')$ is large, since as just discussed such an $\mathcal{A}'$ *always* exists. However, we may be able to prove that for any such $\mathcal{A}$ we can construct an *explicit* attacker $\mathcal{A}'$ achieving large $\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(\mathcal{A}')$. If this is the case, and moreover it is widely believed that no one currently has the power to specify such a collision-finding algorithm with the run-time implied by the reduction, then this in turn contradicts the existence of an adversary $\mathcal{A}$ breaking the security of P with respect to game G. We take this approach to analysing unkeyed CR-hash functions in this thesis and (in line with common usage) abuse terminology to refer to such functions as CR with the understanding that the assumption we are making is that no efficient collision algorithm can currently be constructed, rather than that no such algorithm exists.

Compression functions. If $\mathsf{H} : \mathit{Dom} \to \{0,1\}^\ell$ is such that $\mathit{Dom} = \{0,1\}^m$ for $m > \ell$, then we call H a *compression function.* On the other hand, if H has domain Dom containing points of varying lengths then we call H a *hash function.* We occasionally relax this distinction to refer to both types of function as simply hash functions.

2.3.3 Idealised Models for Hash Functions and Block Ciphers

In this section we give a brief overview of the *random oracle model*, a heuristic model for analysing hash functions. For a full discussion of the model and its pros and cons see e.g., Katz and Lindell [86]. We also give a brief overview of the *ideal permutation model.*

The random oracle model. In some applications, we require a hash function to achieve more than just collision resistance (and other standard properties, such as *preimage resistance*, that we omit here for brevity). Ideally, we would like a hash function to behave like a *random oracle*; that is to say, a black box that returns a random range point in response to each fresh query on a domain point, and that answers consistently on repeated queries. Indeed, there are some (often highly practical) schemes based on hash functions for which it seems infeasible to prove security under any weaker assumption on the hash function. Rather than leaving these schemes unproven, or abandoning our scheme to look for an alternative that can be proved under weaker assumptions (often sacrificing efficiency in the process), we could compromise between the two approaches and replace the concrete hash function in the construction with a random oracle when we construct our proof.

This approach, first formalised by Bellare and Rogaway in their classic (and controversial) paper [22], is called the random oracle model (ROM). More formally, in the ROM we model a hash function $\mathsf{H} : \mathit{Dom} \to \{0,1\}^{\ell}$ as a truly random function $\mathsf{H} \leftarrow_{\$} \mathrm{Func}(\mathit{Dom}, \{0,1\}^{\ell})$ (or its lazy sampled equivalent). All algorithms, and the attacker, are given oracle access to H, and all probabilities are computed over the random choice of H.

A heuristic. The ROM is of course a heuristic. In the real world, our scheme will be instantiated with a concrete hash function, which will be entirely fixed by its specification and can be evaluated by an attacker on any point by simply running its code. More broadly we know that many hash functions, in particular those based on the Merkle-Darmgård transform (see e.g., [155]), are *not* random oracles, since they fall foul of length extension attacks that are not possible for random oracles. Care must therefore be taken when choosing a concrete hash function to instantiate a scheme that was analysed in the ROM. Nonetheless, the ROM has proved incredibly useful for analysing (subject to the heuristic) the security of real world cryptosystems for which a proof otherwise seems infeasible.

Controversy. The ROM has been highly controversial. Goldreich compares the ROM to a “fetish”, worshipped as a “yardstick” while its original purpose as a “sanity check” has been forgotten [71]. Moreover, it has been demonstrated that there are schemes that can be proved secure in the ROM but which are insecure when instantiated with any concrete hash function [44]. That said, these schemes are all highly contrived — so much so that Koblitz and Menezes [93] interpret these results as *support* for the ROM, reasoning that if these artificial schemes are the best counterexamples that can be found then we should feel even more confident in the soundness of the ROM for practical schemes. Most compellingly, no real world scheme proved secure in the ROM has ever been successfully attacked when instantiated with an appropriate hash function.

Conclusion. In line with our focus on practice-oriented provable security, we firmly take the view that a proof of a practical scheme is better than no proof at all, even if it relies on heuristics such as the ROM. From such results, we can reduce the set of flaws that could possibly exist in a scheme to those arising from the concrete hash function, allowing us to eliminate fundamental flaws in the design and problems due to interactions between other components. Trying to persuade practitioners to use highly impractical schemes because they admit standard model proofs is clearly counterproductive, and we can find assurance in the fact that in three decades of real world schemes being analysed in the ROM, none of these have subsequently been broken when instantiated with a good choice of hash function. That said, it is important to keep in mind that results in the ROM are heuristics and great care must be taken when choosing a concrete hash function to instantiate the scheme.

The ideal permutation model. We briefly mention another heuristic model, this time for block cipher based schemes, called the *ideal permutation model* (IPM). We do not directly use the IPM in any of our analyses. However, we do invoke a result of Rogaway and Steinberger [140] on high-rate CR-hashing proved in the IPM in Section 3.5, and so give a brief overview for completeness. The reader is referred to e.g., [41] for a discussion of the IPM and its close relation, the ideal cipher model.

Consider a scheme based on a block cipher $\mathrm{E} : \mathcal{K} \times \{0,1\}^\ell \to \{0,1\}^\ell$ with a set of fixed public keys $K_1, \ldots, K_t \in \mathcal{K}$. (As we shall discuss in Section 3.5, this is a common way of building CR compression functions.) To analyse such a scheme in the IPM, we replace

$\mathrm{E}(K_i, \cdot)$ for each $i \in [1, t]$ with a random permutation $\pi_i \leftarrow_{\$} \mathrm{Perm}(\{0,1\}^\ell)$. The attacker and all other algorithms are given black-box oracle access to each π_i and its inverse π_i^{-1}. All probabilities are taken over the choice of the random permutations.

Block cipher based CR compression functions are typically analysed in the IPM. $\mathcal{A}$ is assumed to run in unbounded time, and its resources are measured in the number of oracle queries made.

This concludes our discussion of cryptographic primitives. In the next section, we introduce a number of notions of entropy and define statistical extractors.

2.4 Information Theoretic Notions

In this section, we define a number of information theoretic notions that we will employ during the subsequent chapters. We begin by defining entropy and statistical indistinguishability, and then describe how a high entropy distribution can be converted into close-to-uniform strings using a statistical extractor. In contrast to the computational security of the primitives in Section 2.3, all attackers considered in this section are unbounded and the corresponding security notions are information theoretic.

2.4.1 Entropy

Entropy is used to measure the degree of unpredictability or uncertainty in a probability distribution. In this section, we recall a number of definitions for min-entropy that we will use in later sections.

Min-entropy. Min-entropy measures the unpredictability of a distribution X in terms of the maximum success probability achievable by an unbounded attacker trying to guess the value of a point $x \leftarrow X$ drawn from that distribution. More formally, let X and Z be distributions. We define the min-entropy of X to be $\mathrm{H}_\infty(X) = -\log(\max_x \Pr[\, X = x \,])$, and the joint min-entropy of X and Z is defined $\mathrm{H}_\infty(X, Z) = -\log(\max_{x,z} \Pr[\, X = x \wedge Z = z \,])$. For a distribution X over $\{0,1\}^n$, it is straightforward to verify that $\mathrm{H}_\infty(X) \leq n$ where the equality is strict for all but the uniform distribution over $\{0,1\}^n$.

Let X and Z be (possibly related) distributions. The conditional min-entropy of X conditioned on Z is defined to capture the maximum success probability that an unbounded attacker trying to guess the value of a point drawn from X can achieve when given a point from Z as side information. We define two variants of conditional min-entropy, which differ in how the point from Z is chosen. For distributions X and Z, the worst-case conditional entropy of X conditioned on Z is defined

$$\mathrm{H}_\infty(X \mid Z) = -\log(\max_{x,z} \Pr\left[\, X = x \mid Z = z \,\right]) \; .$$

Worst-case min-entropy provides a very strong guarantee on the unpredictability of X conditioned on Z that holds regardless of which point from Z is learned. In some settings, this measure is too conservative. For example, X may be highly unpredictable conditioned on all but a handful of values in the support of Z which occur with very small probability. These 'bad' values may force the worst-case min-entropy of X conditioned on Z to be very low despite the fact that, with high probability over the choice of $z \leftarrow Z$, the value of a point $x \leftarrow X|Z = z$ will be unpredictable. In such cases, a notion of conditional min-entropy that averages over the choice of $z \leftarrow Z$ is useful. More formally, the average-case min-entropy of X conditioned on Z is defined

$$\tilde{\mathrm{H}}_\infty(X|Z) = -\log\left(\sum_z \max_x \; \Pr\left[\, X = x \mid Z = z \,\right] \cdot \Pr\left[\, Z = z \,\right]\right) \; .$$

For any pair of distributions X and Z, it holds that $\tilde{\mathrm{H}}_\infty(X|Z) \geq \mathrm{H}_\infty(X|Z)$.

2.4.2 Statistical Distance and Randomness Extractors

We now define statistical extractors which convert samples drawn from a distribution with high min-entropy into close-to-uniform strings. We begin by recalling the notion of statistical distance.

Statistical distance. Statistical distance is a metric to measure the closeness between two distributions. More formally, let X_0 and X_1 be distributions over a finite set $\mathcal{Z}$. Then the statistical distance of X_0 and X_1, denoted $\Delta(X_0, X_1)$, is defined:

$$\Delta(X_0, X_1) = \max_{T \subseteq \mathcal{Z}} \left(\Pr\left[\, X_0 \in T \,\right] - \Pr\left[\, X_1 \in T \,\right]\right) \; .$$

It is straightforward to verify that $\Delta(X_0, X_1) \in [0, 1]$. If $\Delta(X_0, X_1) = 0$, then the distributions X_0 and X_1 are identical. In contrast, $\Delta(X_0, X_1) = 1$ implies that the supports of X_0 and X_1 are disjoint. If two distributions have small statistical distance (where the precise meaning of 'small' will depend on the context), then we say they are *statistically close*.

Statistical indistinguishability. Statistical indistinguishability measures how well an unbounded attacker $\mathcal{A}$ can distinguish between points drawn from distributions X_0 and X_1. More formally, we define the distinguishing advantage of an attacker $\mathcal{A}$ to be

$$\left|\Pr\left[\,\mathcal{A}(x) \Rightarrow 1 \;:\; x \leftarrow X_0\,\right] - \Pr\left[\,\mathcal{A}(x) \Rightarrow 1 \;:\; x \leftarrow X_1\,\right]\right| .$$

Let $T^* = \mathrm{argmax}_{T \subseteq \mathcal{Z}}\Big(\Pr\left[\,X_0 \in T\,\right] - \Pr\left[\,X_1 \in T\,\right]\Big)$. Let $\mathcal{A}$ be the attacker who outputs 1 if $x \in T^*$ and 0 otherwise. It is straightforward to verify that $\mathcal{A}$ achieves advantage $\Delta(X_0, X_1)$ and that their strategy is optimal. As such, $\Delta(X_0, X_1)$ is an upper bound on the distinguishing advantage of any attacker (unbounded or computational) in the above game.

Extractors. A distribution with high min-entropy will be unpredictable but may still be far from uniform. For example, consider taking the uniform distribution over the set of n-bit strings for which the least significant bit is 0. Such a distribution has nearly full min-entropy $n-1$, and yet is trivially distinguishable from $\mathcal{U}_n$. A *statistical extractor*, which we formalise below, can be used to extract close-to-uniform bit strings from high min-entropy distributions.

We say that a random variable X is a *k-source* if $\mathrm{H}_\infty(X) \geq k$. An *extractor* is a function that maps points drawn from a k-source to a close-to-uniform distribution over bit-strings of some fixed length w. It is well-known that deterministic extraction is impossible in general, and so we specify extractors to take as input a uniform seed $A \leftarrow_{\$} \{0,1\}^v$. We define extractors formally below, and then given an example illustrating the impossibility of seedless extraction.

Definition 2.1. *A function* $\mathsf{Ext} : \{0,1\}^* \times \{0,1\}^v \to \{0,1\}^w$ *is said to be a (k,ϵ)-extractor if for all distributions X over* $\{0,1\}^*$ *such that* $\mathrm{H}_\infty(X) \geq k$*, it holds that*

$$\Delta((\mathsf{Ext}(X;A), A), (\mathcal{U}_w, A)) \leq \epsilon ,$$

where $A \leftarrow_{\$} \mathcal{U}_v$*.*

By the classic Leftover Hash Lemma [76], universal hash functions (see e.g., [128] for a formal definition) are optimal and efficient statistical extractors.

We will now show that without seeding it is impossible to extract even a single bit from an imperfect source, even when that source has close to full entropy. The version of this well-known result we present here is taken from [159]. We claim that for any deterministic extractor $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}$, there exists an $(n-1)$-source X such that Ext is constant on X. To see this, we define $\mathcal{X}_0 = \{x \in \{0,1\}^n : \mathsf{Ext}(x) = 0\}$ and $\mathcal{X}_1 = \{0,1\}^n \setminus \mathcal{X}_0$, and notice that there must exist $b \in \{0,1\}$ such that $|\mathcal{X}_b| \geq 2^{n-1}$. We define the $(n-1)$-source X to be the uniform distribution over $\mathcal{X}_b$. It is straightforward to see that, despite having at least $(n-1)$-bits of entropy, the output of Ext over this source will be constant on b. This illustrates the necessity of the seed, which effectively chooses an extractor at random from a family indexed by the seed value.

Online extractors. In Chapter 5, we will require an *online-computable* extractor. We say that an extractor Ext is online-computable with respect to p if it admits an equivalent formulation which correctly computes the value of Ext on each possible input $I \in \cup_{k\in\mathbb{N}}\{0,1\}^{k\cdot p}$ when I is incrementally provided to Ext in chunks of p-bits at a time. We formalise this in the definition below.

Definition 2.2. *An extractor* $\mathsf{Ext} : \{0,1\}^* \times \{0,1\}^v \to \{0,1\}^w$ *is said to be online-computable with respect to p if there exists a pair of efficient algorithms* $\mathsf{iterate} : \{0,1\}^p \times \{0,1\}^p \times \{0,1\}^v \to \{0,1\}^p$ *and* $\mathsf{finalise} : \{0,1\}^p \times \{0,1\}^v \to \{0,1\}^w$ *such that for all inputs* $I = (I_1, \ldots, I_d)$ *where* $I_j \in \{0,1\}^p$ *for* $j \in [1,d]$*, and all seeds* $A \in \{0,1\}^v$*, then after setting* $y_0 = 0^p$*, and* $y_j = \mathsf{iterate}(y_{j-1}, I_j; A)$ *for* $j = 1, \ldots, d$*, it holds that*

$$\mathsf{Ext}(I; A) = \mathsf{finalise}(y_d; A).$$

2.5 Pseudorandom Number Generators

Given that most modern cryptography depends on a good source of random bits, it is unsurprising that pseudorandom number generators — which expand a small high entropy seed into much larger quantities of pseudorandom bits — are ubiquitous in cryptographic protocols and implementations. In this section, we define several varieties of pseudorandom number generator and present their associated security models.

PRNGs with input. A *pseudorandom number generator with input* (PRNG) [60] produces pseudorandom bits and offers strong security guarantees when given continual access to an imperfect source of randomness.

Definition 2.3. *A pseudorandom number generator with input (PRNG) is a tuple of algorithms* $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ *with associated parameter set* (ℓ, p)*, defined as follows:*

- $\mathsf{init} : \to \text{Seed}$ *is a randomised algorithm which takes no input and outputs a public seed* $\mathsf{seed} \in \text{Seed}$.
- $\mathsf{setup} : \text{Seed} \to \mathcal{S}$ *is a randomised algorithm which takes as input a seed* $\mathsf{seed} \in \text{Seed}$ *and returns an initial state* $S_0 \in \mathcal{S}$*, where* $\mathcal{S}$ *denotes the state space of the PRNG.*
- $\mathsf{refresh} : \text{Seed} \times \mathcal{S} \times \{0,1\}^p \to \mathcal{S}$ *takes as input a seed* $\mathsf{seed} \in \text{Seed}$*, a state* $S \in \mathcal{S}$*, and an entropy sample* $I \in \{0,1\}^p$*, and returns a state* $S' \in \mathcal{S}$.
- $\mathsf{next} : \text{Seed} \times \mathcal{S} \to \{0,1\}^\ell \times \mathcal{S}$ *takes as input a seed* $\mathsf{seed} \in \text{Seed}$ *and a state* $S \in \mathcal{S}$*, and returns an output* $R \in \{0,1\}^\ell$*, and an updated state* $S' \in \mathcal{S}$.

Here we have modified the definition of Dodis et al. [60] to: **(1)** allow states to lie in a state space $\mathcal{S}$ rather than being strings of length n; and **(2)** generate the initial state $S_0 \in \mathcal{S}$ via setup rather than choosing S_0 uniformly from $\{0,1\}^n$. These modifications better model real world PRNGs, for which the state may consist of various components and contain non-uniform elements such as counters. Moreover, we will utilise the fact that PRNG states may contain multiple (and possibly non-uniform) components when we construct backdoored PRNGs in Chapter 5.

Distribution samplers. We model the gathering of entropy inputs from the entropy source via a *distribution sampler* [60]. Formally, a distribution sampler $\mathcal{D} : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^p \times \mathbb{R}^{\geq 0} \times \{0,1\}^*$ is a stateful and probabilistic algorithm which takes as input its current state $\sigma \in \{0,1\}^*$ and outputs a tuple (σ', I, γ, z), where $\sigma' \in \{0,1\}^*$ denotes the updated state of the sampler, $I \in \{0,1\}^p$ denotes the entropy sample, $\gamma \in \mathbb{R}^{\geq 0}$ is an entropy estimate for the sample, and $z \in \{0,1\}^*$ denotes a string of side information about the sample. We say that a sampler $\mathcal{D}$ is $(q_{\mathcal{D}}, \gamma^*)$-*legitimate* if for all $j \in [1, q_{\mathcal{D}}]$:

$$\mathrm{H}_\infty(I_j | I_1, \ldots, I_{j-1}, I_{j+1}, \ldots, I_{q_{\mathcal{D}}}, \gamma_1, \ldots, \gamma_{q_{\mathcal{D}}}, z_1, \ldots, z_{q_{\mathcal{D}}}) \geq \gamma_j \ ,$$

where $\sigma_0 = \varepsilon$ and $(\sigma_j, I_j, \gamma_j, z_j) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{j-1})$. Looking ahead, the entropy estimates γ will be used in security definitions to measure how much entropy has entered the PRNG. The side information z is included to model the fact that, in practice, entropy will often be

```
$\mathrm{Rob}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*}$
$b \leftarrow\!\!\$ \{0,1\}$
$\sigma \leftarrow \varepsilon$ ; $\mathsf{seed} \leftarrow\!\!\$ \mathsf{init}$
$S \leftarrow\!\!\$ \mathsf{setup}(\mathsf{seed})$
$c \leftarrow \gamma^*$
$b^* \leftarrow\!\!\$ \mathcal{A}^{\mathrm{Ref,RoR,Get,Set}}(\mathsf{seed})$
Return $(b = b^*)$
```

```
Ref
$(\sigma, I, \gamma, z) \leftarrow\!\!\$ \mathcal{D}(\sigma)$
$S \leftarrow \mathsf{refresh}(\mathsf{seed}, S, I)$
$c \leftarrow c + \gamma$
Return $(\gamma, z)$
```

```
RoR
$(R_0, S) \leftarrow \mathsf{next}(\mathsf{seed}, S)$
If $c < \gamma^*$
  Return $R_0$
  $c \leftarrow 0$
Else $R_1 \leftarrow\!\!\$ \{0,1\}^\ell$
Return $R_b$
```

```
Get
Return $S$
$c \leftarrow 0$

Set($S^*$)
$S \leftarrow S^*$
$c \leftarrow 0$
```

Figure 2.1: Security game Rob for a PRNG $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$.

gathered from system events such as disk timings and key strokes and so an attacker may be able to gain some advantage in predicting the value of the entropy samples by observing the operating environment.

2.5.1 Security Notions for PRNGs

We now recall a number of security notions for PRNGs, as well as a useful result that modularises proofs of PRNG security.

Robustness. Consider the game Rob shown in Figure 2.1. The game is parameterised by an entropy threshold γ^*. We expect security when the entropy in the system is at least this value. Here we have modified game Rob from [60] to: **(1)** generate the initial state via the setup algorithm (as opposed to initialising the PRNG with an ideal random state); and **(2)** remove the Next oracle, which was shown in [51] to be without loss of generality. With this in place, the *robustness* advantage of a $(q_{\mathcal{D}}, \gamma^*)$-legitimate distribution sampler $\mathcal{D}$ and adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) = 2 \cdot \left| \Pr\left[\mathrm{Rob}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \Rightarrow 1 \right] - \frac{1}{2} \right| .$$

We say that $\mathcal{A}$ is a $(q_R, q_{\mathcal{D}}, q_C, q_S)$-adversary if he makes q_R queries to its RoR oracle, a total of q_S queries to its Get and Set oracles, and $q_{\mathcal{D}}$ queries to its Ref oracle where $q_C \leq q_{\mathcal{D}}$ denotes the maximum number of consecutive Ref queries. Looking ahead, we make q_C an explicit parameter (rather than upper bounding q_C by $q_{\mathcal{D}}$) to yield tighter bounds in the results of Sections 4.6 and 4.7 in which q_C will surface as a parameter. For results in which q_C does not surface in the security analysis (in particular, those in Chapter 5) we may omit q_C for brevity.

Forward and backward security. We define games Fwd and Bwd to be restricted variants of game Rob. In game Fwd, the attacker $\mathcal{A}$ is allowed no Set queries, and makes a single Get query after which it may make no further queries. In game Bwd, the attacker makes a single Set query as his first query, and after that may make no Set or Get queries. The forward and backward security advantages for an attacker / sampler pair $(\mathcal{A}, \mathcal{D})$ are respectively defined as:

$$\mathbf{Adv}^{\mathrm{fwd}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) = 2 \cdot \left| \Pr\left[\mathrm{Fwd}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \Rightarrow 1 \right] - \frac{1}{2} \right| \text{ ; and}$$

$$\mathbf{Adv}^{\mathrm{bwd}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) = 2 \cdot \left| \Pr\left[\mathrm{Bwd}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \Rightarrow 1 \right] - \frac{1}{2} \right| .$$

Relations and separations. It is straightforward to verify that robustness implies forward and backward security. However, a PRNG which is both forward and backward secure is not necessarily robust. For a separating example, let $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ be a robust PRNG with state space $\mathcal{S} = \{0,1\}^n$. We define a modified PRNG $\mathsf{PRNG}' = (\mathsf{init}', \mathsf{setup}', \mathsf{refresh}', \mathsf{next}')$ with state space $\mathcal{S}' = \{0,1\}^n \times \{0,1\} \times \{0,1\}^\ell$ as follows. We define init' to be identical to init, and define $\mathsf{setup}'(\cdot)$ to be the algorithm which computes $S \leftarrow_\$ \mathsf{setup}(\cdot)$ and outputs $(S, 0, 0^\ell)$. We define $\mathsf{refresh}'$ to be the algorithm that on input $(\mathsf{seed}, (S, \theta, Y), I)$ for $\theta \in \{0,1\}, Y \in \{0,1\}^\ell$, and $I \in \{0,1\}^p$, computes $S' \leftarrow \mathsf{refresh}(\mathsf{seed}, S, I)$ and returns (S', θ, Y). We define next' to be the algorithm that on input $(\mathsf{seed}, (S, \theta, Y))$ first computes $(R, S') \leftarrow \mathsf{next}(\mathsf{seed}, S)$. If $\theta = 0$, it returns $(R, (S, 0, Y))$; however if $\theta = 1$, it returns $(R, (S, 1, R))$.

It is straightforward to check that PRNG' is still Fwd and Bwd secure. (Since setup' will only return states of the form $(S, 0, 0^\ell)$, an attacker in game Fwd against PRNG can perfectly simulate game Fwd for an attacker against PRNG' by simply querying his own oracles and adding $(0, 0^\ell)$ to the state returned for the attacker's final Get query. Similarly in game Bwd, after the initial Set query the attacker can only make Ref and RoR queries, the output of which are unchanged for the modified PRNG'. As such, an attacker $\mathcal{A}$ in game Bwd against PRNG can perfectly simulate game Bwd for an attacker $\mathcal{A}'$ against PRNG' by querying S to his own Set oracle in response to $\mathcal{A}'$'s initial Set query (S, θ, Y) and then forwarding all other queries directly to his own oracles.) However, an attacker $\mathcal{A}$ can easily win the robustness game as follows. $\mathcal{A}$ queries $(S, 1, 0^n)$ to Set and then makes sufficiently many Ref calls that $c \geq \gamma^*$. $\mathcal{A}$ then queries RoR to receive R^* immediately followed by Get to receive $(S, 1, Y)$. By checking if $Y = R^*$, $\mathcal{A}$ can distinguish a real from random output with overwhelming probability.

```
$\underline{\mathrm{Pres}^{\mathcal{A}}_{\mathsf{PRNG}}}$
$b \leftarrow\!\!\$ \{0,1\}$
$\mathsf{seed} \leftarrow\!\!\$ \mathsf{init}$
$S_0 \leftarrow\!\!\$ \mathsf{setup}(\mathsf{seed})$
$(I_1, \ldots, I_d) \leftarrow\!\!\$ \mathcal{A}(\mathsf{seed})$
For $i = 1, \ldots, d$
    $S_i \leftarrow \mathsf{refresh}(\mathsf{seed}, S_{i-1}, I_i)$
If $(b = 0)$ then $(R^*, S^*) \leftarrow \mathsf{next}(\mathsf{seed}, S_d)$
Else $R^* \leftarrow\!\!\$ \{0,1\}^\ell$ ; $S^* \leftarrow\!\!\$ \mathsf{setup}(\mathsf{seed})$
$b^* \leftarrow\!\!\$ \mathcal{A}(\mathsf{seed}, R^*, S^*)$
Return $(b = b^*)$
```

```
$\underline{\mathrm{Rec}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}}$
$b \leftarrow\!\!\$ \{0,1\}$
$\sigma_0 \leftarrow \varepsilon$ ; $\mathsf{seed} \leftarrow\!\!\$ \mathsf{init}$ ; $\mu \leftarrow 0$
For $k = 1, \ldots, q_{\mathcal{D}}$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d) \leftarrow\!\!\$ \mathcal{A}^{\mathrm{Sam}}(\mathsf{seed}, (\gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}})$
If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
For $i = 1, \ldots, d$
    $S_i \leftarrow \mathsf{refresh}(\mathsf{seed}, S_{i-1}, I_{\mu+i})$
If $(b = 0)$ then $(R^*, S^*) \leftarrow \mathsf{next}(\mathsf{seed}, S_d)$
Else $R^* \leftarrow\!\!\$ \{0,1\}^\ell$ ; $S^* \leftarrow\!\!\$ \mathsf{setup}(\mathsf{seed})$
$b^* \leftarrow\!\!\$ \mathcal{A}(\mathsf{seed}, R^*, S^*, (I_k)_{k>\mu+d})$
Return $(b = b^*)$

$\underline{\mathrm{Sam}()}$
$\mu \leftarrow \mu + 1$
Return $I_\mu$
```

Figure 2.2: Security games Pres and Rec for a PRNG $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$.

2.5.2 Preserving and Recovering Security

A key insight of [60] is that the complex notion of robustness can be decomposed into two simpler notions called *preserving* and *recovering* security. The former models the PRNG's ability to maintain security if the state is secret but the attacker is able to influence the entropy source. The latter models the PRNG's ability to recover from state compromise after sufficient (honestly generated) entropy has entered the system. Consider games Pres and Rec shown in Figure 2.2. Here we have modified the definitions of [60] to have the initial state be generated by setup. For $\mathrm{Gm}^x_y \in \{\mathrm{Pres}^{\mathcal{A}}_{\mathsf{PRNG}}, \mathrm{Rec}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}\}$ we define

$$\mathbf{Adv}^{\mathrm{gm}}_y(x) = 2 \cdot \left| \Pr\left[\, \mathrm{Gm}^x_y \Rightarrow 1 \,\right] - \frac{1}{2} \right| .$$

We say that an attacker $\mathcal{A}$ in game Pres or Rec is a q_C-adversary if the index d (explicitly output by $\mathcal{A}$ in game Rec, and implicitly defined by the number of inputs output by $\mathcal{A}$ in game Pres) is at most q_C. With this in place, the following theorem, which says that Pres plus Rec security implies Rob security, is a straightforward adaptation of the analogous result from [60].

Theorem 2.2. *Let* $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ *be a PRNG with input. Then for any* $(q_{\mathcal{D}}, q_C, q_R, q_S)$*-adversary* $\mathcal{A}$ *and* $(q_{\mathcal{D}}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$ *in game Rob against* PRNG *running in time* T*, there exists* q_C*-adversaries* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *such that*

$$\mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) \le 2q_R \cdot \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{A}_1) + 2q_R \cdot \mathbf{Adv}^{\mathrm{rec}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}_2, \mathcal{D}) ,$$

and moreover, $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *run in time* T*.*

We prove a more involved variant of this theorem in Section 4.5 from which a proof of Theorem 2.2 may easily be recovered. We note that the original result in [60] erroneously omits a factor of two from the right-hand side of the equation; we correct this here, accounting for the additional term in the theorem statement.

PRNG definitions in this thesis. The PRNG model given here shall be used in Chapter 5. In Chapter 4, we will have to substantially modify it in order to accommodate the real-world NIST DRBGs, which have a variety of optional inputs and quirks which do not fit cleanly into this syntax. We shall frequently refer back to this definition in Chapter 4 to emphasise the changes we have made.

This concludes our chapter on preliminaries. With this groundwork in place, we will now employ the provable security concepts described in this chapter to analyse a number of real world problems.

Chapter 3

Fast Message Franking: From Invisible Salamanders to Encryptment

Contents

3.1 Introduction and Motivation

Encrypted messaging. End-to-end (EtE) encrypted messaging applications — such as Whatsapp, Signal, and Facebook Secret Conversations — are used by billions of people each day. The integration of end-to-end encryption in these applications is so seamless that it's not unreasonable to assume that many users are unaware that the technology is even implemented. This is a remarkable illustration of how advanced cryptography has, in the last fifty years, transcended the realms of military and government to become an ubiquitous part of our everyday lives.

The increased uptake of EtE encryption has been divisive, with the discussion around it echoing the privacy versus surveillance debate surrounding cryptography as a whole. EtE encryption is undeniably beneficial for the privacy of users, an especially prescient concern following recent high profile failures such as the Cambridge Analytica scandal [42] and Snowden revelations [124]. EtE encryption can facilitate freedom of expression and censorship avoidance, which as we will reiterate in Chapter 5 is increasingly being viewed as vital for a well-functioning society [126, 136].

At the same time, the debate as to whether an individual's right to privacy trumps that of law enforcement agencies to gather intelligence rages on, exacerbated by atrocities such as the 2015 San Bernardino terrorist attack and subsequent legal dispute between the FBI and Apple over whether the latter should assist in unlocking a deceased perpetrator's iPhone. EtE encryption has been singled out for criticism by, among others, the FBI [106, 117] and the UK Government, with then Home Secretary Amber Rudd memorably causing a backlash in 2017 by claiming that EtE encryption is not for "real people" [141].

3.1.1 Message Franking

While the tension between privacy and surveillance is nothing new in cryptography (and indeed we will return to the latter in Chapter 5), EtE encryption creates another, less obvious, tension. Providers, under increasing pressure to prevent abusive behaviour on their sites, want to support a service for users to report offensive content such as harmful messages, images, or videos. However, if EtE encryption prevents the service provider from learning anything about the message contents, how can a provider verify that the reported message was actually sent and not fabricated by the complainant?

Message franking. The solution to this problem proposed by Facebook — called *message franking* [65,112], where the name comes from 'speaking frankly' — shall be the focus of this chapter. A message franking scheme allows a receiver to cryptographically *prove* to the service provider that a reported message underlies an encrypted message. Inspired by this, Grubbs, Lu, and Ristenpart (GLR) [73] provided the first formal treatment of the problem, and introduced compactly committing authenticated encryption with associated data (ccAEAD) as the key primitive. A secure ccAEAD scheme encrypts messages under a secret key such that a short portion of the ciphertext, called a *binding tag*, serves as a cryptographic commitment to the underlying message (and associated data). GLR detailed appropriate security notions, and proved that the main Facebook message franking approach, called $\mathsf{CtE2}$ here, achieves them. They also introduced a faster custom ccAEAD scheme called Committing Encrypt-and-PRF (CEP).

Message franking in Facebook. The Facebook scheme $\mathsf{CtE2}$ is a composition of an HMAC-based commitment and a standard encrypt-then-MAC AEAD scheme. As such, the scheme requires three full cryptographic passes over the message to compute[1]. We count The CEP construction [73] gets this down to two. However, this still does not match the fastest standard authenticated encryption (AE) schemes such as GCM [107] and OCB [137]. These require roughly one block cipher call (on the same key) per block of message data and, in the case of GCM, some arithmetic operations in $\mathrm{GF}(2^n)$ that are faster than a block cipher invocation. However as observed by GLR, GCM is not compactly committing; one can find two distinct messages and two encryption keys that lead to the same binding tag. This violates a property called *receiver binding*, and could in theory allow a malicious recipient to report a message that was never sent.

Existing ccAEAD schemes are not considered fast enough for all applications of message franking by practitioners [112]. Because of this, Facebook Messenger does not use the secure CtE2 ccAEAD scheme mentioned above to directly encrypt attachment files (which are typically large). Instead they use a hybrid scheme which, at a very high level, encrypts the attachment file under a one-time key with the regular (and very fast) AEAD scheme GCM, and then encrypts and commits to the key using the CtE2 ccAEAD scheme. Despite GCM not being secure when framed as a ccAEAD scheme, its use in Facebook's attachment franking does not immediately imply any concrete attack on Facebook's system. However as we shall see, the fact that GCM is not *robust* [4,66,67,115] — meaning that it is possible to construct ciphertexts which decrypt correctly under different keys — ultimately leads to an attack.

3.1.2 Contributions

In this chapter, we consider the problem of building highly efficient ccAEAD schemes. To motivate this, we first present an attack on Facebook's attachment franking, the ad hoc design of which was necessitated by the lack of fast and secure ccAEAD schemes. Our results are an example of theory meeting practice: the Facebook attack represents, to the best of our knowledge, the first real world attack which exploits the non-robustness of an encryption scheme, and our results on fast and practical ccAEAD build on the wealth of literature on collision resistant (CR) hashing. We discuss our contributions in more detail below.

[1]Throughout this chapter, a pass over data is understood to be an application of a cryptographic scheme such as an encryption scheme or hash function to the data in question. We count each application as a separate pass, regardless of whether the applications can be parallelised.

Breaking Facebook's attachment franking. In the first part of the chapter, we demonstrate an attack against Facebook's attachment franking scheme that allows a malicious sender to send an abusive attachment for which all attempts to report this to Facebook will fail. The cryptographic flaw that enables the attack is that a GCM ciphertext is *not* a commitment to its underlying plaintext; indeed, due to its non-robustness, it is possible to construct a single ciphertext that decrypts to different plaintexts under different keys. The attack exploits the way in which Facebook's server code deduplicates attachments in abuse reports by their ciphertext identifiers. We show that if the attacker sends a GCM ciphertext twice with the same identifier and two different keys, it can ensure that only one decryption ends up in the abuse report. If the GCM ciphertext decrypts to an abusive message under one key, and an innocuous message under the other, then the receiver will receive both the abusive and innocuous messages. However, the human viewing the abuse report will *only* see the innocuous one. It is here that we encounter the invisible salamanders of the title. Following Signal, Facebook calls encrypted messages 'salamanders'. Since this attack allows a malicious sender to craft a message which cannot be seen by Facebook, this can be viewed as an *invisible salamander.*

We responsibly disclosed this vulnerability to Facebook, who helped us understand how our attack works against their systems (much of the abuse handling code is server-side and closed source). The severity of the issue led them to patch their (server-side) systems and to award us a bug bounty. Their fix is ad hoc and involves deduplicating more carefully. However, the vulnerability would have been avoided in the first place by using a fast ccAEAD scheme that provided the binding security properties implicitly assumed of — but not actually provided by — GCM.

Towards faster ccAEAD schemes: encryptment. This message franking failure motivates the need for faster schemes. As mentioned, the best known secure ccAEAD scheme from GLR is two pass, requiring computing both HMAC and AES-CTR mode (or similar) over the message. The fastest standard AE schemes [84, 107, 137] however require just a single pass using a block cipher with a single key. This raises the question of whether we can build ccAEAD schemes that match this performance?

To tackle this question we first abstract out the core technical challenge underlying ccAEAD via a new primitive called *encryptment.* At a high level, encryptment is a one-time secure variant of ccAEAD. The deterministic encryptment algorithm maps a key

K_{EC} and header / message pair (H, M) to an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$. Together these components constitute an encryption of M. Moreover, the *binding tag* B_{EC} represents a commitment to (H, M) that can be verified with the key K_{EC}. Being single-use, encryptment is substantially simpler than ccAEAD, making analyses easier and, we think, design of constructions more intuitive. We will later show how to lift secure encryptment to fully-fledged multi-use ccAEAD via simple and efficient transforms. In the other direction, we show that one can also build encryptment from ccAEAD, making the two primitives equivalent from a theoretical perspective. Encryptment also turns out to be the 'right' primitive for a number of other applications: robust authenticated encryption [67], concealments [56], remotely keyed AE [56], and perhaps even more.

Fast encryptment from fixed-key block ciphers? With our new primitive defined, we turn to building fast encryptment schemes. First we show a negative result: encryptment schemes cannot match the efficiency profile of OCB or GCM in terms of *rate* (where rate is defined to be the number of blocks of input data processed per block cipher call). In fact we rule out any rate-1 scheme that uses just a single block cipher invocation for each block of message with some fixed small set of keys.

This negative result makes use of a connection between encryptment and CR hashing. We first show that for an encryptment scheme to achieve a certain kind of binding security the function that is used to compute the binding tag must be CR. We can then exploit known impossibility results on high-rate fixed-key block cipher-based CR hashing [139, 140, 150] to rule out similarly high rate encryptment schemes. A simple corollary of [140, Thm. 1] is that one cannot prove receiver binding security for any rate-1 fixed-key block cipher-based encryptment scheme. Since OCB and GCM are rate-1, this implies that they cannot yield binding encryptment. Our negative result also rules out rate-1 ccAEAD due to our aforementioned result that (fast) ccAEAD implies (fast) encryptment.

One-pass encryptment from hashing. Given the connection just mentioned, it is natural to turn to CR hashing as a starting point for building fast-as-possible encryptment. Our scheme, which we call Hash Function Chaining (HFC), is based on the classic Merkle-Damgård (MD) [52, 109, 110] construction of a CR hash function. To compute the binding tag, we simply hash the carefully padded header and message using a keyed version of an MD-iterated compression function. The trick which allows us to compute both ciphertext and binding tag in a single pass is to use the intermediate chaining variables generated during this process as random pads to encrypt the message blocks. We prove that our construction is a secure encryptment scheme under the assumption that the underlying compression function is both CR and, when framed as a keyed function, pseudorandom under a weak form of related-key attack. This is a stronger property than regular PRF security, and requires that the function continues to behave like a PRF even if the attacker can query the function under related keys. Here the class of related keys the attacker may use consists of the secret PRF key XORed with a string of the attacker's choosing. We discuss possible choices with which to instantiate the compression function. We also describe how to avoid the related-key PRF assumption, in particular giving a variant of the construction based on the Duplex authenticated-encryption mode [32] using Keccak (SHA-3) [33] that replaces the iterated compression function with a sponge, although the resulting construction is not as fast or elegant as HFC.

From encryptment to ccAEAD. With this in place, we present two efficient transforms which construct multi-opening ccAEAD from secure encryptment. The first transform uses a (standard) secure AEAD scheme. To encrypt a message M, our ccAEAD scheme first generates a one-time encryptment key K_{EC} and then computes an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ for K_{EC}, M. The one-time key K_{EC} is then encrypted under the long-lived AEAD key K using the binding tag B_{EC} as associated data. The resulting ccAEAD ciphertext is the AEAD ciphertext (including its authentication tag) plus $C_{\mathsf{EC}}, B_{\mathsf{EC}}$. The second transform uses just two additional PRF calls to securely convert an encryptment scheme to a ccAEAD scheme. We give a full analysis of both schemes in Sections 3.7.3 and 3.7.4.

Our approach of hashing-based ccAEAD has a number of attractive features. HFC works with any hash function that iterates a secure compression function, giving us a wide variety of options for instantiation. Due to the simplified formalisation via encryptment the security proofs are modular and conceptually straightforward. As already mentioned, our scheme is fast in terms of the number of underlying primitive calls. If instantiated using SHA-256 one can use the SHA hardware instructions [74] now supported on some AMD and ARM processors and that are likely to be incorporated in future Intel processors. Finally, HFC-based ccAEAD is simple to implement.

Other applications. We conclude the chapter by showing that encryptment is a useful abstraction for a variety of other applications. In Section 3.8, we describe how it suffices to build concealments [56] (a conceptually similar but distinct primitive) which in turn can be used to build remotely keyed AE [56]. Previous constructions of these required two passes over the message. Our new encryptment-based approach gives the first single-pass concealments and remotely keyed AE. Finally, encryptment schemes give rise to robust AEAD [67] via the transforms mentioned above (subject to certain conditions on the underlying primitives). We expect that encryptment will find further applications in the future.

3.2 Preliminaries

In this section, we introduce a number of definitions for symmetric encryption and commitment schemes that will be used throughout the chapter.

3.2.1 Authenticated Encryption with Associated Data

A good symmetric encryption (SE) scheme allows two parties who share a secret key to communicate with *confidentiality.* This ensures that, without knowledge of the secret key, an attacker who compromises a ciphertext encrypted under that key can learn nothing useful about the underlying plaintext. In this thesis, we will utilise a class of symmetric encryption schemes which have the additional property of providing *authenticity*; that is to say, a guarantee that a ciphertext was sent by the intended communication partner. Such schemes, called authenticated encryption (AE) schemes, have their earliest formalisations in [19, 24, 87].

An authenticated encryption scheme with associated data (AEAD scheme), first formalised by Rogaway in [133], is an AE scheme which additionally provides authenticity for some metadata (typically referred to as a *header*) which is sent alongside the encrypted message. This is important for many real-world communication protocols in which a header containing networking information such as an IP address must be sent in the clear alongside an encrypted payload. We formally define AEAD schemes below, and throughout the remainder of the section will state definitions in terms of AEAD schemes. The analogous definitions for AE schemes can be recovered by removing all references to the header H from subsequent definitions.

Definition 3.1. *An authenticated encryption scheme with associated data (AEAD scheme) is a tuple of algorithms* $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *with associated key space* $\mathcal{K} \subseteq \Sigma^*$*, header space* $\mathcal{H} \subseteq \Sigma^*$*, message space* $\mathcal{M} \subseteq \Sigma^*$*, ciphertext space* $\mathcal{C} \subseteq \Sigma^*$*, and coin space* $\mathcal{R} \subseteq \Sigma^*$*, defined as follows:*

- *The randomised key generation algorithm* $\mathsf{K} \ : \to \mathcal{K}$ *takes no input, and outputs a secret key* $K \in \mathcal{K}$*. We typically have* K *choose* $K \leftarrow\$ \mathcal{K}$ *and return* K*.*
- *The encryption algorithm* $\mathsf{E} \ : \ \mathcal{K} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C}$ *is a randomised algorithm which takes as input a key* $K \in \mathcal{K}$*, a header* $H \in \mathcal{H}$*, and a message* $M \in \mathcal{M}$*, and outputs a ciphertext* $C \in \mathcal{C}$*.*
- *The decryption algorithm* $\mathsf{D} \ : \ \mathcal{K} \times \mathcal{H} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ *is a deterministic algorithm which takes as input a key* $K \in \mathcal{K}$*, a header* $H \in \mathcal{H}$*, and a ciphertext* $C \in \mathcal{C}$*, and outputs either a message* $M \in \mathcal{M}$ *or the error symbol* $\perp$*.*

We require that AEAD schemes are *perfectly correct*, by which we mean that for all $(K, H, M) \in \mathcal{K} \times \mathcal{H} \times \mathcal{M}$ it holds that $\Pr[\, \mathsf{D}(K, H, \mathsf{E}(K, H, M)) = M \,] = 1$. We additionally require that AEAD schemes are *length regular*, by which we mean that the length of a ciphertext depends only on the length of the underlying message. Formally, we require that there exists a function $\mathsf{clen}_{\mathsf{AE}} \ : \ \mathbb{N} \to \mathbb{N}$ such that for all $(K, H, M) \in \mathcal{K} \times \mathcal{H} \times \mathcal{M}$ it holds that

$$\Pr[\, |C| = \mathsf{clen}_{\mathsf{AE}}(|M|) \ : \ C \leftarrow\$ \mathsf{E}(K, H, M) \,] = 1 \, .$$

In both cases, the probability is over the coins of E. A *nonce-based AEAD scheme* is defined identically to a randomised AEAD scheme, except we redefine E to be deterministic and have E and D take a nonce $N \in \mathcal{N}$ as an input (where $\mathcal{N} \subseteq \Sigma^*$ denotes the nonce space of the scheme) in addition to the inputs specified above. Correctness and length-regularity for nonce-based AEAD schemes are defined analogously.

3.2.2 Security Notions for AEAD Schemes

An AEAD scheme should provide confidentiality and authenticity for the data being encrypted. We formalise these properties via a pair of games ROR and CTXT, which we

```
$\underline{\mathrm{REAL}^{\mathcal{A}}_{\mathsf{AEAD}}}$
$K \leftarrow_\$ \mathsf{K}$
$b^* \leftarrow_\$ \mathcal{A}^{\mathrm{ChalEnc}(\cdot,\cdot)}$
Return $b^*$

$\underline{\mathrm{ChalEnc}(H, M)}$
$C \leftarrow_\$ \mathsf{E}(K, H, M)$
Return $C$

$\underline{\mathrm{RAND}^{\mathcal{A}}_{\mathsf{AEAD}}}$
$K \leftarrow_\$ \mathsf{K}$
$b^* \leftarrow_\$ \mathcal{A}^{\$(\cdot,\cdot)}$
Return $b^*$

$\underline{\$(H, M)}$
$C \leftarrow_\$ \{0,1\}^{\mathsf{clen}_{\mathsf{AE}}(|M|)}$
Return $C$
```

```
$\underline{\mathrm{CTXT}^{\mathcal{A}}_{\mathsf{AEAD}}}$
$K \leftarrow_\$ \mathsf{K}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc}(\cdot,\cdot),\mathrm{ChalDec}(\cdot,\cdot)}$
Return $\mathsf{win}$

$\underline{\mathrm{Enc}(H, M)}$
$C \leftarrow_\$ \mathsf{E}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C)\}$
Return $C$

$\underline{\mathrm{ChalDec}(H, C)}$
If $(H, C) \in \mathcal{Y}$ then
    Return $\perp$
$M \leftarrow \mathsf{D}(K, H, C)$
If $M \neq \perp$ then
    $\mathsf{win} \leftarrow \mathsf{true}$
Return $M$
```

Figure 3.1: Confidentiality (left) and ciphertext integrity (right) games for an AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$.

define below.

Real-or-random security. We target a strong notion of confidentiality, first introduced in the AE setting in [138] and in the AEAD setting in [133], which requires that ciphertexts are indistinguishable from random bit strings of appropriate length. Consider the games REAL and RAND shown in the left hand panel of Figure 3.1 for an AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$. At the start of both games, the challenger generates a key $K \leftarrow_\$ \mathsf{K}$, and the attacker is given access to an oracle to which they may submit header / message pairs (H, M). In game REAL, the oracle returns the corresponding encryption $C \leftarrow_\$ \mathsf{E}(K, H, M)$. In game RAND, the oracle returns a random string of length $\mathsf{clen}_{\mathsf{AE}}(|M|)$. The ROR advantage of an attacker $\mathcal{A}$, who makes at most q queries to their real-or-random encryption oracle, is defined

$$\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{A}, q) = \left|\Pr\left[\mathrm{REAL}^{\mathcal{A}}_{\mathsf{AEAD}} \Rightarrow 1\right] - \Pr\left[\mathrm{RAND}^{\mathcal{A}}_{\mathsf{AEAD}} \Rightarrow 1\right]\right| .$$

Ciphertext integrity. For authenticity, we target the ciphertext integrity notion introduced for AE by Bellare and Namprempre in [19], and which we present here extended to accommodate associated data in the usual way. Intuitively, this notion requires that without knowledge of the secret key, K, it is infeasible to produce previously unseen ciphertexts which decrypt correctly. Consider game CTXT, shown in the right-hand panel of Figure 3.1 for an AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$. At the start of the game, a secret key $K \leftarrow_\$ \mathsf{K}$ is generated, and attacker $\mathcal{A}$ is given access to a pair of oracles Enc and

ChalDec. The attacker may learn ciphertexts encrypted under K by submitting header / message pairs (H, M) to oracle Enc, receiving $C \leftarrow\!\!\$\ \mathsf{E}(K, H, M)$ in response. $\mathcal{A}$ may also submit header / ciphertext pairs (H, C) to the challenge decryption oracle ChalDec, which are then decrypted under $\mathsf{D}(K, \cdot, \cdot)$. If $\mathcal{A}$ queries ChalDec on a pair (H, C) for which C was *not* previously returned in response to an ChalEnc query on $(H, \cdot)$ but which nonetheless decrypts correctly then the game returns true. We define the advantage of an attacker $\mathcal{A}$ in game CTXT, who makes at most q_d queries to oracle ChalDec and q_e queries to oracle Enc, as

$$\mathbf{Adv}^{\mathrm{ctxt}}_{\mathsf{AEAD}}(\mathcal{A}, q_d, q_e) = \Pr\left[\,\mathrm{CTXT}^{\mathcal{A}}_{\mathsf{AEAD}} \Rightarrow \mathsf{true}\,\right].$$

Constructions of AEAD schemes. AEAD schemes are typically constructed from block ciphers (see Section 2.3.1). We will see examples of two AEAD schemes, OCB [99, 134, 137] and GCM [107], later in the chapter.

3.2.3 Commitment Schemes

At a high level, a commitment scheme allows a party to 'commit' to a message without revealing the message contents in a way that can later be independently verified when the message is revealed. In this work we will make use of commitment schemes with an explicit verification algorithm, known as *commitment schemes with verification.* We formalise these in the following definition.

Definition 3.2. *A commitment scheme with verification is a pair of algorithms* $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$ *defined below. Associated to any such scheme is a message space* $\mathcal{M} \subseteq \Sigma^*$*, a commitment space* $\mathcal{CS} \subseteq \Sigma^*$*, and an opening space* $\mathcal{V} \subseteq \Sigma^*$*.*

- *The randomised commitment algorithm* $\mathsf{Com} : \mathcal{M} \to \mathcal{CS} \times \mathcal{V}$ *takes as input a message* $M \in \mathcal{M}$*, and outputs a commitment / opening pair* $(c, vk) \in \mathcal{CS} \times \mathcal{V}$*.*
- *The deterministic verification algorithm* $\mathsf{VerC} : \mathcal{CS} \times \mathcal{V} \times \mathcal{M} \to \{0, 1\}$ *takes as input a commitment, opening, and message tuple* $(c, vk, M) \in \mathcal{CS} \times \mathcal{V} \times \mathcal{M}$*, and outputs a bit* $b \in \{0, 1\}$*.*

We require that commitments returned by CS *are of some fixed length* t*; that is to say that* $\Pr\left[\,|\mathsf{Com}(M)| = t\,\right] = 1$ *for all* $M \in \mathcal{M}$*. A commitment scheme is correct if for all*

$M \in \mathcal{M}$ *it holds that* $\Pr[\mathsf{VerC}(\mathsf{Com}(M), M) = 1] = 1$. *In both cases, the probability is over the coins of* Com.

Binding commitments. We require that commitment schemes satisfy a *binding* property, which says that it is infeasible to find two opening / message pairs $(vk, M), (vk', M')$ such that M and M' are distinct which both successfully verify with the same commitment c. We call this property v-BIND, and define the v-BIND advantage of an attacker $\mathcal{A}$ against a commitment scheme CS to be

$$\mathbf{Adv}^{\mathrm{vBIND}}_{\mathsf{CS}}(\mathcal{A}) = \Pr\Big[b = b' = 1 \wedge M \neq M' \ : \ ((vk, M), (vk', M'), c) \leftarrow\!\!\$\ \mathcal{A}\ ; \\ b \leftarrow \mathsf{VerC}(c, vk, M)\ ; b' \leftarrow \mathsf{VerC}(c, vk', M')\Big]\ ,$$

where the probability is over the coins of $\mathcal{A}$.

Commitment schemes are usually required to satisfy a hiding property, which says that the commitments of two messages M, M' are indistinguishable. Since this will not be explicitly required in this thesis, we omit the formal definition here.

3.3 Invisible Salamanders: Breaking Facebook's Message Franking Scheme

In this section, we demonstrate an attack against Facebook's attachment franking scheme. The attack allows a malicious sender to send an abusive attachment to a user such that all attempts by the user to report this to Facebook will fail.

Attack overview. Facebook uses AES-GCM to encrypt attachments sent via Secret Conversations [65], the end-to-end encryption feature in Messenger. The attack creates a 'colliding' GCM ciphertext which decrypts to an abusive attachment via one key and an innocuous attachment via another. This, combined with the behaviour of Facebook's server-side abuse report generation code, prevents abusive messages from being reported to Facebook. Since messages in Secret Conversations are called 'salamanders' by Facebook (perhaps inspired by the Axolotl ratchet used in Signal, named after an endangered salamander), ensuring Facebook does not see a message essentially makes it an *invisible salamander*. We responsibly disclosed the vulnerability to Facebook. They remediated the vulnerability and awarded a bug bounty for reporting the issue.

Discussion. As mentioned above, the attack hinges on the fact that it is easy to construct GCM ciphertexts which decrypt to different valid messages under different keys. A scheme for which this is infeasible is said to be *robust*. Robust encryption schemes have been well-defined and studied in the literature [4, 66, 67, 115]; however, to the best of our knowledge this is the first real-world attack that has exploited the non-robustness of an encryption scheme. This is an illustration of how security issues which may seem more of theoretical interest than of practical concern can induce real-world vulnerabilities when used as a component in a complex protocol. More generally, the attack is enabled by a subtle interaction between a number of features of the ad hoc franking scheme employed by Facebook, and would have been prevented had a secure ccAEAD scheme been used. This highlights how easily subtle flaws can go unnoticed in protocols, and underscores the importance of using provably secure schemes wherever possible.

Motivation for faster ccAEAD. The ad hoc attachment franking scheme used by Facebook seems likely to have been chosen for efficiency reasons, since existing ccAEAD scheme do not match the performance profile of the fastest AEAD schemes such as GCM and OCB. In the next sections we address this by exploring the limitations of efficient ccAEAD, and construct the first single-pass ccAEAD scheme.

3.3.1 Attack Preliminaries

We begin with a brief overview of the nonce-based AEAD scheme Galois / Counter Mode (GCM) [107,108], which is used as a component in Facebook's attachment franking scheme. We begin by recalling CTR-mode encryption.

CTR-mode encryption. Let $\mathrm{E} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. For a key $K \in \mathcal{K}$, nonce $N \in \{0,1\}^n$, and message $M \in \{0,1\}^{m \cdot n}$ for some $m \in \mathbb{N}$, we define CTR-mode encryption to be the algorithm $\mathsf{CTR.E}$ that on input (K, N, M) outputs $C_1 \parallel \ldots \parallel C_m$, where $C_i \leftarrow \mathrm{E}(N{+}i) \oplus M_i$ for $i = 1, \ldots, m$ and $(M_1, \ldots, M_m) \leftarrow \mathrm{Parse}_n(M)$.

Similarly, we define CTR-mode decryption $\mathsf{CTR.D}$ to be the algorithm that on input $K \in \mathcal{K}$, $N \in \{0,1\}^n$, and ciphertext $C \in \{0,1\}^{m \cdot n}$ outputs $M_1 \,\|\, \ldots \,\|\, M_m$, where $M_i \leftarrow \mathrm{E}(N+i) \oplus C_i$ for $i = 1, \ldots, m$ and $(C_1, \ldots, C_m) \leftarrow \mathrm{Parse}_n(C)$. For simplicity we assume here that the length of both message and ciphertext are multiples of n-bits. This will suffice for our purposes, however it is straightforward to generalise CTR-mode to handle messages of arbitrary length.

An Overview of GCM. The AEAD scheme $\mathsf{GCM} = (\mathsf{GCM.K}, \mathsf{GCM.E}, \mathsf{GCM.D})$ is built from a block cipher $\mathrm{E} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$. At a high level, GCM is an encrypt-then-MAC (EtM) [19] composition of CTR-mode encryption and a Carter-Wegman (CW) [164] message authentication code (MAC). The key generation algorithm $\mathsf{GCM.K}$ simply returns a key $K \leftarrow\!\!\$\, \mathcal{K}$. The encryption and decryption algorithms are shown in Figure 3.2. For simplicity, we have assumed in our presentation that $n = 128$ and that H, M, and C are all multiples of n-bits. On input (K, N, H, M), $\mathsf{GCM.E}$ encrypts M in CTR-mode using K and N. It additionally computes a MAC tag T[2] using the GCM polynomial hash GHASH, also shown in Figure 3.2. GHASH works by taking the header and ciphertext blocks, plus a block encoding their lengths, to be the coefficients of a polynomial over $\mathrm{GF}(2^{128})$ which is then evaluated at $E_K(0^{128})$. The resulting value is masked using the encrypted nonce as a random pad to produce T. Decryption via $\mathsf{GCM.D}$ simply decrypts the input ciphertext using CTR-mode decryption and recomputes and verifies the tag T.

Nonces. The algorithms in Figure 3.2 are an (insecure) simplification of GCM. Firstly, the nonces here are taken to be 128-bit strings. In the standardised version of GCM it is recommended that nonces are 96-bits in length, which are then padded to 128-bits via $N \,\|\, 0^{31} \,\|\, 1$ prior to being used for encryption / decryption as described. If a nonce is not 96-bits, it is padded to a multiple of 128-bits and hashed to 128-bits using GHASH. A second difference is that in the CTR-mode encryption step in the standardised version of GCM only the lower order 32-bits of the nonce are incremented (modulo 2^{32}), whereas here we simply increase the value of the nonce by one (implicitly understood to be modulo 2^{128}). These discrepancies are due to an error in the definition of GCM used in the paper [59] upon which the work in this chapter is based, and unfortunately the error regarding nonce structure renders the version presented here insecure as an AEAD scheme. However, it is straightforward to verify that the attacks presented in this chapter apply analogously to

[2]For simplicity, we omit the optional step to truncate the tag T. It is straightforward to verify that all results presented in this section extend to the case in which the truncation step is included.

```
GCM.E($K, N, H, M$)
$X \leftarrow E_K(0^{128})$
$P \leftarrow E_K(N)$
$C \leftarrow \mathsf{CTR.E}(K, N, M)$
$T \leftarrow P \oplus \mathsf{GHASH}(X, H, C)$
Return $(C, T)$

GCM.D($K, N, H, (C, T)$)
$X \leftarrow E_K(0^{128})$
$P \leftarrow E_K(N)$
$T' \leftarrow P \oplus \mathsf{GHASH}(X, H, C)$
If $T' \neq T$
  Return $\perp$
$M \leftarrow \mathsf{CTR.D}(K, N, C)$
Return $M$
```

```
GHASH($X, H, C$)
$\mathsf{lens} \leftarrow (|H|)_{64} \,\|\, (|C|)_{64}$
$m \leftarrow |C|/128$ ; $h \leftarrow |H|/128$
$(H_1, \ldots, H_h) \leftarrow \mathrm{Parse}_{128}(H)$
$(C_1, \ldots, C_m) \leftarrow \mathrm{Parse}_{128}(C)$
$\mathsf{blen} \leftarrow m + h$
$T \leftarrow \mathsf{lens} \times_{GF} X$
For $i = 1$ to $h$
  $T \leftarrow T \oplus (H_i \times_{GF} X^{\mathsf{blen}+2-i})$
For $i = 1$ to $m$
  $T \leftarrow T \oplus (C_i \times_{GF} X^{\mathsf{blen}+2-i-h})$
Return $T$
```

Figure 3.2: The AEAD scheme $\mathsf{GCM} = (\mathsf{GCM.K}, \mathsf{GCM.E}, \mathsf{GCM.D})$.

the standardised version of GCM, and so the presentation given here is sufficient for our purposes. Indeed, the difference in counter iteration does not affect any of our results, and that regarding nonces only slightly alters the search for a nonce for the advanced variant of the attack (see Section 3.3.4).

GCM is not robust. An encryption scheme is said to be robust if it is infeasible to construct a ciphertext which decrypts correctly under different keys. As we shall see, GCM is *not* robust and it is this lack of robustness that facilitates the attack. We formally define robustness and analyse its relation to binding ccAEAD in Section 3.7.6.

3.3.2 Facebook's Attachment Franking

A diagram of Facebook's franking protocol for attachments (e.g., images and videos) is shown in Figure 3.3. The protocol uses Facebook's ccAEAD scheme for chat messages described in [65, 112] and analysed in [73] (there called CtE2) as a subroutine. (Readers interested in the specifics of CtE2 should consult GLR [73]; the only salient detail here is that it is secure as a ccAEAD scheme.) Certain encryption and HMAC keys as well as details like headers and associated data which are not important to the presentation of the protocol or our attack have been removed for simplicity in the diagram and prose below, see [65, 73] for additional details. For ease of exposition we divide the protocol into three phases: the *sending phase* involving the sender Alice and Facebook, the *receiving phase* involving the receiver Bob and Facebook, and the *reporting phase* between Bob and Facebook.

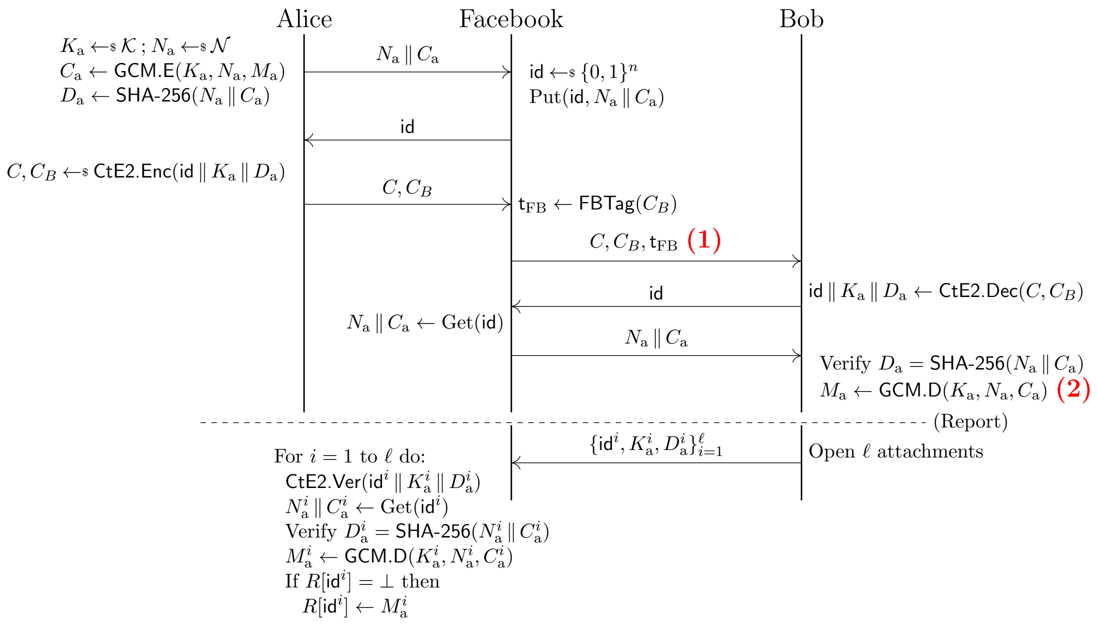

Alice
Facebook
Bob
$K_a \leftarrow_{\$} \mathcal{K}$; $N_a \leftarrow_{\$} \mathcal{N}$
$C_a \leftarrow \mathsf{GCM.E}(K_a, N_a, M_a)$
$D_a \leftarrow \mathsf{SHA\text{-}256}(N_a \,\|\, C_a)$
$N_a \,\|\, C_a$
$\mathsf{id} \leftarrow_{\$} \{0,1\}^n$
$\mathrm{Put}(\mathsf{id}, N_a \,\|\, C_a)$
id
$C, C_B \leftarrow_{\$} \mathsf{CtE2.Enc}(\mathsf{id} \,\|\, K_a \,\|\, D_a)$
C, C_B
$\mathsf{t}_{\mathrm{FB}} \leftarrow \mathsf{FBTag}(C_B)$
$C, C_B, \mathsf{t}_{\mathrm{FB}}$ (1)
id
$\mathsf{id} \,\|\, K_a \,\|\, D_a \leftarrow \mathsf{CtE2.Dec}(C, C_B)$
$N_a \,\|\, C_a \leftarrow \mathrm{Get}(\mathsf{id})$
$N_a \,\|\, C_a$
Verify $D_a = \mathsf{SHA\text{-}256}(N_a \,\|\, C_a)$
$M_a \leftarrow \mathsf{GCM.D}(K_a, N_a, C_a)$ (2)
(Report)
$\{\mathsf{id}^i, K_a^i, D_a^i\}_{i=1}^{\ell}$
Open ℓ attachments
For $i = 1$ to ℓ do:
$\mathsf{CtE2.Ver}(\mathsf{id}^i \,\|\, K_a^i \,\|\, D_a^i)$
$N_a^i \,\|\, C_a^i \leftarrow \mathrm{Get}(\mathsf{id}^i)$
Verify $D_a^i = \mathsf{SHA\text{-}256}(N_a^i \,\|\, C_a^i)$
$M_a^i \leftarrow \mathsf{GCM.D}(K_a^i, N_a^i, C_a^i)$
If $R[\mathsf{id}^i] = \perp$ then
$R[\mathsf{id}^i] \leftarrow M_a^i$

Figure 3.3: Facebook's attachment franking protocol [112, 113]. Certain encryption and HMAC keys have been omitted for simplicity. The sending phase consists of everything from the upper-left corner to the message marked **(1)**. The receiving phase consists of everything strictly after **(1)** and before **(2)**. The reporting phase is below the dashed line. The descriptions of Facebook's behaviour during the reporting phase were paraphrased (with permission) from conversations with Jon Millican, who we thank for his assistance.

Sending phase. Alice begins the sending phase with an attachment M_a to send to Bob. In the first part of the sending phase, Alice generates a key K_a and nonce N_a and encrypts M_a using AES-GCM to obtain a ciphertext C_a. The sender computes the SHA-256 digest D_a of $N_a \,\|\, C_a$, and sends Facebook $N_a \,\|\, C_a$ for storage. Facebook generates a random identifier id, and puts $N_a \,\|\, C_a$ in a key-value data structure with key id. Facebook then sends id to Alice. In the second part of the sending phase, Alice encrypts the message $\mathsf{id} \,\|\, K_a \,\|\, D_a$ using CtE2 to obtain the ccAEAD ciphertext C, C_B. Below, we will call such a message (which contains an identifier, key, and digest) an *attachment metadata* message. Alice sends C, C_B to Facebook, who in turn compute an authentication tag over C_B using FBTag[3] to obtain t_{FB}. Facebook sends $C, C_B, \mathsf{t}_{\mathrm{FB}}$ to the receiver.

Receiving phase. Upon receiving a ciphertext $C, C_B, \mathsf{t}_{\mathrm{FB}}$ from Alice (via Facebook), Bob runs CtE2.Dec on C, C_B to obtain $\mathsf{id} \,\|\, K_a \,\|\, D_a$. Bob then sends id to Facebook, who retrieve the value $N_a \,\|\, C_a$ associated with id from their key-value store and send it to Bob. Bob verifies that $D_a = \mathsf{SHA\text{-}256}(N_a \,\|\, C_a)$ and decrypts C_a to obtain the attachment

[3]FBTag is a MAC algorithm used in Facebook's standard message franking protocol which applies HMAC-SHA-256 keyed with an internal Facebook key to the input plus some metadata; see [65, 73] for full details.

content M_a.

Reporting phase. To report an abusive message, Bob sends all recently received messages to Facebook along with their tags t_FB and the keys with which to verify their ccAEAD commitments (not pictured in the diagram). For each message, Facebook verifies the commitment using $\mathsf{CtE2.Ver}$ and the authentication tag t_FB using its internal HMAC key. Then, if the commitment verifies correctly and the message contains attachment metadata, Facebook retrieves the attachment ciphertext and nonce $N_\mathrm{a} \,\|\, C_\mathrm{a}$ from its key-value store using its identifier id. Facebook verifies that $D_\mathrm{a} = \mathsf{SHA\text{-}256}(N_\mathrm{a} \,\|\, C_\mathrm{a})$ and decrypts C_a with K_a and N_a to obtain the attachment content M_a. If no other attachment metadata message containing identifier id has already been seen, the plaintext M_a is added to the abuse report R. (Looking ahead, this is the application-level behaviour that enables the attack which will violate the one-to-one correspondence between id and plaintext that is assumed here.)

Attack intuition. For our attack threat model, we consider a malicious Alice who wants to send an abusive attachment to Bob, but prevent Bob from reporting it to Facebook. The attachment could be an offensive image (e.g., a picture of abusive text or of a gun) or a video. We focus our discussion below on images.

The attack has two main steps: **(1)** generating a colliding GCM ciphertext; and (**2**) sending it twice to Bob. In step **(1)**, Alice creates two GCM keys and a single GCM ciphertext which decrypts (correctly) to the innocuous attachment under one key and to an abusive attachment under the other key. In step **(2)**, Alice sends the ciphertext to Bob twice using the same identifier, accompanied first by the innocuous key and then by the abusive key. On receiving the two messages, Bob decrypts the image twice and sees both the abusive attachment and the innocuous one. However when Bob reports the conversation to Facebook, its server-side code verifies both decryptions of the image ciphertext but only inserts the innocuous decryption into the abuse report. As such, the human making the abusive-or-not judgement will have no idea that Bob saw the abusive attachment.

We will describe two variants of the attack. We begin with the case in which the second decryption of the colliding ciphertext is junk bytes with no particular structure. This variant is simple but easily detectable, since the junk bytes will not display correctly. We

```
Collide-GCM($K_1, K_2, N_a, C$)
$X_1 \leftarrow E_{K_1}(0^{128})$ ; $X_2 \leftarrow E_{K_2}(0^{128})$
$P_1 \leftarrow E_{K_1}(N_a)$ ; $P_2 \leftarrow E_{K_2}(N_a)$
$m \leftarrow |C|/128 + 1$
$(C_1, \ldots, C_{m-1}) \leftarrow \mathrm{Parse}_{128}(C)$
$\mathsf{lens} \leftarrow (0)_{64} \,\|\, (|C| + 128)_{64}$
$T_1 \leftarrow \mathsf{lens} \times_{GF} X_1$
$T_2 \leftarrow \mathsf{lens} \times_{GF} X_2$
For $i = 1$ to $m - 1$
    $T_1 \leftarrow T_1 \oplus (C_i \times_{GF} X^{m+2-i})$
    $T_2 \leftarrow T_2 \oplus (C_i \times_{GF} X^{m+2-i})$
$T_1 \leftarrow P_1 \oplus T_1$
$T_2 \leftarrow P_2 \oplus T_2$
$C_m \leftarrow (T_1 \oplus T_2) \cdot (X_1 \oplus X_2)^{-1}$
$C_a \leftarrow C \,\|\, C_m$
$T \leftarrow P_1 \oplus \mathsf{GHASH}(X_1, \varepsilon, C_a)$
Return $(C_a, T)$
```

Figure 3.4: The Collide-GCM algorithm, which takes a partial ciphertext C, a nonce N_a, and two keys K_1 and K_2 and computes a tag T and final ciphertext block such that the output nonce / ciphertext / tag triple (N_a, C_a, T) decrypts correctly under both keys. Array indexing is done in terms of 128-bit blocks. We assume all input bit lengths are multiples of 128 for simplicity, and that the input M_a to Collide-GCM is at least two blocks in length. We require that K_1 and K_2 are such that $X_1 \neq X_2$. Arithmetic (addition and multiplication) in GF(2^{128}) is denoted $\oplus$ and $\times_{GF}$ respectively. The function Collide-GCM can take arbitrary headers, but we omit these here for simplicity.

then give a more advanced variant of the attack, in which the second decryption correctly displays an innocuous attachment — in our case, a picture of a kitten.

3.3.3 The Simple Attack

Alice begins the attack with an abusive attachment M_a^{ab}. Alice chooses a nonce N_a and two distinct 128-bit GCM keys K_1 and K_2 such that $X_1 \neq X_2$ where $X_1 = \mathrm{E}_{K_1}(0^{128})$ and $X_2 = \mathrm{E}_{K_2}(0^{128})$ (looking ahead, this condition is necessary since Collide-GCM requires $X_1 \oplus X_2$ to be invertible in GF(2^{128})). Alice then computes a ciphertext C via CTR.E(K_1, N_a, M_a^{ab}), where recall that CTR.E denotes CTR-mode encryption. In Facebook's scheme Alice can choose the keys and the nonce but this is not necessary—any combination of two keys yielding distinct encryptions of 0^{128} and a nonce will work

The ciphertext C is almost, but not quite, the ciphertext Alice will use in the attack. To ensure GCM decryption is correct for both keys, Alice generates a colliding GCM tag and ciphertext using the algorithm Collide-GCM(K_1, K_2, N_a, C) shown in Figure 3.4. When called, Collide-GCM treats its input C as a partial ciphertext, and returns a final ciphertext block C_m and tag T such that the GCM ciphertext $(C \,\|\, C_m, T)$ decrypts correctly under both K_1 and K_2. In more detail, suppose C is $m-1$ blocks in length. Collide-GCM works by computing tags T_1 and T_2 for $C \,\|\, C_m$ using keys K_1 and K_2 respectively, where C_m is treated as an unknown n-bit block. We then set $T_1 \oplus T_2 = 0$, and solve the resulting linear equation for C_m. This ensures that the tags computed under K_1 and K_2 are equal for the extended ciphertext $C_a = C \,\|\, C_m$ and nonce N_a. While we use the final ciphertext block here, a different ciphertext block or a block of associated data could be used instead. The tuple $(N_{\mathrm{a}}, C_{\mathrm{a}}, T)$ correctly decrypts to $M_{\mathrm{a}}^{\mathrm{ab}}$ under K_1 and to another plaintext M_{j} under K_2. However, the plaintext M_{j} will be random bytes with no structure. The advanced variant of our attack in Section 3.3.4 will ensure that M_{j} is a correctly-formatted plaintext.

Sending the colliding ciphertext. Alice continues the sending phase with Facebook, obtaining an identifier id for the nonce / ciphertext pair $N_{\mathrm{a}} \,\|\, C_{\mathrm{a}}$. Alice then creates two attachment metadata messages: $\mathrm{MD}_1 = \mathsf{id} \,\|\, K_2 \,\|\, D_{\mathrm{a}}$ and $\mathrm{MD}_2 = \mathsf{id} \,\|\, K_1 \,\|\, D_{\mathrm{a}}$. Alice completes the remainder of the sending phase twice, first with MD_1 and then with MD_2. (The first message sent is associated to the junk message.) After finishing the receiving phase for MD_1, Bob will decrypt C_{a} with K_2, giving M_{j}. After finishing the receiving phase with MD_2, Bob will decrypt C_{a} with K_1 and see $M_{\mathrm{a}}^{\mathrm{ab}}$. We emphasise that both attachment metadata messages are valid, and no security properties of CtE2 are violated.

Reporting failure. When Bob reports the recent messages, Facebook will verify both MD_1 and MD_2, and check that the digest D_{a} matches the value $N_{\mathrm{a}} \,\|\, C_{\mathrm{a}}$ stored with identifier id. **However Facebook will only insert the first decryption, the plaintext M_{j}, into the abuse report.** The system sees that the second ciphertext has the same SHA-256 hash and identifier, and assumes it's a duplicate; as such, the report contains no trace of the message $M_{\mathrm{a}}^{\mathrm{ab}}$.

3.3.4 Advanced Attack Variant and Proof of Concept

Next we describe the advanced variant of the attack in which both decryptions correctly display as attachments, and our proof-of-concept implementation. Ensuring that both decryptions are valid attachments is important because the simple variant (in which one decryption consists of random bytes) may not have sufficed for a practical exploit if Face-

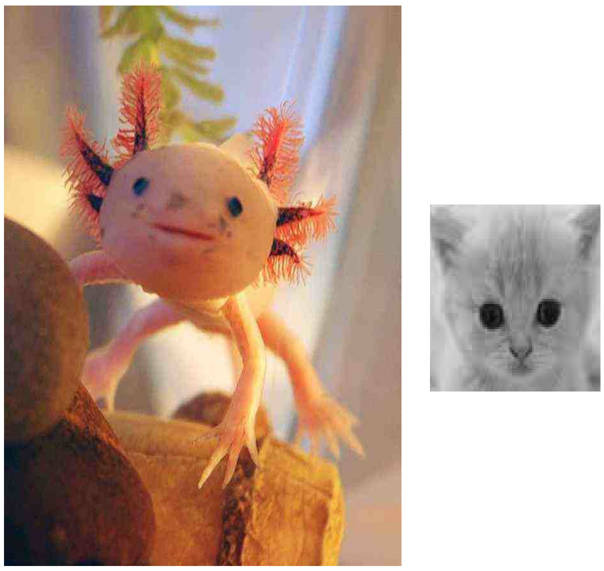


Figure 3.5: Two images with the same GCM ciphertext (C_{a}, T) when encrypted using 16-byte key $K_1 = (\texttt{03})^{16}$ or $K_2 = (\texttt{02})^{16}$, nonce $N_{\text{a}} = 1060666537A$, and associated data $H = (\texttt{ad})^{32}$ (all given in hex where exponentiation indicates repetition). **(Left)** The titular invisible salamander, which is delivered to the recipient but not inserted into the abuse report. **(Right)** An image of a kitten that is put in the recipient's abuse report instead of the salamander.

book only inserted valid images into their abuse reports. We implemented the advanced variant, and crafted a colliding ciphertext for which the 'abusive' decryption M_{a}^{ab} is the image of the axolotl salamander shown in Figure 3.5. The innocuous decryption M_{j} is the image of a kitten in that figure. We verified that both attachments display correctly in Facebook Messenger's browser client. Code for the proof of concept is available on request.

Generating colliding ciphertexts. The only difference between the advanced attack variant and the one described in Section 3.3.3 is the way in which Alice generates the ciphertext C_{a} which is input to Collide-GCM. Instead of simply encrypting the abusive attachment M_{a}^{ab} using CTR.E, Alice first merges M_{a}^{ab} and another innocuous attachment M_{j} using an algorithm $\mathsf{Att\text{-}Merge}(K_1, K_2, M_{\text{a}}^{\text{ab}}, M_{\text{j}})$ which takes the two keys and attachments as input, and outputs a nonce N_{a} and ciphertext C_{a} such that $\mathsf{CTR.D}(K_1, N_{\text{a}}, C_{\text{a}})$ displays M_{a}^{ab} and $\mathsf{CTR.D}(K_2, N_{\text{a}}, C_{\text{a}})$ displays M_{j}. The exact implementation of Att-Merge is file-format-specific, but for most formats Att-Merge has two main steps: **(1)** a *nonce search* to find a nonce that preserves certain file structures in the decryptions of C_a under K_1 and K_2; and **(2)** a *plaintext restructuring step* that expands the plaintexts with random bytes in locations that are ignored by parsers for their respective file formats. We implemented Att-Merge for JPEG and BMP images (the salamander image and the kitten

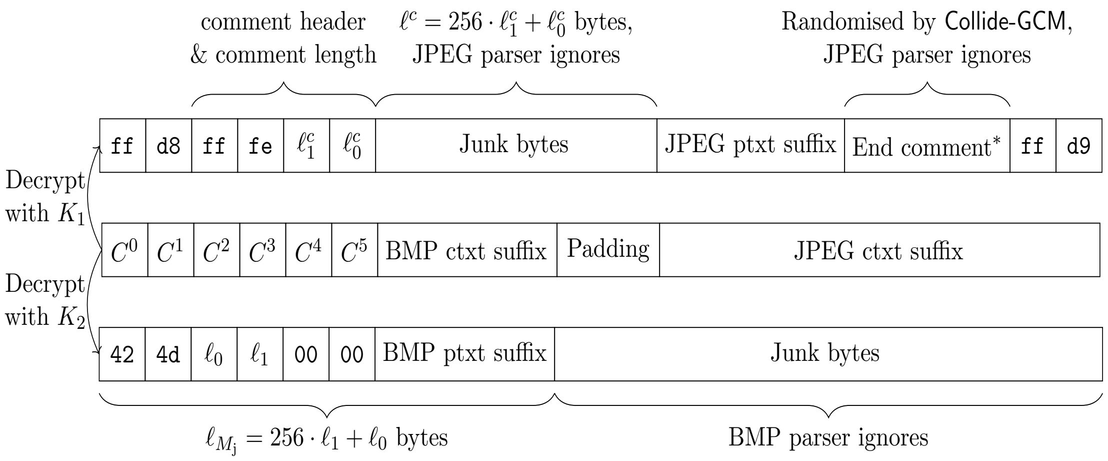


Figure 3.6: Diagram of the JPEG M_a^{ab} **(top)** and BMP M_j **(bottom)** plaintexts output by the plaintext restructuring step, and the corresponding ciphertext **(middle)**. The leftmost block of each file is the first byte. The 'BMP ptxt suffix' is the suffix of the original BMP starting at byte 6. The 'JPEG ptxt suffix' is the bytes of the original JPEG starting at byte 2 and ending before the final two bytes. The fifth and sixth bytes of the JPEG (marked ℓ_1^c and ℓ_0^c) are randomised during nonce search. (*) The region marked 'End comment' begins with the comment header and comment length bytes (which are *not* randomised by Collide-GCM), but we do not depict them for simplicity.

image respectively), so our discussion will focus on these formats.

File formats. Before discussing our implementation of Att-Merge, we will briefly describe the JPEG and BMP file formats.

JPEG files are of the form `ff` || `d8` || `JPEG data` || `ff` || `d9`. The two-byte sequence `ff` || `d8` *must* be the first two bytes, and the two-byte sequence `ff` || `d9` *must* be the final two bytes. There is no file length block in JPEG files — internal data structures have length fields but the total size of the file can be determined only after parsing. JPEG files can contain comments of up to 2^{16} bytes that are ignored by JPEG parsers. Comments are indicated with the two-byte sequence `ff` || `fe`, followed by a big-endian two-byte encoding of the comment length.

BMP files are of the form `42` || `4d` || `<length>` || `BMP data`. BMP files *must* begin with `42` || `4d`, and the next four bytes *must* be the length block. The length block in a BMP file is a four-byte (little-endian) encoding of the file length. All the BMP parsers we used only read the number of bytes indicated in the header and ignore trailing bytes.

Some intuition. At a high level, our Att-Merge proof-of-concept will craft the colliding ciphertext C_a by putting the encryption of the BMP under K_2 and the encryption of the JPEG under K_1 at different byte offsets in C_a. This will, of course, result in some portions of both plaintexts being randomised, but we use several features of the JPEG and BMP file formats to ensure that these random-looking bytes do not prevent the image from being correctly parsed and displayed (see Figure 3.6).

Nonce search. We begin by finding a suitable nonce that will enable ciphertext C_a to decrypt correctly to the different image formats under their respective keys. Since the actual image data occurs at different offsets in the two plaintexts, the nonce search need only find a nonce that preserves certain properties of the file formats for the bytes of the ciphertext that are semantically meaningful in *both* plaintext files. Here we will be concerned with the first four bytes of plaintext.

The structure of the JPEG and BMP files used in the proof-of-concept are shown in Figure 3.6. Looking ahead to the plaintext restructuring step, we will insert a comment immediately after the two-byte JPEG header and so the first four bytes of the JPEG plaintext must be $\texttt{ff} \,\|\, \texttt{d8} \,\|\, \texttt{ff} \,\|\, \texttt{ee}$. The BMP file must begin with the four byte sequence $\texttt{42} \,\|\, \texttt{4d} \,\|\, \ell_0 \,\|\, \ell_1$, where $\ell_0 \,\|\, \ell_1$ denote the first two bytes of the BMP's four byte length field. (Looking ahead, we will require the remaining two bytes of the BMP length field to be $\texttt{00}$, and so $\ell_0 \,\|\, \ell_1$ denotes the BMP image's length.)

To ensure that the attack ciphertext decrypts correctly under both keys, $\mathsf{Att\text{-}Merge}$ must output a nonce N_a and ciphertext C_a such that encrypting the first four bytes of the JPEG (resp. the BMP) under the key stream corresponding to K_1 (resp. K_2) yields the first four bytes of C_a (denoted C^0 through C^3 in Figure 3.6). Modelling the block cipher as an ideal cipher, each nonce $N \in \{0,1\}^{128}$ has roughly a 2^{-32} chance of meeting the required property. For the proof of concept, we wrote a simple Python script to search through nonces until we found 1060666537A, which produces the required collision. Finding that nonce took roughly three hours on a 3.4 GHz quad-core Intel i7.

While this nonce worked for our proof-of-concept, it is unfortunately not of the correct form for GCM as standardised due to the previously mentioned error in the published version of this chapter which incorrectly takes GCM nonces to be 128-bit strings. However, the same method works for the standardised version of GCM using 96-bit nonces, with the only difference being that one would restrict the search to $N \in \{0,1\}^{96} \times \{0^{31} \,\|\, 1\}$ as opposed to $N \in \{0,1\}^{128}$. This does result in a smaller search space, which in turn reduces the expected number of nonces of the correct form. However, since roughly every one in 2^{32} nonces is expected to satisfy the property, we would still expect there to be a sufficiently high proportion of suitable nonces that the search is efficient.

Though fast, fixing the keys and searching through nonces as in the proof-of-concept is not the fastest way to find a collision. Since the keys can be chosen arbitrarily, by fixing a nonce (of the correct form) and using a birthday attack against keys we would expect to produce a collision after only about 2^{16} encryptions.

Plaintext restructuring. After the nonce search, we restructure the plaintexts such that ciphertext C_a decrypts to the correct file format and reveals the correct image under the different keys. For JPEG and BMP images, Att-Merge proceeds by: **(1)** inserting the decryption (under K_1) of the BMP ciphertext into a comment region at the beginning of the JPEG; **(2)** inserting an additional comment at the end of the JPEG so that the bytes randomised by Collide-GCM are ignored by the JPEG parser; and **(3)** appending the decryption (under K_2) of the JPEG ciphertext to the end of the BMP plaintext. See Figure 3.6 for a diagram of the JPEG and BMP plaintexts after restructuring. We now describe each of these steps in more detail.

For step **(1)**, we put the decryption of the BMP ciphertext in the JPEG comment (inserted during the nonce search by placing the two-byte comment sequence `ff` $\|$ `fe` in bytes three and four of the JPEG file). Now the fifth and sixth bytes of the modified JPEG correspond to the comment's length field $\ell_1^c \,\|\, \ell_0^c$, where recall that JPEG comments can be at most 2^{16} bytes long. As we shall see, we cannot choose this length field (without additional brute-forcing during the nonce search). This is because we will place the BMP plaintext M_j in this JPEG comment, which in turn means that the length of the BMP (minus the header) cannot exceed the maximum JPEG comment length of 2^{16} bytes. Since BMP has a four byte length field (bytes three to six of the BMP plaintext), this restriction means that bytes five and size of the BMP — which correspond to the higher-order bytes of the BMP length field — must both be equal to `00`. For a binary string Z we define $Z[a \cdots c]$ to be the bytes of Z from index a to c inclusive (where we index from zero), and let P_{K_2} denote the keystream for the BMP key. Then this restriction implies that $C_a[4, \cdots, 5] = \texttt{00} \,\|\, \texttt{00} \oplus P_{K_2}[4, \ldots, 5]$. Now letting P_{K_1} denote the keystream for the JPEG key and noting that $\ell_1^c \,\|\, \ell_0^c = C_a[4, \ldots, 5] \oplus P_{K_1}[4, \ldots, 5]$, this implies that the

JPEG comment length field $\ell^c_1 \,\|\, \ell^c_o$ will be equal to $P_{K_1}[4,\ldots,5] \oplus P_{K_2}[4,\ldots,5]$.

This fixes the length of the JPEG comment, and we can now make the next bytes of the JPEG the 'decryption' of the BMP data's ciphertext under the JPEG key K_1. Letting $M^{\text{suff}}_{\text{j}}$ be everything but the first six bytes of the BMP M_{j}, then (again indexing from 0) the next bytes of the JPEG will be $M^{\text{suff}}_{\text{j}} \oplus P_{K_2}[6,\cdots,\ell_{M_{\text{j}}}-1] \oplus P_{K_1}[6,\cdots,\ell_{M_{\text{j}}}-1]$ where $\ell_{M_{\text{j}}}$ equals the length of M_{j} in bytes. If the BMP data is too short, we append $\ell^c - \ell_{M_{\text{j}}}$ random bytes to pad the comment up to the length given by the length field. This part of the JPEG comment will be random-looking bytes, but the JPEG parser will ignore it and jump to the byte after the comment.

Before discussing step **(2)**, we will make a few observations. Firstly, as mentioned above, we cannot choose the JPEG comment length $\ell^c = 256 \cdot \ell^c_1 + \ell^c_0$ without additional brute-forcing — rather, it is a random number in the range $[0,\ldots,2^{16}-1]$ and is fixed by the choice of the nonce and the two keys. Thus, smaller BMP files are better than large ones: if the length of the BMP file is $\ell_{M_{\text{j}}}$ and we model the block cipher as an ideal cipher, then each nonce with the four-byte collision we need gives $\Pr\left[\,\ell^c \geq \ell_{M_{\text{j}}} - 6\,\right] \approx (2^{16} - (\ell_{M_{\text{j}}} - 6))/2^{16}$ (where the probability is over the coins of the ideal cipher). In words, the probability of the comment length being greater than or equal to the file length is inversely proportional to the file length. The byte length of a BMP file is directly related to the number of pixels in the image, so the chosen BMP files should be fairly small in dimension to reduce the work required to find a nonce, since a nonce resulting in a too-short comment must be discarded. For example, the kitten image in Figure 3.5 is about one hundred pixels by eighty pixels and is in grayscale so that the number of bytes needed to describe each pixel is minimised. The kitten BMP file is $\ell_{M_{\text{j}}} = 9502$ bytes, and one of the two nonces we found during our search did *not* result in $\ell^c \geq \ell_{M_{\text{j}}} - 6$.

Moreover, we emphasise that nothing in step **(1)** has depended on the contents of either image, only on the length of the BMP. Thus the nonce output by the nonce search phase can be reused to create a colliding ciphertext for any valid JPEG and any BMP of the same length as M_{j} using K_1 and K_2.

We now return to the plaintext restructuring. In step **(2)**, we again expand the JPEG with an additional comment region. This comment region is placed immediately before the end-of-file indicator `ff` $\|$ `d9`. The comment region's length is 44 bytes, so combined with the comment header we add 48 total bytes to the end of the JPEG. This ensures that the second-to-last sixteen-byte block of the JPEG is always ignored by the parser. This is the block of ciphertext we will use to induce a tag collision in Collide-GCM. We could alternatively have used a block of associated data to induce the collision, and dispense with the second JPEG comment entirely. Using a block of ciphertext makes our proof-of-concept more realistic, since Facebook's GCM ciphertexts all have fixed associated data that cannot be modified. We define the modified JPEG file resulting from steps **(1)** and **(2)** to be M'_{a}.

Finally in step **(3)**, we append the 'decryption' under K_2 of the suffix of $\mathsf{CTR.E}(K_1, N_{\mathrm{a}}, M'_{\mathrm{a}})$ beginning at byte $\ell^c + 6$ to the BMP plaintext M_{j}. This step is straightforward — BMP parsers ignore trailing bytes, so our BMP image will still display correctly even when random-looking bytes are appended.

Implementing Collide-GCM. We implemented Collide-GCM in Python 2.7, and verified that colliding ciphertexts can be generated in roughly 45 seconds using an unoptimised implementation of GF(2^{128}) arithmetic. We checked decryption correctness using cryptography.io, a Python cryptography library which uses OpenSSL's GCM implementation. This sufficed as a proof-of-concept exploit for Facebook's engineering team.

3.3.5 Discussion And Mitigation

We chose JPEG and BMP files for our Att-Merge proof of concept because their formats can tolerate random bytes in different regions of the file (the beginning and the end, respectively). We did not try to extend Att-Merge to other common image formats (like PNG, TIFF or GIF), but file format tricks similar to the ones described above can be used to craft ciphertexts that decrypt to images in those formats. As an example, we will sketch a variant of the attack for which the colliding plaintexts are both JPEG files. It is similar to the JPEG / BMP collision described above, except for two differences. First, the JPEG taking the place of the BMP (i.e., the one placed in the beginning comment of the other JPEG) must end in another comment instead of `ff` $\|$ `d9`, which ensures the JPEG parser will ignore the trailing random bytes. Second, a two-byte collision must be found in the final two bytes of the keystream so that both JPEGs end in `ff` $\|$ `d9`. A birthday attack on keys should find keystreams with the correct structure (i.e., the first four and last two bytes are the same) in roughly 2^{24} encryptions. We did not try to implement Att-Merge for video file formats. Such formats are more complex than image formats, but we conjecture it is possible to extend the attack to video files.

Relation to GLR. In [73], GLR proved that CtE2 is a secure ccAEAD scheme. Their proof only applies to CtE2 itself, not to the composition of CtE2 and GCM. Concretely, GLR analysed CtE2 as it is used for text chat messages in Messenger, but did not analyse how it is used for attachments. The Collide-GCM algorithm in Figure 3.4 is related to the r-BIND attack against GCM given by GLR [73]. However, their attack is insufficient to exploit Facebook's attachment franking — it only creates ciphertexts with colliding tags, but not the same ciphertext. Therefore using it against Facebook wouldn't work, because the SHA-256 hashes of the ciphertexts for the two images would not collide. The Collide-GCM algorithm works even if the entire ciphertext, including any headers and the nonce, act as the commitment, and the only opening data is the encryption key.

If Facebook's attachment franking protocol is viewed as a ccAEAD scheme by taking the CtE2 binding tag (i.e., the value C_B output by running $\mathsf{CtE2.Enc}(\mathsf{id} \,\|\, K_\mathrm{a} \,\|\, D_\mathrm{a})$ during the sending phase) to be the compact commitment to the attachment plaintext, the resulting scheme is not vulnerable to GLR's r-BIND attack. This is because C_B commits to both the hash D_a of the nonce / ciphertext pair and to the GCM key K_a.

Mitigating the attack. There are two main ways this attack can be mitigated. The first is a server-software-only patch that ensures abuse reports containing attachments are not deduplicated by attachment identifier. The second is changing the Messenger clients to use a ccAEAD scheme instead of GCM to encrypt attachments. In response to our bug report, Facebook deployed the first mitigation for two main reasons: **(1)** it did not require patching the Messenger clients (an expensive and time-consuming process); and **(2)** changing the client-side crypto while maintaining backwards compatibility with old clients is difficult. Despite requiring less engineering effort, we believe this mitigation has some important drawbacks. Most notably, it leaves the underlying cryptographic issue intact: attachments are still encrypted using GCM. This means that future changes to either the Messenger client or Facebook's server-side code could re-expose the vulnerability. Using a secure ccAEAD scheme in place of GCM for attachment encryption would immediately prevent any deduplication behaviour from being exploited, since the binding security of ccAEAD implies that attachment identifiers uniquely identify the attachment *plaintexts*.

Fast ccAEAD from encryptment. The fact that use of a ccAEAD scheme would have prevented the above attack begs the question of whether a secure ccAEAD scheme can match the efficiency of the fastest AEAD schemes such as GCM. We shall explore this question in the following sections. We begin by introducing a new primitive called encryptment, which serves as a stepping stone to our results on fast ccAEAD.

3.4 A New Primitive: Encryptment

In this section, we introduce a new primitive called an *encryptment scheme*. Encryptment schemes allow both encryption of, and commitment to, a message (see Section 3.2). Moreover, the schemes which we target and ultimately build achieve both security goals with only a *single* pass over the underlying data.

3.4.1 Encryptment Syntax

While the syntax of encryptment schemes is similar to that of the ccAEAD schemes we ultimately look to build, the crucial difference is that we expect far more minimal security notions from encryptment schemes (see Section 3.7 for a more detailed discussion). Looking ahead, we shall see that a secure encryptment scheme is the key building block for more complex primitives such as ccAEAD schemes, robust AE [67], cryptographic concealments [56], and domain extension for authenticated encryption and remotely keyed AE [56], facilitating the construction of very efficient instantiations of these primitives. In Sections 3.7.3 and 3.7.4 we show how to build ccAEAD from encryptment, and discuss the other primitives in Section 3.8.

Encryptment syntax. We define an encryptment scheme to be a tuple of algorithms $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$. Applying the encryptment algorithm EEnc to a given key, header, and message tuple (K_{EC}, H, M) returns a pair $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ which we call an *encryptment*. We refer to encryptment component C_{EC} as the *ciphertext*, and to B_{EC} as the *binding tag*. Together the ciphertext / binding tag pair $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ function as an encryption of M under key K_{EC}, so that given $(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$, the decryptment algorithm EDec can recover the underlying message M. The binding tag B_{EC} simultaneously acts as a commitment to the underlying header and message, with opening K_{EC}; the validity of this commitment to a given pair (H, M) is checked by the verification algorithm EVer. Looking ahead, we will actually require that B_{EC} acts as a commitment to the opening key K_{EC} also, in that it should be infeasible to find $K_{\mathsf{EC}} \neq K'_{\mathsf{EC}}$ which verify the same B_{EC}. We formalise this in the following definition.

Definition 3.3. *An encryptment scheme is a tuple of algorithms* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ *defined as follows. Associated to the scheme is a key space* $\mathcal{K}_{\mathsf{EC}} \subseteq \Sigma^*$*, header space* $\mathcal{H}_{\mathsf{EC}} \subseteq \Sigma^*$*, message space* $\mathcal{M}_{\mathsf{EC}} \subseteq \Sigma^*$*, ciphertext space* $\mathcal{C}_{\mathsf{EC}} \subseteq \Sigma^*$*, and binding tag space* $\mathcal{T}_{\mathsf{EC}} \subseteq \Sigma^*$*.*

- *The randomised key generation algorithm* $\mathsf{EKg} \; : \to \mathcal{K}_{\mathsf{EC}}$ *takes no input, and outputs a key* $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$*.*
- *The encryptment algorithm* $\mathsf{EEnc} : \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}} \to \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$ *is a deterministic algorithm which takes as input a key* $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$*, a header* $H \in \mathcal{H}_{\mathsf{EC}}$*, and a message* $M \in \mathcal{M}_{\mathsf{EC}}$*, and outputs an encryptment* $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$*.*
- *The decryptment algorithm* $\mathsf{EDec} : \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}} \to \mathcal{M}_{\mathsf{EC}} \cup \{\perp\}$ *is a deterministic algorithm which takes as input a key* $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$*, a header* $H \in \mathcal{H}_{\mathsf{EC}}$*, and an encryptment* $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$*, and outputs a message* $M \in \mathcal{M}_{\mathsf{EC}}$ *or the error symbol* $\perp$*.*
- *The verification algorithm* $\mathsf{EVer} : \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}} \times \mathcal{K}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}} \to \{0, 1\}$ *is a deterministic algorithm which takes as input a header* $H \in \mathcal{H}_{\mathsf{EC}}$*, a message* $M \in \mathcal{M}_{\mathsf{EC}}$*, a key* $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$*, and a binding tag* $B_{\mathsf{EC}} \in \mathcal{T}_{\mathsf{EC}}$*, and returns a bit* b*.*

Length regularity and compactness. We impose two requirements on the lengths of encryptments. Firstly, we require *compactness*, by which we mean that the binding tags B_{EC} output by an encryptment scheme are of constant length btlen *regardless* of the length of the underlying message. Secondly, we require *length regularity*, defined analogously to the equivalent notion for AEAD schemes. Formally, we require that there exists a function $\mathsf{clen}_{\mathsf{EC}} : \mathbb{N} \to \mathbb{N}$ such that for all $(K_{\mathsf{EC}}, H, M) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}}$ it holds that $|C_{\mathsf{EC}}| = \mathsf{clen}_{\mathsf{EC}}(|M|)$, where $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$.

Correctness. We define two correctness notions for encryptment schemes, which we

```
$\underline{\mathrm{COR}_{\mathsf{EC}}(H, M)}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$M' \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
$b \leftarrow \mathsf{EVer}(H, M', K_{\mathsf{EC}}, B_{\mathsf{EC}})$
Return $(M = M' \wedge b = 1)$
```

```
$\underline{\text{S-COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
$(C'_{\mathsf{EC}}, B'_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}) = (C'_{\mathsf{EC}}, B'_{\mathsf{EC}}))$
```

Figure 3.7: Correctness games for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$.

formalise via the games COR and S-COR shown in Figure 3.7. We require that *all* encryptment schemes satisfy our all-in-one *correctness* notion, which requires that honestly generated encryptments decrypt to the correct underlying message and successfully verify with probability one. Formally, we say that an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ is correct if for all header / message pairs $(H, M) \in \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}}$, it holds that

$$\Pr\left[\, \mathrm{COR}_{\mathsf{EC}}(H, M) \Rightarrow 1 \,\right] = 1 \ ,$$

where the probability is over the coins of EKg. We additionally define *strong correctness*, which requires that for each tuple $(K_{\mathsf{EC}}, H, M) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{M}_{\mathsf{EC}}$ there is a unique encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$ such that $M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$. We formalise this in game S-COR, and say that an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ is strongly correct if for all tuples $(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, it holds that

$$\Pr\left[\, \text{S-COR}_{\mathsf{EC}}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \Rightarrow 1 \,\right] = 1 \ .$$

While we only require that encryptment schemes satisfy correctness, natural schemes typically possess the stronger property and indeed the schemes we build are strongly correct (which, as we will see, simplifies their security proofs). We note that strong correctness can be added to any encryptment scheme by making EDec recompute an encryptment after decrypting and returning $\perp$ if the two do not match; however, for efficiency we target schemes which achieve strong correctness without this.

3.4.2 Security Goals for Encryptment

We require that encryptment schemes achieve both one-time real-or-random (otROR) security, and a variant of one-time ciphertext integrity (SCU) which requires forging a ciphertext for a given binding tag with a known key; we motivate this variant below.

```
$\text{otROR0}_{\mathsf{EC}}^{\mathcal{A}}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\$ \mathcal{A}^{\text{Enc}(\cdot,\cdot)}$
Return $b$

$\text{Enc}(H, M)$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\bot$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
$\text{otROR1}_{\mathsf{EC}}^{\mathcal{A}}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\$ \mathcal{A}^{\$(\cdot,\cdot)}$
Return $b$

$\$(H, M)$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\bot$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$C_{\mathsf{EC}} \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{EC}}(|M|)}$
$B_{\mathsf{EC}} \leftarrow\$ \{0,1\}^{\mathsf{btlen}}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

Figure 3.8: One-time real-or-random (otROR) games for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$.

Confidentiality. We define otROR security for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ in terms of games otROR0 and otROR1 shown in Figure 3.8. Each game allows an attacker $\mathcal{A}$ to make one query of the form (H, M) to his real-or-random encryption oracle; in game otROR0 he receives back the encryptment of the input under a secret key, and in game otROR1 he receives back random bit strings. For an encryptment scheme EC and adversary $\mathcal{A}$, we define the otROR advantage of $\mathcal{A}$ against EC as

$$\mathbf{Adv}_{\mathsf{EC}}^{\text{ot-ror}}(\mathcal{A}) = \left| \Pr\left[\text{otROR0}_{\mathsf{EC}}^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\text{otROR1}_{\mathsf{EC}}^{\mathcal{A}} \Rightarrow 1\right] \right|,$$

where the probability is over the coins of EKg and $\mathcal{A}$.

Second-ciphertext unforgeability. We also ask that encryptment schemes meet an unforgeability goal that we call second-ciphertext unforgeability (SCU). In this game, the attacker first learns an encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ corresponding to a chosen header / message pair (H, M) under key K_{EC}. We then require that the attacker shouldn't be able to find a *distinct* header and ciphertext pair $(H', C'_{\mathsf{EC}}) \neq (H, C_{\mathsf{EC}})$ such that $\mathsf{EDec}(K_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B_{\mathsf{EC}})$ does not return an error. This should hold even if the attacker knows K_{EC}. Looking ahead, this is a necessary and sufficient condition needed from encryptment when using it to build ccAEAD schemes from fixed domain authenticated encryption.

Formally, the game SCU is shown in Figure 3.9. For an encryptment scheme EC and adversary $\mathcal{A}$, who makes at most q_c queries to their ChalDec oracle, we define the SCU advantage to be

$$\mathbf{Adv}_{\mathsf{EC}}^{\text{scu}}(\mathcal{A}, q_c) = \Pr\left[\text{SCU}_{\mathsf{EC}}^{\mathcal{A}} \Rightarrow \mathsf{true}\right],$$

```
$\underline{\mathrm{SCU}^{\mathcal{A}}_{\mathsf{EC}}}$
$K_{\mathsf{EC}} \leftarrow\!\$\ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\!\$\ \mathcal{A}^{\mathrm{Enc}(\cdot,\cdot),\mathrm{ChalDec}(\cdot,\cdot)}$
Return $\mathsf{win}$

$\underline{\mathrm{Enc}(H, M)}$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$K^*_{\mathsf{EC}} \leftarrow K_{\mathsf{EC}}$ ; $C^*_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$
$B^*_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$ ; $H^* \leftarrow H$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$

$\underline{\mathrm{ChalDec}(H', C'_{\mathsf{EC}})}$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then
    Return $\perp$
If $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$ then
    Return $\perp$
$M' \leftarrow \mathsf{EDec}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

```
$\underline{\mathrm{otCTXT}^{\mathcal{A}}_{\mathsf{EC}}}$
$K_{\mathsf{EC}} \leftarrow\!\$\ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\!\$\ \mathcal{A}^{\mathrm{Enc}(\cdot,\cdot),\mathrm{ChalDec}(\cdot,\cdot,\cdot)}$
Return $\mathsf{win}$

$\underline{\mathrm{Enc}(H, M)}$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then
    Return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$K^*_{\mathsf{EC}} \leftarrow K_{\mathsf{EC}}$ ; $C^*_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$
$B^*_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$ ; $H^* \leftarrow H$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$

$\underline{\mathrm{ChalDec}(H', C'_{\mathsf{EC}}, B'_{\mathsf{EC}})}$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then
    Return $\perp$
If $(H', C'_{\mathsf{EC}}, B'_{\mathsf{EC}}) = (H^*, {C_{\mathsf{EC}}}^*, {B_{\mathsf{EC}}}^*)$
    Return $\perp$
$M' \leftarrow \mathsf{EDec}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B'_{\mathsf{EC}})$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

Figure 3.9: Second-ciphertext unforgeability (SCU) and one-time ciphertext integrity (otCTXT) games for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$.

where the probability is again over the coins of EKg and $\mathcal{A}$.

We note that the challenge decryption oracle ChalDec in game SCU is not strictly necessary, since before $\mathcal{A}$ can query ChalDec he must previously have made an Enc query. As such, $\mathcal{A}$ knows the secret encryptment key and so can simulate the ChalDec oracle himself. Nonetheless, we opted to include ChalDec in the definition for ease of comparison with the similar otCTXT definition which we present now.

Integrity. We may also define a one-time notion of ciphertext integrity (see Section 3.2) for encryptment schemes. Here, the attacker is challenged to output a fresh encryptment which decrypts correctly given a single query to an encryptment oracle. We formalise this via the game otCTXT shown in the right hand panel of Figure 3.9. For an encryptment scheme EC and adversary $\mathcal{A}$, who makes at most q_c queries to oracle ChalDec, we define the otCTXT advantage to be

$$\mathbf{Adv}^{\text{ot-ctxt}}_{\mathsf{EC}}(\mathcal{A}, q_c) = \Pr\left[\,\mathrm{otCTXT}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true}\,\right] .$$

While we do not require that encryptment schemes satisfy the notion of otCTXT security, looking ahead to Section 3.7, those that do (when reframed in the ccAEAD syntax) constitute a secure *one-time ccAEAD scheme*. By this, we mean a ccAEAD scheme for which

```
$\text{sr-BIND}_{\mathsf{EC}}^{\mathcal{A}}$
$((H, M, K_{\mathsf{EC}}), (H', M', K'_{\mathsf{EC}}), B_{\mathsf{EC}}) \leftarrow\!\!\$\ \mathcal{A}$
$b \leftarrow \mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$
$b' \leftarrow \mathsf{EVer}(H', M', K'_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $(H, M, K_{\mathsf{EC}}) = (H', M', K'_{\mathsf{EC}})$ then
   Return $\mathsf{false}$
Return $(b = b' = 1)$
```

```
$\text{s-BIND}_{\mathsf{EC}}^{\mathcal{A}}$
$(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\!\!\$\ \mathcal{A}$
$M' \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M' = \perp$ then return $\mathsf{false}$
$b \leftarrow \mathsf{EVer}(H, M', K_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $b = 0$ then
   Return $\mathsf{true}$
Return $\mathsf{false}$
```

Figure 3.10: Binding notions for an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$.

a fresh secret key is chosen for each encryption. A one-time ccAEAD scheme is suitable for use in applications such as Signal, where ratcheting ensures that each encryption is essentially under a fresh key.

Binding security. We finally require that encryptment schemes satisfy certain binding notions. We start by generalising the receiver binding notion r-BIND for ccAEAD schemes from [73], and adapting the syntax to the encryptment setting. r-BIND security requires that no computationally efficient adversary can find two key, header, and message triples $(H, M, K_{\mathsf{EC}}), (H', M', K'_{\mathsf{EC}})$ and a binding tag B_{EC} such that $(H, M) \neq (H', M')$ and

$$\mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}}) = \mathsf{EVer}(H', M', K'_{\mathsf{EC}}, B_{\mathsf{EC}}) = 1 \ .$$

A simple strengthening of this notion — which we denote sr-BIND (for *strong receiver binding*) — allows the adversary to instead win if $(H, M, K_{\mathsf{EC}}) \neq (H', M', K'_{\mathsf{EC}})$. The pseudocode game sr-BIND is shown in Figure 3.10, where we define the sr-BIND advantage of an adversary $\mathcal{A}$ against EC as

$$\mathbf{Adv}_{\mathsf{EC}}^{\text{sr-bind}}(\mathcal{A}) = \Pr\left[\,\text{sr-BIND}_{\mathsf{EC}}^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] \ .$$

The corresponding game and advantage term for r-BIND security are defined analogously. It is not hard to see that the stronger sr-BIND notion implies the prior r-BIND notion, and indeed is strictly stronger; we will formally prove an analogous result for ccAEAD schemes in Section 3.7. For our purposes, it will simplify our negative results about rate-1 block cipher-based encryptment.

We additionally define the notion of *sender binding.* This ensures that a sender commits to the message underlying an encryptment by requiring that it is infeasible to find an encryptment which decrypts correctly but for which verification fails. Without this requirement, a malicious sender may be able to send an abusive message to a receiver with a faulty commitment such that a receiver is unable to report it. We define sender binding security formally via the game s-BIND in Figure 3.10. We define the s-BIND advantage of an adversary $\mathcal{A}$ against an encryptment scheme EC as

$$\mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{A}) = \Pr\left[\text{s-BIND}^{\mathcal{A}}_{\mathsf{EC}} \Rightarrow \mathsf{true}\right] .$$

Relation to ccAEAD. Given the simpler security properties expected of them, building highly efficient secure encryptment schemes is a more straightforward task than constructing ccAEAD directly. However, as we shall see, encryptment isolates the core complexity of building secure ccAEAD. In Section 3.6, we show how to construct a secure and single-pass encryptment scheme from cryptographic hash functions. In Section 3.7.3 and Section 3.7.4, we give efficient transforms which lift secure encryptment to secure ccAEAD. Together, our results will yield the first single-pass, single-primitive construction of ccAEAD.

Proving SCU security. One reason we have introduced encryptment as a standalone primitive (instead of directly working with the ccAEAD formulation from [73]) is that it simplifies security analyses. A useful tool for these analyses is the following lemma, which states that for any encryptment scheme EC that enjoys strong correctness, the combination of r-BIND and s-BIND security suffice to prove SCU security.

Lemma 3.1. *Let* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ *be a strongly correct encryptment scheme, and consider an attacker $\mathcal{A}$ in the* SCU *game against* EC*. Then there exist attackers $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{A}, q_c) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{B}) + \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{C}) ,$$

and moreover $\mathcal{B}$ and $\mathcal{C}$ both run in the same time as $\mathcal{A}$ plus an $\mathcal{O}(q_c)$ overhead.

At a high level, the proof of Lemma 3.1 proceeds as follows. Let $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$ be the tuple corresponding to $\mathcal{A}$'s single encryption query (H, M) in the SCU game, and suppose that $\mathcal{A}$ wins the game with decryption oracle query (H', C'_{EC}), meaning that $\mathsf{EDec}(K_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B_{\mathsf{EC}}) = M' \neq \bot$ and $(H', C'_{\mathsf{EC}}) \neq (H, C_{\mathsf{EC}})$. The proof first argues that if the scheme is s-BIND-secure, then any ciphertext which decrypts correctly must also verify correctly. As such, it follows that if $(H, M) \neq (H', M')$ then this can be used to construct a winning tuple for an attacker in the r-BIND game against EC; we bound the probability that this occurs with a reduction to r-BIND security. On the other hand, if $(H, M) = (H', M')$ then it must be the case that $C_{\mathsf{EC}} \neq C'_{\mathsf{EC}}$ — but this in turn implies that we have found distinct encryptments which decrypt to the same header and message under K_{EC}, violating strong correctness. We formalise this below.

Proof. We argue by a series of game hops, shown in Figure 3.11. We first define game G_0, which is identical to game SCU against EC except we change the specification of oracle ChalDec to perform a number of additional checks, and set flags depending on the results. These additional steps do not affect the outcome of the game, and so it follows that

$$\mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{A}, q_c) = \Pr[\, G_0 \Rightarrow 1\,] \; .$$

Next we define game G_1, which is identical to game G_0 except now if $\mathcal{A}$ makes a query to ChalDec which decrypts correctly to $M' \neq \bot$, but for which verification fails, then the game sets $M' = \bot$ and so the win flag will not be set. These games run identically until the flag bad_1 is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[\, G_0 \Rightarrow 1\,] - \Pr[\, G_1 \Rightarrow 1\,]| \leq \Pr[\, \mathsf{bad}_1 = \mathsf{true} \text{ in } G_0\,] \; .$$

We bound the probability of this event occurring with a reduction to the s-BIND security of EC. Let $\mathcal{B}$ be an adversary in game s-BIND against EC. $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine as follows. He generates a key $K^*_{\mathsf{EC}} \leftarrow\!\!\$\, \mathsf{EKg}$, and simulates $\mathcal{A}$'s Enc query by computing the required encryptment $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ and returning this, along with K^*_{EC}, to $\mathcal{A}$. He simulates the ChalDec oracle by applying $\mathsf{EDec}(K^*_{\mathsf{EC}}, \cdot, \cdot, B^*_{\mathsf{EC}})$ to the queried header / ciphertext pairs (H', C'_{EC}) and returning the result to $\mathcal{A}$. For queries which decrypt to $M' \neq \bot$, he computes $b' \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$. If he ever finds a tuple $(H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ which decrypts correctly under K^*_{EC} but which does not verify correctly, $\mathcal{B}$ halts and outputs $(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ to his challenger. Notice that such a tuple constitutes a winning query for $\mathcal{B}$, and that the flag bad_1 gets set if and only if such a tuple is found. It follows that

$$\Pr[\, \mathsf{bad}_1 = \mathsf{true} \text{ in } G_0\,] = \Pr\left[\, \text{s-BIND}^{\mathcal{B}}_{\mathsf{EC}} \Rightarrow 1\,\right] = \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{B}) \; . \tag{3.1}$$

Next we define game G_2, which is identical to G_1 except now if $\mathcal{A}$ submits a ChalDec query (H', C'_{EC}) which decrypts correctly to some message $M' \neq \bot$ and for which verification succeeds, then if (H', M') does not equal the challenge header / message pair (H^*, M^*) then ChalDec sets $M' = \bot$ and win does not get set to true. These games run identically until flag bad_2 gets set, and so the Fundamental Lemma of Game Playing implies that

$$\Pr[\, G_1 \Rightarrow 1\,] - \Pr[\, G_2 \Rightarrow 1\,]| \leq \Pr[\, \mathsf{bad}_2 = \mathsf{true} \text{ in } G_1\,] \; .$$

We bound the probability that this event occurs with a reduction to the r-BIND security of EC. Let $\mathcal{C}$ be an adversary in game r-BIND against EC. $\mathcal{C}$ runs $\mathcal{A}$ as a subroutine by choosing a key $K^*_{\mathsf{EC}} \leftarrow\!\!\$\, \mathsf{EKg}$ and simulating oracles Enc and ChalDec as described previously (except he now rejects any ChalDec queries which fail verification as per G_1). Let

$(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ denote the encryptment generated by the Enc query on input (H^*, M^*) with key K^*_{EC}, and notice that the correctness of EC implies that $1 \leftarrow \mathsf{EVer}(H^*, M^*, K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$. If at any point the attacker submits a query (H', C'_{EC}) to ChalDec such that $M' \leftarrow \mathsf{EDec}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ and $1 \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ but for which $(H', M') \neq (H^*, M^*)$, then $\mathcal{C}$ halts and outputs $((H^*, M^*, K^*_{\mathsf{EC}}), (H', M', K^*_{\mathsf{EC}}), B^*_{\mathsf{EC}})$. Such a tuple constitutes a win for $\mathcal{C}$ in his game, and notice that the flag bad_2 is set if and only if such a tuple is found. It follows that

$$\Pr[\,\mathsf{bad}_2 = \mathsf{true} \text{ in } G_1\,] = \Pr\left[\,\text{r-BIND}^{\mathcal{C}}_{\mathsf{EC}} \Rightarrow 1\,\right] = \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{C})\ . \tag{3.2}$$

Finally we define game G_3, which is identical to G_2 except that now if the attacker submits a query (H', C'_{EC}) to oracle ChalDec which decrypts correctly to message M', verifies correctly, and for which $(H', M') = (H^*, M^*)$, then if it is the case that $C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}}$ where C^*_{EC} is the ciphertext derived in the challenge query to Enc, the game sets $M' \leftarrow \perp$ and the win flag remains false. Notice that this game is impossible to win, since the only queries which will not be rejected by the added checks must be such that $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$, but then these themselves are rejected by the first line of pseudocode in ChalDec. It follows that

$$\Pr[\,G_3 \Rightarrow 1\,] = 0\ . \tag{3.3}$$

Notice that these games run identically unless the flag bad_3 is set, and so it holds that

$$|\Pr[\,G_2 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| \leq \Pr[\,\mathsf{bad}_3 = \mathsf{true} \text{ in } G_2\,]\ .$$

We claim that $\Pr[\,\mathsf{bad}_3 = \mathsf{true} \text{ in } G_2\,] = 0$. To see this, notice that bad_3 being set implies the existence of a tuple $(K^*_{\mathsf{EC}}, H^*, C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ such that $M^* \leftarrow \mathsf{EDec}(K^*_{\mathsf{EC}}, H^*, C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ but for which $C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}}$ where we know that $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K^*_{\mathsf{EC}}, H^*, M^*)$ from the challenge query to Enc. Moreover, notice that it must be the case that $(K^*_{\mathsf{EC}}, H^*, C'_{\mathsf{EC}}, B^*_{\mathsf{EC}}) \in \mathcal{K}_{\mathsf{EC}} \times \mathcal{H}_{\mathsf{EC}} \times \mathcal{C}_{\mathsf{EC}} \times \mathcal{T}_{\mathsf{EC}}$, otherwise EDec would return $\perp$. However since EEnc is deterministic, this then violates the assumed strong correctness of the scheme, proving the claim.

Putting together equations (3.1), (3.2), and (3.3) we find that

$$\mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{B}) + \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{C})\ ,$$

thereby concluding the proof.

```
proc. main // $G_0$, $\boxed{G_1}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\$ \mathcal{A}^{\mathsf{Enc}(\cdot,\cdot),\mathsf{Dec}(\cdot,\cdot)}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_0$, $\boxed{G_1}$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$K^*_{\mathsf{EC}} \leftarrow K_{\mathsf{EC}}$ ; $C^*_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$
$B^*_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$ ; $H^* \leftarrow H$ ; $M^* \leftarrow M$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$

proc. ChalDec$(H', C'_{\mathsf{EC}})$ // $G_0$, $\boxed{G_1}$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then return $\perp$
If $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$
    Return $\perp$
$M' \leftarrow \mathsf{EDec}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $M' = \perp$ return $\perp$
$b' \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $b' = 0$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ ; $\boxed{M' \leftarrow \perp}$
If $(H', M') \neq (H^*, M^*)$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
If $(C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}})$ then
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

```
proc. main // $G_2$, $\boxed{G_3}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$\mathsf{win} \leftarrow \mathsf{false}$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$\varepsilon \leftarrow\$ \mathcal{A}^{\mathsf{Enc}(\cdot,\cdot),\mathsf{Dec}(\cdot,\cdot)}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_2$, $\boxed{G_3}$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$K^*_{\mathsf{EC}} \leftarrow K_{\mathsf{EC}}$ ; $C^*_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}}$
$B^*_{\mathsf{EC}} \leftarrow B_{\mathsf{EC}}$ ; $H^* \leftarrow H$ ; $M^* \leftarrow M$
Return $((C_{\mathsf{EC}}, B_{\mathsf{EC}}), K_{\mathsf{EC}})$

ChalDec$(H', C'_{\mathsf{EC}})$ // $G_2$, $\boxed{G_3}$
If $\mathsf{query\text{-}made} = \mathsf{false}$ then return $\perp$
If $(H', C'_{\mathsf{EC}}) = (H^*, C^*_{\mathsf{EC}})$
    Return $\perp$
$M' \leftarrow \mathsf{EDec}(K^*_{\mathsf{EC}}, H', C'_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $M' = \perp$ return $\perp$
$b' \leftarrow \mathsf{EVer}(H', M', K^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$
If $b' = 0$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$ ; $M' \leftarrow \perp$
If $(H', M') \neq (H^*, M^*)$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$ ; $M' \leftarrow \perp$
If $(C'_{\mathsf{EC}} \neq C^*_{\mathsf{EC}})$ then
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$ ; $\boxed{M' \leftarrow \perp}$
If $M' \neq \perp$ then $\mathsf{win} \leftarrow \mathsf{true}$
Return $M'$
```

Figure 3.11: Games for proof of Lemma 3.1.

□

3.4.3 A Simple Encryptment Construction

Given an SE scheme $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ (defined below) and a commitment scheme with verification $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$, it is straightforward to construct an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ by composing the two. The resulting scheme is similar to the $\mathsf{CtE2}$ ccAEAD scheme from [73]; however we make a number of adaptations to the encryptment setting, where in particular the encryption algorithm must be deterministic. The scheme also requires weaker one-time security properties of the underlying primitives than $\mathsf{CtE2}$.

Symmetric encryption. We define the syntax of a *symmetric encryption* (SE) scheme to be identical to that of an AE scheme. In terms of security, we ask less of SE than AE by only requiring that SE achieves ROR security (as opposed to ROR and CTXT security

```
EKg
    $K \leftarrow\!\!\$ \; \mathsf{K}$
    $r_c \leftarrow\!\!\$ \; \mathcal{R}_{\mathsf{CE}}$
    $r_e \leftarrow\!\!\$ \; \mathcal{R}_{\mathsf{SE}}$
    $K_{\mathsf{EC}} \leftarrow (K, r_c, r_e)$
    Return $K_{\mathsf{EC}}$

EEnc($K_{\mathsf{EC}}, H, M$)
    $(K, r_c, r_e) \leftarrow K_{\mathsf{EC}}$
    $(c, vk) \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$
    $B_{\mathsf{EC}} \leftarrow c$
    $C_{\mathsf{EC}} \leftarrow \mathsf{E}(K, vk \,\|\, M; r_e)$
    Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$

EDec($K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}}$)
    $(K, r_c, r_e) \leftarrow K_{\mathsf{EC}}$
    $vk \,\|\, M \leftarrow \mathsf{D}(K, C_{\mathsf{EC}})$
    If $vk \,\|\, M = \bot$ return $\bot$
    $(c', vk') \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$
    If $(c', vk') \neq (B_{\mathsf{EC}}, vk)$
        Return $\bot$
    $C'_{\mathsf{EC}} \leftarrow \mathsf{E}(K, vk \,\|\, M; r_e)$
    If $C'_{\mathsf{EC}} \neq C_{\mathsf{EC}}$ or $C'_{\mathsf{EC}} = \bot$
        Return $\bot$
    Return $M$

EVer($H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}}$)
    $K_{\mathsf{EC}} \leftarrow (K, r_c, r_e)$
    $(c, vk) \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$
    If $c \neq B_{\mathsf{EC}}$ or $c = \bot$
        Return $\bot$
    Return 1
```

Figure 3.12: Construction of a simple encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ from a symmetric encryption scheme $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ and a commitment scheme with verification $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$.

for AE).

Construction. Consider the encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ shown in Figure 3.12. We define key generation EKg to output keys of the form $K_{\mathsf{EC}} = (K, r_c, r_e)$ where $K \leftarrow\!\!\$ \; \mathsf{K}$ is a key for the underlying SE scheme, and r_c, r_e are drawn randomly from the coin spaces $\mathcal{R}_{\mathsf{CS}}, \mathcal{R}_{\mathsf{SE}}$ of the commitment and SE scheme respectively. (Looking ahead, these are coins for the randomised algorithms that will be called by the deterministic EEnc). On input $((K, r_c, r_e), H, M)$, encryptment algorithm EEnc first computes a commitment / opening pair for $H \,\|\, M$ via $(c, vk) \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$ and sets $B_{\mathsf{EC}} = c$. It then encrypts the opening along with the message to give $C_{\mathsf{EC}} \leftarrow \mathsf{Enc}(K, vk \,\|\, M; r_e)$, and outputs encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$. For verification, on input $(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$ where $K_{\mathsf{EC}} = (K, r_c, r_e)$, EVer works by recomputing $(c, vk) \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$ and returns 1 if $c = B_{\mathsf{EC}}$ and 0 otherwise. (Notice that the correctness of $\mathsf{CS} = (\mathsf{Com}, \mathsf{VerC})$ and the fact that $(c, vk) \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$ implies that $1 \leftarrow \mathsf{VerC}(c, vk, H \,\|\, M)$, and so we do not need to explicitly verify the binding tag commitment $B_{\mathsf{EC}} = c$ during decryption.) Decryption EDec works by decrypting C_{EC} under key K to recover message M and opening vk. EDec then uses the coin components of $K_{\mathsf{EC}} = (K, r_c, r_e)$ to recompute the commitment and ciphertext, returning M only if both checks pass.

Security. It is straightforward to see that if SE and CS satisfy one-time notions of real-or-random security then EC satisfies our notion of otROR security. If the commitment scheme is v-BIND secure, this in turn implies that EC is r-BIND secure since any winning tuple in the r-BIND game also breaks the binding of the commitment scheme. (Our scheme is not sr-BIND secure, since neither the SE key, K, nor the encryption coin component, r_e, of $K_{\mathsf{EC}} = (K, r_c, r_e)$ are used by EVer. As such for any tuple $(c, vk, H \,\|\, M)$ such that $(c, vk) \leftarrow \mathsf{Com}(H \,\|\, M; r_c)$ an attacker can win game sr-BIND by outputting $((H, M, (K, r_c, r_e), (H, M, (K', r_c, r'_e)), c)$ for $(K', r'_e) \neq (K, r_e))$. s-BIND security and strong correctness follow, respectively, from the fact that the commitment is verified, and the encryptment recomputed, during decryption. Together these properties imply the scheme is SCU secure by Lemma 3.1. However, such generic compositions are inherently two pass and we seek faster schemes.

3.5 On Efficient Fixed-Key Block Cipher-Based Encryptment

We are interested in building encryptment schemes — and ultimately, more complex primitives such as ccAEAD schemes — from just a block cipher used on a small number of keys and other primitive arithmetic operations (XOR, finite field arithmetic, etc.). Beyond being an interesting theoretical question, there is the practical motivation that the current fastest AEAD schemes, such as OCB, fall into this category. As a simple motivating example illustrating the challenging nature of this task, we will demonstrate that the nonce-based AEAD scheme OCB does *not* satisfy r-BIND security when reframed as an encryptment scheme in the natural way.

3.5.1 Motivating Example: OCB

The offset codebook mode (OCB) was first introduced by Rogaway, Bellare, Black, and Krovetz [137] and later refined by Rogaway [134] and Rogaway and Krovetz [99]. Here we use the formulation given in [134], known as OCB2. Following the publication of the work upon which this chapter is based [59], OCB2 has been demonstrated to be insecure as an AEAD scheme [81]. However, the attack against the r-BIND security of OCB2 that we describe here extends naturally to the other versions of OCB which are still understood to be secure.

Overview. OCB is built from a tweakable block cipher $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \rightarrow \{0,1\}^n$ with tweak space $\mathcal{T} = \{0,1\} \times \{0,1\}^n \times [1 \mathinner{..} 2^{n/2}] \times \{0,1\}$. Figure 3.13 shows $\mathsf{OCB} = (\mathsf{OCB.Kg}, \mathsf{OCB.Enc}, \mathsf{OCB.Dec}, \mathsf{OCB.Ver})$ conceptualised as an encryptment scheme in the

```
$\mathsf{OCB.Enc}(K_{\mathsf{EC}}, M)$
$(K, N) \leftarrow K_{\mathsf{EC}}$
$(M_1, \ldots, M_m) \leftarrow \mathrm{Parse}_n(M)$
For $i = 1$ to $m-1$ do $C_i \leftarrow \tilde{E}_K^{1,N,i,0}(M_i)$
$Pad \leftarrow \tilde{E}^{0,N,m,0}((\mathrm{len}(M_m))_n)$
$C_m \leftarrow M_m \oplus Pad$
$C_{\mathsf{EC}} \leftarrow C_1 \| \cdots \| C_m$
$\Sigma \leftarrow M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^* \oplus Pad$
$B_{\mathsf{EC}} \leftarrow \tilde{E}_K^{0,N,m,1}(\Sigma)$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
$\mathsf{OCB.Dec}(K_{\mathsf{EC}}, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
$(K, N) \leftarrow K_{\mathsf{EC}}$
$(C_1, \ldots, C_m) \leftarrow \mathrm{Parse}_n(C_{\mathsf{EC}})$
For $i = 1$ to $m-1$ do $M_i \leftarrow \tilde{D}_K^{1,N,i,0}(C_i)$
$Pad \leftarrow \tilde{E}^{0,N,m,0}((\mathrm{len}(C_m))_n)$
$M_m \leftarrow C_m \oplus Pad$
$M \leftarrow M_1 \| \cdots \| M_m$
$\Sigma \leftarrow M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^* \oplus Pad$
$B'_{\mathsf{EC}} \leftarrow \tilde{E}_K^{0,N,m,1}(\Sigma)$
If $(B_{\mathsf{EC}} = B'_{\mathsf{EC}})$ then return $M$
Return $\perp$
```

```
$\mathsf{OCB.Ver}(M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$
$(K, N) \leftarrow K_{\mathsf{EC}}$
$(M_1, \ldots, M_m) \leftarrow \mathrm{Parse}_n(M)$
$Pad \leftarrow \tilde{E}_K^{0,N,m,0}((\mathrm{len}(M_m))_n)$
$C_m \leftarrow M_m \oplus Pad$
$\Sigma \leftarrow M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^* \oplus Pad$
$B'_{\mathsf{EC}} \leftarrow \tilde{E}_K^{0,N,m,1}(\Sigma)$
If $B'_{\mathsf{EC}} \neq B_{\mathsf{EC}}$ then return 0
Return 1
```

Figure 3.13: OCB reframed as an encryptment scheme. We write C_m0^* to denote C_m padded with zeroes up to n-bits in length.

natural way; for simplicity we omit associated data from both the construction and the discussion below. Key generation simply picks a key $K \leftarrow\$ \mathcal{K}$ for the underlying TBC and a nonce $N \leftarrow\$ \{0,1\}^n$ and returns $K_{\mathsf{EC}} = (K, N)$. The encryptment and decryptment algorithms are identical to encryption and decryption with OCB, with the exception that the nonce is framed as part of the encryptment key (rather than as an explicit input to the algorithms). Additionally, we interpret the OCB ciphertext (which includes an authentication tag) as a ciphertext and binding tag pair by setting the latter equal to the tag.

During encryptment, ciphertext blocks are computed by partitioning the message into n-bit blocks, encrypting all but the last block using the TBC (with an incrementing counter to ensure that a distinct tweak is used for each block), and encrypting the final message block by using the string Pad (set equal to the encryption of the message length under the TBC) as a random pad. The crucial part of encryptment for our purposes is that the binding tag B_{EC} is computed by XORing together the message blocks and Pad, and encrypting the result using the TBC. Looking ahead, our binding attack will exploit the fact that it is typically straightforward to construct distinct sequences of message blocks which XOR to the same value. Verification simply recomputes and verifies the binding tag; decryptment does the same, having already decrypted each ciphertext block to recover the underlying message.

The attack. OCB has excellent performance properties, with rate-1 encryption and decryption (roughly, one block cipher call per block of message / ciphertext; see Section 3.5.2 for a formal definition) and even faster verification. However, unfortunately OCB is not receiver binding as, intuitively, verification does not provide CR. In more detail, consider the following r-BIND adversary $\mathcal{A}$ against OCB. It computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{OCB.Enc}((K, N), M)$ for arbitrary $(K, N) \in \mathcal{K} \times \{0,1\}^n$ and $M = M_1 \,\|\, M_2 \,\|\, \ldots \,\|\, M_m \in \{0,1\}^*$ such that $m \geq 3$. Attacker $\mathcal{A}$ then sets $M' = \overline{M_1} \,\|\, \overline{M_2} \,\|\, M_3 \,\|\, \ldots \,\|\, M_m$, where $\overline{M_i}$ denotes message block M_i with its least significant bit flipped. Attacker $\mathcal{A}$ then returns $(((K, N), M), ((K, N), M'), B_{\mathsf{EC}})$ to its challenger.

Since the first two message blocks in M, M' have different least significant bits, it is clearly the case that $((K, N), M) \neq ((K, N), M')$. However, since $M_m = M'_m$, and $M_1 \oplus M_2 \oplus \cdots \oplus M_{m-1} = M'_1 \oplus M'_2 \oplus \cdots \oplus M'_{m-1}$, it is straightforward to verify that both messages will result in the same tag B_{EC} when encrypted under (K, N), thereby allowing $\mathcal{A}$ to win game r-BIND with probability one.

3.5.2 Binding Encryptment and Collision Resistant Hashing

In the remainder of this section, we will formally define high-rate encryptment and show how prior results on the impossibility of high-rate CR compression functions can be used to rule out a broad class of high-rate encryptment schemes as well. To this end, we begin by defining the rate of a compression function and an encryptment scheme. We then demonstrate a (mostly syntactic) transform from an encryptment scheme EC to a compression function of the same rate, and show that if EC is r-BIND-secure then the resulting compression function must be CR. We then recall a result from [140] which says that no high-rate compression function with the given parameters can be CR, thereby ruling out the r-BIND-security of EC also.

A connection between hashing and encryptment. Towards showing negative results, we must first define more carefully what we mean by the *rate* of encryptment schemes. We are inspired by (and will later exploit connections to) the definitions of rate from the block cipher-based hash function literature [37, 139, 140].

Consider a compression function $\mathcal{H} : \{0,1\}^{mn} \to \{0,1\}^{rn}$ for $m > r \geq 1$ and $n \geq 1$,

```
$\mathcal{H}(V)$
For $i = 1$ to $\psi$ do
    $X_i \leftarrow f_i(V, Y_1, \ldots, Y_{i-1})$
    $Y_i \leftarrow E_{K_i}(X_i)$
$W \leftarrow g(V, Y_1, \ldots, Y_\psi)$
Return $W$
```

Figure 3.14: A block cipher-based compression function.

which uses $\psi \geq 1$ calls of a block cipher $\mathrm{E} : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^n$ $(m, r, n, \psi, \kappa \in \mathbb{N})$. Following [140] we may write $\mathcal{H}$ as shown in Figure 3.14, where we let $K_1, \ldots, K_\psi \in \{0,1\}^\kappa$ be any *fixed* strings. Further, we let $f_i : \{0,1\}^{(m+(i-1))n} \rightarrow \{0,1\}^n$ for $i \in [1, \ldots, \psi]$ and $g : \{0,1\}^{(m+\psi)n} \rightarrow \{0,1\}^{rn}$ be functions.

The rate of $\mathcal{H}$ is defined to be m/ψ, and so a rate-$\frac{1}{\beta}$ function $\mathcal{H}$ makes β block cipher calls per n-bits of input. For example, a rate-1 $\mathcal{H}$ would achieve a single block cipher call per n-bit block of input. A consequence of the more general results of [140] (see below) is that they rule out rate-1 functions achieving CR security (in the IPM) if an attacker can make $2^{n/4}$ or more ideal permutation queries. We would like to exploit their negative results to similarly rule out rate-1 encryptment schemes.

Rate of encryptment. Consider an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$. Because EEnc is deterministic, we can define a function F which on input (K_{EC}, H, M) computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$ and returns B_{EC}. We now focus attention on encryptment schemes of a certain (natural) form; namely those for which verification on input $\mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$ checks that $F(K_{\mathsf{EC}}, H, M) = B_{\mathsf{EC}}$. (One can generalise this definition by allowing EEnc and EVer to use different functions F, F' to compute the binding tag; the lower bounds given in this section readily extend to this case also.)

With this in place, we can define the rate of verification for encryptment analogously to defining the rate of a hash function $\mathcal{H}$, by saying that EVer has rate-$\frac{1}{\beta}$ if the associated function F makes β block cipher calls per n-bits of header and message data (or equivalently, can process (H, M) of combined length mn-bits using βm block cipher calls). Since the encryptment algorithm EEnc also computes the binding tag via F (in addition to producing the ciphertext), this represents a lower bound on the rate of EEnc also.

Encryptment to hash transform. We can now give a generic, essentially syntactic, transform from an encryptment scheme to a compression function. Fix $m, n \in \mathbb{N}$, and let $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ be an encryptment scheme such that $\{0,1\}^{mn} \subseteq \mathcal{M}_{\mathsf{EC}}$ and $\mathcal{T}_{\mathsf{EC}} = \{0,1\}^{rn}$. Let F be the associated binding tag computation function as per above. We define the compression function $\mathcal{H}_{\mathsf{EC}} : \{0,1\}^{mn} \to \{0,1\}^{rn}$ to be $\mathcal{H}_{\mathsf{EC}}(X) = F(K_{\mathsf{EC}}, \varepsilon, X)$ for $K_{\mathsf{EC}} \in \mathcal{K}_{\mathsf{EC}}$ an arbitrary fixed bit string. Here we take $H = \varepsilon$ so that the number of block cipher calls required to compute F is solely determined by the length of the input X. With this in place, the following theorem, which says that $\mathcal{H}_{\mathsf{EC}}$ is CR if EC is r-BIND-secure, is straightforward to prove.

Theorem 3.1. *Let* EC *be an encryptment scheme, and let* $\mathcal{H}_{\mathsf{EC}} : \{0,1\}^{mn} \to \{0,1\}^{rn}$ *be defined as above. Then for any collision-resistance adversary* $\mathcal{A}$*, we give an explicit* r-BIND *adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}_{\mathsf{EC}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{B}) \ .$$

The adversary $\mathcal{B}$ *runs in the same amount of time as* $\mathcal{A}$*.*

Proof. We define $\mathcal{B}$ to be the attacker who proceeds as follows. $\mathcal{B}$ runs $\mathcal{A}$, and when $\mathcal{A}$ outputs a pair $X, X' \in \{0,1\}^{mn}$, $\mathcal{B}$ checks if $\mathcal{H}_{\mathsf{EC}}(X) = \mathcal{H}_{\mathsf{EC}}(X')$. Note that by the definition of $\mathcal{H}_{\mathsf{EC}}$, such an event implies that computing $(\cdot, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, \varepsilon, X)$ and $(\cdot, B'_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, \varepsilon, X')$ yields colliding binding tags $B_{\mathsf{EC}} = B'_{\mathsf{EC}}$. If such a collision is found, $\mathcal{B}$ outputs $((K_{\mathsf{EC}}, \varepsilon, X), (K_{\mathsf{EC}}, \varepsilon, X'), B_{\mathsf{EC}})$ to his own challenger; otherwise $\mathcal{B}$ outputs $((K_{\mathsf{EC}}, \varepsilon, X), (K_{\mathsf{EC}}, \varepsilon, X''), B_{\mathsf{EC}})$ for $X'' \leftarrow\!\$\ \{0,1\}^{mn}$. By the correctness of EC, it must be the case that if $(\cdot, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$, then $1 \leftarrow \mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$. As such, it is straightforward to verify that if $\mathcal{A}$ finds a winning pair in game CR then $\mathcal{B}$ will win game r-BIND with probability one. Moreover $\mathcal{B}$ runs in the same time as $\mathcal{A}$, thereby implying the claim.

□

Theorem 3.1 allows us to lift known negative results about efficient CR-hashing to the encryptment setting. For example, Rogaway and Steinberger [140] analyse the security of fixed-key block cipher based compression functions $\mathcal{H} : \{0,1\}^{mn} \to \{0,1\}^{rn}$ (constructed as in Figure 3.14) in the IPM. They demonstrate, via an attack, an upper bound on the number of ideal permutation queries an attacker must make to find a collision. We recall [140, Th. 1] below.

Theorem 3.2. *Fix* $m > r \geq 1$ *and* $n > 0$ *(*$m, r, n \in \mathbb{N}$*). Let* $N = 2^n$*. Let* $\mathcal{H} : \{0,1\}^{mn} \to$

$\{0,1\}^{rn}$ be a ψ-call compression function constructed as in Figure 3.14, in which we model E_{K_i} as an ideal permutation $\pi_i \leftarrow\!\$\ \mathrm{Perm}(\{0,1\}^n)$ for each $i \in [1, \psi]$. Then there exists an explicit adversary $\mathcal{A}$ making $q = \psi \cdot (N^{1-(\frac{m-r}{\psi})} + 1)$ ideal permutation queries and achieving

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H}}(\mathcal{A}) = 1\ .$$

The proof of the result utilises the pigeonhole principle, which says that if you place m balls in n bins for $m > n$ then there must be at least one bin with two balls in it. To see how the bound is derived in [140], let $p = \lfloor q/\psi \rfloor$, let $N = 2^n$, and consider the construction of $\mathcal{H}$ given in Figure 3.14. The CR attacker $\mathcal{A}$ proceeds as follows. $\mathcal{A}$ selects the p points $Y_1, \ldots, Y_p \in \{0,1\}^n$ which have the largest preimage sets $f_1^{-1}(Y) = \{X \in \{0,1\}^{mn} : f_1(X) = Y\}$ under the first function f_1. Since $f_1 : \{0,1\}^{mn} \to \{0,1\}^n$, one can think of f_1 placing the N^m domain points into N bins, labelled with the points in $\{0,1\}^n$. It follows that the p most occupied 'bins' (corresponding to labels $Y_1, \ldots, Y_p$) contain at least $p \cdot N^{m-1}$ points. The attacker $\mathcal{A}$ queries $Y_1, \ldots, Y_p$ to π_1 (recall we are working in the IPM). Letting $V_1 = \cup_{i=1}^{p} f_1^{-1}(Y_i)$, the above argument implies that $|V_1| > p \cdot N^{m-1}$ and that $\mathcal{A}$ now knows the value that each of these points maps to during the computation of $\mathcal{H}$ following the application of π_1. $\mathcal{A}$ then chooses the p points $Y_1', \ldots, Y_p' \in \{0,1\}^n$ which have the most pre-images under f_2 in V_1. This set of pre-images must contain at least $p^2 \cdot N^{m-2}$ points. $\mathcal{A}$ now queries π_2 on each of these points, and learns how to compute the value of these $p^2 \cdot N^{m-2}$ points under $\mathcal{H}$ at the point of applying π_2. Repeating this process for each of the ψ permutations, it follows that $\mathcal{A}$ will finish having made $\psi \cdot p < q$ queries and will be able to compute the value of all points in a set of size at least $p^\psi \cdot N^{m-\psi} > N$ under $\mathcal{H}$. The pigeonhole principle then implies that the desired collision will be amongst these points.

The above result is striking for certain parameter settings. For example, consider a compression function $\mathcal{H} : \{0,1\}^{2n} \to \{0,1\}^n$ making a single permutation call ($\psi = 1$). Then an attacker can find a collision in $\mathcal{H}$ with just two ideal permutation queries. Invoking the relation between CR compression functions and encryptment in Theorem 3.1 immediately yields the following corollary.

Corollary 3.1. *Fix $m > r \geq 1$ and $n > 0$ ($m, r, n \in \mathbb{N}$). Let $N = 2^n$. Let EC be an encryptment scheme of the structure given in Theorem 3.1. Suppose that: **(1)** $\{0,1\}^{mn} \subseteq \mathcal{M}$; **(2)** $\mathcal{T}_{\mathsf{EC}} = \{0,1\}^{rn}$; and **(3)** the function F used by EEnc and EVer to compute the* *binding tag on messages of length mn is of the form shown in Figure 3.14. Then there exists an explicit adversary $\mathcal{A}$ making $q = \psi \cdot (N^{1-(\frac{m-r}{\psi})} + 1)$ ideal permutation queries and achieving*

$$\mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC}}(\mathcal{A}) = 1 .$$

This immediately rules out r-BIND security for rate-1 encryptment schemes that achieve the efficiency of OCB, i.e., having $\psi = m$, m arbitrarily large, and $r = 1$. In the minimal case that $m = 2$ (two block messages), then $\mathcal{A}$ only requires $q = 2$ queries to succeed. Stronger results ruling out rate-$\frac{1}{2}$ verification and encryptment can be similarly lifted from [140, Th. 2] under some technical conditions about the verification function and the adversary.

Extensions. One can modify our definitions so the key K_i for the i^{th} cipher call can be picked from the set $\{K_1, \ldots, K_\psi\}$ as a function of the current round and messages instead of being used in a fixed order; Rogaway and Steinberger refer to this as the no-fixed order model (though it first appeared in [37]). A negative result based on [37, Th. 5] would rule out encryptment using any rate-1 no-fixed order verification algorithm.

The results above were cast in terms of r-BIND security, but extend to sr-BIND security because the latter implies the former. We conjecture that these lower bounds can be extended to block cipher-based *robust* encryption schemes (in the sense of [67], see Section 3.7.6); we leave this as an open problem for future work.

Ultimately these negative results indicate that for an r-BIND-secure encryptment scheme, the best we can hope for is either a rate-$\frac{1}{3}$ construction based on a block cipher with a small set of fixed keys, or to allow rekeying with each block of message. We therefore turn to building as efficient-as-possible constructions.

Relation to ccAEAD. In Section 3.7.5, we will describe how the existence of an r-BIND-secure ccAEAD scheme for which encryption achieves a given rate implies the existence of an r-BIND-secure encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ such that EEnc achieves the same rate, and so the results of this section exclude the existence of rate-1 or rate-$\frac{1}{2}$ ccAEAD encryption also.

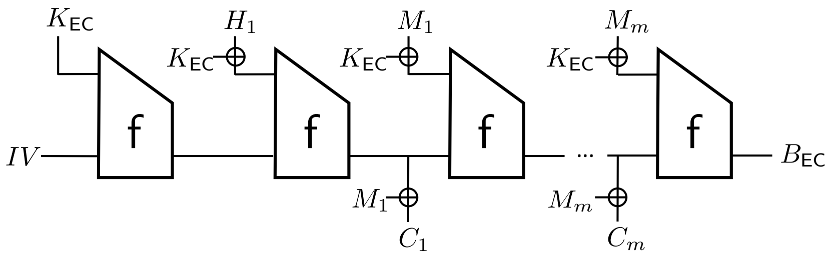


Figure 3.15: Encryptment in the HFC scheme for a 1-block header and m-block message. For simplicity the diagram does not show the details of padding.

3.6 Building Encryptment from Hashing

In this section, we turn our attention to building secure and efficient encryptment schemes. While the results of the previous section imply that we cannot achieve binding encryptment matching the rate of OCB and GCM, we will build an encryptment scheme which *does* match their efficiency in the metric of being single-pass. As we shall see in Section 3.7, our scheme — called *hash function chaining* (HFC) — can be lifted to single-pass, many-time secure ccAEAD via simple and efficient transforms. Our construction requires stronger than usual security of the underlying compression function f (regarded as a PRF keyed on its second input) — namely a form of related-key attack (RKA) security. Later in the section we describe variants of the HFC scheme that avoid the RKA requirement, although these are not as fast or elegant as the original scheme.

Construction overview. As one might expect given the close relationship between binding encryptment and CR hashing, we take the latter as our starting point. A slightly simplified version of our construction — called *hash function chaining* (HFC) — is shown in Figure 3.15 (padding details are omitted), where f is a compression function. In summary, the scheme hashes the key, associated data and message data (the latter two of which are repeatedly XORed with the key). Intermediate chaining variables from the hash computation are used as pads to encrypt the message data, while the final chaining variable constitutes the binding tag.

Intuitively, (strong) receiver binding derives from the collision resistance of the underlying hash function. We XOR the key into all the associated data and message blocks to ensure that every application of the compression function is keyed. This is critical; just prepending (or both prepending and appending) the key to the data leads to a scheme whose confidentiality is easily broken. Likewise, one cannot dispense with the additional initial block that simply processes the key, otherwise the encoding of the key, associated data, and message would not be injective and binding attacks result.

3.6.1 Notation, Padding Schemes, and Iterated Functions

Before defining the full scheme, we first give some additional definitions that will simplify the presentation.

Padding schemes. Our scheme utilises a padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \overline{\mathrm{PadM}}, \mathrm{PadSuf}, \mathrm{Pad})$, which we now formalise. The padding scheme is parameterised by positive integers $d, n \in \mathbb{N}$; however, we omit these in the notation for simplicity. We assume $d \geq n \geq 128$.

The header and message padding functions PadH, PadM take as input a pair (H, M), and return tuples $(H_1, \ldots, H_h)$ and $(M'_1, \ldots, M'_{m-1})$ respectively. We require that $H_i, M'_j \in \{0,1\}^d$ for $i = 1, \ldots, h$, and $j = 1, \ldots, m-1$. We abuse notation to let $\mathrm{PadH}(H, M)$ (resp. $\mathrm{PadM}(H, M)$) denote the concatenation of the blocks returned by these algorithms. The online padding algorithm $\overline{\mathrm{PadM}}$ takes as input a tuple (H, M_i, i) where $M_i \in \{0,1\}^n$, and outputs $M'_i \in \{0,1\}^d$. We require that for any pair (H, M), it holds that $\overline{\mathrm{PadM}}(H, M_i, i) = M'_i$ for $i = 1, \ldots, m-1$, where $(M'_1, \ldots, M'_{m-1}) \leftarrow \mathrm{PadM}(H, M)$. This allows PadM to be computed in an online manner, with the message data being delivered in n-bit blocks. Finally, we define PadSuf to be the algorithm which takes as input $\ell_H, \ell_M \in \mathbb{N}^2$ and a string $X \in \{0,1\}^{\leq d}$, and outputs a string Y such that $\mathrm{Trunc}_{|X|}(Y) = X$ and Y is a multiple of d-bits. The full padding function is then defined to be $\mathrm{Pad}(H, M) = \mathrm{PadH}(H, M) \,\|\, \mathrm{PadM}(H, M) \,\|\, \mathrm{PadSuf}(|H|, |M|, M_m)$ where M_m denotes the final message block in the output of $\mathrm{Parse}_n(M)$. Note that d divides $|\mathrm{Pad}(H, M)|$.

Padding scheme properties. To prove the security of HFC as an encryptment scheme we will require the padding scheme Pad to be *injective*; that is to say that if $\mathrm{Pad}(H, M) = \mathrm{Pad}(H', M')$ then it must be the case that $(H, M) = (H', M')$. An example of such a padding scheme is shown in Figure 3.16, and we shall assume that HFC is instantiated with this scheme unless stated otherwise. Our padding scheme is a variant of MD-strengthening. The header padding algorithm PadH simply pads the header H to a multiple of d-bits with

```
PadH($H, M$)
    $(H_1, \ldots, H_h) \leftarrow \text{Parse}_d(H \,\|\, 0^{d-|H| \bmod d})$
    Return $(H_1, \ldots, H_h)$

PadSuf($\ell_H, \ell_M, M_m$)
    $p \leftarrow \min\{i \in \mathbb{N} \cup \{0\} \;:\; d \,|\, ((\ell_M \bmod n) + i + 128)\}$
    Return $M_m \,\|\, 0^p \,\|\, (\ell_H)_{64} \,\|\, (\ell_M)_{64}$
```

```
PadM($H, M$)
    $(M_1, \ldots, M_m) \leftarrow \text{Parse}_n(M)$
    For $i = 1, \ldots, m-1$
        $M'_i \leftarrow M_i \,\|\, 0^{d-n}$
    Return $(M'_1, \ldots, M'_{m-1})$
```

```
$\overline{\text{PadM}}(H, M_i, i)$
    $M'_i \leftarrow M_i \,\|\, 0^{d-n}$
    Return $M'_i$
```

Figure 3.16: Padding scheme $\mathsf{PadS} = (\text{PadH}, \text{PadM}, \overline{\text{PadM}}, \text{PadSuf}, \text{Pad})$. We require that $\ell_H, \ell_M \in \mathbb{N}$.

zeroes, and parses the result as a sequence of d-bit blocks. The message padding algorithm parses the message M into a series of n-bit blocks, pads all but the last of these to d-bits with zeroes (as we will see, this is required for decryptability), and returns the result. The suffix padding algorithm pads the final message block to a multiple of d-bits having, in the MD-strengthening step, appended the length of the header and message both encoded as 64-bit strings. Looking ahead, we will not rely on the strengthening for its traditional purpose of forming a suffix-free padding scheme in our HFC construction. Instead, we use strengthening only for injectivity and will assume more of the compression function f from which HFC is built.

Extending to otCTXT-security. If we would additionally like to prove the otCTXT-security of HFC, we require that the padding scheme satisfies the stronger property of *prefix-freeness*, meaning that if $(H, M) \neq (H', M')$ then $\text{Pad}(H, M)$ is not a prefix of $\text{Pad}(H', M')$. This can be achieved by, for example, adding frame-bits to distinguish the final block from the rest of the header and message data.

Iterated functions. Finally, we introduce notation for iterated compression functions. Let $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ be a function for some $d \geq n \geq 128$, let $D^+ = \cup_{i \geq 1} \{0,1\}^{i \cdot d}$, and let $V_0 \in \{0,1\}^n$. Then $\mathsf{f}^+ : \{0,1\}^n \times D^+ \to \{0,1\}^n$ denotes the *iteration* of f, where $\mathsf{f}^+(V_0, X_1 \,\|\, \cdots \,\|\, X_m) = V_m$ is computed via $V_i = \mathsf{f}(V_{i-1}, X_i)$ for $1 \leq i \leq m$.

With this in place, we are ready to present our encryptment scheme.

3.6.2 The HFC Encryptment Scheme

The HFC encryptment scheme $\mathsf{HFC} = (\mathsf{HFC.Kg}, \mathsf{HFC.Enc}, \mathsf{HFC.Dec}, \mathsf{HFC.Ver})$ is based on a compression function $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$. The encryptment and decryptment algorithms are presented in Figure 3.17.

```
$\mathsf{HFC.Enc}(K_{\mathsf{EC}}, H, M)$
$(H_1, \ldots, H_h) \leftarrow \mathrm{PadH}(H, M)$
$(M_1, \ldots, M_m) \leftarrow \mathrm{Parse}_n(M)$
$(M'_1, \ldots, M'_{m-1}) \leftarrow \mathrm{PadM}(H, M)$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
$V_h \leftarrow \mathsf{f}^+\big(V_0, (K_{\mathsf{EC}} \oplus H_1) \,\|\, \cdots \,\|\, (K_{\mathsf{EC}} \oplus H_h)\big)$
$C_{\mathsf{EC}} \leftarrow \varepsilon$
For $i = 1, \ldots, m-1$ do
   $C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+i-1} \oplus M_i)$
   $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i))$
$C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+m-1} \oplus M_m)$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(\mathrm{PadSuf}(|H|, |M|, M_m))$
$B_{\mathsf{EC}} \leftarrow \mathsf{f}^+(V_{h+m-1}, (K_{\mathsf{EC}} \oplus M'_m) \,\|\, (K_{\mathsf{EC}} \oplus M'_{m+1}))$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
$\mathsf{HFC.Dec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
$(H_1, \ldots, H_h) \leftarrow \mathrm{PadH}(H, C_{\mathsf{EC}})$
$(C_1, \ldots, C_m) \leftarrow \mathrm{Parse}_n(C_{\mathsf{EC}})$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
$V_h \leftarrow \mathsf{f}^+\big(V_0, (K_{\mathsf{EC}} \oplus H_1) \,\|\, \cdots \,\|\, (K_{\mathsf{EC}} \oplus H_h)\big)$
For $i = 1, \ldots, m-1$ do
   $M_i \leftarrow V_{h+i-1} \oplus C_i$ ; $M'_i \leftarrow \overline{\mathrm{PadM}}(H, M_i, i)$
   $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i))$
$M_m \leftarrow V_{h+m-1} \oplus C_m$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(\mathrm{PadSuf}(|H|, |C_{\mathsf{EC}}|, M_m))$
$B'_{\mathsf{EC}} \leftarrow \mathsf{f}^+(V_{h+m-1}, (K_{\mathsf{EC}} \oplus M'_m) \,\|\, (K_{\mathsf{EC}} \oplus M'_{m+1}))$
If $B'_{\mathsf{EC}} \neq B_{\mathsf{EC}}$ then
   Return $\perp$
Return $M_1 \,\|\, \cdots \,\|\, M_m$
```

Figure 3.17: The HFC encryptment scheme built from a compression function $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ and padding scheme $\mathsf{PadS} = (\mathrm{PadH}, \mathrm{PadM}, \overline{\mathrm{PadM}}, \mathrm{PadSuf}, \mathrm{Pad})$. Here $K_{\mathsf{EC}} \in \{0,1\}^d$ and $IV \in \{0,1\}^n$ is a fixed public constant.

The key generation algorithm $\mathsf{HFC.Kg}$ chooses $K_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^d$. Encryptment first pads the header and message using the padding functions PadH and PadM respectively. We let $IV \in \{0,1\}^n$ be a fixed constant value (also called an *initialisation vector*). The scheme computes an initial chaining variable as $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$. It then hashes

$$\mathrm{Pad}(H, M) = \mathrm{PadH}(H, M) \,\|\, \mathrm{PadM}(H, M) \,\|\, \mathrm{PadSuf}(|H|, |M|, M_m)$$

(where M_m denotes the final block returned by $\mathrm{Parse}_n(M)$) with f^+ (i.e., the iteration of the compression function f), where the secret encryptment key K_{EC} is XORed into each d-bit block prior to hashing. The final chaining variable produced by this process forms the binding tag B_{EC}. Notice that while the compression function takes d-bit inputs, the way in which the message data is padded means we only process n-bits of message in each compression function call; looking ahead, this is to ensure decryptability. We will see that the collision resistance of the iterated hash function when instantiated with a collision resistant compression function for which it is hard to invert the IV implies the sr-BIND security of the construction.

Rather than running a separate encryption algorithm alongside this process to encrypt the message, we instead generate ciphertext blocks by XORing the message blocks M_i with intermediate chaining variables, yielding $C_i = V_{h+i-1} \oplus M_i$ for $1 \leq i \leq m$ where h denotes the number of header blocks. Recall that in our notation $X \oplus Y$ silently truncates the longer string to the length of the shorter string, and so only the n-bits of message data in each d-bit padded message block is XORed with the n-bit chaining variable. Similarly, if message M is such that $|M| \bmod n = r$, then the final ciphertext block produced by this process is truncated to the leftmost r-bits. The properties of the compression function ensure that the chaining variables are pseudorandom, thus yielding the required otROR security. By 'reusing' chaining variables as random pads we can achieve encryptment with no additional overhead over just computing the binding tag, yielding better efficiency (see further discussion below).

Decryptment via $\mathsf{HFC.Dec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ begins by padding H into d-bit blocks via $\mathrm{PadH}(H, M)$ and parsing C_{EC} into n-bit blocks. The algorithm computes the initial chaining variable as $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$, then hashes the padded header as in encryptment. The scheme then recovers the first message block M_1 by XORing the resulting chaining variable into the first ciphertext block C_1. This is padded via the online padding function $\overline{\mathrm{PadM}}(H, M_1, 1)$ and then used to compute the next chaining variable via application of f, and so on. Notice how at most n-bits of message data is recovered in each such step; this is why we must process only n-bits of message data in each compression function call, else the decryptor would be unable to compute the next chaining variable. Finally $\mathsf{HFC.Dec}$ recomputes and verifies the binding tag, returning the message only if this check passes.

On input $(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})$, the verification algorithm (not shown) pads the header and message to $\mathrm{PadH}(H, M) \,\|\, \mathrm{PadM}(H, M) \,\|\, \mathrm{PadSuf}(|H|, |M|, M_m)$, XORs K_{EC} into every block, and hashes the resulting string with f^+ with initial chaining variable $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$, checking that the output matches the binding tag B_{EC}.

Efficiency. The efficiency of the scheme (in terms of throughput) depends on the parameters d, n, where recall that $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^n$. As discussed previously at most n-bits of message data can be processed in each compression function call. As such, HFC achieves optimal throughput when $d = n$. In this case, no padding is applied to the message blocks (beyond padding the last block to d-bits), and so computing the full encryptment incurs *no overhead* over simply hashing the header and message with the iteration of f.

```
AltPadH$(H, M)$
$(H_1, \ldots, H_h) \leftarrow \text{Parse}_{d-n}(H)$
$(M_1, \ldots, M_m) \leftarrow \text{Parse}_n(M)$
$\beta \leftarrow \min(m, h)$
If $\beta = h$ then
  Return $\varepsilon$
Else $(H_{\beta+1}, \ldots, H_h) \leftarrow \text{Parse}_d(H_{\beta+1} \,\|\, \ldots \,\|\, H_h)$
  $H_h \leftarrow H_h \,\|\, 0^{d-|H| \bmod d}$
Return $(H_{\beta+1}, \ldots, H_h)$
```

```
AltPadM$(H, M)$
$(H_1, \ldots, H_h) \leftarrow \text{Parse}_{d-n}(H)$
$(M_1, \ldots, M_m) \leftarrow \text{Parse}_n(M)$
$\beta \leftarrow \min(m, h)$
For $i = 1, \ldots, \beta$ do
  $X_i \leftarrow M_i \,\|\, H_i$
If $\beta = h$ then for $i = \beta + 1, \ldots, m - 1$
  $M'_i \leftarrow M_i \,\|\, 0^{d-n}$
Return $(X_1, \ldots, X_\beta, M'_{\beta+1}, \ldots, M'_{m-1})$
```

```
$\overline{\text{AltPadM}}(H, M_i, i)$
$(H_1, \ldots, H_h) \leftarrow \text{Parse}_{d-n}(H)$
If $i \leq h$ then
  $M'_i \leftarrow M_i \,\|\, H_i$
Else $M'_i \leftarrow M_i \,\|\, 0^{d-n}$
Return $M'_i$
```

Figure 3.18: Alternative padding scheme AltPadS.

If $d > n$ then some throughput is lost due to padding. We present an alternative padding scheme for this case which recovers some throughput by padding message blocks with header data. In more detail, consider the padding scheme $\mathsf{AltPadS} = (\text{AltPadH}, \text{AltPadM}, \overline{\text{AltPadM}}, \text{AltPadSuf}, \text{AltPad})$ shown in Figure 3.18, where AltPadSuf is defined identically to PadSuf in Figure 3.16. The scheme parses the message data into n-bit blocks and the header data into $(d-n)$-bits blocks. It then constructs d-bit blocks by padding the message blocks with header blocks. Any header / message data remaining after this process is padded unambiguously similarly to the previous scheme (Figure 3.16). It is straightforward to verify that the resulting padding function $\text{AltPad}(H, M) = \text{AltPadH}(H, M) \,\|\, \text{AltPadM}(H, M) \,\|\, \text{AltPadSuf}(|H|, |M|, M_m)$ is injective.

3.6.3 Analysing the HFC Encryptment Scheme

We now analyse the security of HFC relative to the security goals detailed in Section 3.4.

Strong receiver binding. We begin by proving that HFC satisfies strong receiver binding provided the underlying padding function is injective. Observe that the binding tag computation performed by the encryptment algorithm HFC.Enc is equivalent to hashing an encoding of the input tuple (K_{EC}, H, M) with f^+. The encoding — which consists of padding the header and message, XORing the key K_{EC} into each block, and then prepending K_{EC} — is injective provided the padding function Pad is injective. As such, any tuple breaking the sr-BIND security of HFC corresponds to a collision against f^+.

A well-known result [52] gives that f^+ is CR provided the underlying compression function is both CR and such that it is hard to find an input which hashes to the initialisation vector IV. Standard compression functions satisfy both properties. The conditions on d, n in the theorem can be relaxed; the conditions arise from our choice of padding.

Theorem 3.3. *Let* HFC *be as shown in Figure 3.17, built from a compression function* $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *for* $d \geq n \geq 128$ *and instantiated with an injective padding scheme* PadS*. Then for any adversary* $\mathcal{A}$ *in game* sr-BIND *against* HFC*, we can construct an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{f}^+}(\mathcal{B}) ,$$

where adversary $\mathcal{B}$ *runs in the same time as* $\mathcal{A}$*.*

Proof. We first introduce some notation. Recall that $\text{Pad}(H, M) = \text{PadH}(H, M) \,\|\, \text{PadM}(H, M) \,\|\, \text{PadSuf}(|H|, |M|, M_m)$. For a header / message pair (H, M) such that $(X_1, \ldots, X_\ell) \leftarrow \text{Parse}_d(\text{Pad}(H, M))$, we define

$$\mathcal{P}(K_{\mathsf{EC}}, H, M) = K_{\mathsf{EC}} || (X_1 \oplus K_{\mathsf{EC}}) || \ldots || (X_\ell \oplus K_{\mathsf{EC}}) .$$

Notice that HFC computes the binding tag of a triple (K_{EC}, H, M) as $B_{\mathsf{EC}} = \mathsf{f}^+(IV, \mathcal{P}(K_{\mathsf{EC}}, H, M))$.

Consider an attacker $\mathcal{B}$ in the CR game against f^+. $\mathcal{B}$ runs the sr-BIND attacker $\mathcal{A}$ as a subroutine. Now suppose that $\mathcal{A}$ wins game sr-BIND against HFC with tuple $((K_{\mathsf{EC}}, H, M), (K'_{\mathsf{EC}}, H', M'), B_{\mathsf{EC}})$. Since verification $\mathsf{HFC.Ver}$ works by recomputing the binding tag and checking for equality, a win for $\mathcal{A}$ corresponds to finding $(K_{\mathsf{EC}}, H, M) \neq (K'_{\mathsf{EC}}, H', M')$ such that $\mathsf{f}^+(IV, \mathcal{P}(K_{\mathsf{EC}}, H, M)) = \mathsf{f}^+(IV, \mathcal{P}(K'_{\mathsf{EC}}, H', M')) = B_{\mathsf{EC}}$. We claim that such a win for $\mathcal{A}$ allows $\mathcal{B}$ to construct a winning collision in the CR game against f^+. To see this notice that $\mathcal{P}$ is injective, meaning that for all (K_{EC}, H, M), $(K'_{\mathsf{EC}}, H', M')$, $\mathcal{P}(K_{\mathsf{EC}}, H, M) = \mathcal{P}(K'_{\mathsf{EC}}, H', M')$ implies that $(K_{\mathsf{EC}}, H, M) = (K'_{\mathsf{EC}}, H', M')$. To justify this, notice that since K_{EC} is prepended to the output of $\mathcal{P}$ the padded strings are clearly different if $K_{\mathsf{EC}} \neq K'_{\mathsf{EC}}$. On the other hand if $K_{\mathsf{EC}} = K'_{\mathsf{EC}}$, then $\mathcal{P}(K_{\mathsf{EC}}, H, M) = \mathcal{P}(K'_{\mathsf{EC}}, H', M')$ implies that $\text{Pad}(H, M) = \text{Pad}(H', M')$. This in turns implies that $(H, M) = (H', M')$ by the injectivity of $\text{Pad}(H, M)$.

As such, any winning tuple $((K_{\mathsf{EC}}, H, M), (K'_{\mathsf{EC}}, H', M'), B_{\mathsf{EC}})$ for $\mathcal{A}$ breaking the sr-BIND security of HFC corresponds to a winning pair $(\mathcal{P}(K_{\mathsf{EC}}, H, M), \mathcal{P}(K'_{\mathsf{EC}}, H', M'))$ for $\mathcal{B}$ in the CR game against f^+. It follows that

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{f}+}(\mathcal{B}) \ ,$$

where $\mathcal{B}$ runs in the same time as $\mathcal{A}$, as required. □

Sender binding and correctness. Since the binding tag is recomputed and verified during decryption with $\mathsf{HFC.Dec}$, it is clear that an attacker can never construct a ciphertext which decrypts correctly but for which verification fails. As such, for all attackers $\mathcal{C}$ it holds that

$$\mathbf{Adv}^{\text{s-bind}}_{\mathsf{HFC}}(\mathcal{C}) = 0 \ .$$

Similarly, it is straightforward to verify that the scheme is strongly correct. Moreover, since sr-BIND security implies r-BIND security, we know that for any attacker $\mathcal{A}$ in game r-BIND against HFC there exists an attacker $\mathcal{A}'$ running in the same time as $\mathcal{A}$ such that $\mathbf{Adv}^{\text{r-bind}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{HFC}}(\mathcal{A}')$. With these three conditions in place, applying Lemma 3.1 and Theorem 3.3 implies that for any attacker $\mathcal{A}$ in the SCU game against HFC, there exists an attacker $\mathcal{B}$ such that

$$\mathbf{Adv}^{\text{scu}}_{\mathsf{HFC}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{f}+}(\mathcal{B}) \ .$$

One-time confidentiality. All that remains to prove that HFC is a secure encryptment scheme is to bound its otROR security. We analyse this in the next theorem, by reducing the otROR security of HFC to the RKA-PRF security [17] of f for a specific class of related-key deriving functions.

RKA-PRF security. Our variant of RKA-PRF security allows an attacker to query a keyed function F under linearly related keys; we require that F behaves like an independent random function for each related key. Let $F : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ be a function, and consider the games RKA-PRF0 and RKA-PRF1 defined as follows. In both games the attacker is given access to an oracle to which he may submit queries of the form $(X, Y) \in \{0,1\}^n \times \{0,1\}^d$. In game RKA-PRF0, the oracle returns $F(X, Y \oplus K_{\text{prf}})$ where $K_{\text{prf}} \in \{0,1\}^d$ is a PRF key. In game RKA-PRF1, the oracle returns a random value for each fresh query and answers consistently for repeated queries. The *linear-only RKA-PRF* advantage of an adversary $\mathcal{A}$ who makes at most q queries to their real-or-random oracle is defined

$$\mathbf{Adv}_{F}^{\oplus\text{-prf}}(\mathcal{A}, q) = \left|\Pr\left[\,\text{RKA-PRF0}_{F}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\text{RKA-PRF1}_{F}^{\mathcal{A}} \Rightarrow 1\,\right]\right| \ .$$

We bound the otROR security of HFC by a game hopping argument, first arguing that we can replace the compression function with lazily sampled random functions (one for each related key) by a reduction to the RKA-PRF security of f, and then replacing the random function outputs with random bit strings using a birthday bound argument. We formalise this in the following theorem.

Theorem 3.4. *Let* HFC *be as shown in Figure 3.17, built from a compression function* $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *for* $d \geq n \geq 128$ *and instantiated with an injective padding scheme* PadS*. Let* $\mathcal{A}$ *be an otROR adversary whose encryption query totals at most* ℓ *blocks of* d *bits after padding. Then there exists an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}_{\mathsf{HFC}}^{\text{ot-ror}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}, \ell+1) + \frac{(\ell+1)^2}{2^{n+1}} \ .$$

$\mathcal{B}$ runs in the same time as $\mathcal{A}$ plus an $\mathcal{O}(\ell)$ overhead.

Proof. We argue by a series of game hops, shown in Figure 3.19. We begin by defining G_0, which is equivalent to game otROR0 against HFC; it follows that

$$\Pr\left[\,\text{otROR0}_{\mathsf{HFC}}^{\mathcal{A}} \Rightarrow 1\,\right] = \Pr\left[\,G_0 \Rightarrow 1\,\right] .$$

Next we define game G_1, which is identical to G_0 except for each $Y \in \{0,1\}^d$ we replace $\mathsf{f}(\cdot, K_{\mathsf{EC}} \oplus Y)$ with a lazily sampled random function. We claim that there exists an attacker $\mathcal{B}$ in game RKA-PRF against f with the claimed run time, such that

$$\left|\Pr\left[\,G_0 \Rightarrow 1\,\right] - \Pr\left[\,G_1 \Rightarrow 1\,\right]\right| \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}, \ell+1) \ .$$

To see this, let $\mathcal{B}$ proceed as follow. $\mathcal{B}$ simulates $\mathcal{A}$'s encryption oracle using its own RoR oracle as follows. On input (H, M), $\mathcal{B}$ pads and partitions the message as per the scheme. $\mathcal{B}$ then computes the encryptment following the pseudocode description of encryptment algorithm $\mathsf{HFC.Enc}$, submitting a query (V, Y) to his RoR oracle every time $\mathsf{HFC.Enc}$ would compute $\mathsf{f}(V, (K_{\mathsf{EC}} \oplus Y))$. Notice that $\mathcal{B}$ makes precisely $\ell + 1$ queries; one for each of the ℓ blocks of padded header and message data, plus an additional query to compute the initial chaining variable $V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$. $\mathcal{B}$ then returns the resulting ciphertext and binding tag $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ to $\mathcal{A}$ and at the end of the game outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}$'s oracle is returning real compression function outputs then this perfectly simulates game G_0, otherwise it perfectly simulates game G_1; a straightforward argument then implies the claim.

Next we define game G_2 (not shown), which is identical to G_1 except the lazily sampled random functions no longer maintain consistency in repeated queries, instead returning a random string in response to each query. Games G_1 and G_2 run identically unless the random function is queried on the same input twice. Notice that this may only occur if one of the randomly sampled n-bit chaining variables collides with either another randomly sampled n-bit chaining variable or with the initialisation vector $IV \in \{0,1\}^n$. Supposing $\mathcal{A}$'s query is such that the message / header have a combined length of ℓ d-bit blocks after padding, then a standard argument using a birthday bound implies that

$$|\Pr[\, G_1 \Rightarrow 1\,] - \Pr[\, G_2 \Rightarrow 1\,]| \leq \frac{(\ell+1)^2}{2^{n+1}} \ ,$$

where the plus one in the numerator again follows from the extra compression function call required to compute V_0. Moreover notice that in game G_2 both C_{EC} and B_{EC} are identically distributed to random bit strings, and so

$$\Pr[\, G_2 \Rightarrow 1\,] = \Pr\left[\, \text{otROR1}^{\mathcal{A}}_{\mathsf{HFC}} \Rightarrow 1\,\right] .$$

Combining the above via a standard argument then completes the proof.

$\square$

One-time integrity and ccAEAD security. In the following theorem, we bound the otCTXT security of HFC under the additional assumption that the padding scheme used is *prefix-free*. As discussed in Section 3.4, this result, combined with those above, implies that HFC (reframed in the ccAEAD syntax) is a one-time secure ccAEAD scheme. Afterwards, we will discuss why prefix-freeness is necessary.

The following theorem bounds the advantage of an attacker $\mathcal{A}$ in game otCTXT against HFC. The proof first argues that we can replace compression function calls with random function outputs, via a reduction to the RKA-PRF security of f. It then uses the prefix-freeness of the padding scheme to argue that (barring accidental collisions amongst intermediate chaining variables, accounting for the birthday bound term in the security bound), each binding tag is computed as the result of a fresh query to the random function and so the probability that $\mathcal{A}$ can guess the value of an unseen binding tag is small. Formally arguing this point is surprisingly subtle, and constitutes the main technical challenge in the proof. With this in place we then bound $\mathcal{A}$'s guessing probability, accounting

```
proc. main // $G_0$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^d$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Enc}(\cdot,\cdot)}$
Return $b$

proc. Enc$(H, M)$ // $G_0$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$H_1, \ldots, H_h \leftarrow \mathrm{PadH}(H, M)$
$M_1, \ldots, M_m \leftarrow \mathrm{Parse}_n(M)$
$M'_1, \ldots, M'_{m-1} \leftarrow \mathrm{PadM}(H, M)$
$V_0 \leftarrow \mathsf{f}(IV, K_{\mathsf{EC}})$
For $i = 1, \ldots, h$
  $V_i \leftarrow \mathsf{f}(V_{i-1}, (K_{\mathsf{EC}} \oplus H_i))$
$C_{\mathsf{EC}} \leftarrow \varepsilon$
For $i = 1, \ldots, m-1$ do
  $C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+i-1} \oplus M_i)$
  $V_{h+i} \leftarrow \mathsf{f}(V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i))$
$C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+m-1} \oplus M_m)$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}_d(\mathrm{PadSuf}(|H|, |M|, M_m))$
$\alpha \leftarrow \max\{i \ : \ |M'_{m+i}| > 0\}$
For $i = 0, \alpha$
  $V_{h+m+i} \leftarrow \mathsf{f}(V_{h+m+i-1}, (K_{\mathsf{EC}} \oplus M'_{m+i}))$
$B_{\mathsf{EC}} \leftarrow V_{h+m+\alpha}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

```
proc. main // $G_1$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \{0,1\}^d$
$\mathsf{query\text{-}made} \leftarrow \mathsf{false}$
$b \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Enc}(\cdot,\cdot)}$
Return $b$

proc. Enc$(H, M)$ // $G_1$
If $\mathsf{query\text{-}made} = \mathsf{true}$ then return $\perp$
$\mathsf{query\text{-}made} \leftarrow \mathsf{true}$
$H_1, \ldots, H_h \leftarrow \mathrm{PadH}(H, M)$
$M_1, \ldots, M_m \leftarrow \mathrm{Parse}_n(M)$
$M'_1, \ldots, M'_{m-1} \leftarrow \mathrm{PadM}(H, M)$
$F[IV, K_{\mathsf{EC}}] \leftarrow\!\!\$\ \{0,1\}^n$ ; $V_0 \leftarrow F[IV, K_{\mathsf{EC}}]$
For $i = 1, \ldots, h$
  If $F[V_{i-1}, (K_{\mathsf{EC}} \oplus H_i)] = \perp$ then
    $F[V_{i-1}, (K_{\mathsf{EC}} \oplus H_i)] \leftarrow\!\!\$\ \{0,1\}^n$
  $V_i \leftarrow F[V_{i-1}, (K_{\mathsf{EC}} \oplus H_i)]$
$C_{\mathsf{EC}} \leftarrow \varepsilon$
For $i = 1, \ldots, m-1$ do
  $C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+i-1} \oplus M_i)$
  If $F[V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i)] = \perp$ then
    $F[V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i)] \leftarrow\!\!\$\ \{0,1\}^n$
  $V_{h+i} \leftarrow F[V_{h+i-1}, (K_{\mathsf{EC}} \oplus M'_i)]$
$C_{\mathsf{EC}} \leftarrow C_{\mathsf{EC}} \,\|\, (V_{h+m-1} \oplus M_m)$
$M'_m, M'_{m+1} \leftarrow \mathrm{Parse}(\mathrm{PadSuf}(|H|, |M|, M_m))$
$\alpha \leftarrow \max\{i \ : \ |M'_{m+i}| > 0\}$
For $i = 0, \alpha$
  If $F[V_{h+m+i-1}, (K_{\mathsf{EC}} \oplus M'_{m+i})] = \perp$ then
    $F[V_{h+m+i-1}, (K_{\mathsf{EC}} \oplus M'_{m+i})] \leftarrow\!\!\$\ \{0,1\}^n$
  $V_{h+m+i} \leftarrow F[V_{h+m+i-1}, (K_{\mathsf{EC}} \oplus M'_{m+i})]$
$B_{\mathsf{EC}} \leftarrow V_{h+m+\alpha}$
Return $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$
```

Figure 3.19: Games for proof of Theorem 3.4.

for the final term in the bound.

Theorem 3.5. *Let* HFC *be as shown in Figure 3.17, built from a compression function* $\mathsf{f} \ : \ \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *for* $d \geq n \geq 128$ *and instantiated with a prefix-free padding scheme* PadS*. Let* $\mathcal{A}$ *be an otCTXT adversary who makes* q_c ChalDec *queries among which there are* q_1 *distinct header / ciphertext pairs* (H, C_{EC})*. Suppose that the header / message (from the single query to oracle* Enc*) and the* q_1 *distinct header / ciphertexts (from the queries to oracle* ChalDec*) have a combined length of* ℓ *blocks of* d *bits after padding. Then there exists an adversary* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\text{ot-ctxt}}_{\mathsf{HFC}}(\mathcal{A}, q_c) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}, \ell+1) + \frac{(\ell+1)^2}{2^{n+1}} + \frac{q_c}{2^n - \ell - 1} \ .$$

The adversary $\mathcal{B}$ *runs in the same time as* $\mathcal{A}$ *plus an* $\mathcal{O}(\ell)$ *overhead.*

Proof. We argue by a series of game hops. We begin by defining game G_0 to be equivalent to game otCTXT against HFC. We may assume without loss of generality that $\mathcal{A}$ never repeats a query to ChalDec nor queries the tuple $(H^*, C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ where $(C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ is the encryptment returned in response to $\mathcal{A}$'s Enc query (H^*, M^*), since such a query will always result in $\perp$ being returned. More generally, we may assume that $\mathcal{A}$ never makes a ChalDec query of the form $(H^*, C^*_{\mathsf{EC}}, B_{\mathsf{EC}})$ for some $B_{\mathsf{EC}} \neq B^*_{\mathsf{EC}}$, where again starred values correspond to the challenge Enc query, since due to the strong correctness of the scheme the binding tag must be incorrect and so decryptment will return an error. Finally, we may assume that $\mathcal{A}$ never makes a ChalDec query containing e.g., a malformed block or a point outside the header / ciphertext space of the scheme which would lead the 'first phase' of decryptment (by which we mean the steps which recover the message underlying the ciphertext) to return an error, since this cannot possibly correspond to a winning query. As such, to each ChalDec query $(H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ made by $\mathcal{A}$ we may associate a pair (H, M) which will be recovered in the first decryptment phase from (H, C_{EC}) (even if $\mathsf{HFC.Dec}$ ultimately returns an error due to the incorrect binding tag). Moreover, due to the strong correctness of the scheme it follows that if $(H, C_{\mathsf{EC}}) \neq (H', C'_{\mathsf{EC}})$ then the associated header / message pairs $(H, M), (H, M')$ will be distinct also.

With this in place we now define game G_1, which is identical to G_0 except each fresh query to the compression function f is answered with an n-bit random bit string as opposed to the output of the function. A lookup table is maintained to ensure that responses to repeated queries are consistent. An analogous argument to that made in the proof of Theorem 3.4 implies that there exists an attacker $\mathcal{B}$ in the RKA-PRF game against f with the claimed query budget such that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}, \ell + 1) .$$

Next we define game G_2, which is identical to G_1 except we modify oracle ChalDec to decrypt queries for which the header / ciphertext pair (H, C_{EC}) have previously been queried to ChalDec by table look-up. In more detail, table D is initialised to $\perp$. Each time ChalDec is queried on a tuple $(H, C_{\mathsf{EC}}, B'_{\mathsf{EC}})$ such that $D[H, C_{\mathsf{EC}}] = \perp$, ChalDec runs $\mathsf{HFC.Dec}$ and sets $D[H, C_{\mathsf{EC}}] = (M, B_{\mathsf{EC}})$ where M is the message recovered during decryption and B_{EC} is the (correct) binding tag for this pair which is computed during decryption. Subsequently if ChalDec is queried on a tuple $(H, C_{\mathsf{EC}}, B'_{\mathsf{EC}})$ such that $D[H, C_{\mathsf{EC}}] \neq \perp$ it simply checks if $B_{\mathsf{EC}} = B'_{\mathsf{EC}}$, returning M if so and $\perp$ otherwise. This is a purely syntactic change which does not affect the outcome of the game, and so it follows that

$$\Pr[G_1 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1] .$$

Next we define game G_3, which is identical to game G_2 except if one of the random strings sampled to respond to a fresh query to f collides with a previously sampled string or the initialisation vector IV then we sample again such that this is not the case. Notice that these games run identically unless such a collision occurs, an event we denote coll. The Fundamental Lemma of Game Playing therefore implies that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \Pr[\mathsf{coll} \text{ in } G_2] \,.$$

We now bound this probability. Let (H^0, M^0) denote the query made by $\mathcal{A}$ to Enc. Suppose that, extracting the header and ciphertext components of $\mathcal{A}$'s q_c ChalDec queries, these contain q_1 distinct and chronologically ordered pairs $(H^1, C^1_{\mathsf{EC}}), \ldots, (H^{q_1}, C^{q_1}_{\mathsf{EC}})$, and let $(H^1, M^1), \ldots, (H^{q_1}, M^{q_1})$ denote the pairs of headers and underlying messages corresponding to these queries. Moreover, following from the discussion at the beginning of the proof, it must be the case that $(H^i, M^i) \neq (H^j, M^j)$ for $0 \leq i < j \leq q_1$. Recall that the total padded length of these messages in d-bit blocks is equal to ℓ. Notice that at most $(\ell + 1)$ strings will be sampled while processing these queries (the plus one term arising from the initial query of (IV, K_{EC}) required to compute the initial chaining variable V_0), and so a birthday bound implies that

$$\Pr[\mathsf{coll} \text{ in } G_2] \leq \frac{(\ell+1)^2}{2^{n+1}} \,.$$

We now argue that in game G_3 each binding tag $B^0_{\mathsf{EC}}, \ldots, B^{q_1}_{\mathsf{EC}}$ is computed as the result of a fresh query to f, and is chosen randomly from a set of size at least $2^n - \ell - 1$. To see this, consider computing the binding tag for each of the padded strings $\mathrm{Pad}(H^0, M^0), \ldots,$ $\mathrm{Pad}(H^{q_1}, M^{q_1})$ in G_3, which are of length $\ell_0, \ldots, \ell_{q_1}$ in d-bit blocks respectively. To $\mathrm{Pad}(H^i, M^i)$, we let $(V^i_0, \ldots, V^i_{\ell_i})$ denote the set of chaining variables passed through during the binding tag computation, where $V^i_0 = \mathsf{f}(IV, K_{\mathsf{EC}}) = V^j_0$ for all $0 \leq i < j \leq q_1$, and $B^i_{\mathsf{EC}} = V^i_{\ell_i}$. We let $|\mathrm{LCP}(i,j)|$ denote the length of the longest common prefix in d-bit blocks of $\mathrm{Pad}(H^i, M^i)$ and $\mathrm{Pad}(H^j, M^j)$. More formally, we let $|\mathrm{LCP}(i,j)| = \max\{k : \mathrm{Trunc}_{k \cdot d}(\mathrm{Pad}(H^i, M^i)) = \mathrm{Trunc}_{k \cdot d}(\mathrm{Pad}(H^j, M^j))\}$. We let $|\mathrm{LCP}(j)| = \max_{0 \leq i < j} |\mathrm{LCP}(i,j)|$; that is to say, the length of the longest prefix in d-bit blocks that $\mathrm{Pad}(H^j, M^j)$ shares with any previously processed padded message. We abuse notation to let M^j_k denote the kth d-bit block of $\mathrm{Pad}(H^j, M^j)$.

We claim that the following two statements hold: **(1)** Let $1 \leq j \leq q_1$, and $p = |\mathrm{LCP}(j)|$. Then for all $0 \leq i < j$ such that $\mathrm{LCP}(i,j) = p$, it holds that $p < \min(\ell_i, \ell_j)$; and **(2)** consider the binding tag computation for padded messages $\mathrm{Pad}(H^i, M^i)$ and $\mathrm{Pad}(H^j, M^j)$ where $0 \leq i < j \leq q_1$, and suppose that of the associated series of chaining variables it holds that $V^i_k = V^j_{k'}$ for some $0 \leq k \leq \ell_i$, $0 \leq k' \leq \ell_j$. Then it must be the case that $k = k'$ and the first k blocks of $\mathrm{Pad}(H^i, M^i)$ and $\mathrm{Pad}(H^j, M^j)$ are equal.

To see that **(1)** holds, first recall that the padding scheme Pad is prefix-free. Since by assumption $(H^i, M^i) \neq (H^j, M^j)$ for all $0 \leq i < j \leq q_1$, it must be the case that $\mathrm{Pad}(H^i, M^i)$ is not a prefix of $\mathrm{Pad}(H^j, M^j)$ or vice versa. As such they must differ in at least one d-bit block, and so it follows that $|\mathrm{LCP}(i,j)| < \min(\ell_i, \ell_j)$.

For **(2)**, note that if $k = k' = 0$ then the statement is vacuously true, and so we suppose for the remainder of the proof that at least one of k, k' is non-zero. Suppose for a contradiction there exist $\mathrm{Pad}(H^i, M^i), \mathrm{Pad}(H^j, M^j)$ which share a common chaining variable $V_k^i = V_{k'}^j$, but for which the claim does not hold. Since the strings returned in response to compression function calls are sampled without replacement in G_3, distinct queries will always return distinct responses. Suppose that $k, k' > 0$ (the case in which one of k, k' equals 0 is entirely analogous). Then $V_k^i, V_{k'}^j$ are computed by querying $(V_{k-1}^i, (K_{\mathsf{EC}} \oplus M_k^i))$ and $(V_{k'-1}^j, (K_{\mathsf{EC}} \oplus M_{k'}^j))$ to f respectively. As such, it must be the case that $V_{k-1}^i = V_{k'-1}^j$ and $M_k^i = M_k^j$. Repeatedly applying this argument yields that $V_{k-\alpha}^i = V_{k'-\alpha}^j$ and $M_{k-\alpha+1}^i = M_{k'-\alpha+1}^j$ for all $\alpha = 1, \ldots, \min(k, k')$. Suppose first that $k \neq k'$, and suppose without loss of generality that $k < k'$. This implies that $V_0^i = V_{k'-k}^j$ where $k' - k \geq 1$. Moveover, since $V_0^i = \mathsf{f}(IV, K_{\mathsf{EC}})$ this implies that $V_{k'-k-1}^j = IV$. However, this is impossible since $V_{k'-k-1}^j$ must have been computed by via a call to f and in the hop to G_3 we effectively removed IV from the range of f. Therefore it must be the case that $k = k'$, and so by the previous argument $(M_1^i, \ldots, M_k^i) = (M_1^j, \ldots, M_k^j)$ also, proving the claim.

With this in place, consider computing the binding tag for a message $\mathrm{Pad}(H^j, M^j)$, where $p = |\mathrm{LCP}(j)|$. Let $\mathrm{Prev}_j = \{\mathrm{Pad}(H^k, M^k)\}_{0 \leq k < j}$ be the set of padded header / message pairs resulting from the first $j-1$ queries to ChalDec and the single query to Enc. Let $\mathcal{X}_j = \{\mathrm{Pad}(H^i, M^i) \in \mathrm{Prev}_j \ : \ \mathrm{LCP}(i,j) = p\}$; in other words, the set of previously processed padded header / message pairs which share a prefix of length p blocks with $\mathrm{Pad}(H^j, M^j)$. Due to the consistency in responses to repeated queries to the compression function, it is clearly the case that for all $\mathrm{Pad}(H^i, M^i) \in \mathcal{X}_j$ it holds that $(V_0^i, \ldots, V_p^i) = (V_0^j, \ldots, V_p^j)$. Since by **(1)** it must be the case that $p < \ell_i, \ell_j$, it follows that $M_{p+1}^i, M_{p+1}^j \neq \varepsilon$, and moreover the next chaining variable for $\mathrm{Pad}(H^j, M^j)$ will be computed via querying $(V_p^j, (K_{\mathsf{EC}} \oplus M_{p+1}^j))$ to f.

We claim that this will be a fresh query to f and so will be answered with a string chosen uniformly from the set of at least $2^n - \ell - 1$ unused points. To see this, notice that for all $\mathrm{Pad}(H^i, M^i) \in \mathcal{X}_j$ it holds that $M^i_{p+1} \neq M^j_{p+1}$, since otherwise this would contradict the maximality of p. Now **(2)** implies that V^j_p cannot equal V^i_k for any $\mathrm{Pad}(H^i, M^i)$, $0 \le i < j$ and $0 \le k \le \ell_i$ *unless* $\mathrm{Pad}(H^i, M^i) \in \mathcal{X}_j$ and $k = p$. This implies that V^j_p has not been queried to f at any point during the computation of the first j binding tags other than during the computation of the $(p+1)^{\text{st}}$ chaining variable for strings in $\mathcal{X}_j$. Since we have already argued that all such queries have a distinct message block and so do not collide, it follows that $(V^j_p, (K_{\mathsf{EC}} \oplus M^j_{p+1}))$ represents a fresh query to f, and so V^j_{p+1} is chosen uniformly from the set of available points. This in turn forces the query made to compute the next chaining variable to be distinct from all points previously queried, and so repeatedly applying the same argument implies the binding tag $B^j_{\mathsf{EC}} = V^j_{\ell_j}$ is the result of a fresh query also. Finally, since at most $(\ell + 1)$ chaining variables are sampled while processing the padded queries of combined length ℓ d-bit blocks, and the random strings returned in response to fresh f queries are sampled without replacement (and from a set excluding $IV \in \{0,1\}^n$), it follows that each binding tag is chosen from a set of size at least $(2^n - \ell - 1)$, proving the claim.

We conclude by bounding the probability that $\mathcal{A}$ makes a winning query to ChalDec, an event we denote win. For $1 \le i \le q_1$, we mark the event that $\mathcal{A}$ makes a query $(H^i, C^i_{\mathsf{EC}}, B'_{\mathsf{EC}})$ to ChalDec such that $B'_{\mathsf{EC}} = B^i_{\mathsf{EC}}$ by setting a flag win_i. Suppose that $\mathcal{A}$ makes a_i queries with header / ciphertext (H^i, C^i_{EC}), and notice that $\sum_{i=1}^{q_1} a_i = q_c$. It follows that

$$\begin{aligned}
\Pr[\,\mathsf{win} = \mathsf{true}\,] &= \Pr[\,\vee_{i=1}^{q_1} \mathsf{win}_i = \mathsf{true}\,] \\
&\le \sum_{i=1}^{q_1} \sum_{j=1}^{a_i} \Pr\left[\mathsf{win}_i \text{ set by } j^{\text{th}} \text{ query of form } (H^i, C^i_{\mathsf{EC}}, \cdot)\right] \\
&\le \sum_{i=1}^{q_1} \sum_{j=1}^{a_i} \frac{1}{2^n - \ell - 1} \\
&\le \frac{q_c}{2^n - \ell - 1}\,.
\end{aligned}$$

The first inequality follows from a union bound. The second inequality follows since each binding tag is chosen uniformly from a set of size at least $2^n - \ell - 1$. The final inequality follows since $\sum_{i=1}^{q_1} a_i = q_c$. □

Necessity of prefix-free padding. We note that without a prefix-free padding scheme,

HFC is not otCTXT secure in general. Indeed for the padding scheme of Figure 3.16, an attacker can construct a pair (H, M) for his Enc query such that a prefix of $\mathrm{Pad}(H, M)$ is equal to $\mathrm{Pad}(H, M')$ for some $(H, M) \neq (H, M')$. By choosing (H, M) such that the binding tag for (H, M') is among the random pads used to encrypt M, $\mathcal{A}$ can then recover the correct binding tag and random pads for (H, M') from the ciphertext C_{EC} corresponding to (H, M). This in turn enables $\mathcal{A}$ to construct a valid forgery for (H, M').

The attack. For an example of this attack, suppose that HFC is instantiated with a compression function $\mathsf{f} : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ and the padding scheme PadS shown in Figure 3.16. We will demonstrate an efficient adversary $\mathcal{A}$ such that

$$\mathbf{Adv}^{\text{ot-ctxt}}_{\mathsf{HFC}}(\mathcal{A}, 1) = 1 .$$

Let $\mathcal{A}$ be the attacker in game otCTXT who proceeds as follows. $\mathcal{A}$ picks an arbitrary message $M^* \in \{0,1\}^{128}$. It is straightforward to verify that $\mathrm{Pad}(\varepsilon, M^*) = M_1' \,\|\, M_2'$ where $M_1' = M^*$ and $M_2' = (0)_{64} \,\|\, (128)_{64}$. As such, letting $V_0 = \mathsf{f}(IV, K_{\mathsf{EC}})$, it holds that the encryptment under K_{EC} of (ε, M^*) will be

$$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) = (V_0 \oplus M_1', \mathsf{f}^{+}(V_0, (K_{\mathsf{EC}} \oplus M_1') \,\|\, (K_{\mathsf{EC}} \oplus M_2'))) . \tag{3.4}$$

$\mathcal{A}$ uses his single Enc query to submit $(H, M) = (\varepsilon, M^* \,\|\, (0)_{64} \,\|\, (128)_{64} \,\|\, \hat{M})$ for some arbitrary $\hat{M} \in \{0,1\}^{128}$ to his Enc oracle. It is straightforward to verify that for the padding scheme in Figure 3.16 it holds that $\mathrm{Pad}(H, M) = M_1' \,\|\, M_2' \,\|\, M_3' \,\|\, M_4'$, where $M_1' = M^*$, $M_2' = (0)_{64} \,\|\, (128)_{64}$, $M_3' = \hat{M}$, and $M_4' = (0)_{64} \,\|\, (384)_{64}$. As such, the encryptment received by $\mathcal{A}$ in response to their Enc query will be of the form $(C_1 \,\|\, C_2 \,\|\, C_3, B_{\mathsf{EC}}')$ where $C_1 = V_0 \oplus M_1'$, $C_2 = \mathsf{f}^{+}(V_0, (K_{\mathsf{EC}} \oplus M_1')) \oplus M_2'$, $C_3 = \mathsf{f}^{+}(V_0, (K_{\mathsf{EC}} \oplus M_1') \,\|\, (K_{\mathsf{EC}} \oplus M_2')) \oplus M_3'$, and $B_{\mathsf{EC}}' = \mathsf{f}^{+}(V_0, (K_{\mathsf{EC}} \oplus M_1') \,\|\, \ldots \,\|\, (K_{\mathsf{EC}} \oplus M_4'))$. In particular, notice that C_1 is identical to C_{EC} in Equation 3.4, and that moreover the random pad used to produce C_3 is identical to the binding tag in Equation 3.4. As such, $(C_1, C_3 \oplus \hat{M})$ is the correct encryptment of (ε, M^*) under K_{EC}, and so by submitting $(\varepsilon, (C_1, C_3 \oplus \hat{M}))$ to his ChalDec oracle $\mathcal{A}$ will win game otCTXT with probability one having made a single ChalDec query.

The attack may still be executed for compression functions with $d > n$, subject to it being possible to encode the length of M in a manner consistent with the zero padding performed by the scheme. Moreover, $\mathcal{A}$ will have to brute-force the $(d - n)$-bits of the binding tag which are truncated when encrypting the n-bit message block.

otCTXT without prefix-free padding. It seems likely that small modifications to the HFC construction would achieve otCTXT security while only requiring an injective padding scheme. For example, consider a modified scheme HFC' which takes as input a key $(K_{\mathsf{EC}}, K'_{\mathsf{EC}}) \leftarrow\!\!\$\ \{0,1\}^d \times \{0,1\}^d$. Modified encryptment $\mathsf{HFC.Enc}'$ on input $((K_{\mathsf{EC}}, K'_{\mathsf{EC}}), H, M)$ computes $\mathsf{HFC.Enc}(K_{\mathsf{EC}}, H, M) = (C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and outputs $(C_{\mathsf{EC}}, (B_{\mathsf{EC}}, \mathsf{f}(B_{\mathsf{EC}}, K'_{\mathsf{EC}})))$, where we call the latter value in the tuple the *authentication tag*. To decrypt $(H, C_{\mathsf{EC}}, (B_{\mathsf{EC}}, B'_{\mathsf{EC}}))$, $\mathsf{HFC.Dec}'$ computes $\mathsf{HFC.Dec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and checks that $B'_{\mathsf{EC}} = \mathsf{f}(B_{\mathsf{EC}}, K'_{\mathsf{EC}})$, returning $\perp$ if either of these steps fails. Let $(H^*, C^*_{\mathsf{EC}}, (B^*_{\mathsf{EC}}, B^{*'}_{\mathsf{EC}}))$ denote the encryptment arising from the attacker's Enc query in game otCTXT against HFC'. Now the attacker will be required to guess a pseudorandom authentication tag in order to create a successful forgery *unless* they can find $(H, C_{\mathsf{EC}}) \neq (H^*, C^*_{\mathsf{EC}})$ such that $\mathsf{HFC.Dec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}}) \neq \perp$, which in turn breaks the SCU security of HFC. We leave formalising this intuition, and proving the properties of such a modified scheme, to future work.

3.6.4 Instantiating the HFC Scheme

We conclude this section with a discussion of the different compression functions that may be used to instantiate the HFC encryptment scheme.

The obvious (and probably best) choice to instantiate f is the $\mathsf{SHA\text{-}256}$ or $\mathsf{SHA\text{-}512}$ compression function. These provide good software performance, and there is a shift towards widespread hardware support in the form of the Intel SHA instructions [5, 74, 160]. Extensive cryptanalysis for the CR (e.g., [100, 145]), preimage resistance (e.g., [75, 89]), and RKA-PRP security of the associated SHACAL block cipher (e.g., [79, 90, 91, 103]) gives confidence in its security.

Other options, although in some cases less well-studied cryptanalytically, include $\mathsf{SHA\text{-}3}$ finalists. In particular a variant of the HFC construction using a sponge-based mode such as Keccak, in which the key is fed to the sponge prior to hashing the message blocks, would allow us to avoid the RKA assumption; we discuss this in more detail below. We could also remove the RKA assumption by using a compression function with a dedicated key input such as $\mathsf{LP231}$ [139]. $\mathsf{BLAKE2b}$ [7] is a variant of the BLAKE hash function and was also a $\mathsf{SHA\text{-}3}$ finalist. Its compression function is not explicitly block cipher-based; it is built from a variant of the quarter-round function of the ChaCha20 stream cipher [27]. It is believed to have security comparable to SHA-256, but is notable for efficiency — on some platforms it outperforms even MD5 in software.

Another approach would be to use AES via a PGV compression function [125]. We focus our discussion further on Davies-Meyer (DM), one of the secure PGV constructions [38]. Letting $\mathrm{E} : \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher, DM is defined as $\mathsf{DM}(V, M) = \mathrm{E}(M, V) \oplus V$. For HFC, an important advantage of DM over other PGV compression functions (e.g., Matyas-Meyer-Oseas) is that the block cipher keys do not depend on intermediate chaining values and so key scheduling for the entire (encoded) message can be done up front. It is straightforward to verify that the linear-only RKA-PRF security of DM is inherited from the linear-only RKA-PRF security of the underlying cipher. An obvious choice for E would be AES-128. On systems with AES-NI, HFC instantiated with AES-DM will have very good performance, although not quite as fast as AES-GCM or AES-OCB given the need to rekey for every block. The RKA security of AES has been studied extensively, and known attacks do not falsify the assumptions we need [35, 36]. However, analysed in the ICM, a birthday attack implies that an attacker can find a collision in DM in roughly $2^{n/2}$ queries to the ideal cipher. By Theorem 3.3, this implies that HFC instantiated with AES-DM cannot achieve sr-BIND security of more than 2^{64}-bits, which is in general insufficient in practice.

Encryptment from sponges. Our HFC construction has a number of similarities to the AE scheme SpongeWrap built using the SHA-3 winner Keccak [32, 33]. The SpongeWrap scheme uses a large permutation $\pi : \{0,1\}^{r_s+c_s} \to \{0,1\}^{r_s+c_s}$ for some $r_s, c_s \in \mathbb{N}$. This permutation is used to iteratively hash the message, with the resulting output forming the binding tag. Portions of the intermediate chaining variables are simultaneously used as pads to mask the message. The key difference between this scheme and HFC is that the large state size offered by Keccak means that there is no need to key each call to π — this is done indirectly via the extra c_s bits of state that are not used for outputs.

We now describe our SpongeWrap-like encryptment scheme SPE. It works as follows. Encryptment of a sequence of padded message blocks $M_1, \ldots, M_m$ each of length r_s-bits with a key $K_{\mathsf{EC}} \in \{0,1\}^{r_s}$ first sets $Y_0 = \pi(K_{\mathsf{EC}} \,\|\, 0^{c_s})$. The scheme then iteratively computes $C_i = Y_{i-1} \oplus M_i$ where $Y_i = \pi(Y_{i-1} \oplus (M_i \,\|\, 0^{c_s}))$ for $1 \le i \le m$. The binding tag B_{EC} is then set to be the leftmost r_s bits of $\pi(Y_m)$, and the scheme outputs encryptment

$(C_1 \parallel \cdots \parallel C_m, B_{\mathsf{EC}})$. Decryption and verification work in the natural way. With a suitable padding scheme, SPE easily extends to handle associated data and arbitrary length messages.

The security analyses from [32] imply that SPE as an encryptment scheme achieves otROR and SCU security. Tighter bounds can likely be obtained using techniques from [83]. r-BIND security follows from the collision resistance of Keccak, and it is straightforward to verify that the scheme is also sender binding and strongly correct. As such, SPE may be a good choice should SHA-3 be implemented widely.

From encryptment to ccAEAD. This concludes our analysis of the HFC encryptment scheme. In the next section we will show how to build fully-fledged multi-use ccAEAD from encryptment, and thereby use HFC as the basis for the first single-pass secure ccAEAD scheme.

3.7 Compactly Committing AEAD from Encryptment

In this section, we consider the main topic of the chapter: building efficient and secure compactly committing AEAD (ccAEAD). First, we recall the formal notions for ccAEAD following the treatment given by GLR [73] and compare these to encryptment. With this in place, we show in Sections 3.7.3 and 3.7.4 how to build ccAEAD from encryptment with very efficient transforms. Moreover, in Section 3.7.5 we show how to construct secure encryptment from ccAEAD in a way that transfers our negative results from Section 3.5 to ccAEAD, and in Section 3.7.6 we investigate the relationship between robust AEAD and encryptment.

3.7.1 ccAEAD Syntax and Correctness

We begin by defining a syntax for ccAEAD schemes. As we shall see, encryptment can be viewed as a one-time secure, deterministic variant of ccAEAD. We further discuss the differences between the two primitives later in the section.

Definition 3.4. *A ccAEAD scheme is a tuple of algorithms* $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$

with associated key space $\mathcal{K} \subseteq \Sigma^*$, header space $\mathcal{H} \subseteq \Sigma^*$, message space $\mathcal{M} \subseteq \Sigma^*$, ciphertext space $\mathcal{C} \subseteq \Sigma^*$, opening space $\mathcal{K}_f \subseteq \Sigma^*$, and binding tag space $\mathcal{T} \subseteq \Sigma^*$, defined as follows.

- *The randomised key generation algorithm* $\mathsf{CKg} : \to \mathcal{K}$ *takes no input, and outputs a secret key $K \in \mathcal{K}$.*
- *The randomised encryption algorithm* $\mathsf{CEnc} : \mathcal{K} \times \mathcal{H} \times \mathcal{M} \to \mathcal{C} \times \mathcal{T}$ *takes as input a tuple $(K, H, M) \in \mathcal{K} \times \mathcal{H} \times \mathcal{M}$, and outputs a ciphertext / binding tag pair $(C, C_B) \in \mathcal{C} \times \mathcal{T}$.*
- *The deterministic decryption algorithm* $\mathsf{CDec} : \mathcal{K} \times \mathcal{H} \times \mathcal{C} \times \mathcal{T} \to (\mathcal{M} \times \mathcal{K}_f) \cup \{\bot\}$ *takes as input a tuple $(K, H, C, C_B) \in \mathcal{K} \times \mathcal{H} \times \mathcal{C} \times \mathcal{T}$, and outputs a message / opening pair $(M, K_f) \in \mathcal{M} \times \mathcal{K}_f$ or the error symbol $\bot$.*
- *The deterministic verification algorithm* $\mathsf{CVer} : \mathcal{H} \times \mathcal{M} \times \mathcal{K}_f \times \mathcal{T} \to \{0,1\}$ *takes as input a tuple $(H, M, K_f, C_B) \in \mathcal{H} \times \mathcal{M} \times \mathcal{K}_f \times \mathcal{T}$, and outputs a bit $b \in \{0,1\}$.*

Correctness and compactness. Correctness for ccAEAD schemes is defined identically to the COR correctness notion for encryptment schemes (Figure 3.7), except in the ccAEAD case the probability is now over the coins of CEnc also. We additionally require length regularity; that is to say that the length of ccAEAD ciphertexts C depends only on the length of the underlying message. Formally, we require that there exists a function $\mathsf{clen}_{\mathsf{CE}} : \mathbb{N} \to \mathbb{N}$ such that for all $(K, H, M) \in \mathcal{K} \times \mathcal{H} \times \mathcal{M}$ it holds that

$$\Pr\left[\, |C| = \mathsf{clen}_{\mathsf{CE}}(|M|) \;:\; (C, C_B) \leftarrow\!\!\$\; \mathsf{CEnc}(K, H, M) \,\right] = 1 \;,$$

where the probability is over the coins of CEnc. Finally we require that binding tags C_B are *compact*, by which we mean that all C_B returned by a ccAEAD scheme are of constant length blen.

Comparison with encryptment. With this in place, we highlight the key differences between encryptment and ccAEAD. The overarching difference is that encryptment schemes are single-use (i.e., a key is only ever used to encrypt a single message), whereas ccAEAD schemes are multi-use. To support this the encryption algorithm for a ccAEAD scheme is randomised, whereas for encryptment this algorithm is deterministic. This allows us to construct schemes that enjoy security in the face of attackers that can obtain multiple encryptions. We note that nonce-based ccAEAD [73] can also achieve multi-use security. We focus on randomised ccAEAD here, and leave the problem of building fast

```
$\text{MO-REAL}^{\mathcal{A}}_{\mathsf{CE}}$:
$K \leftarrow\$ \mathsf{CKg}$
$b' \leftarrow\$ \mathcal{A}^{\text{Enc,Dec,ChalEnc}}$
Return $b'$

$\text{Enc}(H, M)$
$(C, C_B) \leftarrow\$ \mathsf{CEnc}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, C_B)\}$
Return $(C, C_B)$

$\text{Dec}(H, C, C_B)$
If $(H, C, C_B) \notin \mathcal{Y}$ then
  Return $\perp$
$(M, K_f) \leftarrow \mathsf{CDec}(K, H, C, C_B)$
Return $(M, K_f)$

$\text{ChalEnc}(H, M)$
$(C, C_B) \leftarrow\$ \mathsf{CEnc}(K, H, M)$
Return $(C, C_B)$
```

```
$\text{MO-RAND}^{\mathcal{A}}_{\mathsf{CE}}$:
$K \leftarrow\$ \mathsf{CKg}$
$b' \leftarrow\$ \mathcal{A}^{\text{Enc,Dec,ChalEnc}}$
Return $b'$

$\text{Enc}(H, M)$
$(C, C_B) \leftarrow\$ \mathsf{CEnc}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, C_B)\}$
Return $(C, C_B)$

$\text{Dec}(H, C, C_B)$
If $(H, C, C_B) \notin \mathcal{Y}$ then
  Return $\perp$
$(M, K_f) \leftarrow \mathsf{CDec}(K, H, C, C_B)$
Return $(M, K_f)$

$\text{ChalEnc}(H, M)$
$(C, C_B) \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{CE}}(|M|)} \times \{0,1\}^{\mathsf{blen}}$
Return $(C, C_B)$
```

```
$\text{MO-CTXT}^{\mathcal{A}}_{\mathsf{CE}}$:
$K \leftarrow\$ \mathsf{CKg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\text{Enc,Dec,ChalDec}}$
Return $\mathsf{win}$

$\text{Enc}(H, M)$
$(C, C_B) \leftarrow\$ \mathsf{CEnc}(K, H, M)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, C_B)\}$
Return $(C, C_B)$

$\text{Dec}(H, C, C_B)$
Return $\mathsf{CDec}(K, H, C, C_B)$

$\text{ChalDec}(H, C, C_B)$
If $(H, C, C_B) \in \mathcal{Y}$ then
  Return $\perp$
$(M, K_f) \leftarrow \mathsf{CDec}(K, H, C, C_B)$
If $M \neq \perp$ then
  $\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_f)$
```

Figure 3.20: Confidentiality (left and middle) and ciphertext integrity (right) games for ccAEAD.

nonce-based ccAEAD to future work.

Another difference is that encryptment schemes are restricted to use the same key for verification as they use for encryptment, whereas ccAEAD schemes output an explicit opening key K_f during decryption. There is no requirement that this equals the secret key used for encryption. Again, outputting an opening key distinct from the encryption key allows for ccAEAD schemes that maintain confidentiality and integrity even after some ciphertexts produced under a given encryption key have been opened.

Comparison with AEAD. The definition of an AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ can be recovered from the above definition of ccAEAD by noticing that each algorithm can be defined identically to their ccAEAD variants, except in the AEAD case we view the ciphertext / binding tag pair as a single ciphertext and modify decryption to no longer output the opening.

3.7.2 Security Notions for ccAEAD Schemes

We now define security notions for ccAEAD schemes following GLR. They adapt the familiar security notions of real-or-random (ROR) ciphertext indistinguishability [138] and ciphertext integrity (SCU) [19] to the ccAEAD setting. We focus on GLR's *multi-opening* (MO) security notions. MO-ROR (resp. MO-CTXT) requires that if multiple messages are encrypted under the same key, then learning the message / opening pair (M, K_f) for some of the resulting ciphertexts does not compromise the ROR (resp. CTXT) security of the remaining unopened ciphertexts. This precludes schemes which e.g., have the opening key K_f equal to the secret encryption key K.

Confidentiality. Games MO-REAL and MO-RAND are shown in Figure 3.20. In both variants, the attacker is given access to an oracle ChalEnc to which he may submit message / header pairs. This oracle returns real (resp. random) ciphertext / binding tag pairs in game MO-REAL (resp. MO-RAND). The attacker is then challenged to distinguish between the two games. To model multi-opening security the attacker is also given a pair of encryption / decryption oracles, Enc and Dec, and may submit the (real) ciphertexts generated via a query to the former to the latter, learning the openings of these ciphertexts in the process. The challenge decryption oracle will return $\perp$ for any ciphertext not generated via the encryption oracle. This is because the definition targets a CPA-style notion which requires ROR security to hold even when an attacker may learn openings, as opposed to a chosen ciphertext-style definition in which the attacker may decrypt any ciphertext that does not imply a trivial win. We define the advantage of an attacker $\mathcal{A}$ in game MO-ROR against a ccAEAD scheme CE who makes q_c queries to ChalEnc, q_e queries to Enc, and q_d queries to Dec, as

$$\mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{A}, q_c, q_e, q_d) = \left|\Pr\left[\text{MO-REAL}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow 1\right] - \Pr\left[\text{MO-RAND}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow 1\right]\right| .$$

Ciphertext integrity. Ciphertext integrity guarantees that an attacker cannot produce a fresh ciphertext which will decrypt correctly. The multi-opening adaptation to the ccAEAD setting MO-CTXT is shown in Figure 3.20. The attacker $\mathcal{A}$ is given access to an encryption oracle Enc and a challenge decryption oracle ChalDec. The attacker wins if he submits a ciphertext to ChalDec which decrypts correctly and which was not the result of a previous query to the encryption oracle. To model multi-opening security, the attacker is given access to a further oracle Dec via which he may decrypt ciphertexts and learn the corresponding openings. The advantage of an attacker $\mathcal{A}$ in game MO-CTXT against a ccAEAD scheme CE, who makes q_c ChalDec queries, q_e Enc queries, and q_d Dec queries, is defined

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE}}(\mathcal{A}, q_c, q_e, q_d) = \Pr\left[\text{MO-CTXT}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow \mathsf{true}\right] .$$

```
sr-BIND$^{\mathcal{A}}_{\mathsf{CE}}$
$((H, M, K_f), (H', M', K'_f), C_B) \leftarrow\$ \mathcal{A}$
$b \leftarrow \mathsf{CVer}(H, M, K_f, C_B)$
$b' \leftarrow \mathsf{CVer}(H', M', K'_f, C_B)$
If $(H, M, K_f) = (H', M', K'_f)$
    Return false
Return $(b = b' = 1)$
```

```
s-BIND$^{\mathcal{A}}_{\mathsf{CE}}$
$(K, H, C, C_B) \leftarrow\$ \mathcal{A}$
$(M', K'_f) \leftarrow \mathsf{CDec}(K, H, C, C_B)$
If $(M', K'_f) = \perp$ then return false
$b \leftarrow \mathsf{CVer}(H, M', K'_f, C_B)$
If $b = 0$
    Return true
Return false
```

Figure 3.21: Binding notions for a ccAEAD scheme $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$.

Relation to AEAD security notions. We note that the familiar ROR and CTXT notions for AEAD schemes can be recovered from the corresponding ccAEAD games in Figure 3.20 by reframing the ccAEAD scheme as an AEAD scheme as described previously, removing access to oracle Dec in all games, and removing Enc in MO-REAL and MO-RAND. Advantage functions are defined analogously. Since here we are removing attacker capabilities, it follows that MO-ROR and MO-CTXT security for a ccAEAD scheme implies ROR and CTXT security for the derived AEAD scheme also.

Receiver and sender binding. Binding games for ccAEAD schemes are shown in Figure 3.21. Strong receiver binding for ccAEAD schemes is the same as for encryptment (see Figure 3.10) except the attacker outputs openings K_f, K'_f rather than encryptment keys $K_{\mathsf{EC}}, K'_{\mathsf{EC}}$ as part of his guess. The sender binding game for a ccAEAD scheme challenges an attacker $\mathcal{A}$ to output a tuple (K, H, C, C_B) such that $(M, K_f) \leftarrow \mathsf{CDec}(K, H, C, C_B)$ does not equal $\perp$ but $\mathsf{CVer}(H, M, K_f, C_B) = 0$. This is the same as the associated game for encryptment, except that the opening K_f recovered during decryption is used for verification rather than the key output by $\mathcal{A}$. For Gm $\in$ {sr-BIND, s-BIND}, we define

$$\mathbf{Adv}^{\mathrm{gm}}_{\mathsf{CE}}(\mathcal{A}) = \Pr\left[\, \mathrm{Gm}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow \mathsf{true} \,\right] .$$

Given the similarities, we abuse notation by using the same names for ccAEAD binding notions as in the encryptment case; which version will be clear from the context.

Relation between r-BIND **and** sr-BIND**.** In the following theorem we prove that sr-BIND security implies r-BIND security for ccAEAD schemes; our results readily extend to encryptment also. We then show that the converse does not hold by constructing a scheme which is receiver binding but for which strong receiver binding can be trivially broken.

Theorem 3.6. *Let* $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ *be a ccAEAD scheme. Then for any attacker* $\mathcal{A}$ *in game* r-BIND *against* CE*, there exists an attacker* $\mathcal{B}$ *in game* sr-BIND *against* CE *such that*

$$\mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{B}) \ ,$$

and moreover $\mathcal{B}$ *runs in the same time* $\mathcal{A}$*. On the other hand, there exists a CE scheme* $\mathsf{CE}' = (\mathsf{CKg}', \mathsf{CEnc}', \mathsf{CDec}', \mathsf{CVer}\text{'})$ *which is* r-BIND *secure but for which there exists an efficient attacker* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{C}) = 1 \ .$$

Proof. To prove the first claim, let $\mathcal{A}$ be an attacker in game r-BIND against CE. We then define $\mathcal{B}$ to be the attacker in game sr-BIND against CE who simply runs $\mathcal{A}$; eventually $\mathcal{A}$ halts and outputs $((H, M, K_f), (H', M', K'_f), C_B)$ and $\mathcal{B}$ returns the same tuple to his challenger. Then

$$\begin{aligned}
&\mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{A}) = \Pr\left[\text{r-BIND}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow \mathsf{true}\right] \\
&= \Pr\left[\mathsf{CVer}(H, M, K_f, C_B) = \mathsf{CVer}(H', M', K'_f, C_B) = 1 \wedge (H, M) \neq (H', M')\right] \\
&\leq \Pr\left[\mathsf{CVer}(H, M, K_f, C_B) = \mathsf{CVer}(H', M', K'_f, C_B) = 1 \wedge (H, M, K_f) \neq (H', M', K'_f)\right] \\
&= \Pr\left[\text{sr-BIND}^{\mathcal{B}}_{\mathsf{CE}} \Rightarrow \mathsf{true}\right] \\
&= \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}}(\mathcal{B}) \ ,
\end{aligned}$$

where all probabilities are over the coins of $\mathcal{A}$ (recall that CVer is deterministic), thereby proving the claim.

To prove that the converse does not hold, we now define a CE scheme which is r-BIND secure but not sr-BIND secure. Take any CE scheme $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ which is r-BIND secure and for which opening space $\mathcal{K}_f = \{0,1\}^n$ (for example, one may take the CEP scheme from [73]). We define a modified scheme $\mathsf{CE}' = (\mathsf{CKg}', \mathsf{CEnc}', \mathsf{CDec}', \mathsf{CVer}')$ as follows. We let $\mathsf{CKg}', \mathsf{CEnc}', \mathsf{CDec}'$ be identical to $\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}$. We define the opening space of the modified scheme to be $\mathcal{K}'_f = \{0,1\}^{n+1}$, so that $\mathcal{K}_f \subset \mathcal{K}'_f$. Moreover, we define CVer' to be the algorithm which on input (H, M, K_f, C_B) computes $\overline{K_f} = \text{Trunc}_n(K_f)$ and returns the output of $\mathsf{CVer}(H, M, \overline{K_f}, C_B)$.

It is straightforward to see that the modified scheme CE' is r-BIND secure. Indeed for any attacker $\mathcal{E}$ in game r-BIND against CE', we may define an attacker $\mathcal{F}$ in game r-BIND against CE who simply runs $\mathcal{E}$ and outputs $(H, M, \text{Trunc}_n(K_f)), (H', M', \text{Trunc}_n(K'_f)), C_B)$

```
CKg
    $K \leftarrow\$ \mathsf{K}$
    Return $K$

CEnc$(K, H, M)$
    $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
    $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
    $C_{\mathsf{AE}} \leftarrow\$ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
    Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

CDec$(K, H, (C, C_B))$
    $(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
    $K_{\mathsf{EC}} \leftarrow \mathsf{D}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
    If $K_{\mathsf{EC}} = \perp$ then return $\perp$
    $M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
    If $M = \perp$ then return $\perp$
    Return $(M, K_{\mathsf{EC}})$

CVer$(H, M, K_f, C_B)$
    $b \leftarrow \mathsf{EVer}(H, M, K_f, C_B)$
    Return $b$
```

Figure 3.22: A generic transform from an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ and a standard AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ to a multi-opening ccAEAD scheme $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}] = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$.

where $((H, M, K_f), (H', M', K'_f), C_B)$ is the tuple output by $\mathcal{E}$. It is easy to see that any winning output for $\mathcal{E}$ implies a winning output for $\mathcal{F}$ also, and so $\mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}'}(\mathcal{E}) \leq \mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{F})$.

However, an adversary $\mathcal{C}$ may win game sr-BIND with probability 1 by taking any tuple (H, M, K_f, C_B) for which $1 \leftarrow \mathsf{CVer}(H, M, K_f, C_B)$, and submitting $((H, M, K_f \,\|\, 0), (H, M, K_f \,\|\, 1), C_B)$ to his challenger. Now $(H, M, K_f \,\|\, 0) \neq (H, M, K_f \,\|\, 1)$, but since $\mathsf{CVer}'(H, M, K_f \,\|\, 0) = \mathsf{CVer}(H, M, K_f \,\|\, 1) = 1$ it holds that $\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}'}(\mathcal{C}) = 1$, implying the result. □

Relation to robust AEAD. Given that both target certain binding notions, a natural question is whether an sr-BIND secure ccAEAD scheme is robust [67] when reframed as an AEAD scheme (see Section 3.7.1) and vice versa. In Section 3.7.6, we show that neither notion implies the other in generality. We also discuss the conditions under which the ccAEAD schemes we build from secure encryptment are robust.

3.7.3 A Transform From Encryptment to ccAEAD via AEAD

We now turn to building ccAEAD from encryptment. Fix an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ and a standard AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$. Let $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}] = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ be the ccAEAD scheme shown in Figure 3.22.

Overview. Key generation CKg for $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ simply runs $K \leftarrow\$ \mathsf{K}$ and outputs K. To encrypt a header / message (H, M), CEnc uses the key generation algorithm of the encryptment scheme to generate a one-time encryptment key $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$ and computes the encryptment of the header and message via $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$. The scheme then uses the encryption algorithm of the AEAD scheme to encrypt the one-time key K_{EC} with header B_{EC}, producing $C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$, and outputs $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. On input (K, H, C, C_B) where $C = (C_{\mathsf{EC}}, C_{\mathsf{AE}})$ and $C_B = B_{\mathsf{EC}}$, CDec computes $K_{\mathsf{EC}} \leftarrow \mathsf{D}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$ and if $K_{\mathsf{EC}} = \perp$ returns $\perp$, since this clearly indicates that C_{AE} is invalid. The recovered encryptment key K_{EC} is in turn used to recover the message via $M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$. If $M = \perp$, the scheme returns $\perp$; otherwise, it returns (M, K_{EC}) as the message / opening pair. CVer simply applies the verification algorithm EVer of the underlying encryptment scheme to the input tuple and returns the result.

Notice that by including the binding tag B_{EC} as the header in the authenticated encryption, this ensures the integrity of B_{EC}. If we did not authenticate B_{EC}, then an attacker could trivially break the MO-CTXT-security of the scheme by using an Enc query to obtain ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ for a pair (H, M) and submitting that ciphertext to Dec to recover the opening key K_{EC}. With this, $\mathcal{A}$ can easily create a valid forgery by computing $(C'_{\mathsf{EC}}, B'_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H', M')$ for some distinct header / message pair and outputting $((C'_{\mathsf{EC}}, C_{\mathsf{AE}}), B'_{\mathsf{EC}})$. Including the binding tag as the header in the AEAD ciphertext means that an attacker trying to replicate the above mix-and-match attack must create a forgery for an encryptment binding tag and key already returned as the result of an Enc query, thus violating the SCU security of the underlying encryptment scheme.

Security of the transform. We now analyse the security of the ccAEAD scheme $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$. We note that the ciphertext length regularity function $\mathsf{clen}_{\mathsf{CE}} : \mathbb{N} \to \mathbb{N}$ for $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ is defined to be that which on input $|M| \in \mathbb{N}$ outputs $\mathsf{clen}_{\mathsf{EC}}(|M|) + \mathsf{clen}_{\mathsf{AE}}(\kappa)$, where $\mathsf{clen}_{\mathsf{EC}}$ and $\mathsf{clen}_{\mathsf{AE}}$ are the length functions associated to EC and AEAD respectively and κ denotes the length of encryptment keys returned by EKg. Binding tags produced by $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ are of the same length btlen as the tags produced by EC.

MO-ROR **security.** We begin by analysing the confidentiality of $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$. The proof of the following theorem first uses a reduction to the ROR security of AEAD to argue that we can replace the encryptions C_{AE} generated in Enc and ChalEnc in MO-REAL with random bit strings of length $\mathsf{clen}_{\mathsf{AE}}(\kappa)$. We then use a hybrid argument and reductions to the otROR security of EC to argue that the encryptments generated in ChalEnc can be replaced with random strings also. A final reduction to the ROR security of AEAD

```
proc. main // $G_0$
$K \leftarrow\$ \mathsf{Kg}$
$b' \leftarrow\$ \mathcal{A}^{\text{Enc,Dec,ChalEnc}}$
Return $b'$

proc. Enc$(H, M)$ // $G_0$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\$ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$// $G_0$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\bot$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \bot$ then return $\bot$
$(M, K_{\mathsf{EC}}) \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $(M, K_{\mathsf{EC}}) = \bot$ then return $\bot$
Return $(M, K_{\mathsf{EC}})$

proc. ChalEnc$(H, M)$// $G_0$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\$ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$
```

```
proc. main // $G_1$, $\boxed{G_2}$
$K \leftarrow\$ \mathsf{Kg}$
$b' \leftarrow\$ \mathcal{A}^{\text{Enc,Dec,ChalEnc}}$
Return $b'$

proc. Enc$(H, M)$// $G_1$, $\boxed{G_2}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\$ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\boxed{C_{\mathsf{AE}} \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{AE}}(\kappa)}}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_1$, $\boxed{G_2}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\bot$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$(M, K_{\mathsf{EC}}) \leftarrow D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$
Return $(M, K_{\mathsf{EC}})$

proc. ChalEnc$(H, M)$ // $G_1$, $\boxed{G_2}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\$ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\boxed{C_{\mathsf{AE}} \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{AE}}(\kappa)}}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$
```

```
proc. main // $G_3$, $\boxed{G_4}$
$K \leftarrow\$ \mathsf{Kg}$
$b' \leftarrow\$ \mathcal{A}^{\text{Enc,Dec,ChalEnc}}$
Return $b'$

proc. Enc$(H, M)$// $G_3$, $\boxed{G_4}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{AE}}(\kappa)}$
$\boxed{C_{\mathsf{AE}} \leftarrow\$ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})}$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$// $G_3$, $\boxed{G_4}$
If $(H, C, C_B) \notin \mathcal{Y}$ then
    Return $\bot$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$(M, K_{\mathsf{EC}}) \leftarrow D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$
Return $(M, K_{\mathsf{EC}})$

proc. ChalEnc$(H, M)$ // $G_3$, $\boxed{G_4}$
$K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$
$C_{\mathsf{EC}} \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{EC}}(|M|)}$
$B_{\mathsf{EC}} \leftarrow\$ \{0,1\}^{\mathsf{btlen}}$
$C_{\mathsf{AE}} \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{AE}}(\kappa)}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$
```

Figure 3.23: Games for proof of Theorem 3.7.

allows us to revert Enc to producing real AEAD ciphertexts and reach a game equivalent to MO-RAND.

Theorem 3.7. *Let* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ *be an encryptment scheme with key space* $\mathcal{K}_{\mathsf{EC}} = \{0,1\}^{\kappa}$ *and binding tag space* $\mathcal{T}_{\mathsf{EC}}$. *Let* $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be an AEAD scheme scheme such that* $\mathcal{K}_{\mathsf{EC}} \subseteq \mathcal{M}$ *and* $\mathcal{T}_{\mathsf{EC}} \subseteq \mathcal{H}$, *where* $\mathcal{H}$ *and* $\mathcal{M}$ *denote the header and message space of* AEAD *respectively. Let* CE[EC, AEAD] *be the ccAEAD scheme presented in Figure 3.22. Then for any adversary* $\mathcal{A}$ *in the* MO-ROR *game against* CE, *there exists adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\text{ror}}_{\mathsf{AEAD}}(\mathcal{B}_1, q_c + q_e) + \mathbf{Adv}^{\text{ror}}_{\mathsf{AEAD}}(\mathcal{B}_2, q_e) + q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}) \ .$$

Adversaries $\mathcal{B}$ *and* $\mathcal{C}$ *run in the same time as* $\mathcal{A}$ *with an* $\mathcal{O}(q_c + q_e + q_d)$ *overhead.*

Proof. Let $\mathcal{A}$ be an attacker in game MO-ROR against CE[EC, AEAD]. We argue by a series of game hops, shown in Figure 3.23. We begin by defining game G_0, which is identical to game MO-REAL against CE[EC, AEAD].

Next we define game G_1, which is identical to G_0 except decryption by oracle Dec is performed via table look-up. In more detail, we begin with an array D initially set to $\bot$.

When Enc computes a ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ under key K_{EC} in response to some query (H, M), it sets $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] = (M, K_{\mathsf{EC}})$. For subsequent decryption queries of the form $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$, Dec now returns the pair (M, K_{EC}) stored at entry $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$. We need not maintain a look up table for ciphertexts generated by oracle ChalEnc since Dec only returns decryptions of ciphertexts generated via oracle Enc. It is easy to see that this is a purely syntactic change and the two games are functionally equivalent, and so

$$\Pr[G_0 \Rightarrow 1] = \Pr[G_1 \Rightarrow 1] .$$

Next we define game G_2, which is identical to game G_1 except all ciphertexts encrypted under the AEAD scheme $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ are replaced with random ciphertexts of appropriate length. We claim that there exists an adversary $\mathcal{B}_1$ in the ROR-security game against AEAD such that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_1, q_c + q_e) .$$

Adversary $\mathcal{B}_1$ proceeds as follows. $\mathcal{B}_1$ runs $\mathcal{A}$ as a subroutine. In response to Enc and ChalEnc queries on input (H, M), $\mathcal{B}_1$ generates $K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$, computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$ and then queries $(B_{\mathsf{EC}}, K_{\mathsf{EC}})$ to his challenge encryption oracle, receiving C_{AE} in return. $\mathcal{B}_1$ returns ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ to $\mathcal{A}$, and for Enc queries additionally sets $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}] = (M, K_{\mathsf{EC}})$. To simulate the Dec oracle, $\mathcal{B}_1$ returns $\perp$ if the header / ciphertext pair queried was not the result of a previous query to Enc and returns the pair (M, K_{EC}) stored at $D[H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}]$ otherwise. At the end of the game $\mathcal{B}_1$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}_1$'s encryption oracle is returning real ciphertexts as in game REAL against AEAD then this perfectly simulates game G_1, and if the oracle is returning random bit strings as in game RAND then this perfectly simulates game G_2. It follows that

$$\begin{aligned}|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| &= \left|\Pr\left[\mathrm{REAL}^{\mathcal{B}_1}_{\mathsf{AEAD}} \Rightarrow 1\right] - \Pr\left[\mathrm{RAND}^{\mathcal{B}_1}_{\mathsf{AEAD}} \Rightarrow 1\right]\right| \\ &= \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_1, q_c + q_e) ,\end{aligned}$$

where $\mathcal{B}_1$'s query budget follows from noting that $\mathcal{B}_1$ makes one query to his RoR oracle for each of $\mathcal{A}$'s $q_c + q_e$ Enc and ChalEnc queries.

Next we define game G_3, which is identical to G_2 except that during ChalEnc queries the ciphertext / binding tag pairs produced by the encryptment scheme EC are replaced with random bit strings of appropriate length. We claim that there exists an adversary $\mathcal{C}$ in the otROR game against EC such that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}) .$$

To see this, we define a series of hybrid games $H_0, \ldots, H_{q_c}$ where H_0 is identical to game G_2, H_{q_c} is identical to game G_3, and H_i is identical to game H_{i-1} except during the i^{th} ChalEnc query the output of EEnc is replaced with random bit strings. A standard hybrid argument implies that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \sum_{i=0}^{q_c-1} |\Pr[H_i \Rightarrow 1] - \Pr[H_{i+1} \Rightarrow 1] . \quad (3.5)$$

We now bound the gap between these game. Fix an index $i \in [0, q_c - 1]$ and let $\mathcal{C}_i$ be an attacker in the otROR game against EC. $\mathcal{C}_i$ runs $\mathcal{A}$ as a subroutine, simulating his oracles as follows. For Enc / Dec queries, $\mathcal{A}$ simulates the oracles by executing the pseudocode descriptions in game G_2 (or equivalently G_3). In response to ChalEnc queries, attacker $\mathcal{C}_i$ responds to the first i queries by choosing random $C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}$ of appropriate length and returning $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. For the $(i+1)^{\text{th}}$ query, $\mathcal{C}_i$ queries his own encryption oracle on the query pair (H, M) to receive back $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and returns $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$ for random C_{AE}. For the remaining ChalEnc queries, $\mathcal{C}_i$ generates ciphertexts as per the pseudocode description of the oracle in game G_2. At the end of the game, adversary $\mathcal{C}_i$ outputs whatever bit $\mathcal{A}$ does.

Notice that if $\mathcal{C}_i$'s encryption oracle returns real ciphertexts as in game otROR0 this perfectly simulates game H_i, otherwise it perfectly simulates game H_{i+1}. It follows that

$$\begin{aligned} |\Pr[H_i \Rightarrow 1] - \Pr[H_{i+1} \Rightarrow 1]| &= \left|\Pr\left[\text{otROR0}^{\mathcal{C}_i}_{\mathsf{EC}} \Rightarrow 1\right] - \Pr\left[\text{otROR1}^{\mathcal{C}_i}_{\mathsf{EC}} \Rightarrow 1\right]\right| \\ &= \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}_i) . \end{aligned}$$

Substituting this into equation (3.5), and defining $\mathcal{C}$ to be the attacker who chooses $i \leftarrow\$ [0, q_c - 1]$ and runs attacker $\mathcal{C}_i$, proves the claim.

Notice that in game G_3, oracle ChalEnc always returns a random ciphertext / binding tag pair of appropriate length in response to queries. Next we define game G_4 to be the same as G_3, except we revert oracle Enc to generate ciphertexts C_{AE} by running E rather than by choosing random ciphertexts. A reduction to the ROR-security of AEAD analogous to that described above, and noting that $\mathcal{A}$ makes q_e Enc queries, implies that there exists an adversary $\mathcal{B}_2$ in game ROR against AEAD such that

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_2, q_e) .$$

In particular, notice that oracle Enc always returns real ciphertexts in game G_4. We define a final transition to game G_5 (not shown) which does away with the look-up table and instead runs CDec on the relevant queries to Dec; again this is a purely syntactic change, and so

$$\Pr[G_4 \Rightarrow 1] = \Pr[G_5 \Rightarrow 1] .$$

Moreover, it is straightforward to verify that game G_5 is identical to game MO-RAND. Putting this all together via a standard argument, it follows that there exists adversaries $\mathcal{B}_1, \mathcal{B}_2$, $\mathcal{C}$ with the claimed run times and query budgets, such that

$$\mathbf{Adv}^{\mathrm{mo\text{-}ror}}_{\mathsf{CE}}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_1, q_c + q_e) + \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{AEAD}}(\mathcal{B}_2, q_e) + q_c \cdot \mathbf{Adv}^{\mathrm{ot\text{-}ror}}_{\mathsf{EC}}(\mathcal{C}) .$$

concluding the proof. □

MO-CTXT **security.** We now analyse the integrity of $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$. The proof of the following theorem first argues that if $\mathcal{A}$ submits a fresh query $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$ to ChalDec which decrypts correctly but for which no prior Enc query with header H returned $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$, then this violates the CTXT security of AEAD. This implies that any *other* fresh query which decrypts correctly must share an underlying encryptment key K_{EC} with a response to an Enc query. We then construct a reduction showing that any such query decrypting successfully breaks the SCU security of EC. Combining these arguments using a hybrid argument over the q_e Enc queries then completes the proof.

Theorem 3.8. *Let* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ *be an encryptment scheme with key space* $\mathcal{K}_{\mathsf{EC}} = \{0,1\}^{\kappa}$ *and binding tag space* $\mathcal{T}_{\mathsf{EC}}$. *Let* $\mathsf{AEAD} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be an AEAD scheme scheme such that* $\mathcal{K}_{\mathsf{EC}} \subseteq \mathcal{M}$ *and* $\mathcal{T}_{\mathsf{EC}} \subseteq \mathcal{H}$, *where* $\mathcal{H}$ *and* $\mathcal{M}$ *denote the header and message space of* AEAD *respectively. Let* $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ *be the ccAEAD scheme built from* EC *according to Figure 3.22. Then for any adversary* $\mathcal{A}$ *in the* MO-CTXT *game against* CE*, there exists adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\mathrm{mo\text{-}ctxt}}_{\mathsf{CE}[\mathsf{EC},\mathsf{AEAD}]}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\mathrm{ctxt}}_{\mathsf{AEAD}}(\mathcal{B}, q_c + q_d, q_e) + q_e \cdot \mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{C}, 1) .$$

Adversaries $\mathcal{B}$ *and* $\mathcal{C}$ *run in the same time as* $\mathcal{A}$ *with an* $\mathcal{O}(q_c + q_e + q_d)$ *overhead.*

Proof. We argue by a series of game hops, shown in Figure 3.24. We begin by defining

```
proc. main // $G_0$:
$K \leftarrow\!\!\$\ \mathsf{Kg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc},\mathrm{Dec},\mathrm{ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_0$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{Kg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_0$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then return $\perp$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_0$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then return $\perp$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
proc. main // $G_1, \boxed{G_2}$:
$K \leftarrow\!\!\$\ \mathsf{CKg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc},\mathrm{Dec},\mathrm{ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_1, \boxed{G_2}$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{Kg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_1, \boxed{G_2}$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \perp$
    Return $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow 1$
    $\boxed{\text{Return } \perp}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$
    $\mathsf{bad}_2 \leftarrow 1$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_1, \boxed{G_2}$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow 1$
    $\boxed{\text{Return } \perp}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$
    $\mathsf{bad}_2 \leftarrow 1$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
proc. main // $G_3$:
$K \leftarrow\!\!\$\ \mathsf{CKg}$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc},\mathrm{Dec},\mathrm{ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_3$
$K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}})\}$
$D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_3$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \perp$
    Return $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow 1$
    Return $\perp$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$
    $\mathsf{bad}_2 \leftarrow 1$
    Return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_3$
$(C_{\mathsf{EC}}, C_{\mathsf{AE}}) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C_{\mathsf{EC}}, C_{\mathsf{AE}}, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
    Return $\perp$
$K_{\mathsf{EC}} \leftarrow \mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$
If $K_{\mathsf{EC}} = \perp$ then return $\perp$
If $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\perp\}$
    $\mathsf{bad}_1 \leftarrow 1$
    Return $\perp$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$
    $\mathsf{bad}_2 \leftarrow 1$
    Return $\perp$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

Figure 3.24: Games for proof of Theorem 3.8.

game G_0, which is equivalent to MO-CTXT against $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$. It follows that

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE}}(\mathcal{A}, q_c, q_e, q_d) = \Pr[\, G_0 \Rightarrow 1\,] \,.$$

Next we define game G_1, which is identical to G_0 except we maintain a look-up table D of header / ciphertext pairs which were returned by oracle Enc; subsequently, the decryption of such ciphertexts in oracles Dec and ChalDec is performed via table look-up. Entries in the table are of the form $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = (M, K_{\mathsf{EC}})$, and we write e.g., $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}]$ to denote the set

$$\{(M, K_{\mathsf{EC}}) \ : D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = (M, K_{\mathsf{EC}}) \text{ for } H \in \mathcal{H}, C_{\mathsf{EC}} \in \mathcal{C}\} \,.$$

This is a purely syntactic change. We also set a number of bad flags, but these too do not affect the outcome of the game. Moreover, we modify oracle Dec so that if the attacker $\mathcal{A}$ submits a query which decrypts correctly but which does not correspond to a previous query to Enc (and is thus not stored in the look-up table), the win flag is set to true. This change can only increase the attacker's chance of success, and so it follows that

$$\Pr[\, G_0 \Rightarrow 1\,] \le \Pr[\, G_1 \Rightarrow 1\,] \,.$$

Next we define game G_2, which is identical to G_1 except we change the way in which oracles Dec and ChalDec respond to queries. Namely now if the attacker submits a query to Dec or ChalDec of the form $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$ such that $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \bot$ and $\mathsf{Dec}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}}) \neq \bot$, it checks if $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \{\bot\}$ and if so returns $\bot$. These games run identically unless the bad_1 flag is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[\, G_1 \Rightarrow 1\,] - \Pr[\, G_2 \Rightarrow 1\,]| \le \Pr[\, \mathsf{bad}_1 = 1 \text{ in } G_1\,] \,.$$

We bound this probability with a reduction to the CTXT security of AEAD. Let $\mathcal{B}$ be an attacker in the CTXT game against AEAD who runs $\mathcal{A}$ as a subroutine as follows. To simulate oracle Enc on query (H, M), $\mathcal{B}$ generates an encryptment key $K_{\mathsf{EC}} \leftarrow\!\!\$\, \mathsf{EKg}$, computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$, queries $(B_{\mathsf{EC}}, K_{\mathsf{EC}})$ to his own encryption oracle receiving C_{AE} in return, and returns $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. $\mathcal{B}$ maintains a look-up table of queries to Enc. To simulate Dec and ChalDec queries for $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$, $\mathcal{B}$ checks if $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \bot$; if so he returns the stored (M, K_{EC}) if the query was to the Dec oracle and $\bot$ if the query was to ChalDec. If no such entry is stored, $\mathcal{B}$ submits the pair $(B_{\mathsf{EC}}, C_{\mathsf{AE}})$ to his own challenge decryption oracle. If $\bot$ is returned, he returns $\bot$ to $\mathcal{A}$; otherwise he simulates the rest of the query as per the pseudocode description. Notice that flag bad_1 being set corresponds to $\mathcal{A}$ making a decryption query $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$ for which $(B_{\mathsf{EC}}, C_{\mathsf{AE}})$ does not correspond to an encryption query made by $\mathcal{B}$ in his game, but nonetheless $\mathsf{D}(K, B_{\mathsf{EC}}, C_{\mathsf{AE}})$ does not return an error. As such, this corresponds to a winning query for $\mathcal{B}$. It follows that

$$\begin{aligned}\Pr\left[\,\mathsf{bad}_1 = 1 \text{ in } G_1\,\right] &= \Pr\left[\,\mathrm{CTXT}^{\mathcal{B}}_{\mathsf{AEAD}} \Rightarrow 1\,\right] \\ &\leq \mathbf{Adv}^{\mathrm{ctxt}}_{\mathsf{AEAD}}(\mathcal{B}, q_c + q_d, q_e)\ ,\end{aligned}$$

where $\mathcal{B}$'s query budget follows since he makes at most one decryption oracle query for each of $\mathcal{A}$'s Dec and ChalDec queries, and one encryption oracle query for each of $\mathcal{A}$'s Enc queries. Notice that in game G_2, all encryptment keys K_{EC} recovered and input to EDec during Dec and ChalDec queries correspond to keys generated in response to Enc queries. As such, for each query $(H, ((C_{\mathsf{EC}}, B_{\mathsf{EC}}), C_{\mathsf{AE}}))$ which may successfully decrypt in game G_2 there must be an entry of the form $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = (\cdot, K_{\mathsf{EC}})$. By the correctness of the encryption scheme, all table entries of that form must share the same key K_{EC} and this will be the key which is recovered during decryption.

Next we define game G_3, which is identical to G_2 except we again change the way in which Dec, ChalDec respond to queries. Now if the attacker makes a query of the form $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$ such that $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \{\perp\}$, and $\mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ does not return an error (where K_{EC} is the key underlying $(B_{\mathsf{EC}}, C_{\mathsf{AE}})$) but for which $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{EC}}] = \perp$, we return $\perp$. Notice that this restriction makes the game impossible to win, since Dec and ChalDec will reject any ciphertext not previously returned by Enc. As such it follows that

$$\Pr\left[\,G_3 \Rightarrow 1\,\right] = 0\ .$$

These two games run identically unless the flag bad_2 is set, and so the Fundamental Lemma of Game Playing implies that

$$\left|\Pr\left[\,G_2 \Rightarrow 1\,\right] - \Pr\left[\,G_3 \Rightarrow 1\,\right]\right| \leq \Pr\left[\,\mathsf{bad}_2 = 1 \text{ in } G_2\,\right]\ .$$

We now bound this probability with a reduction to the SCU security of EC.

Suppose that attacker $\mathcal{A}$ makes q_e Enc queries, and let $\mathcal{C}$ be an attacker in the SCU game against EC who proceeds as follows. $\mathcal{C}$ chooses a key $K \leftarrow\!\!\$\ \mathsf{CKg}$, an index $i \leftarrow\!\!\$\ [1, q_e]$, and runs $\mathcal{A}$ as a subroutine. $\mathcal{C}$ simulates all apart from the i^{th} Enc query by choosing a key $K_{\mathsf{EC}} \leftarrow\!\!\$\ \mathsf{EKg}$, computing $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$ on the given input, setting $C_{\mathsf{AE}} \leftarrow\!\!\$\ \mathsf{E}(K, B_{\mathsf{EC}}, K_{\mathsf{EC}})$, and returning ciphertext $((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}})$. For the i^{th} query, $\mathcal{C}$ submits $\mathcal{A}$'s query (H^*, M^*) to his encryptment oracle, receiving $((C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}}), K^*_{\mathsf{EC}})$

in return. $\mathcal{C}$ computes $C^*_{\mathsf{AE}} \leftarrow\!\!\$\; \mathsf{E}(K, B^*_{\mathsf{EC}}, K^*_{\mathsf{EC}})$ and returns $((C^*_{\mathsf{EC}}, C^*_{\mathsf{AE}}), B^*_{\mathsf{EC}})$ to $\mathcal{A}$. $\mathcal{C}$ maintains a look-up table of the ciphertexts generated in response to Enc queries as described previously.

$\mathcal{C}$ simulates oracles Dec and ChalDec for queries $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$ by checking if there is an entry in the look-up table of the form $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = (M, K_{\mathsf{EC}})$. If not he returns $\perp$, but if so he computes $\mathcal{M} \cup \{\perp\} \ni X \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and returns the output to $\mathcal{A}$. If $\mathcal{A}$ ever makes a decryption query $(H, ((C_{\mathsf{EC}}, C_{\mathsf{AE}}), B_{\mathsf{EC}}))$ such that $D[\cdot, \cdot, B_{\mathsf{EC}}, C_{\mathsf{AE}}] \neq \{\perp\}$ and which decrypts correctly, but for which $D[H, C_{\mathsf{EC}}, B_{\mathsf{EC}}, C_{\mathsf{AE}}] = \perp$, the bad_2 flag will be set. If additionally such a query has $(B_{\mathsf{EC}}, C_{\mathsf{AE}}) = (B^*_{\mathsf{EC}}, C^*_{\mathsf{AE}})$ and so corresponds to the i^{th} Enc query in which $\mathcal{C}$ inserted his challenge, then $\mathcal{C}$ submits the tuple (H, C_{EC}) to his challenge decryption oracle.

If such an event occurs, then by definition $\mathsf{EDec}(K^*_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ does not return $\perp$, where K^*_{EC} is the key associated to $(B^*_{\mathsf{EC}}, C^*_{\mathsf{AE}})$ and that which was used in $\mathcal{C}$'s challenge. At the same time, we know that $D[H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}}, C^*_{\mathsf{AE}}] = \perp$, so $(H, C_{\mathsf{EC}}) \neq (H^*, C^*_{\mathsf{EC}})$. Therefore, this constitutes a winning query for $\mathcal{C}$ in the SCU game.

Let $(B^1_{\mathsf{EC}}, C^1_{\mathsf{AE}}), \ldots, (B^{q_e}_{\mathsf{EC}}, C^{q_e}_{\mathsf{AE}})$ denote the set of ciphertexts returned in response to the q_e Enc queries made by $\mathcal{A}$, and $\mathsf{bad}(B_{\mathsf{EC}}, C_{\mathsf{AE}})$ denote the ciphertext in the query which resulted in bad_2 being set (by a previous transition, the only ciphertexts which could cause bad_2 to be set must correspond to an Enc query). We write $\mathcal{C}^j$ for the attacker who inserts his challenge at the j^{th} Enc query. Then taking a union bound, it follows that

$$\begin{aligned}
\Pr\left[\, \mathsf{bad}_2 = 1 \text{ in } G_2 \,\right] &\leq \sum_{j=1}^{q_e} \Pr\left[\, \mathsf{bad}_2 = 1 \text{ in } G_2 \wedge \mathsf{bad}(B_{\mathsf{EC}}, C_{\mathsf{AE}}) = (B^j_{\mathsf{EC}}, C^j_{\mathsf{AE}}) \,\right] \\
&\leq \sum_{j=1}^{q_e} \Pr\left[\, \mathrm{SCU}^{\mathcal{C}^j}_{\mathsf{EC}} \Rightarrow 1 \,\right] \\
&= q_e \cdot \mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{C}, 1) \;,
\end{aligned}$$

where $\mathcal{C}$'s query budget follows since he makes at most one ChalDec query. Putting this altogether, it follows that

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE[EC,AEAD]}}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\mathrm{ctxt}}_{\mathsf{AEAD}}(\mathcal{B}, q_c + q_d, q_e) + q_e \cdot \mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{C}, 1) \;,$$

which concludes the proof. □

Binding security. It is straightforward to verify that $\mathsf{CE}[\mathsf{EC}, \mathsf{AEAD}]$ inherits s-BIND and sr-BIND security directly from EC, and so we have shown that the transform yields a secure ccAEAD scheme. Moreover, by reframing CE as a regular AEAD scheme as described in Section 3.7.1, our transform yields a ROR and CTXT secure single-pass AEAD scheme. To implement the transform the underlying AEAD scheme must be instantiated. One can use, for example, OCB.

3.7.4 A Transform From Encryptment to ccAEAD via a Compression Function

In this section, we present an alternative approach for building ccAEAD from encryptment, which lifts an encryptment scheme to fully-fledged ccAEAD at the cost of just two extra compression function calls. The scheme uses an RKA-PRF, most simply re-using the compression function f used in HFC. This transform is also conceptually elegant as it may be built from a single cryptographic primitive.

Overview. Consider the scheme $\mathsf{CE}[\mathsf{EC}, \mathsf{f}] = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ shown in Figure 3.25, built from an encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ and a compression function $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$. Key generation simply returns a random d-bit key $K \leftarrow_{\$} \{0,1\}^d$. On input (K, H, M), the encryption algorithm CEnc chooses a random nonce $N \leftarrow_{\$} \{0,1\}^n$ and uses this to derive a one-time encryptment key via $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$. This key is then used to encrypt the header and message via $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$. The algorithm then uses the long-lived key to compute an authentication tag over the binding tag, $T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$, and outputs $((C_{\mathsf{EC}}, T_B), B_{\mathsf{EC}})$. Notice how the domains of the compression function calls used to compute the one-time keys and the tags are domain-separated; looking ahead, this shall be utilised in the proof of MO-CTXT security. Decryption with CDec first verifies the authentication tag, returning an error if this step fails. The algorithm then uses N to re-derive K_{EC} and decrypts $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ to return M, finally returning (K_{EC}, M) if all steps succeed. Verification with CVer simply verifies the binding tag using the verification algorithm of the underlying encryptment scheme.

We now analyse the security of the ccAEAD scheme $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ resulting from the compression function transform.

```
CKg
  $K \leftarrow_{\$} \{0,1\}^d$
  Return $K$

CEnc$(K, H, M)$
  $N \leftarrow_{\$} \{0,1\}^n$
  $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
  $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
  $T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
  Return $((N, C_{\mathsf{EC}}, T_B), B_{\mathsf{EC}})$

CDec$(K, H, (C, C_B))$
  $(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
  $T'_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
  If $T_B \neq T'_B$ then return $\perp$
  $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
  $M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
  If $M = \perp$ then return $\perp$
  Return $(M, K_{\mathsf{EC}})$

CVer$(H, M, K_f, C_B)$
  $b \leftarrow \mathsf{EVer}(H, M, K_f, C_B)$
  Return $b$
```

Figure 3.25: A transform $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ for building a ccAEAD scheme from an encryptment scheme EC and a compression function $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$. The strings fpad and spad are fixed and distinct.

MO-ROR **security.** We will begin by analysing the confidentiality of $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$. The proof of the following theorem first uses a reduction to the RKA-PRF security of f to argue that we can replace compression function calls in game MO-REAL with random function queries. With this in place, we remove collisions in the nonces sampled in Enc and ChalEnc which then allows us to replace all encryptment keys generated by these oracles with independent random strings. This then allows a reduction to the otROR security of EC via which we can replace all encryptments generated in ChalEnc with random strings; modulo accidental collisions amongst the binding tags (which are accounted for in the bound), we may then replace the corresponding authentication tags T_B with random strings also. After a final few games hops to revert to sampling binding tags and nonces at random, and to return to using f in Enc, we reach a game equivalent to MO-RAND. We note that for game MO-RAND, it is straightforward to verify that the associated ciphertext length function $\mathsf{clen}_{\mathsf{CE}}$ for $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ is that which, on input $|M| \in \mathbb{N}$, outputs $\mathsf{clen}_{\mathsf{EC}}(|M|) + 2n$. Binding tags produced by $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ are of the same length as the tags produced by EC, $\mathsf{blen} = \mathsf{btlen}$.

Theorem 3.9. *Let* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ *be an encryptment scheme with binding tag space* $\mathcal{T}_{\mathsf{EC}} \subseteq \{0,1\}^n$ *and such that key generation via* EKg *is equivalent to choosing* $K_{\mathsf{EC}} \leftarrow_{\$} \{0,1\}^n$*. Let* $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ *be a compression function, and let* fpad *and* spad *be fixed and distinct d-bit strings. Let* $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ *be the ccAEAD scheme built from* EC *and* f *according to Figure 3.25. Then for any adversary* $\mathcal{A}$ *in game* MO-ROR *against* CE*, there exists adversaries* $\mathcal{B}_1, \mathcal{B}_2$*, and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}_1, 2 \cdot (q_e + q_c)) + \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}_2, 2q_e) + q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}) + \frac{(q_c + q_e)^2}{2^n} + \frac{8q_e \cdot q_c + 3q_c \cdot (q_c - 1)}{2^{\mathsf{blen}+1}} .$$

Adversaries $\mathcal{B}_1, \mathcal{B}_2$*, and* $\mathcal{C}$ *run in the same time as* $\mathcal{A}$ *plus an* $\mathcal{O}(q_c + q_e + q_d)$ *overhead.*

Proof. We argue by a series of game hops, shown in Figures 3.26 and 3.27. For brevity, when transitioning between games we occasionally add flags and / or remove redundant code without stating this in the following proof; it is straightforward to verify that all such modifications do not alter the distribution of the game in question. We begin by defining game G_0 (not shown), which is equivalent to game MO-REAL against $\mathsf{CE}[\mathsf{EC}, \mathsf{f}]$ with the exception that decryption in oracle Dec is performed by table look-up. As discussed in the analogous argument of Theorem 3.7, performing decryption by table look-up does not affect the distribution of the game and so this game is equivalent to game MO-REAL.

Next we define game G_1, which is identical to G_0 except: **(1)** we add a flag in Enc and ChalEnc; and **(2)** replace $\mathsf{f}(\cdot, K \oplus Y)$ with a lazily sampled random function F for each $Y \in \{\mathsf{fpad}, \mathsf{spad}\}$. It is straightforward to verify that **(1)** does not affect the outcome of the game. For **(2)**, we claim that there exists an attacker $\mathcal{B}_1$ in game RKA-PRF against f, running in the same time as $\mathcal{A}$ plus an $\mathcal{O}(q_c + q_e + q_d)$ overhead, such that

$$|\Pr[\, G_1 \Rightarrow 1\,] - \Pr[\, G_0 \Rightarrow 1\,]| \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}_1, 2 \cdot (q_e + q_c)) \, .$$

$\mathcal{B}_1$ runs $\mathcal{A}$ as a subroutine, simulating $\mathcal{A}$'s oracle following the pseudocode description in G_0. In particular for each function call $\mathsf{f}(X, K \oplus Y)$, $\mathcal{B}_1$ queries (X, Y) to his real-or-random function oracle, and uses the response to run the remainder of the game. At the end of the game, $\mathcal{B}_1$ outputs whatever bit $\mathcal{A}$ does. It is straightforward to verify that if $\mathcal{B}_1$ receives real outputs in his challenge then this perfectly simulates G_0; otherwise it perfectly simulates G_1. Noting that each query to Enc and ChalEnc induces two f calls then implies the claim.

Next we define game G_2, which is identical to G_1 except we modify Enc and ChalEnc to sample nonces without replacement. These games run identically until the flag bad_1 is set. Since $(q_c + q_e)$ nonces are sampled in both games, invoking the Fundamental Lemma of Game Playing and a birthday bound implies that

$$|\Pr[\, G_2 \Rightarrow 1\,] - \Pr[\, G_1 \Rightarrow 1\,]| \leq \Pr[\, \mathsf{bad}_1 = \mathsf{true} \text{ in } G_2\,] \leq \frac{(q_c + q_e)^2}{2^{n+1}} \, .$$

Next we define game G_3, which is identical to G_2 except instead of deriving encryptment keys via $K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$ in Enc and ChalEnc, we instead sample these uniformly at random $K_{\mathsf{EC}} \leftarrow\!\!\$\, \{0,1\}^n$. Since in G_2 these keys are derived by applying a random function to a point which is queried nowhere else in the game, these games are identically distributed and so

$$\Pr[\, G_3 \Rightarrow 1\,] = \Pr[\, G_2 \Rightarrow 1\,] \, .$$

Next we define game G_4, which is identical to G_3 except in ChalEnc we replace each encryptment output $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$ with random bits strings of appropriate length. A straightforward reduction to the otROR security of EC, using a hybrid argument over the q_c ChalEnc queries and a series of attackers $\mathcal{C}_i$ for $i \in [1, q_c]$, each of whom simulates ChalEnc for $\mathcal{A}$ by choosing random $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ for the first $i-1$ queries, and inserting their challenge in the i^{th} query, implies that there exists an adversary $\mathcal{C}$ in game otROR against EC with the claimed run time, such that

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq q_c \cdot \mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC}}(\mathcal{C}) .$$

Next we define game G_5, which is identical to G_4 except we modify ChalEnc such that if one of the (now randomly sampled) binding tags collides with a binding tag previously generated in Enc or randomly sampled in ChalEnc (indicated by the sets $\mathcal{Q}_E$ and $\mathcal{Q}_C$ respectively), then the binding tag is resampled such that this is not the case. These games run identically until the flag bad_2 is set. Since q_c binding tags are sampled in ChalEnc, and at the point of sampling the i^{th} such tag there are at most $q_e + (i-1)$ points in the set $\mathcal{Q}_E \cup \mathcal{Q}_C$ with which the binding tag may collide, it follows that

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_2 = \mathsf{true} \text{ in } G_5] \leq \frac{2q_e \cdot q_c + q_c \cdot (q_c - 1)}{2^{\mathsf{blen}+1}} .$$

Next we define game G_6, which is identical to G_5 except we modify ChalEnc to sample authentication tags uniformly at random $T_B \leftarrow\!\!\$\ \{0,1\}^n$ rather than computing these via $T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$. Consider the set of all authentication tags computed in Enc and ChalEnc during the course of G_5. Due to the previous transition, the binding tags sampled in ChalEnc cannot collide with any previously sampled binding tag. As such, each query $\mathsf{f}(B_{\mathsf{EC}}, \mathsf{spad})$ will be on a previously unqueried point. Moreover, that point will be queried nowhere else in the game (and so the distribution of these tags is precisely that of G_6) *unless* a subsequent Enc query also produces binding tag B_{EC}, an event we indicate with the flag bad_3. Both games run identically until the flag bad_3 is set; as such, invoking the Fundamental Lemma of Game Playing implies that

$$\begin{aligned}
|\Pr[G_6 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| &\leq \Pr[\mathsf{bad}_3 = \mathsf{true} \text{ in } G_5] \\
&\leq \Pr[\mathsf{bad}_3 = \mathsf{true} \text{ in } G_4] + \frac{2q_e \cdot q_c + q_c \cdot (q_c - 1)}{2^{\mathsf{blen}+1}} \\
&\leq \frac{4q_e \cdot q_c + q_c \cdot (q_c - 1)}{2^{\mathsf{blen}+1}} .
\end{aligned}$$

The second inequality follows from a previous bound on the gap between G_4 and G_5. The final inequality follows since for each Enc query in G_4 there are at most q_c randomly sampled points in the set $\mathcal{Q}_c$, and so the probability of bad_3 being set in this game is upper bounded by $\frac{q_e \cdot q_c}{2^{\mathsf{blen}}}$

Next we define game G_7 to be identical to G_6, except we revert to sampling binding tags uniformly at random in ChalEnc. These games run identically until the flag bad_2 is set, and so an entirely analogous argument to that made previously implies that

$$|\Pr[\,G_7 \Rightarrow 1\,] - \Pr[\,G_6 \Rightarrow 1\,]| \leq \frac{2q_e \cdot q_c + q_c \cdot (q_c - 1)}{2^{\mathsf{blen}+1}} \;.$$

In game G_8, we revert to generating encryptment keys via the random function in Enc; by an analogous argument to that made previously, this is a purely syntactic change, and

$$\Pr[\,G_8 \Rightarrow 1\,] = \Pr[\,G_7 \Rightarrow 1\,] \;.$$

In game G_9, we revert to sampling nonces with replacement. Again these games run identically until the flag bad_1 is set; it follows that

$$|\Pr[\,G_9 \Rightarrow 1\,] - \Pr[\,G_8 \Rightarrow 1\,]| \leq \frac{(q_c + q_e)^2}{2^{n+1}} \;.$$

Finally, we define game G_{10} to be identical to G_9, except we replace the random function F with the compression function f. An analogous argument to that made previously, noting that in these games no function queries are made in ChalEnc and $2q_e$ function queries are made in Enc, implies that there exists an attacker $\mathcal{B}_2$ in game RKA-PRF against f such that

$$|\Pr[\,G_9 \Rightarrow 1\,] - \Pr[\,G_8 \Rightarrow 1\,]| \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}_1, 2q_e) \;.$$

Moreover, G_{10} is identical to game MO-RAND against $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$ (modulo decryption performed by table look-up, which as mentioned above does not alter the distribution of the game). As such, combining the above via a standard argument implies that there exists attacker's $\mathcal{B}_1, \mathcal{B}_2$, and $\mathcal{C}$ with the claimed run times, such that

$$\begin{aligned}\mathbf{Adv}_{\mathsf{CE}}^{\text{mo-ror}}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}_1, 2\cdot(q_e + q_c)) + \mathbf{Adv}_{\mathsf{f}}^{\oplus\text{-prf}}(\mathcal{B}_2, 2q_e)\\ + q_c \cdot \mathbf{Adv}_{\mathsf{EC}}^{\text{ot-ror}}(\mathcal{C}) + \frac{(q_c + q_e)^2}{2^n} + \frac{8q_e \cdot q_c + 3q_c \cdot (q_c - 1)}{2^{\mathsf{blen}+1}} \;.\end{aligned}$$

□

MO-CTXT **security.** Next, we analyse the MO-CTXT security of $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$. As in the proof of Theorem 3.9, we first argue that we can replace compression function calls with

```
proc. main // $G_0$
$K \leftarrow\$ \{0,1\}^d$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $b'$

proc. Enc$(H, M)$ // $G_0$
$N \leftarrow\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_0$
If $(H, C, C_B) \notin \mathcal{Y}$
  Return $\perp$
Return $D[H, C, B_{\mathsf{EC}}]$

proc. ChalEnc$(H, M)$ // $G_0$
$N \leftarrow\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
Return $(C, B_{\mathsf{EC}})$
```

```
proc. main // $G_1$, $\boxed{G_2}$
$K \leftarrow\$ \{0,1\}^d$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $b'$

proc. Enc$(H, M)$ // $G_1$, $\boxed{G_2}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
  $\mathsf{bad}_1 \leftarrow 1$
  $\boxed{N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1}$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_1$, $\boxed{G_2}$
If $(H, C, C_B) \notin \mathcal{Y}$
  Return $\perp$
Return $D[H, C, B_{\mathsf{EC}}]$

proc. ChalEnc$(H, M)$ // $G_1$, $\boxed{G_2}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
  $\mathsf{bad}_1 \leftarrow 1$
  $\boxed{N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1}$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
Return $(C, B_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_1$, $\boxed{G_2}$
If $F[X, Y] \neq \perp$
  $F[X, Y] \leftarrow\$ \{0,1\}^n$
Return $F[X, Y]$
```

```
proc. main // $G_3$, $\boxed{G_4}$
$K \leftarrow\$ \{0,1\}^d$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $b'$

proc. Enc$(H, M)$ // $G_3$, $\boxed{G_4}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
  $\mathsf{bad}_1 \leftarrow 1$
  $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow\$ \{0,1\}^n$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
If $B_{\mathsf{EC}} \in \mathcal{Q}_C$ then $\mathsf{bad}_3 \leftarrow \mathsf{true}$
$\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup \{B_{\mathsf{EC}}\}$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_3$, $\boxed{G_4}$
If $(H, C, C_B) \notin \mathcal{Y}$
  Return $\perp$
Return $D[H, C, B_{\mathsf{EC}}]$

proc. ChalEnc$(H, M)$ // $G_3$, $\boxed{G_4}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$ then
  $\mathsf{bad}_1 \leftarrow 1$
  $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow\$ \{0,1\}^n$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$\boxed{(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{EC}}(|M|) \times \{0,1\}^{\mathsf{btlen}}}}$
If $B_{\mathsf{EC}} \in \mathcal{Q}_E \cup \mathcal{Q}_C$
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathcal{Q}_C \leftarrow \mathcal{Q}_C \cup \{B_{\mathsf{EC}}\}$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
Return $(C, B_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_3$, $\boxed{G_4}$
If $F[X, Y] \neq \perp$ then
  $F[X, Y] \leftarrow\$ \{0,1\}^n$
Return $F[X, Y]$
```

Figure 3.26: Games for the proof of Theorem 3.9.

random function queries via a reduction to the RKA-PRF security of f. We then modify the sampling of nonces so that each encryptment key corresponding to an Enc query or a decryption oracle query with a previously unseen nonce can be replaced with a random string. This then allows a reduction to the SCU security of EC to argue that the attacker is unlikely to win with a decryption query for which the encryptment key and binding tag correspond to a previous Enc query. The proof then argues that to win, the attacker must either correctly guess the value of a previously unseen authentication tag (which, since uniformly distributed over $\{0,1\}^n$ in the modified game, accounts for the $\frac{(q_c+q_d)}{2^n}$ term in the bound), or break the s-BIND or sr-BIND security of EC.

Theorem 3.10. *Let* $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ *be an encryptment scheme with binding tag space* $\mathcal{T}_{\mathsf{EC}} \subseteq \{0,1\}^n$ *and such that key generation via* EKg *is equivalent to choosing*

```
proc. main // $G_5$, $\boxed{G_6}$
$K \leftarrow\$ \{0,1\}^d$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $b'$

proc. Enc$(H, M)$ // $G_5$, $\boxed{G_6}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
    $\mathsf{bad}_1 \leftarrow 1$
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow\$ \{0,1\}^n$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$\boxed{\text{If } B_{\mathsf{EC}} \in \mathcal{Q}_C \text{ then } \mathsf{bad}_3 \leftarrow \mathsf{true}}$
$\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup \{B_{\mathsf{EC}}\}$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_5$, $\boxed{G_6}$
If $(H, C, C_B) \notin \mathcal{Y}$
    Return $\perp$
Return $D[H, C, B_{\mathsf{EC}}]$

proc. ChalEnc$(H, M)$ // $G_5$, $\boxed{G_6}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
    $\mathsf{bad}_1 \leftarrow 1$
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow\$ \{0,1\}^n$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{EC}}(|M|) \times \{0,1\}^{\mathsf{btlen}}}$
If $B_{\mathsf{EC}} \in \mathcal{Q}_E \cup \mathcal{Q}_C$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
    $B_{\mathsf{EC}} \leftarrow\$ \{0,1\}^{\mathsf{btlen}} \setminus \mathcal{Q}_E \cup \mathcal{Q}_C$
$\mathcal{Q}_C \leftarrow \mathcal{Q}_C \cup \{B_{\mathsf{EC}}\}$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$\boxed{T_B \leftarrow\$ \{0,1\}^n}$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
Return $(C, B_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_5$, $\boxed{G_6}$
If $F[X, Y] \neq \perp$ then
    $F[X, Y] \leftarrow\$ \{0,1\}^n$
Return $F[X, Y]$
```

```
proc. main // $G_7$, $\boxed{G_8}$
$K \leftarrow\$ \{0,1\}^d$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $b'$

proc. Enc$(H, M)$ // $G_7$, $\boxed{G_8}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
    $\mathsf{bad}_1 \leftarrow 1$
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow\$ \{0,1\}^n$
$\boxed{K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup \{B_{\mathsf{EC}}\}$
If $B_{\mathsf{EC}} \in \mathcal{Q}_C$ then $\mathsf{bad}_3 \leftarrow \mathsf{true}$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_7$, $\boxed{G_8}$
If $(H, C, C_B) \notin \mathcal{Y}$
    Return $\perp$
Return $D[H, C, B_{\mathsf{EC}}]$

proc. ChalEnc$(H, M)$ // $G_7$, $\boxed{G_8}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
    $\mathsf{bad}_1 \leftarrow 1$
    $N \leftarrow\$ \{0,1\}^n \setminus \mathcal{Q}_1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow\$ \{0,1\}^n$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{EC}}(|M|) \times \{0,1\}^{\mathsf{btlen}}}$
If $B_{\mathsf{EC}} \in \mathcal{Q}_E \cup \mathcal{Q}_C$
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
$\mathcal{Q}_C \leftarrow \mathcal{Q}_C \cup \{B_{\mathsf{EC}}\}$
$T_B \leftarrow\$ \{0,1\}^n$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
Return $(C, B_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_7$, $\boxed{G_8}$
If $F[X, Y] \neq \perp$ then
    $F[X, Y] \leftarrow\$ \{0,1\}^n$
Return $F[X, Y]$
```

```
proc. main // $G_9$, $\boxed{G_{10}}$
$K \leftarrow\$ \{0,1\}^d$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $b'$

proc. Enc$(H, M)$ // $G_9$, $\boxed{G_{10}}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
    $\mathsf{bad}_1 \leftarrow 1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$
$\boxed{K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, C, B_{\mathsf{EC}}] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_9$, $\boxed{G_{10}}$
If $(H, C, C_B) \notin \mathcal{Y}$
    Return $\perp$
Return $D[H, C, B_{\mathsf{EC}}]$

proc. ChalEnc$(H, M)$ // $G_9$, $\boxed{G_{10}}$
$N \leftarrow\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
    $\mathsf{bad}_1 \leftarrow 1$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow\$ \{0,1\}^{\mathsf{clen}_{\mathsf{EC}}(|M|) \times \{0,1\}^{\mathsf{btlen}}}$
$T_B \leftarrow\$ \{0,1\}^n$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
Return $(C, B_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_9$, $\boxed{G_{10}}$
If $F[X, Y] \neq \perp$ then
    $F[X, Y] \leftarrow\$ \{0,1\}^n$
Return $F[X, Y]$
```

Figure 3.27: Further games for the proof of Theorem 3.9.

$K_{\mathsf{EC}} \leftarrow_{\$} \{0,1\}^n$. Let $\mathsf{f} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ be a compression function, and let fpad and spad be fixed and distinct d-bit strings. Let $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$ be the ccAEAD scheme built from EC and f according to Figure 3.25. Then for any adversary $\mathcal{A}$ in the MO-CTXT *game against CE, there exists adversaries $\mathcal{B}$, $\mathcal{C}$, $\mathcal{E}$, and $\mathcal{F}$, such that*

$$\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE}[\mathsf{EC},\mathsf{f}]}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}, 2 \cdot q) + q_e \cdot \mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{C}, 1) + \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{E}) + \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{F}) + \frac{q^2 + 2(q_e+1)\cdot(q_c+q_d) + q_e \cdot (q_e-1)}{2^{n+1}} ,$$

where $q = q_c + q_e + q_d$. All attackers run in the same amount of time as $\mathcal{A}$, plus an $\mathcal{O}(q)$ overhead.

Proof. We argue by a series of game hops, shown in Figures 3.28 and 3.29. We begin by defining game G_0, which is equivalent to MO-CTXT against $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$. Next we define game G_1, which is identical to G_0 except: **(1)** we maintain a table D of header / ciphertext pairs which were returned by oracle Enc; subsequently, the decryption of such ciphertexts in oracle Dec is performed via table look-up; and **(2)** we allow $\mathcal{A}$ to win if they submit a previously unseen ciphertext to Dec that decrypts correctly. **(1)** is clearly a syntactic change and **(2)** can only increase $\mathcal{A}$'s success probability; it follows that

$$\Pr[G_0 \Rightarrow 1]| \leq \Pr[G_1 \Rightarrow 1] .$$

Next we define G_2, which is identical to G_1 except we replace compression function calls with lazily sampled random function queries; an analogous argument to that made in the proof of Theorem 3.9, and noting that each oracle makes two f calls per query, implies that there exists an attacker $\mathcal{B}$ with the claimed run time, such that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}, 2 \cdot q) ,$$

where $q = (q_c + q_e + q_d)$.

Next we define game G_3, which is identical to G_2 except we modify nonce sampling in Enc. In more detail, if a sampled nonce $N \leftarrow_{\$} \{0,1\}^n$ collides with a nonce sampled to respond to a previous Enc query or submitted as part of a Dec or ChalDec query (indicated by the set $\mathcal{Q}_1$) in G_3, then we resample the nonce such that this is not the case. These games run identically until the flag bad_1 is set. For each of the $i \in [1, q_e]$ Enc queries, there are at most $(q_c + q_d + (i-1))$ points with which the nonce sampled to respond to that query could collide; summing over each query and invoking the Fundamental Lemma of Game

Playing implies that

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \frac{2q_e \cdot (q_c + q_d) + q_e \cdot (q_e - 1)}{2^{n+1}} .$$

Next we define game G_4, which is identical to G_3 except we modify the way in which encryptment keys are generated in all oracles. Namely, in each Enc query and each Dec / ChalDec query on a previously unseen nonce, the oracle chooses $K_{\mathsf{EC}} \leftarrow\!\!\$ \{0,1\}^n$ as opposed to deriving the key via $K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$ where F denotes the lazily sampled random function. In oracles Dec and ChalDec the correct encryptment key for previously seen nonces are recovered by table look-up. Since in G_3 nonces in Enc are sampled without replacement, and by definition the modified decryption oracle queries are on a previously unseen nonce, it follows that all such keys are derived as the result of a fresh query to the random function and so both games are identically distributed. We obtain:

$$\Pr[G_4 \Rightarrow 1] = \Pr[G_3 \Rightarrow 1] .$$

Next we define game G_5, which is identical to G_4 except now Dec and ChalDec return $\perp$ in response to any query $(H, ((N, C_{\mathsf{EC}}, T_B), B_{\mathsf{EC}}))$ for which $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \{\perp\}$ regardless of whether the query decrypts correctly. These games run identically until the flag bad_2 is set, so

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_2 = 1 \text{ in } G_5] .$$

We bound this probability via a reduction to the SCU security of EC. Let $\mathcal{C}$ be an attacker in the SCU game against EC who proceeds as follows. $\mathcal{C}$ chooses a key $K \leftarrow\!\!\$ \mathsf{CKg}$, an index $i \leftarrow\!\!\$ [1, q_e]$, and runs $\mathcal{A}$ as a subroutine. $\mathcal{C}$ simulates all apart from the i^{th} Enc query by first choosing a random nonce $N \leftarrow\!\!\$ \{0,1\}^n \setminus \mathcal{Q}_1$, a random key $K_{\mathsf{EC}} \leftarrow\!\!\$ \{0,1\}^n$, and computing $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$ on the given input. $\mathcal{C}$ simulates the lazily sampled random function F by choosing random strings in response to each fresh query, and sets $T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$, finally returning ciphertext $((N, C_{\mathsf{EC}}, T_B), B_{\mathsf{EC}})$ to $\mathcal{A}$. For the i^{th} query, $\mathcal{C}$ chooses $N^* \leftarrow\!\!\$ \{0,1\}^n \setminus \mathcal{Q}_1$, submits $\mathcal{A}$'s query (H^*, M^*) to his encryptment oracle, receiving $((C^*_{\mathsf{EC}}, B^*_{\mathsf{EC}}), K^*_{\mathsf{EC}})$ in return. $\mathcal{C}$ computes $T^*_B \leftarrow F(B^*_{\mathsf{EC}}, \mathsf{spad})$ and returns $((N^*, C^*_{\mathsf{EC}}, T^*_B), B^*_{\mathsf{EC}})$ to $\mathcal{A}$. $\mathcal{C}$ maintains look-up tables of the ciphertexts generated in response to Enc queries, and the encryptment keys sampled in all oracles, as per the pseudocode description of the game.

$\mathcal{C}$ simulates oracles Dec and ChalDec for queries $(H, ((N, C_{\mathsf{EC}}, T_B), B_{\mathsf{EC}}))$ by first checking if there is an entry in the decryption look-up table of the form $D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] = (M, K_{\mathsf{EC}})$, returning (M, K_{EC}) if this is an Dec query and $\perp$ otherwise. If not, $\mathcal{C}$ verifies T_B

and returns an error if the check fails. If the nonce N corresponds to a previous query, $\mathcal{C}$ retrieves the corresponding encrytpment key K_{EC} from the key look-up table. If not, they sample a fresh encryptment key $K_{\mathsf{EC}} \leftarrow\!\!\$\, \{0,1\}^n$ and update the look-up table accordingly. Either way, $\mathcal{C}$ decrypts $M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$, again returning an error if this step fails. If the query is such that $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] \neq \{\perp\}$ then $\mathcal{A}$ returns $\perp$; otherwise they return (M, K_{EC}). Notice that if the query is such that $\perp$ is returned despite the ciphertext decrypting correctly, then the flag bad_2 will be set. If additionally such a query has $(N, B_{\mathsf{EC}}) = (N^*, B^*_{\mathsf{EC}})$, then this query corresponds to the key K^*_{EC} and binding tag B^*_{EC} produced in the i^{th} Enc query in which $\mathcal{C}$ inserted his challenge. In this case, $\mathcal{C}$ submits the tuple (H, C_{EC}) to his challenge decryption oracle. It is straightforward to verify that $\mathcal{C}$ perfectly simulates G_5 for $\mathcal{A}$, and that moreover if the above event occurs then $\mathcal{C}$ has found a new tuple $(H, C_{\mathsf{EC}}, B^*_{\mathsf{EC}})$ which decrypts correctly under K^*_{EC}, constituting a win for $\mathcal{C}$. Since there are at most q_e Enc queries in which $\mathcal{C}$ may insert their challenge, and $\mathcal{C}$ makes at most one ChalDec query, a standard argument implies that

$$\Pr\left[\,\mathsf{bad}_2 = 1 \text{ in } G_5\,\right] \leq q_e \cdot \mathbf{Adv}^{\mathrm{scu}}_{\mathsf{EC}}(\mathcal{C}, 1)\ .$$

Next we define game G_6, which is identical to G_5 except we modify oracles Dec and ChalDec such that if the attacker submits a fresh query $(H, ((N, C_{\mathsf{EC}}, T_B), B_{\mathsf{EC}})\ A)$ for which $D[\cdot, N, \cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}$ and which decrypts correctly with the modified checks, but for which $D[\cdot, \cdot, \cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}$ (indicating that the binding tag B_{EC} was not previously returned in response to an Enc query), then the oracle returns $\perp$. These games run identically until the flag bad_3 is set. We now bound the probability of this event occurring. Suppose $\mathcal{A}$ makes $q_1 \leq (q_c + q_d)$ Dec and ChalDec queries for which $D[\cdot, \cdot, \cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}$, among which there are $q_2 \leq q_1$ distinct binding tags $B^1_{\mathsf{EC}}, \ldots, B^{q_2}_{\mathsf{EC}}$. To each such binding tag B^i_{EC}, there is precisely one correct authentication tag $T^i_B = F(B^i_{\mathsf{EC}}, \mathsf{spad})$. Since by assumption $D[\cdot, \cdot, \cdot, B^i_{\mathsf{EC}}, \cdot] = \{\perp\}$, it follows that at the point of the first decryption query on B^i_{EC} the random function has not been queried on this binding tag. As such at the point of $\mathcal{A}$ making this query the value of T^i_B is uniformly distributed over $\{0,1\}^n$, and so is guessed in a single decryption query with probability $\frac{1}{2^n}$. Suppose $\mathcal{A}$ makes a_i queries on binding tag B^i_{EC}; then a union bound implies that bad_3 is set with one of these queries with probability $\frac{a_i}{2^n}$. Taking a union bound over each binding tag $B^i_{\mathsf{EC}} \in [1, q_2]$, and since $\sum_{i=1}^{q_2} a_i = q_1 \leq (q_c + q_d)$, it follows that

$$|\Pr\left[\,G_6 \Rightarrow 1\,\right] - \Pr\left[\,G_5 \Rightarrow 1\,\right] \leq \Pr\left[\,\mathsf{bad}_3 = 1 \text{ in } G_6\,\right] \leq \sum_{i=1}^{q_2} \frac{a_i}{2^n} \leq \frac{(q_c + q_d)}{2^n}\ .$$

Next we define game G_7, which is identical to G_6 now encryptment keys generated in all oracles are sampled without replacement. In particular, notice that if two oracle queries contain distinct nonces in G_7 then their associated encryptment keys will be distinct also. These games run identically until the flag bad_4 is set. Since at most $q = q_c + q_e + q_d$ encryptment keys are sampled, it is straightforward to verify that

$$|\Pr[G_7 \Rightarrow 1] - \Pr[G_8 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_4 = 1 \text{ in } G_7] \leq \frac{q^2}{2^{n+1}} .$$

Next we define game G_8 to be identical to G_7, except that now if a query is made to Dec or ChalDec which decrypts correctly with the added checks, but for which the binding tag fails to verify, then the oracle returns $\perp$. It is straightforward to verify that both games can be perfectly simulated by an attacker $\mathcal{E}$ in game s-BIND against EC, and that any such query corresponds to a win in that game; it follows that

$$|\Pr[G_8 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{E}) .$$

Now in game G_8, any previously unseen query $(H^*, ((N^*, C^*_{\mathsf{EC}}, T^*_B), B^*_{\mathsf{EC}}))$ which causes win to be set in Dec or ChalDec must: **(1)** decrypt correctly; **(2)** verify correctly under the associated key K^*_{EC}; and **(3)** be such that B^*_{EC} was returned in response to a previous Enc query, but N^* was not returned by any Enc query. We claim that there exists an attacker $\mathcal{F}$ in game sr-BIND against EC such that

$$\Pr[G_8 \Rightarrow 1] = \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{F}) .$$

To see this, notice that game G_8 can be perfectly simulated by an attacker $\mathcal{F}$ in game sr-BIND against EC with the claimed run time. Suppose that $\mathcal{A}$ submits a winning query $(H^*, ((N^*, C^*_{\mathsf{EC}}, T^*_B), B^*_{\mathsf{EC}}))$ in the simulated game. Since in G_8 encryptment keys are sampled without replacement, queries with distinct nonces have different associated keys. Since by **(3)** N^* is distinct from all nonces generated to respond to Enc queries, it must hold that the encryptment key K^*_{EC} derived when decrypting this query will be distinct from all those generated during Enc queries also. Also by **(3)**, there must have been an Enc query which returned a tuple $(H, ((N, C_{\mathsf{EC}}, T_B), B^*_{\mathsf{EC}}))$ where $N \neq N^*$. By the correctness of EC, this binding tag must verify correctly under its associated key K_{EC}. If this event occurs, then $\mathcal{A}$ has found a winning tuple $((H, M, K_{\mathsf{EC}}), (H^*, M^*, K^*_{\mathsf{EC}}), B^*_{\mathsf{EC}})$, where $K_{\mathsf{EC}} \neq K^*_{\mathsf{EC}}$ by the discussion above and M, M^* denote the messages underlying the respective ciphertexts (and which are well-defined, since by assumption both ciphertexts decrypt correctly). This implies the claim.

Putting this altogether via a standard argument implies that

$$\begin{aligned}\mathbf{Adv}^{\text{mo-ctxt}}_{\mathsf{CE[EC,f]}}(\mathcal{A}, q_c, q_e, q_d) \leq \mathbf{Adv}^{\oplus\text{-prf}}_{\mathsf{f}}(\mathcal{B}, 2 \cdot q) + q_e \cdot \mathbf{Adv}^{\text{scu}}_{\mathsf{EC}}(\mathcal{C}, 1) + \mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC}}(\mathcal{E}) \\ + \mathbf{Adv}^{\text{sr-bind}}_{\mathsf{EC}}(\mathcal{F}) + \frac{q^2 + 2(q_e+1)\cdot(q_c+q_d) + q_e \cdot (q_e - 1)}{2^{n+1}} \, ,\end{aligned}$$

thereby concluding the proof.

□

Binding security. We omit proofs for the sr-BIND and s-BIND security of $\mathsf{CE[EC,f]}$; the transform inherits these properties directly from EC.

This concludes our analysis of transforms from encryptment to ccAEAD. Next, we will establish an equivalence between the two by showing how to construct encryptment from ccAEAD.

3.7.5 A Transform From ccAEAD to Encryptment

We now describe a transform which builds a secure encryptment scheme from any secure ccAEAD scheme, thereby establishing an equivalence between the two. Moreover, the rate of encryptment in the transformed construction is exactly that of ccAEAD encryption; as such, the negative results on rate-1 encryptment from Section 3.5 extend to the ccAEAD case also.

Let $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ be a ccAEAD scheme, with associated coin space $\mathcal{R}$. Then we may construct an encryptment scheme $\mathsf{EC[CE]} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ from CE as follows. We define EKg to be the algorithm which generates a ccAEAD key $K \leftarrow\!\!\$\, \mathsf{CKg}$, chooses random coins $R \leftarrow\!\!\$\, \mathcal{R}$ where $\mathcal{R}$ denotes the coin space of the ccAEAD scheme, and outputs $K_{\mathsf{EC}} = K \,\|\, R$. The deterministic encryptment algorithm on input $(K \,\|\, R, H, M)$ uses the encryption algorithm of the ccAEAD scheme with coins fixed to R to compute $(C, C_B) \leftarrow \mathsf{CEnc}(K, H, M; R)$, returning $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) = (C, C_B)$. In particular, notice that the rate of EEnc is *exactly* that of CEnc.

We define EDec to be the algorithm which on input $(K \,\|\, R, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ first uses the de-

```
proc. main // $G_0$
$K \leftarrow_{\$} \{0,1\}^d$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_0$
$N \leftarrow_{\$} \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_0$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$T'_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
If $T'_B \neq T_B$ then return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then return $\bot$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_0$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
  Return $\bot$
$T'_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
If $T'_B \neq T_B$ then return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

```
proc. main // $G_1$, $\boxed{G_2}$
$K \leftarrow_{\$} \{0,1\}^d$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_1$, $\boxed{G_2}$
$N \leftarrow_{\$} \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$\boxed{K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$\boxed{T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})}$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_1$, $\boxed{G_2}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \neq \bot$
  Return $(M, K_{\mathsf{EC}})$
$T'_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$\boxed{T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})}$
If $T'_B \neq T_B$ then return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$\boxed{K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_1$, $\boxed{G_2}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
  Return $\bot$
$T'_B \leftarrow \mathsf{f}(B_{\mathsf{EC}}, K \oplus \mathsf{spad})$
$\boxed{T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})}$
If $T'_B \neq T_B$ then return $\bot$
$K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$
$\boxed{K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_2$
If $F[X, Y] = \bot$ then
  $F[X, Y] \leftarrow_{\$} \{0,1\}^n$
Return $F[X, Y]$
```

```
proc. main // $G_3$, $\boxed{G_4}$
$K \leftarrow_{\$} \{0,1\}^d$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_3$, $\boxed{G_4}$
$N \leftarrow_{\$} \{0,1\}^n$
If $N \in \mathcal{Q}_1$
  $\mathsf{bad}_1 \leftarrow 1$
  $N \leftarrow_{\$} \{0,1\}^n \setminus \{\mathcal{Q}_1\}$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$
$\boxed{K[N] \leftarrow_{\$} \{0,1\}^n}$
$\boxed{K_{\mathsf{EC}} \leftarrow K[N]}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_3$, $\boxed{G_4}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
If $D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \neq \bot$
  Return $(M, K_{\mathsf{EC}})$
$T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
If $T'_B \neq T_B$ then return $\bot$
$K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$
$\boxed{\text{If } K[N] = \bot}$
  $\boxed{K[N] \leftarrow_{\$} \{0,1\}^n}$
$\boxed{K_{\mathsf{EC}} \leftarrow K[N]}$
If $M = \bot$ then return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_3$, $\boxed{G_4}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
  Return $\bot$
$T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
If $T'_B \neq T_B$ then return $\bot$
$K_{\mathsf{EC}} \leftarrow F(N, \mathsf{fpad})$
$\boxed{\text{If } K[N] = \bot}$
  $\boxed{K[N] \leftarrow_{\$} \{0,1\}^n}$
$\boxed{K_{\mathsf{EC}} \leftarrow K[N]}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \bot$ then return $\bot$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_3$, $\boxed{G_4}$
If $F[X, Y] = \bot$ then
  $F[X, Y] \leftarrow_{\$} \{0,1\}^n$
Return $F[X, Y]$
```

Figure 3.28: Games for proof of Theorem 3.10.

```
proc. main // $G_5, G_6, G_7, G_8$
$K \leftarrow\!\!\$ \{0,1\}^d$ ; $\mathsf{win} \leftarrow \mathsf{false}$
$\mathcal{A}^{\mathrm{Enc,Dec,ChalDec}}$
Return $\mathsf{win}$

proc. Enc$(H, M)$ // $G_5$, $\boxed{G_6}$
$N \leftarrow\!\!\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
  $N \leftarrow\!\!\$ \{0,1\}^n \setminus \{\mathcal{Q}_1\}$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K[N] \leftarrow\!\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow K[N]$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_5$, $\boxed{G_6}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
If $D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \neq \perp$
  Return $(M, K_{\mathsf{EC}})$
$T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
If $T'_B \neq T_B$ then return $\perp$
If $K[N] = \perp$
  $K[N] \leftarrow\!\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow K[N]$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[\cdot, N, \cdot B_{\mathsf{EC}}, \cdot] \neq \{\perp\}$
  $\mathsf{bad}_2 \leftarrow \mathsf{true}$ ; return $\perp$
$\boxed{\text{If } D[\cdot,\cdot,\cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}}$
  $\boxed{\mathsf{bad}_3 \leftarrow \mathsf{true} \text{ ; return } \perp}$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_5$, $\boxed{G_6}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
  Return $\perp$
$T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
If $T'_B \neq T_B$ then return $\perp$
If $K[N] = \perp$
  $K[N] \leftarrow\!\!\$ \{0,1\}^n$
$K_{\mathsf{EC}} \leftarrow K[N]$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[\cdot, N, \cdot B_{\mathsf{EC}}, \cdot] \neq \{\perp\}$
  Return $\perp$
$\boxed{\text{If } D[\cdot,\cdot,\cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}}$
  $\boxed{\mathsf{bad}_3 \leftarrow \mathsf{true} \text{ ; return } \perp}$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. $F(X, Y)$ // $G_5, G_6, G_7, G_8$
If $F[X, Y] \neq \perp$ then
  $F[X, Y] \leftarrow\!\!\$ \{0,1\}^n$
Return $F[X, Y]$
```

```
proc. Enc$(H, M)$ // $G_7$, $\boxed{G_8}$
$N \leftarrow\!\!\$ \{0,1\}^n$
If $N \in \mathcal{Q}_1$
  $N \leftarrow\!\!\$ \{0,1\}^n \setminus \{\mathcal{Q}_1\}$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
$K[N] \leftarrow\!\!\$ \{0,1\}^n$
If $K[N] \in \mathcal{Q}_2$
  $\mathsf{bad}_4 \leftarrow \mathsf{true}$ ; $K[N] \leftarrow\!\!\$ \{0,1\}^n / \{\mathcal{Q}_2\}$
$K_{\mathsf{EC}} \leftarrow K[N]$
$\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{K_{\mathsf{EC}}\}$
$(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, H, M)$
$T_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
$C \leftarrow (N, C_{\mathsf{EC}}, T_B)$
$\mathcal{Y} \leftarrow \mathcal{Y} \cup \{(H, C, B_{\mathsf{EC}})\}$
$D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \leftarrow (M, K_{\mathsf{EC}})$
Return $(C, B_{\mathsf{EC}})$

proc. Dec$(H, C, C_B)$ // $G_7$, $\boxed{G_8}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
If $D[H, N, C_{\mathsf{EC}}, B_{\mathsf{EC}}, T_B] \neq \perp$
  Return $(M, K_{\mathsf{EC}})$
$T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
If $T'_B \neq T_B$ then return $\perp$
If $K[N] = \perp$
  $K[N] \leftarrow\!\!\$ \{0,1\}^n$
  If $K[N] \in \mathcal{Q}_2$
    $\mathsf{bad}_4 \leftarrow \mathsf{true}$ ; $K[N] \leftarrow\!\!\$ \{0,1\}^n \setminus \{\mathcal{Q}_2\}$
$K_{\mathsf{EC}} \leftarrow K[N]$
$\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{K_{\mathsf{EC}}\}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[\cdot, N, \cdot B_{\mathsf{EC}}, \cdot] \neq \{\perp\}$
  Return $\perp$
If $D[\cdot,\cdot,\cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}$
  Return $\perp$
$\boxed{b \leftarrow \mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})}$
$\boxed{\text{If } b = 0 \text{ return } \perp}$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$

proc. ChalDec$(H, C, C_B)$ // $G_7$, $\boxed{G_8}$
$(N, C_{\mathsf{EC}}, T_B) \leftarrow C$ ; $B_{\mathsf{EC}} \leftarrow C_B$
$\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{N\}$
If $(H, C, B_{\mathsf{EC}}) \in \mathcal{Y}$ then
  Return $\perp$
$T'_B \leftarrow F(B_{\mathsf{EC}}, \mathsf{spad})$
If $T'_B \neq T_B$ then return $\perp$
If $K[N] = \perp$
  $K[N] \leftarrow\!\!\$ \{0,1\}^n$
  If $K[N] \in \mathcal{Q}_2$
    $\mathsf{bad}_4 \leftarrow \mathsf{true}$ ; $K[N] \leftarrow\!\!\$ \{0,1\}^n \setminus \{\mathcal{Q}_2\}$
$K_{\mathsf{EC}} \leftarrow K[N]$
$\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \cup \{K_{\mathsf{EC}}\}$
$M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$
If $M = \perp$ then return $\perp$
If $D[\cdot, N, \cdot B_{\mathsf{EC}}, \cdot] \neq \{\perp\}$
  Return $\perp$
If $D[\cdot,\cdot,\cdot, B_{\mathsf{EC}}, \cdot] = \{\perp\}$
  Return $\perp$
$\boxed{b \leftarrow \mathsf{EVer}(H, M, K_{\mathsf{EC}}, B_{\mathsf{EC}})}$
$\boxed{\text{If } b = 0 \text{ return } \perp}$
$\mathsf{win} \leftarrow \mathsf{true}$
Return $(M, K_{\mathsf{EC}})$
```

Figure 3.29: Further games for proof of Theorem 3.10.

cryption algorithm of the ccAEAD scheme to compute $(M, K_f) \leftarrow \mathsf{CDec}(K, H, C_{\mathsf{EC}}, B_{\mathsf{EC}})$. It then checks if $\mathsf{CEnc}(K, H, M; R) = (C_{\mathsf{EC}}, B_{\mathsf{EC}})$ and if so returns M; otherwise it returns $\perp$. Notice that re-encrypting and comparing the encryptment in this way ensures that the encryptment scheme has strong correctness.

For EVer there are two cases. If CE is such that it outputs its key as the opening (i.e., $K_f = K$) then we define EVer to be the algorithm which on input $(H, M, K \parallel R, B_{\mathsf{EC}})$ simply computes $b \leftarrow \mathsf{CVer}(H, M, K, B_{\mathsf{EC}})$ and returns the result. If CE does not fall in this class then EVer needs to recover the opening key before verifying the binding tag. EVer can always do this (for both classes of ccAEAD scheme) given $(H, M, K \parallel R, B_{\mathsf{EC}})$ by re-computing $(C', C'_B) \leftarrow \mathsf{CEnc}(K, H, M; R)$ followed by $(M', K'_f) \leftarrow \mathsf{CDec}(K, H, C', C'_B)$, finally returning $b \leftarrow \mathsf{CVer}(H, M, K'_f, B_{\mathsf{EC}})$ (or 0 if any of these intermediate steps return an error). The security of the derived scheme EC[CE] is described in the following theorem.

Theorem 3.11. *Let* CE= (CKg, CEnc, CDec, CVer) *be a ccAEAD scheme. Then there exists a strongly correct encryptment scheme* EC[CE]= (EKg, EEnc, EDec, EVer) *such that for all adversaries $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, we give adversaries $\mathcal{B}, \mathcal{C}, \mathcal{E}$ such that*

$$\mathbf{Adv}^{\text{ot-ror}}_{\mathsf{EC[CE]}}(\mathcal{A}_1) \leq \mathbf{Adv}^{\text{mo-ror}}_{\mathsf{CE}}(\mathcal{B}, 1, 0, 0) \ ;$$

$$\mathbf{Adv}^{\text{r-bind}}_{\mathsf{EC[CE]}}(\mathcal{A}_2) \leq \mathbf{Adv}^{\text{r-bind}}_{\mathsf{CE}}(\mathcal{C}) \ ; \ \textit{and}$$

$$\mathbf{Adv}^{\text{s-bind}}_{\mathsf{EC[CE]}}(\mathcal{A}_3) \leq \mathbf{Adv}^{\text{s-bind}}_{\mathsf{CE}}(\mathcal{E}) \ .$$

Moreover, adversaries $\mathcal{B}, \mathcal{C}, \mathcal{E}$ run in the time of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ respectively, and the rate of EEnc *is precisely that of* CEnc*.*

Since the encryptment is recomputed during decryption with EDec, it is straightforward to see that the scheme is strongly correct. We sketch the proof of the remainder of the three-part theorem below, where we consider the more complex case in which CE does not output its key as the opening; the proof of the alternative case is entirely analogous.

(Part 1) It is easy to see that an attacker $\mathcal{B}$ in game MO-ROR against the ccAEAD scheme CE can perfectly simulate game otROR for an attacker $\mathcal{A}_1$ against the derived encryptment scheme EC[CE] by submitting $\mathcal{B}$'s encryption query to his own encryption oracle and returning the result; therefore a reduction to the MO-ROR security of CE implies the first result.

(Part 2) To see that $\mathsf{EC}[\mathsf{CE}]$ is receiver binding, notice that to win game r-BIND against $\mathsf{EC}[\mathsf{CE}]$ attacker $\mathcal{A}_2$ must output a tuple

$$((K \,\|\, R, H, M), (K' \,\|\, R', H', M'), C_B)$$

such that $(H, M) \neq (H', M')$ but for which

$$\mathsf{CVer}(H, M, K_f, C_B) = \mathsf{CVer}(H', M', K'_f, C_B) = 1$$

where

$$(M, K_f) = \mathsf{CDec}(K, H, \mathsf{CEnc}(K, H, M; R)), \text{ and}$$
$$(M', K'_f) = \mathsf{CDec}(K', H', \mathsf{CEnc}(K', H', M'; R')) \,.$$

Consider an adversary $\mathcal{C}$ in game r-BIND against CE who runs $\mathcal{A}_2$ as a subroutine. When $\mathcal{A}_2$ outputs $((K \,\|\, R, H, M), (K' \,\|\, R', H', M'), C_B)$, $\mathcal{C}$ runs CEnc then CDec on both tuples to obtain each of the openings, and returns $((K_f, H, M), (K'_f, H', M'), C_B)$ to his own challenger. This will be a winning tuple for $\mathcal{C}$ if $\mathcal{A}_2$'s output is a winning tuple in their game.

(Part 3) Let $\mathcal{F}$ be an attacker who runs $\mathcal{A}_3$ as a subroutine. When $\mathcal{A}_3$ outputs $(K \,\|\, R, H, C, C_B)$, adversary $\mathcal{F}$ outputs (K, H, C, C_B) to their own challenger. Suppose that $(K \,\|\, R, H, C, C_B)$ is a winning tuple in game s-BIND against $\mathsf{EC}[\mathsf{CE}]$. From the description of EDec, this implies that $\mathsf{CDec}(K, H, C, C_B) = (M', K'_f) \neq \bot$, and so the tuple output by $\mathcal{F}$ to their challenger in game s-BIND against CE decrypts correctly also. Moreover, by the description of EVer, it must hold that $\mathsf{CVer}(H, M, K'_f, C_B) = 0$ where $(M', K'_f) \leftarrow \mathsf{CDec}(K, H, \mathsf{CEnc}(K, H, M; R))$. Since the ciphertext is recomputed during decryption with EDec, it must be the case that $\mathsf{CEnc}(K, H, M; R) = (C, C_B)$, which by assumption decrypts correctly. As such, the key K'_f with which verification is performed by EVer is exactly that which will be recovered and used to perform verification in game s-BIND against CE. Therefore any winning tuple against $\mathsf{EC}[\mathsf{CE}]$ corresponds to a winning tuple for $\mathcal{F}$ against CE, implying the claim.

Finally, combining Lemma 3.1, Theorem 3.11, and that $\mathsf{EC}[\mathsf{CE}]$ is strongly correct implies that $\mathsf{EC}[\mathsf{CE}]$ is SCU secure.

```
$\underline{\mathrm{FROB}^{\mathcal{A}}_{\mathsf{CE}}}$
$((K, H), (K', H'), (C, C_B)) \leftarrow\$ \mathcal{A}$
If $K = K'$ then return 0
$M_1 \leftarrow \mathsf{CDec}(K, H, (C, C_B))$
$M_2 \leftarrow \mathsf{CDec}(K', H', (C, C_B))$
Return $(M_1 \neq \bot \wedge M_2 \neq \bot)$
```

Figure 3.30: The full robustness (FROB) security game for a ccAEAD scheme $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$.

3.7.6 Relationship to Robust Encryption: Relations and Separations

We conclude this section by discussing the relationship between ccAEAD and robust encryption. An encryption scheme is said to be *robust* if it is infeasible to find two distinct keys which decrypt the same ciphertext to a valid message. Robust encryption was first formalised in the public-key setting by Abdalla et al. in [4] and later extended in [66,115]. Farshim et al. provide the first treatment of robust AE in [67]. Given the similar goals of robust encryption and binding ccAEAD, it is natural to wonder whether there is some kind of equivalence between the two.

Full robustness. In Figure 3.30 we adapt the definition of *full robustness* (FROB) for AE of Farshim et al. to the ccAEAD setting. The original FROB definition of [67] can be recovered from that shown in Figure 3.30 by replacing the ciphertext / binding tag pair (C, C_B) with a single ciphertext C and removing all references to headers. The attacker $\mathcal{A}$ is challenged to output a tuple $((K, H), (K', H'), (C, C_B))$ such that $K \neq K'$, but the ciphertext / binding tag pair (C, C_B) decrypts correctly under both keys and the corresponding headers. The FROB advantage for an attacker $\mathcal{A}$ against a ccAEAD scheme CE is defined

$$\mathbf{Adv}^{\mathrm{frob}}_{\mathsf{CE}}(\mathcal{A}) = \Pr\left[\,\mathrm{FROB}^{\mathcal{A}}_{\mathsf{CE}} \Rightarrow 1\,\right],$$

where the probability is over the coins of CEnc and $\mathcal{A}$.

Separations between FROB **and** sr-BIND **security.** Intuitively, both robust encryption and ccAEAD target some notion of binding security. Thus a natural question is whether an sr-BIND secure ccAEAD scheme is also FROB secure, and vice versa. It turns out that the two notions are orthogonal and neither implies the other in generality.

The intuition for this is that breaking robustness requires finding distinct keys which both *decrypt* the same ciphertext correctly, whereas breaking sr-BIND security requires finding distinct tuples $(H, M, K_f) \neq (H', M', K'_f)$ which both *verify* the same binding tag correctly. Since there is no requirement that decrypting with distinct keys will result in distinct openings being recovered during decryption, breaking robustness does not necessarily translate into a win for an attacker in game sr-BIND. However, we note that an attacker can only break FROB security without breaking sr-BIND security if the winning tuple uses the *same* headers for both decryptions and both keys decrypt the ciphertext to the *same* underlying message. Such an attack would seem to be of little concern in verifiable abuse reporting (and perhaps other settings as well).

Likewise, since any verification performed during decryption uses the opening output by CDec as opposed to one specified by an attacker, one may modify the verification algorithm of a robust ccAEAD scheme in such a way that it becomes easy to produce two distinct openings which verify the same binding tag (thereby breaking the sr-BIND security of the scheme), but in such a way that these 'bad' openings will never be recovered during normal decryption, leaving the robustness of the scheme unaffected.

The case is different if the ccAEAD scheme is such that it always outputs its key as the opening (as is the case for many single-opening ccAEAD schemes). If such a scheme is sender binding — so ciphertexts which decrypt correctly must also verify correctly — then sr-BIND security implies FROB security. It is straightforward to verify that FROB security does not imply sr-BIND security for such schemes in general (for a separating example, we can modify an FROB secure ccAEAD scheme which outputs its encryption key as the opening such that it becomes easy to find distinct headers which verify with the same key / message / binding tag tuple, but decryption is left unchanged). We formalise this intuition and provide separating examples in the following theorem.

Theorem 3.12. *(1) There exists a ccAEAD scheme* $\mathsf{CE}_1 = (\mathsf{CKg}_1, \mathsf{CEnc}_1, \mathsf{CDec}_1, \mathsf{CVer}_1)$ *which is* FROB*-secure, but for which there exists an efficient attacker* $\mathcal{B}$ *such that*

$$\mathbf{Adv}^{\text{sr-bind}}_{\mathsf{CE}_1}(\mathcal{B}) = 1 \ .$$

(2) There exists a ccAEAD scheme $\mathsf{CE}_2 = (\mathsf{CKg}_2, \mathsf{CEnc}_2, \mathsf{CDec}_2, \mathsf{CVer}_2)$ *which is* sr-BIND*-secure, but for which there exists an efficient attacker* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{frob}}_{\mathsf{CE}_2}(\mathcal{C}) = 1 \ .$$

(3) Let $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ *be a ccAEAD scheme which outputs its encryption key as the opening. Then for any attacker* $\mathcal{A}$ *in game* FROB *against* CE*, there exists*

adversaries $\mathcal{E}, \mathcal{F}$ such that

$$\mathbf{Adv}^{\mathrm{frob}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{s\text{-}bind}}_{\mathsf{CE}}(\mathcal{E}) + \mathbf{Adv}^{\mathrm{sr\text{-}bind}}_{\mathsf{CE}}(\mathcal{F}) \ ,$$

and moreover $\mathcal{E}, \mathcal{F}$ run in the same time as $\mathcal{A}$.

Proof. We start with **(1)**. Let $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ be an FROB-secure ccAEAD scheme for which all valid openings K_f are of a fixed length of n-bits. We then construct a modified scheme $\mathsf{CE}_1 = (\mathsf{CKg}_1, \mathsf{CEnc}_1, \mathsf{CDec}_1, \mathsf{CVer}_1)$ which is identical to CE except we define CVer_1 to be the algorithm which on input (H, M, K_f, C_B) sets $K'_f = \mathrm{Trunc}_n(K_f)$ and returns the result of $\mathsf{CVer}(H, M, K'_f, C_B)$. Notice that decryption is identical in both schemes; therefore any winning tuple in game FROB against CE_1 is a winning tuple for CE also, violating the assumed robustness of CE. However, an attacker $\mathcal{B}$ can win the sr-BIND security game with probability one by outputting $((H, M, K_f \,\|\, 0), (H, M, K_f \,\|\, 1), C_B)$ where (H, M, K_f, C_B) is any tuple such that $\mathsf{CVer}(H, M, K_f, C_B) = 1$, thereby proving the first claim.

To prove **(2)**, let $\mathsf{CE} = (\mathsf{CKg}, \mathsf{CEnc}, \mathsf{CDec}, \mathsf{CVer})$ be an sr-BIND-secure ccAEAD scheme for which all keys output by CKg are bit-strings of length κ. We define a modified scheme $\mathsf{CE}_2 = (\mathsf{CKg}_2, \mathsf{CEnc}_2, \mathsf{CDec}_2, \mathsf{CVer}_2)$ which is identical to CE except we define CDec_2 to be the algorithm which on input a tuple (K, H, C, C_B) computes $K' = \mathrm{Trunc}_\kappa(K)$ and returns the output of $\mathsf{CDec}(K', H, C, C_B)$. Since verification in CE_2 is unchanged, the modified scheme CE_2 inherits the sr-BIND security of CE. However, an attacker $\mathcal{C}$ can win the FROB game against CE_2 with probability one by choosing a key K and message M, computing $(C, C_B) \leftarrow\!\$\ \mathsf{CEnc}(H, K, M)$, and outputting the tuple $((K \,\|\, 0, H), (K \,\|\, 1, H), (C, C_B))$.

For **(3)**, consider an attacker $\mathcal{A}$ in game FROB against CE. Then any winning tuple $((K, H), (K', H'), (C, C_B))$ for $\mathcal{A}$ must be such that $K \neq K'$ and $\mathsf{CDec}(K, H, (C, C_B)) = (M, K_f) \neq \perp$ and $\mathsf{CDec}(K', H', (C, C_B)) = (M', K'_f) \neq \perp$. Since by definition CDec outputs its key as the opening, it follows that $K_f \neq K'_f$ also. If $\mathcal{A}$ has found a tuple $(K, H, (C, C_B))$ which decrypts successfully to (M, K) but for which $\mathsf{CVer}(H, M, K, C_B) = 0$, then he has broken the s-BIND security of CE. If this is not the case, then $\mathcal{A}$ has found $(K, H, M) \neq (K', H', M')$ which both verify correctly with binding tag C_B, breaking the sr-BIND security of CE. Reductions to the appropriate adversaries then imply the claim. $\square$

Robustness of ccAEAD transforms. In Sections 3.7.3 and 3.7.4 we presented two transforms which allow the construction of secure ccAEAD from encryptment. It is straightforward to verify that the ccAEAD scheme $\mathsf{CE}[\mathsf{EC},\mathsf{AEAD}] = (\mathsf{CKg},\mathsf{CEnc},\mathsf{CDec},$ $\mathsf{CVer})$ built from the first transform using a secure AEAD scheme (Section 3.7.3) is robust provided the underlying AEAD scheme is itself robust. In fact, provided the underlying encryptment scheme is sr-BIND and s-BIND secure then the AEAD scheme need only satisfy a weaker property that it is infeasible for an attacker to find a tuple $((K,H),(K',H'),C_{\mathsf{AE}})$ such that $K \neq K'$ and $\mathsf{D}(K,H,C_{\mathsf{AE}}) = \mathsf{D}(K',H',C_{\mathsf{AE}})$ where D is the decryption algorithm of the AEAD scheme. The second transform $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$ (Section 3.7.4) which uses a compression function f results in a robust ccAEAD scheme provided that f is collision-resistant and the underlying encryptment scheme is both sr-BIND and s-BIND-secure. We formalise this and provide a proof sketch in the following lemma.

Lemma 3.2. *Let* $\mathsf{CE}[\mathsf{EC},\mathsf{f}] = (\mathsf{CKg},\mathsf{CEnc},\mathsf{CDec},\mathsf{CVer})$ *be the ccAEAD scheme in Section 3.7.4, built from a compression function* f *and encryptment scheme* $\mathsf{EC} = (\mathsf{EKg},\mathsf{EEnc},\mathsf{EDec},\mathsf{EVer})$. *Then for any attacker* $\mathcal{A}$ *in game* FROB *against* CE*, there exist attackers* $\mathcal{B}$*,* $\mathcal{C}$*, and* $\mathcal{E}$ *such that*

$$\mathbf{Adv}^{\mathrm{frob}}_{\mathsf{CE}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{f}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{s\text{-}bind}}_{\mathsf{EC}}(\mathcal{C}) + \mathbf{Adv}^{\mathrm{sr\text{-}bind}}_{\mathsf{EC}}(\mathcal{E}) ,$$

and moreover, all adversaries run in the same time as $\mathcal{A}$*.*

Proof. (Sketch) Recall that for $\mathsf{CE}[\mathsf{EC},\mathsf{f}]$, encryption on input (K,H,M) computes $K_{\mathsf{EC}} \leftarrow \mathsf{f}(N, K \oplus \mathsf{fpad})$ for random N and a fixed constant fpad, then $(C_{\mathsf{EC}},B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}},H,M)$, finally outputting ciphertext / binding tag pair

$$(C,C_B) = ((N,C_{\mathsf{EC}},\mathsf{f}(B_{\mathsf{EC}},K \oplus \mathsf{spad})),B_{\mathsf{EC}}) .$$

Let $\mathcal{A}$ be an attacker in the FROB game against the resulting ccAEAD scheme, and suppose $\mathcal{A}$ outputs a winning tuple

$$((K,H),(K',H'),((N,C_{\mathsf{EC}},\mathsf{f}(B_{\mathsf{EC}},K \oplus \mathsf{spad})),B_{\mathsf{EC}})) .$$

This implies that $K \neq K'$, and that the ciphertext decrypts correctly under both keys. Let $K_{\mathsf{EC}} = \mathsf{f}(N, K \oplus \mathsf{fpad})$ and $\mathsf{f}(N, K' \oplus \mathsf{fpad})$. Suppose further that $K_{\mathsf{EC}} = K'_{\mathsf{EC}}$; then this translates into a winning pair for an attacker $\mathcal{B}$ in the CR-game against f. On the other hand, suppose that $K_{\mathsf{EC}} \neq K'_{\mathsf{EC}}$ but nonetheless $\mathsf{EDec}(K_{\mathsf{EC}},H,C_{\mathsf{EC}},B_{\mathsf{EC}}) = M \neq \perp$ and $\mathsf{EDec}(K'_{\mathsf{EC}},H',C_{\mathsf{EC}},B_{\mathsf{EC}}) = M' \neq \perp$. Suppose that one (or both) of the input tuples does not verify despite decrypting correctly; then $\mathcal{A}$ has found a tuple which allows an attacker $\mathcal{C}$ to win the s-BIND game against EC. If such an event does not occur, then $\mathcal{A}$

has found tuples $(K_{\mathsf{EC}}, H, M) \neq (K'_{\mathsf{EC}}, H', M')$ which both verify correctly with binding tag B_{EC}. This corresponds to a winning tuple for an attacker $\mathcal{E}$ in game sr-BIND against CE. Putting this together proves the claim. □

Applications of encryptment. We have just seen how to build single-pass ccAEAD from encryptment. We will now conclude this chapter by showing that encryptment is a useful building block to enable single-pass constructions of other primitives.

3.8 Extensions

The applications of encryptment extend beyond ccAEAD, with our single-pass HFC encryptment scheme offering new and efficient instantiations of a variety of primitives. In this section we describe several of these primitives and their applications.

Concealment schemes. In [56], Dodis et al. introduce the notion of *concealment* schemes, a primitive which has a number of applications in the context of authenticated encryption. In particular, we will discuss its use in domain extension of AE and remotely keyed AE later in the section. A concealment scheme is defined to be a pair of algorithms $\mathsf{ConS} = (\mathsf{conceal}, \mathsf{open})$. The randomised concealment algorithm $\mathsf{conceal}$ takes as input a message M and outputs a pair *hider* and *binder*, $(h, b) \leftarrow\$ \mathsf{conceal}(M)$. The deterministic opening algorithm open takes as input a pair (h, b) and outputs either the underlying message M or $\perp$ depending on whether (h, b) is in the range of $\mathsf{conceal}(M)$.

Security for concealment schemes requires that no efficient attacker can distinguish between the hiders of two chosen messages, where we define an attacker $\mathcal{A}$'s distinguishing advantage as

$$\Pr\left[\, b = b' \;:\; (M_0, M_1) \leftarrow\$ \mathcal{A} \;;\; b \leftarrow\$ \{0,1\} \;;\; h_b \leftarrow\$ \mathsf{conceal}(M_b) \;; b' \leftarrow\$ \mathcal{A}(h_b) \,\right] .$$

We similarly require that no efficient attacker can find two distinct hiders which open the same binder; that is to say a tuple (h, h', b) such that $h \neq h'$ and $\mathsf{open}(h, b) \neq \perp$ and $\mathsf{open}(h', b) \neq \perp$. There is additionally a compactness property in that for a concealment scheme to be considered non-trivial it must be the case that the binder b is much shorter than the underlying message M, $|b| \ll |M|$.

One-pass concealment. In [56], the authors show how to construct concealment schemes using a one-time secure symmetric encryption scheme $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ for which the keys $K \leftarrow\$ \mathsf{K}$ are short and a collision resistant hash function H. On input M, $\mathsf{conceal}$ chooses a random key K and sets $h = \mathsf{Enc}(K, M)$ and $b = K||\mathsf{H}(h)$. To open (h, b), open returns $\mathsf{D}(K, h)$ if $\mathsf{H}(h) = b$ and $\perp$ otherwise. However this requires two passes for both the encryption and the hashing step.

In contrast, given an otROR, r-BIND, and s-BIND secure encryptment scheme $\mathsf{EC} = (\mathsf{EKg}, \mathsf{EEnc}, \mathsf{EDec}, \mathsf{EVer})$ with strong correctness, one can construct a secure concealment scheme which achieves the desired security goals with just a single pass over the data. We assume the header $H = \varepsilon$ in the subsequent discussion, and so omit this input from algorithms. The concealment scheme $\mathsf{ConS}_{\mathsf{EC}} = (\mathsf{conceal}_{\mathsf{EC}}, \mathsf{open}_{\mathsf{EC}})$ is defined as follows. To conceal a message M, $\mathsf{conceal}_{\mathsf{EC}}$ generates a key $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$ and computes $(C_{\mathsf{EC}}, B_{\mathsf{EC}}) \leftarrow \mathsf{EEnc}(K_{\mathsf{EC}}, M)$. The scheme outputs (h, b) where $h = C_{\mathsf{EC}}$ and $b = K_{\mathsf{EC}}||B_{\mathsf{EC}}$. To open (h, b), open returns $M \leftarrow \mathsf{EDec}(K_{\mathsf{EC}}, C_{\mathsf{EC}}, B_{\mathsf{EC}})$.

The otROR security of the encryptment scheme immediately implies the hiding security of the concealment scheme. Similarly to break the binding of the concealment scheme, an adversary $\mathcal{A}$ must find a tuple $(C_{\mathsf{EC}}, C'_{\mathsf{EC}}, K_{\mathsf{EC}}||B_{\mathsf{EC}})$ such that $C_{\mathsf{EC}} \neq C'_{\mathsf{EC}}$, $\mathsf{EDec}(K_{\mathsf{EC}}, C_{\mathsf{EC}}, B_{\mathsf{EC}}) = M$, $\mathsf{EDec}(K_{\mathsf{EC}}, C'_{\mathsf{EC}}, B_{\mathsf{EC}}) = M'$, and both $M, M' \neq \perp$. This translates to a win for an attacker $\mathcal{B}$ in game r-BIND against EC who outputs $((K_{\mathsf{EC}}, \epsilon, M), (K'_{\mathsf{EC}}, \epsilon, M'), B_{\mathsf{EC}})$. To see that this is a winning tuple for $\mathcal{B}$, notice that since EC is s-BIND secure this implies that it is infeasible to find a tuple $(K_{\mathsf{EC}}, \varepsilon, C_{\mathsf{EC}}, B_{\mathsf{EC}})$ which decrypts to a valid message M but for which $\mathsf{EVer}(\varepsilon, M, K_{\mathsf{EC}}, B_{\mathsf{EC}}) = 0$. Similarly, the fact that EC is strongly correct means that it must be the case that $M \neq M'$, since there is a unique encryptment $(C_{\mathsf{EC}}, B_{\mathsf{EC}})$ corresponding to each key / message pair. Together, these facts imply that the tuple output by $\mathcal{B}$ breaks the r-BIND security of EC.

Applications of one-pass concealment. With this encryptment-based concealment scheme we can give one-pass instantiations of domain extension for AE and remotely-keyed AE (RKAE). Domain extension for AE takes an AE scheme for which the message space consists of only 'short' messages and allows it to encrypt much longer messages. The modified encryption algorithm on input (K, M) is defined to first compute the concealment of M via $(h, b) \leftarrow\$ \mathsf{conceal}(M)$, and then output ciphertext $(h, \mathsf{E}(K, b))$.

In remotely-keyed AE, a secure but computationally-limited device holding a long-term AE key offloads most of the computational work of encrypting and decrypting to an untrusted but more powerful device. In [56] the authors show a 'canonical' RKAE scheme from any concealment scheme, and so we can apply this result to build RKAE from encryptment via $\mathsf{ConS}_{\mathsf{EC}}$. An interesting question for future work is defining and constructing remotely-keyed ccAEAD schemes, and establishing whether applying the RKAE transform of [56] to $\mathsf{ConS}_{\mathsf{EC}}$ gives a remotely-keyed ccAEAD scheme.

Verifiable outsourced storage from encryptment. In the majority of this work we have viewed encryptment as a combined encryption and commitment scheme, but an alternate way to view it is as a kind of three-party secret sharing scheme for the header and message pair. If EEnc runs EKg internally instead of accepting a key as input, the (now randomised) EEnc algorithm will output $(C_{\mathsf{EC}}, B_{\mathsf{EC}}, K_{\mathsf{EC}})$ on input H, M. None of these three values in isolation reveals any information about the message M. We will denote as $\mathsf{SShare}(H, M)$ the randomised algorithm which on input (H, M) first computes $K_{\mathsf{EC}} \leftarrow\$ \mathsf{EKg}$, and then outputs $(\mathsf{EEnc}(K_{\mathsf{EC}}, H, M), K_{\mathsf{EC}})$.

This secret-sharing viewpoint of encryptment gives an optimally-efficient and verifiable way to store large files on untrusted cloud storage. There are three parties: an untrusted storage provider, a public ledger providing integrity but not confidentiality (such as a blockchain), and the user. When the user wants to store file M with header H, it runs $\mathsf{SShare}(H, M)$. It stores C_{EC} with the cloud provider, posts B_{EC} on the public ledger, and retains K_{EC} in its own trusted storage. Since K_{EC} is small, the user can store it in hardened local storage such as a TPM. Likewise, since B_{EC} is small the cost of storing it on the public ledger is minimised. In addition to minimising local storage overhead for the user, storing B_{EC} in a public ledger gives the user the ability to prove misbehaviour on the part of the cloud provider, such as deleting part of the file or trying to modify it.

3.9 Conclusion

In this chapter we used the problem of verifiable abuse reporting in Facebook's EtE encrypted messenger as a springboard to conduct a multi-layered provable security study of ccAEAD.

We began by demonstrating an attack against Facebook's attachment franking protocol, which allows a malicious sender to send an abusive message to a user such that all attempts by the user to report this to Facebook will fail. The attack hinges on the fact that the component AEAD scheme GCM is not robust, a property that, while well-studied in the literature, had not previously been abused in a real world attack. This is an interesting example of how provable security both pre-empted and indeed may have prevented the vulnerability had there been an attempt at constructing a security proof for the scheme, or the non-robustness of GCM been considered.

Facebook opted to use an ad hoc (and ultimately flawed) scheme due to a dearth of sufficiently fast ccAEAD schemes, which raises the question of whether ccAEAD can ever match the efficiency of the fastest AEAD schemes. We define a new primitive called encryptment, which is a simple, one-time variant of ccAEAD and a useful stepping stone via which to construct secure ccAEAD. We take a detour into the literature on collision-resistant hashing to rule out the existence of secure encryptment (and by implication ccAEAD schemes) matching the rate of the fastest AEAD schemes such as OCB and GCM. We then use the classic Merkle-Damgård collision-resistant hash function as the basis for building a single-pass encryptment scheme, which in turn yields the first single-pass ccAEAD scheme via simple and efficient transforms. The development of our single-pass ccAEAD schemes is a nice illustration of the power of the rigorously formulated security definitions that are integral to the provable security paradigm. By pinning down precise syntax and security properties for encryptment and ccAEAD, we are able to make connections between distinct areas of cryptography and develop a simple modular approach to the complex task of building secure ccAEAD.

We conclude with a brief discussion of open problems and directions for future work.

Further work and open problems. The most efficient instantiations of our HFC encryptment scheme (and the ccAEAD schemes built from it) require a strong RKA-PRF assumption on the underlying compression function. Constructing schemes that match the efficiency of HFC without the RKA assumption is a key direction for future work.

We hope that encryptment will find uses beyond the construction of ccAEAD. Fleshing out the details of the encryptment applications given in Section 3.8, and finding new uses for encryptment, is an interesting open problem. Following the initial publication of the work upon which this chapter is based [59], a number of subsequent works have considered different problems related to message franking, such as building schemes that enable finer-grained reporting of abuse [101], message franking channels [80], and public key message franking [158]. It seems likely that as our understanding of EtE encryption grows, more desirable properties of message franking schemes will emerge. Developing new models and ever more efficient constructions seems likely to be a fruitful line of work in the years to come.

Chapter 4

A Provable Security Analysis of NIST SP 800-90A

Contents

4.1 Introduction and Motivation

4.1.1 Cryptographic Standards and NIST SP 800-90A

Standardisation. Designing secure cryptographic algorithms is a notoriously challenging task. As cryptography becomes ever more ubiquitous in applications, implementors are required to choose and implement primitives suitable for their purposes. With the dangers of 'rolling your own crypto' being widely known, and sifting through the cryptographic literature a time-consuming task even for a full-time cryptographer, standardising cryptographic algorithms is vital to provide implementors with easy access to secure and reliable cryptographic tools. Ideally, a standard would only include algorithms which have been heavily vetted by the research community in an open selection process, giving assurance in the soundness of the suggested algorithms. Standardisation also provides a degree of consistency across implementations, by encouraging practitioners to use a small set of approved primitives. Standardisation is a key part of the cryptographic ecosystem which, when it works well (for example, in the case of TLS 1.3, widely considered a success story of standardisation) distills the best ideas from both the research community and practitioners into millions of real-world implementations.

A subverted standard. The result (and intended impact) of standardising an algorithm is to encourage its widespread adoption. This, combined with the ubiquity of PRNGs in cryptographic applications, perhaps explains why NIST Special Publication 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators (NIST SP 800-90A) [10] was targeted by the NSA. Following the Snowden leaks, it was reported how the NSA had designed the Dual-EC and — despite concerns raised by members of the cryptographic community, who pointed out its low-speed, output biases, and known capability to be backdoored [70, 147, 153] — assiduously lobbied for its inclusion in standards as part of a wider program to subvert the standardisation process [9, 122–124]. In 2014, the Dual-EC was withdrawn from NIST SP 800-90A, and a number of subsequent works analysed the exploitability of the Dual-EC [46, 47] and the process by which it came to be standardised [31].

Analysis in this thesis. The next two chapters both take their root in the fall-out of the Dual-EC controversy. In Chapter 5, we will return to backdoored PRNGs to analyse the limits of what can and can't be accomplished by these subverted algorithms. In this chapter, we focus our attention on the *other* PRNGs which remain in NIST SP 800-90A.

4.1.2 Background on the Standard

Three PRNGs remain in the current revision of NIST SP 800-90A — HASH-DRBG, HMAC-DRBG and CTR-DRBG. These PRNGs — which respectively use a hash function, HMAC, and a block cipher as their basic building blocks — are widely used. Indeed, these are the *only* approved PRNGs for cryptographic software or hardware seeking FIPS certification [1, 161]. However, these algorithms have received surprisingly little formal analysis to date.

While aspects of these constructions have been analysed [43,78,85,142,152,168] and some implementation considerations discussed [28], these works tend to make significant simplifying assumptions and / or treat only certain algorithms rather than the constructions as a whole (see Section 4.1.4 for a full discussion of related work). There has not to date been a deeper analysis of these standardised DRBGs, either investigating the stronger security properties claimed in the standard or taking into account the (considerable) flexibility in their specification.

The constructions provided in NIST SP 800-90A are somewhat nonstandard. Even the term DRBG is rare, if not absent from the literature, which favours the term PRNG. Similarly the NIST DRBGs — which return variable length (and sizable) outputs upon request and support a variety of optional inputs and parameters — do not fit cleanly into the usual PRNG syntax [60] (see Section 2.5). With only a limited amount of formal analysis in the literature to date, coupled with the fact that the standardisation of these algorithms did not follow from a competition or widely publicly vetted process, this leaves pseudorandom number generation in large parts of software relying on relatively unanalysed algorithms.

Security claims. The standard claims that each of the NIST DRBGs is 'backtracking resistant' and 'prediction resistant'. The former property guarantees that in the event of a state compromise, output produced prior to the point of compromise remains secure. The latter property ensures that if the state is compromised and subsequently reseeded with sufficient entropy then security will be recovered. Defined informally in the standard, these can easily be seen to correspond to the Fwd and Bwd security notions given in Section 2.5. Somewhat surprisingly, to the best of our knowledge neither of these properties have been formally investigated and proved. In fact, the NIST DRBG algorithms which are responsible for initial state generation and reseeding do not seem to have been analysed at all in prior work.

Discussion. A number of factors may have contributed to this lack of analysis. It seems likely that the attention lavished on the Dual-EC resulted in the other PRNGs in NIST SP 800-90A being somewhat overlooked by the research community. Secondly, the understanding of what a PRNG should achieve has moved on significantly since the NIST DRBGs were standardised in 2006. Indeed, the concept of robustness for PRNGs [60] was not formalised until 2013, and so perhaps when the standard was introduced ad-hoc PRNG designs were more palatable. Finally, the NIST DRBGs are based on fairly run-of-the mill concepts such as running a hash function in counter mode, and yet simultaneously display design quirks which significantly complicate analysis and defy attempts at a modular treatment. As such, they perhaps fall between the cracks of not being 'interesting' enough to tackle for an attention-grabbing result, and yet too tricky for a straightforward proof.

4.1.3 Contributions

In this chapter, we conduct a thorough investigation into the security of the NIST SP 800-90A DRBGs, with a focus on HASH-DRBG and HMAC-DRBG. We pay particular attention to flexibilities in the specification of these algorithms, which are frequently abstracted away in previous analysis. We set out to analyse the algorithms as they are specified and used, and so sometimes make heuristic assumptions in our modelling (namely, working in the random oracle model (ROM) and assuming an oracle-independent entropy source). We felt this to be more constructive than modifying the constructions solely to derive a proof under weaker assumptions, and explain the rationale behind all such decisions.

Robustness proofs. As discussed in Section 4.1.2, the notion of robustness [60] captures both backtracking and prediction resistance and is the 'gold-standard' for PRNG security. For our main technical results, we analyse HASH-DRBG and HMAC-DRBG within this framework. As a (somewhat surprising) negative result, we show that implementations of HMAC-DRBG for which optional strings of additional input are not always included in next calls are *not* in fact forward secure. This is contrary to claims in the standard that the NIST DRBGs are backtracking resistant. This highlights the power of security proofs to surface subtle flaws in algorithms which at first sight may seem obviously secure, and underscores the importance of paying attention to implementation choices.

As positive results, we prove that HASH-DRBG and HMAC-DRBG (called with additional input) are robust in the ROM. The first result is fully general, while the latter is with respect to a class of entropy sources which includes those approved by the standard.

Challenges. A key challenge is that the NIST DRBGs do not appear to have been designed with a security proof in mind. As such, seemingly innocuous design decisions turn out to significantly complicate matters. The first step is to reformulate robustness for the ROM. Our modeling is inspired by Gazi and Tessaro's treatment of robustness in the ideal permutation model [69]. We must make various adaptations to accommodate the somewhat unorthodox interface of the NIST DRBGs, and specifying the model requires some care. It is for this reason that we focus on HASH-DRBG and HMAC-DRBG in this thesis, since they map naturally into the same framework. Providing a similar treatment for CTR-DRBG would require different techniques, and is an important direction for future work. The extent to which we must modify the syntax of existing security notions simply to be able to express the NIST DRBG interface within it illustrates the challenge of applying the clean ideas from academic cryptography to the far more messy real world, and highlights the gap that remains between theory and practice.

At first glance, it may seem obvious that a PRNG built from a random oracle will produce random looking bits. However, formally proving that the constructions survive the strong forms of compromise required to be robust is far from trivial. While the proofs employ fairly standard techniques, certain design features of the algorithms introduce unexpected complexities and some surprisingly fiddly analysis. Throughout this process, we highlight points at which a minor design modification would have allowed for a simpler proof.

Implementation flexibilities. We counter these formal and (largely) positive results by offering a more informal discussion of flexibilities in the standard. We argue that when the NIST DRBGs are used to produce many blocks of output per request — a desirable implementation choice in terms of efficiency, and permitted by the standard — then the usual security models may overlook important attack vectors against these algorithms. Taking a closer look, we propose an informal security model in which we suppose an attacker compromises part of the state of the DRBG — for example, through a side-channel attack — *during* an output generation request. Reconsidered within this framework, we find that each of the constructions admits vulnerabilities which allow an attacker to recover unseen output. We find a further flaw in a certain variant of CTR-DRBG which allows an attacker who compromises the state to also recover strings of additional input — which may contain secrets — previously fed to the DRBG. While our attacks are theoretical in nature, we follow this up with an analysis of the open-source OpenSSL and mbed TLS CTR-DRBG implementations which shows that the implementation decisions we highlight as potentially problematic are taken by implementors in the real world. We conclude the chapter with a number of reflections and recommendations for the safe use of these DRBGs.

4.1.4 Related Work

As discussed, the PRNGs in NIST SP 800-90A have received fairly minimal formal analysis to date; we provide an overview of related work here. A handful of prior works have analysed the NIST DRBGs as deterministic PRGs with an idealised initialisation procedure. That is to say, they prove that the next algorithm of the DRBG produces pseudorandom bits when applied to an *ideally random* initial state (e.g., $S_0 = (K_0, V_0, cnt_0)$ for uniformly random K_0, V_0 in the case of CTR-DRBG and HMAC-DRBG). This is a substantial simplification; in the real world these state components must be derived from the entropy source using the setup algorithm. Campagna [43] and Shrimpton and Terashima [152] provide such a treatment of CTR-DRBG, while Hirose [78] and Ye et al. [168] give proofs for HMAC-DRBG. This latter work also provides a formal verification of the mbedTLS implementation of HMAC-DRBG. None of these works model initial state generation or reseeding; as far as we are aware, ours is the first work to analyse these algorithms for HASH-DRBG and HMAC-DRBG. With the exception of [78], they do not model the use of additional input. Moreover, pseudorandomness of output is a much weaker security model than robustness and does not allow any form of state compromise. Kan [85] considers the assumptions underlying the security claims of the DRBGs (e.g., does the hash function underlying HASH-DRBG need to be pseudorandom, pre-image resistant, and so on). To our knowledge, this is the only previous work to consider HASH-DRBG. However, the work contains little formal analysis; proof attempts are non-standard and consider highly simplified variants of output production, and there are a number of inaccuracies (such as concluding CTR-DRBG is inherently flawed and should never be used since a block cipher in CTR-mode will never produce colliding blocks). This is also the only prior work we know of that considers forward security (although this is incorrectly assumed to be implied by the primitives underlying the DRBG in question, e.g., the fact that the hash function underlying HASH-DRBG cannot be inverted).

In [142], Ruhault claims a potential attack against the robustness of CTR-DRBG. However, the specification of the BCC function in that work (a CBC-MAC-like function which is used by the CTR-DRBG_df algorithm, see Section 4.3) is different to that provided by the standard. Namely in [142], BCC is defined to split the input $IV \parallel S$ into n 128-bit blocks ordered from right to left as $[B_n, \ldots, B_1]$. However, in the standard these blocks are ordered left to right $[B_1, \ldots, B_n]$. This leads to the blocks being processed in a different order by BCC. The attack from [142] does not work when the correct BCC function is used, and it does not seem possible to recover the attack.

4.2 Preliminaries

We begin by introducing the PRNG syntax that shall be used in this chapter. As we shall see, the NIST DRBGs have a number of optional inputs and flexibilities which are not accommodated by the usual PRNG syntax of Definition 2.3. We present our PRNG definition below, and then discuss our choice of syntax.

Definition 4.1. *A PRNG with input is a tuple of deterministic algorithms* PRNG = (setup, refresh, next) *with associated parameters* $(p, \overline{p}, \alpha, \beta_{max})$*, defined as follows:*

- setup : $\text{Seed} \times \{0,1\}^{p \leq i \leq \overline{p}} \times \mathcal{N} \rightarrow \mathcal{S}$ *takes as input a seed* seed $\in$ Seed*, an entropy sample* $I \in \{0,1\}^{p \leq i \leq \overline{p}}$*, and a nonce* $N \in \mathcal{N}$ *(where* Seed*,* $\mathcal{N}$*, and* $\mathcal{S}$ *denote the seed space, nonce space, and state space of the PRNG respectively), and returns an initial state* $S_0 \in \mathcal{S}$*.*
- refresh : $\text{Seed} \times \mathcal{S} \times \{0,1\}^{p \leq i \leq \overline{p}} \rightarrow \mathcal{S}$ *takes as input a seed* seed $\in$ Seed*, a state* $S \in \mathcal{S}$*, and an entropy sample* $I \in \{0,1\}^{p \leq i \leq \overline{p}}$*, and returns a state* $S' \in \mathcal{S}$*.*
- next : $\text{Seed} \times \mathcal{S} \times \mathbb{N}^{\leq \beta_{max}} \times \{0,1\}^{\leq \alpha} \rightarrow \{0,1\}^{\leq \beta_{max}} \times \mathcal{S}$ *takes as input a seed* seed $\in$ Seed*, a state* $S \in \mathcal{S}$*, a parameter* $\beta \in \mathbb{N}^{\leq \beta_{max}}$*, and a string of additional input addin* $\in \{0,1\}^{\leq \alpha}$*, and returns an output* $R \in \{0,1\}^{\beta}$ *and an updated state* $S' \in \mathcal{S}$*.*

If a PRNG always has seed $= \varepsilon$ *or addin* $= \varepsilon$ *(indicating that, respectively, a seed or additional input is never used), then we omit these parameters.*

Discussion. Our PRNG definition modifies Definition 2.3 as follows. Firstly, following Shrimpton and Terashima [151] in their modelling of the Intel RNG, we have removed the init algorithm from our PRNG specification. We instead assume that the seed is generated externally and supplied to the PRNG. This modification makes sense when modelling real-world PRNGs such as the NIST DRBGs for which an explicit seed generation algorithm (or as we shall see, a seed) is not specified. In this work, we assume that seeds are sampled uniformly from the seed space of the DRBG. Also following [151], we define setup to be a deterministic algorithm which constructs the initial state of the PRNG from samples drawn from the entropy source (as opposed to using truly random coins). This modification allows us to model real-world PRNGs which must construct their initial state from an imperfect entropy source. We further modify the syntax of setup from [151] to have it take an entropy sample and nonce as input, as per the specification of the NIST DRBGs. In contrast, in [151] setup takes no input but may have access to the entropy source.

Finally, we extend the definitions of refresh and next from Definition 2.3 to allow: **(1)**

entropy samples I to be bit-strings of arbitrary length $|I| \in [p, \overline{p}]$, as opposed to some fixed length p; **(2)** variable length outputs up to length β_{max} to be requested via the parameter β; and **(3)** the option to include strings of additional input $addin \in \{0,1\}^{\leq\alpha}$ in next calls. As we shall see in Section 4.3, this is necessary to capture the interface of the corresponding algorithms for the NIST DRBGs. NIST SP 800-90A uses the term *deterministic random bit generator* (DRBG) instead of the more familiar PRNG. We use these terms interchangeably.

4.3 Overview of NIST SP 800-90A

NIST SP 800-90A defines three DRBG mechanisms, HASH-DRBG, HMAC-DRBG, and CTR-DRBG. The former two are based on an approved hash function, and the latter on an approved block cipher. The full list of approved primitives is given in the standard; as an example, we provide the parameters for HASH-DRBG and HMAC-DRBG using SHA-256, and CTR-DRBG based on AES-128, in Table 4.1. We will begin with an overview of the standard, and then describe the specification of each DRBG in turn.

Algorithms and mapping into Definition 4.1. The standard specifies a tuple of (Instantiate, Reseed, Generate) algorithms for each of the DRBGs. These algorithms map directly into the (setup, refresh, next) algorithms in the PRNG model of Definition 4.1. For consistency with the usual PRNG syntax, we refer to the NIST DRBG algorithms as (setup, refresh, next) throughout. The NIST DRBGs are not specified to take a seed; as such when mapping these into the syntax of Definition 4.1 we take seed $= \varepsilon$ and omit this parameter from subsequent definitions. We discuss the lack of a seed further in Section 4.5. The standard also defines an update algorithm for CTR-DRBG and HMAC-DRBG and derivation functions for HASH-DRBG and CTR-DRBG. These algorithms are used to derive new state variables from provided data, and are called as subroutines by the PRNG algorithms. The component algorithms of HASH-DRBG and HMAC-DRBG are shown in Figure 4.1 and those for CTR-DRBG are shown in Figure 4.2. We discuss these in more detail in Section 4.4.

DRBG functions. The setup, refresh, and next algorithms underly (respectively) the `Instantiate`, `Reseed`, and `Generate` functions of the DRBG. When called, these functions

Table 4.1: Table showing sample parameter settings for the NIST DRBGs. All quantities are given in bits.

	CTR-DRBG with df	CTR-DRBG w/out df	HMAC-DRBG	HASH-DRBG
Underlying primitive	AES-128	AES-128	HMAC-SHA-256	SHA-256
Highest supported security strength	128	128	256	256
State component lengths	$\|K\| = 128$, $\|V\| = 128$	$\|K\| = 128$, $\|V\| = 128$	$\|K\| = 256$, $\|V\| = 256$	$\|V\| = 440$, $\|C\| = 440$
Output block length	128	128	256	256
Min entropy for setup and refresh	security strength	security strength	security strength	security strength
Min entropy input length	128	128	256	256
Max entropy input length	2^{35}	256	2^{35}	2^{35}
Max length of *addin*	2^{32}	256	2^{32}	2^{32}
Max no. of bits / request	2^{19}	2^{19}	2^{19}	2^{19}
Max no. of requests between reseeds	2^{48}	2^{48}	2^{48}	2^{48}

check the validity of the request (e.g., that the number of requested bits does not exceed β_{max}) and return an error if these checks fail. If not, the function fetches the internal state of the DRBG, along with any other inputs required by the algorithms (such as entropy inputs, a nonce, and so on), and the underlying algorithm is applied to these inputs. The resulting outputs are returned to the caller and / or used to update the internal state, and the successful status of the call is indicated to the caller. Here we abstract away this process to avoid cluttering our exposition. To this end, we assume that all required inputs are provided to the algorithm in question (without modelling how these are fetched), and assume that all inputs and requests are valid, omitting the success / error notifications. The NIST DRBGs are also specified to have an `Uninstantiate` function which erases the internal state, and a `Health Test` function which is used to test whether the other functions in an implementation are performing correctly. We do not model these functions in this work.

The DRBG state. The standard defines the *working state* of a DRBG to be the set of stored variables which are used to produce pseudorandom output. The *internal state* is then defined to be the working state plus administrative information which indicates the security strength of the instantiation and whether prediction resistance is supported (see below). We typically omit administrative information as this shall be clear from the context. By the 'state' of the DRBG (denoted S), we mean the working state unless otherwise specified.

Entropy sources and instantiation. A DRBG must have access to an *approved entropy source* during initial state generation via setup. This may either be a live entropy source as approved in NIST SP 800-90B [157] or a (truly) random bit generator as per NIST SP 800-90C [11]. The DRBG uses the function `Get_entropy_input`() to request an entropy sample I of length within the range $[p, \overline{p}]$ (see Definition 4.1) containing a given amount of entropy (discussed further below). For all DRBG mechanisms except CTR-DRBG implemented without a derivation function, the `Instantiate` function must also acquire a nonce. Nonces must either contain $\gamma^*/2$-bits of min-entropy or not be expected to repeat more than such a value would. Examples of suitable nonces given in the standard include strings drawn from the entropy source, time stamps, and sequence numbers. Once the initial state has been constructed, the DRBG is said to be *instantiated.* A DRBG implementation can support multiple simultaneous instantiations which are differentiated between using state handles. Here we assume that each DRBG supports a single instantiation, and so omit handles.

Reseeding. If a DRBG implementation has continual access to an entropy source, then the implementation is said to support *prediction resistance.* In this case, entropy samples drawn from the source may be periodically incorporated into the DRBG state via refresh. We assume that implementations always support prediction resistance, and omit the parameters indicating this from the state and function calls. Reseeds may be explicitly requested by the consuming application or triggered by a request in a next call (in which case, a refresh is performed before the state is passed to next). Additionally, a DRBG implementation specifies a parameter $reseed\_interval$, which indicates the maximum number of output generation requests before a reseed is forced. For all approved DRBG variants, $reseed\_interval$ may be at most 2^{48}, with the exception of CTR-DRBG instantiated with 3-KeyTDEA for which $reseed\_interval$ may not exceed 2^{32}. The number of next calls since the last refresh is recorded by a state component called a reseed counter (cnt). The reseed counter is not a security critical state variable, and throughout this work we assume that it is publicly known. In keeping with the usual modelling of PRNGs, we assume here that reseeds are always explicitly requested; this is without loss of generality.

Security strength. An instantiation of a DRBG is parameterised by a *security strength* γ^*. A DRBG may be instantiated at any $\gamma^* \in \{112, 128, 192, 256\}$ provided this does not exceed the highest strength supported by the implementation (which in turn depends on the underlying primitive, see Table 4.1 for examples). The standard requires that all entropy samples used in initial state generation and reseeding contain at least γ^*-bits of entropy. In contrast, robustness for PRNGs [60] requires that a PRNG is secure when reseeded with sets of entropy samples which *collectively* contain γ^*-bits of entropy. Looking ahead to Section 4.6, we analyse HASH-DRBG with respect to this stronger notion.

Output generation. Outputs of varying lengths up to β_{max} bits may be requested by the caller via the parameter β which forms part of the input to the next function. For all approved DRBGs β_{max} may be as large as 2^{19}, with the exception of CTR-DRBG instantiated with 3-KeyTDEA for which β_{max} may not exceed 2^{13}.

Additional input. The standard gives the option for strings of additional input (denoted *addin*) to be provided to the DRBG by the caller during next calls. These inputs may be public or predictable (e.g., device serial numbers and time stamps), or may contain secrets. If these inputs do contain entropy, they may provide a buffer in the event of a system failure or compromise. Here we typically assume that additional input is either always or never included during next calls.

The standard also allows optional additional input to be included in refresh calls, and during setup (in the form of a *personalisation string*). For brevity, we do not model these inputs in this work, and omit them from our presentation of the algorithms. However, since these additional inputs are simply appended to the entropy input in refresh calls and the nonce in setup calls, our analysis immediately extends to the case in which these inputs are included (and our results are stronger for not relying on any entropy that they may contain).

4.4 Algorithms

In this section, we describe the component algorithms for each of the NIST DRBGs. When presenting the algorithms in pseudocode, we write 'Require :' and 'Ensure :' to denote, respectively, the input and output of an algorithm; this is both for clarity and to be consistent with their presentation in NIST SP 800-90A.

4.4.1 HASH-DRBG.

HASH-DRBG is built from an approved cryptographic hash function $\mathsf{H} : \{0,1\}^{\leq \omega} \rightarrow \{0,1\}^{\ell}$. The working state is defined $S = (V, C, cnt)$, where the counter $V \in \{0,1\}^{L}$ and constant $C \in \{0,1\}^{L}$ are the security critical state variables and recall that *cnt* denotes the reseed counter. The constant C is added into the counter V during each state update, but is itself only updated following a reseed. The standard does not explicitly state the purpose of C; however slides by Kelsey from 2004 [88] mention how HASH-DRBG will "Hash with constant to avoid duplicating other hash computations". As such, it would appear that the purpose of the constant is to ensure that if a previous counter V is duplicated at some point in a different refresh period then the inclusion of the (almost certainly distinct) counter in the subsequent state update prevents the previous sequence of states being repeated.

Algorithms. The component algorithms for HASH-DRBG are shown in Figure 4.1. Both the setup and refresh algorithms derive a new state by applying the derivation function HASH-DRBG_df to the entropy input and (in the case of refresh) the previous counter. Output generation via next proceeds as follows. If additional input is used in the call, it is hashed and added into the counter V (lines 3 - 5). Output blocks are then produced by hashing the counter in CTR-mode (lines 7 - 10). At the conclusion of the call, the counter V is hashed with a distinct prefix prepended, and the resulting string — along with the constant C and reseed counter *cnt* — are added into V (lines 12 - 13).

4.4.2 HMAC-DRBG

HMAC-DRBG is based on the function $\mathsf{HMAC} : \{0,1\}^{\ell} \times \{0,1\}^{\leq \omega} \rightarrow \{0,1\}^{\ell}$, which is built from an approved hash function. The working state is of the form $S = (K, V, cnt)$, where the key $K \in \{0,1\}^{\ell}$ and counter $V \in \{0,1\}^{\ell}$ are the security critical state variables.

Algorithms. The component algorithms of HMAC-DRBG are shown in Figure 4.1. The setup and refresh algorithms of HMAC-DRBG both use the update subroutine to incorporate an entropy sample I into K and V. For setup, these variables are initialised to $K = 0\text{x}00\ldots00$ and $V = 0\text{x}01\ldots01$ prior to this process. The next algorithm for HMAC-DRBG proceeds as follows. If additional input is used, this is incorporated into K and V via the update function (lines 3 - 4). Output is then generated by iteratively computing $V \leftarrow \mathsf{HMAC}(K, V)$ and concatenating the resulting strings (lines 6 - 8). At the conclusion of the call, both key and counter are updated via the update function (line 10).

4.4.3 CTR-DRBG

CTR-DRBG is built from an approved block cipher $\mathrm{E} : \{0,1\}^{\kappa} \times \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell}$. The working state is defined $S = (K, V, cnt)$, where the key $K \in \{0,1\}^{\kappa}$ and counter $V \in \{0,1\}^{\ell}$ are the security critical state variables.

Algorithms. There are two variants of CTR-DRBG depending on whether a derivation function is used. Omitting the derivation function improves efficiency (see Section 4.8.6); however, this option may only be taken if the DRBG has access to a 'full entropy source' which returns statistically close to uniform (as opposed to just high entropy) strings. As such, this represents an implementation trade-off between efficiency and flexibility.

Component algorithms for CTR-DRBG are shown in Figure 4.2. For brevity we omit the CTR-DRBG refresh algorithm since it is not analysed in this work, and only include the setup algorithm for the case in which a derivation function is used. The next algorithm for CTR-DRBG proceeds as follows. First, any additional input is incorporated into the state via the update function (line 8). If a derivation function is used, the string of additional input is conditioned into a $(\kappa + \ell)$-bit string with CTR-DRBG_df prior to this process (line 5). If a derivation function is not used, the additional input string is restricted to be at most $(\kappa + \ell)$-bits in length. Output blocks are then iteratively generated using the block cipher in CTR-mode (lines 11 - 13). At the conclusion of the call, both K and V are updated via an application of the update function (line 15).

```
HASH-DRBG_df
Require: $input\_string$, $(num\_bits)_{32}$
Ensure: $req\_bits$
$temp \leftarrow \varepsilon$ ; $m \leftarrow \lceil num\_bits/\ell \rceil$
For $i = 1, \ldots, m$
    $temp \leftarrow temp \,\|\, \mathsf{H}((i)_8 \,\|\, (num\_bits)_{32} \,\|\, input\_string)$
$req\_bits \leftarrow \mathrm{left}(temp, num\_bits)$
Return $req\_bits$
```

```
HASH-DRBG setup
Require: $I, N$
Ensure: $S_0 = (V_0, C_0, cnt_0)$
$seed\_material \leftarrow I \,\|\, N$
$V_0 \leftarrow \mathsf{HASH\text{-}DRBG\_df}(seed\_material, L)$
$C_0 \leftarrow \mathsf{HASH\text{-}DRBG\_df}(0x00 \,\|\, V_0, L)$
$cnt_0 \leftarrow 1$
Return $(V_0, C_0, cnt_0)$
```

```
HASH-DRBG refresh
Require: $S = (V, C, cnt), I$
Ensure: $S' = (V', C', cnt')$
$seed\_material \leftarrow 0x01 \,\|\, V \,\|\, I$
$V' \leftarrow \mathsf{HASH\text{-}DRBG\_df}(seed\_material, L)$
$C' \leftarrow \mathsf{HASH\text{-}DRBG\_df}(0x00 \,\|\, V', L)$
$cnt' \leftarrow 1$
Return $(V', C', cnt')$
```

```
HASH-DRBG next
Require: $S = (V, C, cnt), \beta, addin$
Ensure: $R, S' = (V', C', cnt')$
1. If $cnt > reseed\_interval$
2.     Return reseed_required
3. If $addin \neq \varepsilon$
4.     $w \leftarrow \mathsf{H}(0x02 \,\|\, V \,\|\, addin)$
5.     $V \leftarrow (V + w) \bmod 2^L$
6. $data \leftarrow V$ ; $temp_R \leftarrow \varepsilon$ ; $n \leftarrow \lceil \beta/\ell \rceil$
7. For $j = 1, \ldots, n$
8.     $r \leftarrow \mathsf{H}(data)$
9.     $data \leftarrow (data + 1) \bmod 2^L$
10.    $temp_R \leftarrow temp_R \,\|\, r$
11. $R \leftarrow \mathrm{left}(temp_R, \beta)$
12. $H \leftarrow \mathsf{H}(0x03 \,\|\, V)$
13. $V' \leftarrow (V + H + C + cnt) \bmod 2^L$
14. $C' \leftarrow C$ ; $cnt' \leftarrow cnt + 1$
15. Return $R, (V', C', cnt')$
```

```
HMAC-DRBG update
Require: $provided\_data, K, V$
Ensure: $K, V$
$K \leftarrow \mathsf{HMAC}(K, V \,\|\, 0x00 \,\|\, provided\_data)$
$V \leftarrow \mathsf{HMAC}(K, V)$
If $provided\_data \neq \varepsilon$
    $K \leftarrow \mathsf{HMAC}(K, V \,\|\, 0x01 \,\|\, provided\_data)$
    $V \leftarrow \mathsf{HMAC}(K, V)$
Return $(K, V)$
```

```
HMAC-DRBG setup
Require: $I$, $N$
Ensure: $S_0 = (K_0, V_0, cnt_0)$
$seed\_material \leftarrow I \,\|\, N$
$K \leftarrow 0x00 \ldots 00$
$V \leftarrow 0x01 \ldots 01$
$(K_0, V_0) \leftarrow \mathsf{update}(seed\_material, K, V)$
$cnt_0 \leftarrow 1$
Return $(K_0, V_0, cnt_0)$
```

```
HMAC-DRBG refresh
Require: $S = (K, V, cnt), I$
Ensure: $S' = (K', V', cnt')$
$seed\_material \leftarrow I$
$(K', V') \leftarrow \mathsf{update}(seed\_material, K, V)$
$cnt' \leftarrow 1$
Return $(K', V', cnt')$
```

```
HMAC-DRBG next
Require: $S = (K, V, cnt)$, $\beta, addin$
Ensure: $R, S' = (K', V', cnt')$
1. If $cnt > reseed\_interval$
2.     Return reseed_required
3. If $addin \neq \varepsilon$
4.     $(K, V) \leftarrow \mathsf{update}(addin, K, V)$
5. $temp \leftarrow \varepsilon$ ; $n \leftarrow \lceil \beta/\ell \rceil$
6. For $j = 1, \ldots, n$
7.     $V \leftarrow \mathsf{HMAC}(K, V)$
8.     $temp \leftarrow temp \,\|\, V$
9. $R \leftarrow \mathrm{left}(temp, \beta)$
10.$(K', V') \leftarrow \mathsf{update}(addin, K, V)$
11. $cnt' \leftarrow cnt + 1$
12. Return $R, (K', V', cnt')$
```

Figure 4.1: Component algorithms for HASH-DRBG and HMAC-DRBG.

```
CTR-DRBG_df
Require: $input\_string$, $num\_bits \leq 512$
Ensure: $req\_bits$
$L \leftarrow (\mathsf{len}(input\_string)/8)_{32}$ ; $N \leftarrow (num\_bits/8)_{32}$
$Z \leftarrow L \,\|\, N \,\|\, input\_string \,\|\, 0x80$
While $\mathsf{len}(Z) \bmod \ell \neq 0$
    $Z \leftarrow Z \,\|\, 0x00$
$temp \leftarrow \varepsilon$ ; $i \leftarrow 0$
$K \leftarrow \mathrm{left}(0x000102...1D1E1F, \kappa)$
While $len(temp) < \kappa + \ell$
    $IV \leftarrow (i)_{32} \,\|\, 0^{\ell-32}$
    $temp \leftarrow temp \,\|\, \mathsf{BCC}(K, (IV \,\|\, Z))$
    $i \leftarrow i + 1$
$K \leftarrow \mathrm{left}(temp, \kappa)$
$X \leftarrow \mathrm{select}(temp, \kappa + 1, \kappa + \ell)$
$temp \leftarrow \varepsilon$
While $len(temp) < num\_bits$
    $X \leftarrow E(K, X)$
    $temp \leftarrow temp \,\|\, X$
$req\_bits \leftarrow \mathrm{left}(temp, num\_bits)$
Return $req\_bits$
```

```
BCC
Require: $K, data$
Ensure: $output\_block$
$chain \leftarrow 0^{\ell}$
$n \leftarrow len(data)/\ell$
$(B_1, \ldots, B_n) \leftarrow \mathrm{Parse}_{\ell}(data)$
For i = 1, ..., n
    $input\_block \leftarrow chain \oplus B_i$
    $chain \leftarrow E(K, input\_block)$
$output\_block \leftarrow chain$
Return $output\_block$
```

```
CTR-DRBG update
Require: $provided\_data, K, V$
Ensure: $K, V$
$temp \leftarrow \varepsilon$ ; $m \leftarrow \lceil (\kappa + \ell)/\ell \rceil$
For $j = 1, \ldots, m$
    $V \leftarrow (V + 1) \bmod 2^{\ell}$ ; $C \leftarrow E(K, V)$
    $temp \leftarrow temp \,\|\, C$
$temp \leftarrow \mathrm{left}(temp, (\kappa + \ell))$
$temp \leftarrow temp \oplus provided\_data$
$K \leftarrow \mathrm{left}(temp, \kappa)$
$V \leftarrow \mathrm{right}(temp, \ell)$
Return $K, V$
```

```
CTR-DRBG setup
Require: $I$, $nonce$
Ensure: $S_0 = (K_0, V_0, cnt_0)$
$seed\_material \leftarrow I \,\|\, nonce$
$seed\_material \leftarrow \mathsf{CTR\text{-}DRBG\_df}(seed\_material, (\kappa + \ell))$
$K \leftarrow 0^{\kappa}$ ; $V \leftarrow 0^{\kappa}$
$(K_0, V_0) \leftarrow \mathsf{update}(seed\_material, K, V)$
$cnt_0 \leftarrow 1$
Return $(K_0, V_0, cnt_0)$
```

```
CTR-DRBG next
Require: $S = (K, V, cnt), \beta, addin$
Ensure: $R, S' = (K', V', cnt')$
1. If $cnt > reseed\_interval$
2.     Return reseed_required
3. If $addin \neq \varepsilon$
4.     If derivation function used then
5.         $addin \leftarrow \mathsf{CTR\text{-}DRBG\_df}(addin, (\kappa + \ell))$
6.     Else if $len(addin) < (\kappa + \ell)$ then
7.         $addin \leftarrow addin \,\|\, 0^{(\kappa + \ell - len(addin))}$
8.     $(K, V) \leftarrow \mathsf{update}(addin, K, V)$
9. Else $addin \leftarrow 0^{\kappa + \ell}$
10. $temp \leftarrow \varepsilon$ ; $n \leftarrow \lceil \beta/\ell \rceil$
11. For $j = 1, \ldots, n$
12.     $V \leftarrow (V + 1) \bmod 2^{\ell}$ ; $r \leftarrow E(K, V)$
13.     $temp \leftarrow temp \,\|\, r$
14. $R \leftarrow \mathrm{left}(temp, \beta)$
15. $(K', V') \leftarrow \mathsf{update}(addin, K, V)$
16. $cnt' \leftarrow cnt + 1$
17. Return $R, (K', V', cnt')$
```

Figure 4.2: Component algorithms for CTR-DRBG. The CTR-DRBG setup algorithm shown is for the case in which a derivation function is used.

4.5 Robustness in the Random Oracle Model

As discussed in Section 1.2.1, the stronger security properties of backtracking and prediction resistance claimed in the standard have never been formally investigated. To address this, we analyse HASH-DRBG and HMAC-DRBG in the robustness framework of [60] (see Section 2.5). This models a powerful attacker who is able to compromise the state and influence the entropy source of the PRNG, and encapsulates both backtracking and prediction resistance. In this section, we adapt the robustness model from Section 2.5 to accommodate the NIST DRBGs, and introduce the notion of robustness in the ROM.

Distribution sampler. We begin by tweaking the definition of a $(q_{\mathcal{D}}, \gamma^*)$-legitimate distribution sampler definition to model the additional entropy sample (which recall must contain γ^* bits of entropy) with which the NIST DRBGs are seeded during setup. Formally, we say that a sampler $\mathcal{D}$ is $(q_{\mathcal{D}}^{+}, \gamma^*)$-*legitimate* if: **(1)** for all $j \in [1, q_{\mathcal{D}} + 1]$:

$$\mathrm{H}_{\infty}(I_j | I_1, \ldots, I_{j-1}, I_{j+1}, \ldots, I_{q_{\mathcal{D}}+1}, \gamma_1, \ldots, \gamma_{q_{\mathcal{D}}+1}, z_1, \ldots, z_{q_{\mathcal{D}}+1}) \geq \gamma_j \ ,$$

where $\sigma_0 = \varepsilon$ and $(\sigma_j, I_j, \gamma_j, z_j) \leftarrow_{\$} \mathcal{D}(\sigma_{j-1})$; and **(2)** it holds that $\gamma_1 \geq \gamma^*$. It is straightforward to see that to any sequence of *Get_entropy_input*() calls made by the DRBG, we can define an associated sampler.

We note that NIST SP 800-90B [157] defines the entropy estimate of sample I as simply $\mathrm{H}_{\infty}(I)$, as opposed to conditioning on other samples and associated data. However, the tests specified in NIST SP 800-90B estimate entropy using multiple samples drawn from the source. As such, it seems reasonable to assume that entropy sources satisfy the conditional entropy requirement used above.

4.5.1 Defining Robustness in the ROM

Our positive results about HASH-DRBG and HMAC-DRBG will be in the random oracle model (ROM). As such, the first step in our analysis is to adapt the notion of robustness to the ROM.

Robustness. Consider the game $\overline{\mathrm{Rob}}$ shown in Figure 4.3, for a PRNG PRNG with associated parameter set $(p, \overline{p}, \alpha, \beta_{max})$ and attacker / sampler pair $\mathcal{A}$, $\mathcal{D}$. As with game

Rob (Section 2.5.1), the game is parameterised by an entropy threshold γ^*. Mapping the NIST DRBGs into this model, we take γ^* equal to the security strength of the instantiation. At the start of the game, we choose a random function $\mathsf{H} \leftarrow\!\$\, \mathcal{H}$ where $\mathcal{H}$ denotes the set of all functions of a given domain and range. All of the PRNG algorithms have access to H which we indicate in superscript (e.g., $\mathsf{setup}^{\mathsf{H}}$). Unlike the modelling of robustness in the IPM of Gazi and Tessaro [69], we do *not* give the sampler $\mathcal{D}$ access to H for reasons we discuss below. To the best of our knowledge this is the first work to consider robustness in the ROM, and our security model may be useful to analyse other PRNGs beyond HASH-DRBG and HMAC-DRBG. Game $\overline{\text{Rob}}$ additionally modifies game Rob to accommodate our PRNG syntax (including the use of additional input, discussed below), and similarly to [151] generates the initial state using the deterministic $\mathsf{setup}^{\mathsf{H}}$ algorithm seeded with the first entropy sample output by the sampler (as opposed to via the randomised setup algorithm).

The game is implicitly parameterised by the nonce distribution $\mathcal{N}$ used by PRNG, where we write $N \leftarrow \mathcal{N}$ to denote sampling a nonce. Since nonces may be predictable (e.g., if a sequence number is used) we assume $\mathcal{N}$ is public and that the nonce sampled by the challenger at the start of an execution of $\overline{\text{Rob}}$ is given to $\mathcal{A}$ as part of the input $\overline{\gamma_1}$. Similarly, we assume the attacker can choose the strings of additional input which may be included in next calls. These are conservative assumptions, since any entropy in these values can only make the attacker's job harder. We assume that an attacker either always or never includes additional input in RoR queries.

With this in place, the $\overline{\text{Rob}}$ advantage of an adversary $\mathcal{A}$ and a $(q_{\mathcal{D}}^{+}, \gamma^*)$-legitimate sampler $\mathcal{D}$ is defined

$$\mathbf{Adv}^{\overline{\text{rob}}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A},\mathcal{D}) = 2 \cdot \left| \Pr\left[\overline{\text{Rob}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \Rightarrow 1 \right] - \frac{1}{2} \right| .$$

We say that $\mathcal{A}$ is a $(q_{\mathsf{H}}, q_R, q_{\mathcal{D}}, q_C, q_S)$-adversary if it makes q_{H} queries to the random oracle H, q_R queries to its RoR oracle, a total of q_S queries to its Get and Set oracles, and $q_{\mathcal{D}}$ queries to its Ref oracle of which at most q_C are consecutive.

Fixed length variant. While we define the general game $\overline{\text{Rob}}$ here, our robustness proofs will be in a slightly restricted game $\overline{\text{Rob}}_{\beta}$ in which the attacker may only request outputs of some fixed length $\beta \leq \beta_{max}$ in RoR queries. This simplifying assumption is to avoid further complicating security bounds with parameters indicating the length of the output requested in each RoR query. The analogous results for the fully general game

$\overline{\text{Rob}}$ can be recovered as a straightforward extension of our proofs.

Standard model forward security. Our negative result about the forward security of HMAC-DRBG shall be in the standard model. We define game $\overline{\text{Fwd}}$ to be a restricted variant of $\overline{\text{Rob}}$ in which: **(1)** we no longer sample a random oracle $\mathsf{H} \leftarrow\!\!\$\, \mathcal{H}$ at the start of the game and remove oracle access to H from all algorithms; and **(2)** the attacker $\mathcal{A}$ is allowed no Set queries, and makes a single Get query after which they may make no further queries. The forward security advantage of $(\mathcal{A}, \mathcal{D})$ is defined

$$\mathbf{Adv}^{\overline{\text{fwd}}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) = 2 \cdot \left| \Pr\left[\overline{\text{Fwd}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \Rightarrow 1 \right] - \frac{1}{2} \right| .$$

The problem of seeding. It is well known that deterministic extraction from imperfect sources is impossible in general (see Section 2.4), which is why the PRNG in game $\overline{\text{Rob}}$ is initialised with a public seed seed which crucially is chosen independently of the entropy source. In the literature on provably robust PRNGs, the seed is typically one or more uniform bit strings [60, 62, 69].

Unfortunately (for our analysis), none of the NIST DRBGs are specified to take a seed (i.e., seed $= \varepsilon$ in our modelling). Moreover, HMAC-DRBG and HASH-DRBG have no state components or inputs which can be reframed as a seed without adding substantial assumptions. Indeed, any seed derived from the entropy source will not satisfy the required independence criteria. For example, this rules out reframing the constant C which is a state component of HASH-DRBG as a seed. Likewise, the standard allows the nonce used by setup to be sampled from the source, rendering this also unsuitable in general. While for simplicity we omit the optional personalisation strings and additional input fed to setup and refresh, respectively, we stress that since these are provided by the caller and are arbitrary these would not in general be suitable had they been included. For example, there is nothing to prevent the caller using past PRNG outputs, which clearly depend on the source, as additional input.

At this point we are faced with two choices. We either: **(1)** give sampler $\mathcal{D}$ access to H (as in the robustness in the IPM model of [69]), and either modify the NIST DRBGs to accommodate a random seed or restrict our analysis to implementations for which additional input is sufficiently independent of the source to suffice as a seed. Or: **(2)** do not give $\mathcal{D}$ access to H. In this case, the oracle H with respect to which security

```
$\overline{\mathrm{Rob}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*}$
$\mathsf{H} \leftarrow\$ \mathcal{H}$ ; $b \leftarrow\$ \{0,1\}$ ; $N \leftarrow \mathcal{N}$
$\sigma \leftarrow \varepsilon$ ; $\mathsf{seed} \leftarrow\$ \mathrm{Seed}$
$(\sigma, I, \gamma, z) \leftarrow\$ \mathcal{D}(\sigma)$
$S \leftarrow \mathsf{setup}^{\mathsf{H}}(\mathsf{seed}, I, N)$
$c \leftarrow \gamma^*$ ; $\overline{\gamma} \leftarrow (\gamma, z, N)$
$b^* \leftarrow\$ \mathcal{A}^{\mathrm{Ref,RoR,Get,Set,}\mathsf{H}}(\mathsf{seed}, \overline{\gamma})$
Return $(b = b^*)$

proc. $\mathsf{H}(X)$
Return $\mathsf{H}(X)$
```

```
Ref
$(\sigma, I, \gamma, z) \leftarrow\$ \mathcal{D}(\sigma)$
$S \leftarrow \mathsf{refresh}^{\mathsf{H}}(\mathsf{seed}, S, I)$
$c \leftarrow c + \gamma$
Return $(\gamma, z)$
```

```
RoR$(\beta, addin)$
$(R_0, S) \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S, \beta, addin)$
If $c < \gamma^*$
    Return $R_0$
    $c \leftarrow 0$
Else $R_1 \leftarrow\$ \{0,1\}^{\beta}$
Return $R_b$
```

```
Get
Return $S$
$c \leftarrow 0$

Set$(S^*)$
$S \leftarrow S^*$
$c \leftarrow 0$
```

Figure 4.3: Security game $\overline{\mathrm{Rob}}$ for a PRNG $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$.

analysis is carried out is chosen randomly and independently of the entropy source, and so serves the same purpose as a seed. We take the latter approach for a number of reasons. Firstly, we wish to analyse the NIST DRBGs as they are specified and used. As such either modifying the construction or greatly restricting the number of implementations we can reason about (as per **(1)**) solely to facilitate the analysis seems counterproductive. Secondly, as pointed out in [151], generating a seed is challenging in practice due to the necessary independence from the entropy source. Moreover, given the litany of tests which approved entropy sources in NIST SP 800-90B are subjected to, it seems reasonable to assume that the pathological sources used to illustrate e.g., deterministic extraction impossibility results are unlikely to pass these tests in practice.

4.5.2 Preserving and Recovering Security in the ROM

Theorem 2.2 describes a convenient result which says that a PRNG is robust if it satisfies two simpler properties called preserving and recovering security. We will now adapt these notions to accommodate the NIST DRBGs, and reprove the result.

Recall that Pres and Rec as defined in Section 2.5 require that after a next call the resulting PRNG state is indistinguishable from a state generated via setup. This is utilised in the hybrid argument proof of Theorem 2.2, which implicitly assumes that at all points in game Rob, the ideal distribution of the state is identical to that of states output by setup. This presents two problems for the NIST DRBGs. Firstly, states for the NIST DRBGs are often easily distinguishable from those output by setup due to the reseed counter cnt which is initialised to $cnt = 1$ and iterates through each successive next call. Worse, we shall see that the state of $\mathsf{HASH\text{-}DRBG}$ takes on a completely different distribution following a next call than it does after setup and $\mathsf{refresh}$. An additional problem is that the

```
$\overline{\text{Init}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}$
$\mathsf{H} \leftarrow\!\$\; \mathcal{H}$ ; $b \leftarrow\!\$\; \{0,1\}$ ; $N \leftarrow \mathcal{N}$
$\sigma_0 \leftarrow \varepsilon$ ; $\mathsf{seed} \leftarrow\!\$\; \text{Seed}$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\$\; \mathcal{D}(\sigma_{k-1})$
If $(b = 0)$ then $S_0^* \leftarrow \mathsf{setup}^{\mathsf{H}}(\mathsf{seed}, I_1, N)$
Else $S_0^* \leftarrow\!\$\; \mathsf{M}^{\mathsf{H}}(\varepsilon)$
$b^* \leftarrow\!\$\; \mathcal{A}^{\mathsf{H}}(\mathsf{seed}, S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1}, N)$
Return $(b = b^*)$

$\overline{\text{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta}$
$\mathsf{H} \leftarrow\!\$\; \mathcal{H}$ ; $b \leftarrow\!\$\; \{0,1\}$
$\mathsf{seed} \leftarrow\!\$\; \text{Seed}$
$(S_0', I_1, \ldots, I_d, addin) \leftarrow\!\$\; \mathcal{A}^{\mathsf{H}}(\mathsf{seed})$
$S_0 \leftarrow\!\$\; \mathsf{M}^{\mathsf{H}}(S_0')$
For $i = 1, \ldots, d$
    $S_i \leftarrow \mathsf{refresh}^{\mathsf{H}}(\mathsf{seed}, S_{i-1}, I_i)$
If $(b = 0)$ then $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S_d, \beta, addin)$
Else $R^* \leftarrow\!\$\; \{0,1\}^{\beta}$ ; $S^* \leftarrow \mathsf{M}^{\mathsf{H}}(S_d)$
$b^* \leftarrow\!\$\; \mathcal{A}^{\mathsf{H}}(\mathsf{seed}, R^*, S^*)$
Return $(b = b^*)$
```

```
$\overline{\text{Rec}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}$
$\mathsf{H} \leftarrow\!\$\; \mathcal{H}$ ; $b \leftarrow\!\$\; \{0,1\}$
$\sigma_0 \leftarrow \varepsilon$ ; $\mathsf{seed} \leftarrow\!\$\; \text{Seed}$ ; $\mu \leftarrow 1$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\$\; \mathcal{D}(\sigma_{k-1})$
$(S_0, d, addin) \leftarrow\!\$\; \mathcal{A}^{\mathsf{H},\text{Sam}}(\mathsf{seed}, I_1, (\gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > (q_{\mathcal{D}} + 1)$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
For $i = 1, \ldots, d$
    $S_i \leftarrow \mathsf{refresh}^{\mathsf{H}}(\mathsf{seed}, S_{i-1}, I_{\mu+i})$
If $(b = 0)$ then $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S_d, \beta, addin)$
Else $R^* \leftarrow\!\$\; \{0,1\}^{\beta}$ ; $S^* \leftarrow\!\$\; \mathsf{M}^{\mathsf{H}}(S_d)$
$b^* \leftarrow\!\$\; \mathcal{A}(\mathsf{seed}, R^*, S^*, (I_k)_{k>\mu+d})$
Return $(b = b^*)$

Sam()
$\mu \leftarrow \mu + 1$
Return $I_\mu$

$\mathsf{H}(X)$
Return $\mathsf{H}(X)$
```

Figure 4.4: Security games $\overline{\text{Init}}$, $\overline{\text{Pres}}$, and $\overline{\text{Rec}}$ for a PRNG $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ and masking function $\mathsf{M} : \mathcal{S} \cup \{\varepsilon\} \rightarrow \mathcal{S}$.

NIST DRBGs must construct initial states from an entropy sample (as opposed to using random coins as in Theorem 2.2), and so will typically output states which themselves are an approximation of an ideal state distribution.

Having run up against similar problems during their treatment of the Intel RNG, Shrimpton and Terashima [151] made two key adaptations which we shall use here. Firstly, they introduce the notion of *masking functions* which take as input a state and output an ideally distributed variant of that state, thereby preserving predictable elements such as counters while randomising unpredictable elements such as keys. They also define a new security notion Init which models how well the states returned by setup emulate an idealised initial state distribution. This allows a bounded transition to an easier-to-analyse game in which the PRNG is initialised with an ideally random state.

For our modelling, we will adapt the definitions of Shrimpton and Terashima [151] to accommodate the NIST DRBGs, which as we shall see requires some care. We additionally extend these definitions to include a random oracle, similarly to the treatment of robustness in the IPM of [69].

Security games. Consider games $\overline{\text{Pres}}$, $\overline{\text{Rec}}$, and $\overline{\text{Init}}$ shown in Figure 4.4. To avoid further complicating our analysis the notions given here are for $\overline{\text{Rob}}_{\beta}$, the variant of $\overline{\text{Rob}}$

in which $\mathcal{A}$ always requests outputs of β-bits in RoR queries. Here we have adapted the notions of [151] in the natural way to accommodate: **(1)** a random oracle; and **(2)** our PRNG syntax. It is straightforward to extend our analysis to accommodate variable length outputs.

All games are defined with respect to a masking function, which we define formally to be a randomised function $\mathsf{M} : \mathcal{S} \cup \{\varepsilon\} \rightarrow \mathcal{S}$ where $\mathcal{S}$ denotes the state space of the PRNG. Here, we have extended the definition of [151] to include ε in the domain of the masking function (implicitly assuming that ε does not lie in the state space of the PRNG; if this is not the case then any distinguished symbol may be used instead). We discuss the reasons for this adaptation in Section 4.6. We give the masking function access to the random oracle, indicated by M^{H}. We make a number of further modifications. Firstly in $\overline{\mathrm{Init}}$, we require S_0^* to be indistinguishable from $\mathsf{M}^{\mathsf{H}}(\varepsilon)$ as opposed to $\mathsf{M}^{\mathsf{H}}(S_0^*)$ as in [151]. Secondly, during the computation of the challenge in $\overline{\mathrm{Pres}}$ and $\overline{\mathrm{Rec}}$ we apply the masking function to the state S_d which was *input* to $\mathsf{next}^{\mathsf{H}}$ as opposed to the state S^* which is *output* by $\mathsf{next}^{\mathsf{H}}$. Finally, in $\overline{\mathrm{Pres}}$ we allow $\mathcal{A}$ to output $S \in \mathcal{S} \cup \{\varepsilon\}$ at the start of his challenge rather than $S \in \mathcal{S}$. In all cases, this is to accommodate the somewhat complicated state distribution of $\mathsf{HASH\text{-}DRBG}$ (see Section 4.6).

For all $\mathrm{Gm}_y^x \in \{\overline{\mathrm{Init}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}^{\mathcal{A},\mathcal{D}}, \overline{\mathrm{Pres}}_{\mathsf{PRNG},\mathsf{M},\beta}^{\mathcal{A},}, \overline{\mathrm{Rec}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}^{\mathcal{A},\mathcal{D}}\}$, we define

$$\mathbf{Adv}_y^{\mathrm{gm}}(x) = 2 \cdot \left| \Pr\left[\,\mathrm{Gm}_y^x \Rightarrow 1\,\right] - \frac{1}{2} \right| .$$

An adversary in $\overline{\mathrm{Init}}$ is said to be a q_{H}-adversary if it makes q_{H} queries to its H oracle. An adversary in game $\overline{\mathrm{Pres}}$ or $\overline{\mathrm{Rec}}$ is said to be a (q_{H}, q_C)-adversary if it makes q_{H} queries to its H oracle and always outputs $d \leq q_C$.

With this in place, the following theorem — which says that $\overline{\mathrm{Init}}$, $\overline{\mathrm{Pres}}$, and $\overline{\mathrm{Rec}}$ security collectively imply $\overline{\mathrm{Rob}}$ security — is an adaptation of the analogous results from [69, 151]. As a bonus, employing a slightly different line of argument with two series of hybrid arguments means our proof holds for arbitrary masking functions, lifting the restriction from [151] that masking functions possess a property called idempotence.

Theorem 4.1. *Let* $\mathsf{PRNG} = (\mathsf{setup}^{\mathsf{H}}, \mathsf{refresh}^{\mathsf{H}}, \mathsf{next}^{\mathsf{H}})$ *be a PRNG with input with associated parameter set* $(p, \overline{p}, \alpha, \beta_{max})$*, built from a hash function* H *which we model as a random oracle. Suppose that each invocation of* $\mathsf{refresh}^{\mathsf{H}}$ *and* $\mathsf{next}^{\mathsf{H}}$ *makes at most* q_{ref} *and* q_{nxt} *queries to* H *respectively. Let* $\mathsf{M}^{\mathsf{H}} : \mathcal{S} \cup \{\varepsilon\} \rightarrow \mathcal{S}$ *be a masking function for which* *each invocation of* M^{H} *makes at most* q_{M} H *queries. Then for any* $(q_{\mathsf{H}}, q_R, q_{\mathcal{D}}, q_C, q_S)$*-adversary* $\mathcal{A}$ *and* $(q_{\mathcal{D}}^{+}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$ *in game* $\overline{\mathrm{Rob}}_{\beta}$ *against* PRNG*, there exists a* $(q_{\mathsf{H}} + q_{\mathcal{D}} \cdot q_{ref} + q_R \cdot q_{nxt})$*-adversary* $\mathcal{A}_1$ *and* $(q_{\mathsf{H}} + q_{\mathcal{D}} \cdot q_{ref} + q_R \cdot (q_{nxt} + q_{\mathsf{M}}), q_C)$*-adversaries* $\mathcal{A}_2, \mathcal{A}_3$*, such that*

$$\begin{aligned}\mathrm{Adv}^{\overline{\mathrm{rob}}}_{\mathsf{PRNG},\gamma^*,\beta}(\mathcal{A},\mathcal{D}) \leq 2 \cdot \mathrm{Adv}^{\overline{\mathrm{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}_1,\mathcal{D}) \\ + 2q_R \cdot \mathrm{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}_2) + 2q_R \cdot \mathrm{Adv}^{\overline{\mathrm{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A}_3,\mathcal{D})\ .\end{aligned}$$

Proof. Let $(\mathcal{A}, \mathcal{D})$ be an attacker / sampler pair in game $\overline{\mathrm{Rob}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*,\beta}$ against PRNG. We shall construct a hybrid argument based upon the attacker's RoR queries, where by assumption $\mathcal{A}$ makes q_R such queries. We say that a RoR query is uncompromised if $c \geq \gamma^*$ at the point of the query; otherwise we say it is compromised. We further divide uncompromised RoR queries into those which are *preserving* and those which are *recovering.* We say a RoR query is recovering if $c < \gamma^*$ at some point between the previous RoR query (or if there have been no previous RoR queries, the start of the game) and the current one inclusive. For a recovering RoR query, we call the most recent query for which $c \leftarrow 0$ the most recent entropy drain (mRED) query. If an uncompromised query is not recovering, then it is said to be preserving.

Let game G_0^* be equivalent to game $\overline{\mathrm{Rob}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*,\beta}$ with challenge bit $b = 0$. Let $(\mathcal{A}, \mathcal{D})$ be an attacker / sampler pair in G_0^*. We begin by defining game G_0, which is identical to G_0^* except we replace the line $S \leftarrow \mathsf{setup}^{\mathsf{H}}(\mathsf{seed}, I, N)$ with $S \leftarrow\!\$\ \mathsf{M}^{\mathsf{H}}(\varepsilon)$. We bound the gap between these games with a reduction to the $\overline{\mathrm{Init}}$ security of PRNG. Let $(\mathcal{A}_1, \mathcal{D})$ be an attacker / sampler pair in game $\overline{\mathrm{Init}}$, where $\mathcal{A}_1$ proceeds as follows. When $\mathcal{A}_1$ receives his challenge state S^*, the seed seed, and the entropy samples, estimates, side information, and nonce $((I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1}, N)$, he passes $(\mathsf{seed}, (\gamma_1, z_1, N))$ to $\mathcal{A}$. $\mathcal{A}_1$ begins to simulate game $\overline{\mathrm{Rob}}$ with challenge bit $b = 0$ for $\mathcal{A}$, using the challenge state S^* as the initial state for the simulated game. In particular, $\mathcal{A}_1$ uses the remaining entropy samples and estimates to simulate $\mathcal{A}$'s Ref calls, and uses his own H oracle to answer $\mathcal{A}$'s H queries and to simulate the $\mathsf{refresh}^{\mathsf{H}}$ and $\mathsf{next}^{\mathsf{H}}$ algorithms. At the end of the game, $\mathcal{A}_1$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{A}$ received a real state in his challenge then this perfectly simulates G_0^*; otherwise it perfectly simulates G_0. Moreover, notice that $\mathcal{A}_1$ makes at most $(q_{\mathsf{H}} + q_R \cdot q_{nxt} + q_{\mathcal{D}} \cdot q_{ref})$ queries to H (this total follows from the q_{H} queries that $\mathcal{A}$ makes to H and which $\mathcal{A}_1$ forwards to his own oracle, plus the q_{ref} (resp. q_{nxt}) H

queries which $\mathcal{A}_1$ makes to simulate the $q_{\mathcal{D}}$ (resp. q_R) Ref (resp. RoR) queries. It follows that

$$|\Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_0^*] - \Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_0]| \leq \mathrm{Adv}^{\overline{\mathrm{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}_1, \mathcal{D}) \ .$$

We now define a series of modified games, where for each $i \in [0, q_R - 1]$, G_{i+1} is identical to game G_i except in the event that the $(i+1)^{\mathrm{st}}$ RoR query is uncompromised. In this case then instead of computing the output / state via $(R, S') \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S, \beta, \mathit{addin})$ where S denotes the current state of the PRNG, the challenger returns $R \leftarrow_{\$} \{0,1\}^{\beta}$ to $\mathcal{A}$ regardless of the challenge bit and sets the state of the PRNG to $\mathsf{M}^{\mathsf{H}}(S)$ (i.e., the mask of the state which was input to next to satisfy the output request). This new state is used to run the rest of the game. We also introduce an intermediate game $G_{(i+\frac{1}{2})}$ defined such that if the $(i+1)^{\mathrm{st}}$ RoR query is preserving then the challenger acts as in Game $(i+1)$, whereas if the $(i+1)^{\mathrm{st}}$ RoR query is recovering the challenger acts as in Game i. We bound the gaps between these pairs of games in the following lemma.

Lemma 4.1. *For any $(q_{\mathsf{H}}, q_R, q_{\mathcal{D}}, q_C, q_S)$-adversary $\mathcal{A}$ and sampler $\mathcal{D}$, and for all $i \in [0, q_R - 1]$, there exist $(q_{\mathsf{H}} + q_{\mathcal{D}} \cdot q_{\mathit{ref}} + q_R \cdot (q_{\mathit{nxt}} + q_{\mathsf{M}}), q_C)$-adversaries $\mathcal{A}_2^i$, $\mathcal{A}_3^i$ such that*

$$\left|\Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_i] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+1)}\right]\right|$$
$$\leq \mathrm{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}_2^i) + \mathrm{Adv}^{\overline{\mathrm{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A}_3^i, \mathcal{D}) \ .$$

To see this, notice that for each $i \in [0, q_R - 1]$ the triangle inequality implies that the left-hand side of the above equation is upper bounded by

$$\left|\Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_i] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+\frac{1}{2})}\right]\right|$$
$$+ \left|\Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+\frac{1}{2})}\right] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+1)}\right]\right| .$$

We begin by showing that there exists an adversary $\mathcal{A}_2^i$ such that

$$\left|\Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_i] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+\frac{1}{2})}\right]\right| \leq \mathrm{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}_2^i) \ .$$

We may assume without loss of generality that the $(i+1)^{\mathrm{st}}$ query is preserving, otherwise games G_i and are $G_{(i+\frac{1}{2})}$ are identical. Attacker $\mathcal{A}_2^i$ proceeds as follows. $\mathcal{A}_2^i$ runs $\mathcal{A}$ as a subroutine, using his H oracle as well as the code of the sampler $\mathcal{D}$ to simulate game G_i up to and including the i^{th} RoR query. In more detail: on input seed seed, $\mathcal{A}_2^i$ first computes $(\sigma_1, I_1, \gamma_1, z_1) \leftarrow_{\$} \mathcal{D}(\sigma_0)$, and passes $(\mathsf{seed}, (\gamma_1, z_1, N))$ to $\mathcal{A}$ where $N \leftarrow \mathcal{N}$. $\mathcal{A}_2^i$ simulates

$\mathcal{A}$'s H oracle by querying his own H oracle and returning the response. For a Ref query, $\mathcal{A}_2^i$ uses the code of the sampler to compute $(\sigma', I, \gamma, z) \leftarrow\!\!\$\ \mathcal{D}(\sigma)$ and returns (γ, z) to $\mathcal{A}$, but does not yet update the state. For Get, Set, and RoR queries, $\mathcal{A}_2^i$ takes the state S which immediately followed the last non-Ref query (or chooses $S \leftarrow\!\!\$\ \mathsf{M}^{\mathsf{H}}(\varepsilon)$ if this is the first non-Ref query and so no such state exists) and uses $\mathsf{refresh}^{\mathsf{H}}$ to refresh that state with all entropy samples which would have been incorporated due to Ref queries since the previous call, ultimately producing state S'. For Get / Set queries, $\mathcal{A}_2^i$ returns / overwrites the state S' as required. For the first $(i-1)$ RoR queries, if the query is uncompromised then $\mathcal{A}_2^i$ samples $R \leftarrow\!\!\$\ \{0,1\}^\beta$ and sets the state of the generator to $S'' \leftarrow\!\!\$\ \mathsf{M}^{\mathsf{H}}(S')$. If the query is compromised then $\mathcal{A}_2^i$ simply computes $(R, S'') \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S', \beta, \mathit{addin})$. In both cases, $\mathcal{A}$ returns R to $\mathcal{A}$ and continues running the game with state S''. $\mathcal{A}$ simulates the i^{th} RoR query (which must be uncompromised, since we have assumed that the $(i+1)^{\text{st}}$ query is preserving) as just described, but does not mask the state S'. (Since the $(i+1)^{\text{st}}$ RoR query is preserving, we know that $\mathcal{A}$ must only make Ref queries between the i^{th} and $(i+1)^{\text{st}}$ RoR queries. Looking ahead, $\mathcal{A}_2^i$ will insert his challenge in the $(i+1)^{\text{st}}$ RoR query, and S' will be masked as part of the challenge computation.)

To simulate the $(i+1)^{\text{st}}$ RoR query with associated additional input addin, $\mathcal{A}_2^i$ passes the state S' which immediately followed the i^{th} RoR query (or $S' = \varepsilon$ if $i = 0$ and so no such state exists[1]), as well as addin and all entropy inputs $I_1, \ldots, I_d$ which were output by the simulated sampler in response to Ref queries made in between the i^{th} and $(i+1)^{\text{st}}$ RoR queries, to his challenger. Since by assumption $\mathcal{A}$ makes at most q_C consecutive Ref queries, it follows that $d \leq q_C$. $\mathcal{A}_2^i$ receives (R^*, S^*) in response and returns R^* to $\mathcal{A}$. $\mathcal{A}_2^i$ continues running the game with state S^* and the simulated algorithms $\mathsf{refresh}^{\mathsf{H}}$ and $\mathsf{next}^{\mathsf{H}}$. At the end of the game, $\mathcal{A}_2^i$ outputs whatever bit $\mathcal{A}$ does.

Notice that if $\mathcal{A}_2^i$'s challenge bit is 0 and he receives the real output and state in his challenge then this perfectly simulates G_i; otherwise he receives a random output and a masked state, and this perfectly simulates game $G_{(i+\frac{1}{2})}$. It follows that

$$\begin{aligned}
&\left| \Pr\left[\, \mathcal{A} \Rightarrow 1 \text{ in } G_i \,\right] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+\frac{1}{2})} \right] \right| \\
&\qquad\qquad = \left| \Pr\left[\, \mathcal{A}_2^i \Rightarrow 1 \mid b = 0 \,\right] - \Pr\left[\, \mathcal{A}_2^i \Rightarrow 1 \mid b = 1 \,\right] \right| \leq \mathrm{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}_2^i)\,.
\end{aligned}$$

[1]Recall that we modified $\overline{\text{Pres}}$ to allow $\mathcal{A}$ to output ε at the start of the challenge. Looking ahead, this is to allow an attacker in $\overline{\text{Pres}}$ against $\mathsf{HASH\text{-}DRBG}$ to simulate this case, since for $\mathsf{HASH\text{-}DRBG}$ states output by setup are distributed differently to those following next calls (see Section 4.6).

Moreover, notice that $\mathcal{A}_2^i$ makes at most $(q_{\mathsf{H}} + q_{\mathcal{D}} \cdot q_{ref} + q_R \cdot (q_{nxt} + q_{\mathsf{M}}))$ queries to H. Here the additional $q_R \cdot q_M$ term arises since $\mathcal{A}_2^i$ masks at most $(q_R - 1)$ PRNG states (one for every uncompromised RoR query with index $j \in [1, i]$) plus the initial state $S_0 \leftarrow\$ \mathsf{M}^{\mathsf{H}}(\varepsilon)$). The other terms arise from $\mathcal{A}_2^i$ answering $\mathcal{A}$'s H queries and simulating the $\mathsf{refresh}^{\mathsf{H}}$ and $\mathsf{next}^{\mathsf{H}}$ algorithms.

Next, we show that for all $i \in [0, q_R - 1]$ there exists an adversary $\mathcal{A}_3^i$ such that

$$\left| \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+\frac{1}{2})} \right] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+1)} \right] \right| \leq \mathrm{Adv}^{\overline{\mathrm{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A}_3^i, \mathcal{D}) \;.$$

We may again assume without loss of generality that the $(i+1)$st RoR query is recovering otherwise games $G_{(i+\frac{1}{2})}$ and $G_{(i+1)}$ are identical. Attacker $\mathcal{A}_3^i$ proceeds as follows. $\mathcal{A}_3^i$ is given seed seed, I_1, and $(\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1}$, passes $(\mathsf{seed}, \gamma_1, z_1, N)$ to $\mathcal{A}$ where $N \leftarrow \mathcal{N}$, and begins to simulate game G_i. $\mathcal{A}_3^i$ simulates $\mathcal{A}$'s oracles for the first i queries as described above with two differences. Firstly in response to a Ref query, $\mathcal{A}_3^i$ returns the relevant entropy estimate / side information (which were given to $\mathcal{A}_3^i$ at the start of his challenge) to $\mathcal{A}$. Then at the point of the next non-Ref query, $\mathcal{A}_3^i$ queries Sam repeatedly to get all entropy samples which should have been incorporated into the state during that period and uses these to update the state. (In contrast, in the previous simulation $\mathcal{A}_2^i$ ran the code of the sampler himself.) The second difference is that if the ith RoR query is uncompromised, $\mathcal{A}$ returns $R \leftarrow\$ \{0,1\}^\beta$ to $\mathcal{A}$ as before but now also sets the state to $S'' \leftarrow\$ \mathsf{M}^{\mathsf{H}}(S')$ where S' denotes the state of the PRNG as updated with entropy samples at the point of the query. (In the previous simulation $\mathcal{A}_2^i$ left this state unmasked.)

When $\mathcal{A}$ makes his $(i+1)$st RoR query with associated additional input *addin*, $\mathcal{A}_3^i$ locates the mRED query (which must exist somewhere between the ith RoR query inclusive and the $(i+1)$st RoR query exclusive since we have assumed that the $(i+1)$st RoR query is recovering). Moreover, there must only be Ref queries between the mRED query and $(i+1)$st RoR query (since any Get / Set queries would contradict the maximality of the mRED query). $\mathcal{A}_3^i$ submits the state which immediately followed the mRED query to his challenger, along with d set equal to the number of Ref calls which $\mathcal{A}$ made between the mRED query and the $(i+1)$st RoR query, and *addin*. Again since $\mathcal{A}$ makes at most q_C consecutive Ref queries, it must be the case that $d \leq q_C$ also. $\mathcal{A}_3^i$ receives (R^*, S^*) in response, along with the remaining entropy samples. $\mathcal{A}_3^i$ returns R^* to $\mathcal{A}$, and uses the state S^* along with the entropy samples to simulate the remainder of the game. At the end of the game $\mathcal{A}_3^i$ outputs whatever bit $\mathcal{A}$ does.

Notice that if $\mathcal{A}_3^i$'s challenge bit is 0 and so he receives the real output and state in his challenge then this perfectly simulates $G_{(i+\frac{1}{2})}$; otherwise, he receives a random output and a masked state and this perfectly simulates game $G_{(i+1)}$. It follows that

$$\left|\Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+\frac{1}{2})}\right] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } G_{(i+1)}\right]\right|$$
$$= \left|\Pr\left[\mathcal{A}_3^i \Rightarrow 1 \mid b = 0\right] - \Pr\left[\mathcal{A}_3^i \Rightarrow 1 \mid b = 1\right]\right| \leq \text{Adv}^{\overline{\text{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^* q_{\mathcal{D}},\beta}(\mathcal{A}_3^i, \mathcal{D}) ,$$

where an analogous argument to that made above verifies the query complexity of $\mathcal{A}_3^i$.

Now in game G_{q_R}, each of $\mathcal{A}$'s uncompromised RoR queries is answered with a random bit string and a masked state. To go back to using unmasked states while still returning random outputs to $\mathcal{A}$, we define a second sequence of hybrid games where game $\overline{G}_{q_R}$ is identical to game G_{q_R}, and for each $i \in [0, q_R - 1]$ game $\overline{G}_i$ is identical to game $\overline{G}_{(i+1)}$ except if the $(i+1)^{\text{st}}$ query is uncompromised. In this case, for the $(i+1)^{\text{st}}$ query the challenger computes $(R, S'') \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S', \beta, \mathit{addin})$ where S' denotes the state at the point of the query and returns $R \leftarrow\!\!\$\ \{0,1\}^\beta$ to the attacker, but continues running the game with the real state S'' as opposed to using the masked version of that state $\mathsf{M}^{\mathsf{H}}(S')$. We define intermediate games $\overline{G}_{(i+\frac{1}{2})}$ analogously to $G_{(i+\frac{1}{2})}$. An analogous argument to that used above then implies the following lemma.

Lemma 4.2. *For any $(q_{\mathsf{H}}, q_R, q_{\mathcal{D}}, q_C, q_S)$-adversary $\mathcal{A}$ and sampler $\mathcal{D}$, and for all $i \in [0, q_R - 1]$, there exist $(q_{\mathsf{H}} + q_{\mathcal{D}} \cdot q_{ref} + q_R \cdot (q_{nxt} + q_{\mathsf{M}}), q_C)$-adversaries $\overline{\mathcal{A}}_2^i$, $\overline{\mathcal{A}}_3^i$ such that*

$$\left|\Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } \overline{G}_i\right] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } \overline{G}_{(i+1)}\right]\right|$$
$$\leq \text{Adv}^{\overline{\text{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\overline{\mathcal{A}}_2^i) + \text{Adv}^{\overline{\text{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\overline{\mathcal{A}}_3^i, \mathcal{D}) .$$

The proof of the above lemma is identical to that in the previous case, except that now $\overline{\mathcal{A}}_2^i$ and $\overline{\mathcal{A}}_3^i$ return random outputs in response to all uncompromised RoR queries made by $\mathcal{A}$. Similarly, we let $\overline{G}_0^*$ be identical to game $\overline{G}_0$, except we compute the initial state via $S_0 \leftarrow \mathsf{setup}^{\mathsf{H}}(\mathsf{seed}, I_1, N)$ rather than as $\mathsf{M}^{\mathsf{H}}(\varepsilon)$. An analogous argument to that used above again implies that there exists an adversary $\overline{\mathcal{A}_1}$ making at most $(q_{\mathsf{H}} + q_R \cdot q_{nxt} + q_{\mathcal{D}} \cdot q_{ref})$

H queries such that

$$\left|\Pr\left[\,\mathcal{A} \Rightarrow 1 \text{ in } \overline{G}_0\,\right] - \Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } \overline{G}_0^*\right]\right| \leq \mathrm{Adv}^{\overline{\mathrm{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\overline{\mathcal{A}_1},\mathcal{D}) \ .$$

Moreover, notice that game G_0^* is identical to game Rob with challenge bit $b = 1$. Putting this altogether, a standard argument implies that there exists adversaries $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ with the claimed query budgets such that

$$\begin{aligned}\mathrm{Adv}^{\overline{\mathrm{rob}}}_{\mathsf{PRNG},\gamma^*,\beta}(\mathcal{A},\mathcal{D}) \leq 2 \cdot \mathrm{Adv}^{\overline{\mathrm{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}_1,\mathcal{D}) &\\ + 2q_R \cdot \mathrm{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}_2) + 2q_R \cdot \mathrm{Adv}^{\overline{\mathrm{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A}_3,\mathcal{D}) \ .&\end{aligned}$$

□

Tightness. Unfortunately, due to the hybrid argument taken over the q_R RoR queries made by $\mathcal{A}$, Theorem 4.1 is not tight. This is exacerbated in the ROM, since the attacker in each of the q_R hybrid reductions must make enough H queries to simulate the whole of game $\overline{\mathrm{Rob}}$ for $\mathcal{A}$. This hybrid argument accounts for the q_R coefficients in the bound and in the attacker query budgets. This seems inherent to the proof technique and is present in the analogous results of [60, 69, 151]. Developing a technique to obtain tighter bounds is an important open question.

4.6 Analysis of HASH-DRBG

We now present our analysis of the robustness of HASH-DRBG, in which the underlying hash function $\mathsf{H} : \{0,1\}^{\leq\omega} \to \{0,1\}^{\ell}$ is modelled as a random oracle. Our proof is with respect to the masking function M^{H} shown in Figure 4.5. To avoid further complicating security bounds we assume that HASH-DRBG is never called with additional input; we expect extending the proof to include additional input to be straightforward.

Challenges. Certain features of HASH-DRBG significantly complicate the proof, and necessitated adaptations in our security modelling (Section 4.5). For example, notice that the distributions of states returned by $\mathsf{setup}^{\mathsf{H}}$ and $\mathsf{refresh}^{\mathsf{H}}$ are quite different from the distribution of states S' where $(R, S') \leftarrow \mathsf{next}^{\mathsf{H}}(S, \beta)$. To model this in games $\overline{\mathrm{Init}}, \overline{\mathrm{Pres}},$

```
$\underline{\mathsf{M}^{\mathsf{H}}(S)}$
If $S = \varepsilon$
    $V' \leftarrow_{\$} \{0,1\}^L$
    $C' \leftarrow \mathsf{HASH\text{-}DRBG\_df}^{\mathsf{H}}(0\mathtt{x}00||V, L)$
    $cnt' \leftarrow 1$
Else $(V, C, cnt) \leftarrow S$
    $H \leftarrow_{\$} \{0,1\}^{\ell}$
    $V' \leftarrow (V + C + cnt + H) \bmod 2^L$
    $C' \leftarrow C$ ; $cnt' \leftarrow cnt + 1$
$S' \leftarrow (V', C', cnt')$
Return $S'$
```

Figure 4.5: Masking function for proof of Theorem 4.2

and $\overline{\text{Rec}}$ we extended the domain of M to include the empty string ε to indicate that an idealised state of the first form should be returned (for example, when modelling initial state generation in $\overline{\text{Init}}$). We also modified $\overline{\text{Pres}}$ to allow $\mathcal{A}$ to output ε at the start of the challenge. This is because the proof of Theorem 4.1 requires $\mathcal{A}$ to be able to insert his $\overline{\text{Pres}}$ challenge in any RoR query in a simulation of game Rob. To simulate the first RoR query, $\mathcal{A}$ must be able to request a masked state of the form returned by setup at the start of his challenge; for all other queries, he requires a masked state of the type returned by next. Since for HASH-DRBG these states are distributed differently, we must distinguish between the two in $\overline{\text{Pres}}$. Juggling these different state distributions complicates proofs, in particular introducing multiple cases into the proof of $\overline{\text{Pres}}$ security.

Moreover, consider the distribution of $S' \leftarrow_{\$} \mathsf{M}^{\mathsf{H}}(S)$ for $S \in \mathcal{S}$, which serves to idealise the distribution of the state $S' = (V', C', cnt')$ as updated following an output generation request. It would be convenient to argue that V' is uniformly distributed over $\{0,1\}^L$. However, since V' is chosen uniformly from the window $[V + C + cnt, V + C + cnt + (2^{\ell} - 1)]$ where $S = (V, C, cnt)$ and $L > \ell + 1$ (see below), this is clearly not the case. To accommodate this dependency between the updated state S' and the previous state S, we have modifed games $\overline{\text{Pres}}$ and $\overline{\text{Rec}}$ so that it is S which is masked instead of S'. More minor issues, such as: **(1)** not properly separating the domain of queries made by $\mathsf{setup}^{\mathsf{H}}$ to produce the counter V from those made to produce the constant C; and **(2)** the way in which L is not a multiple of ℓ for the approved hash functions; make certain steps in the analysis more fiddly than they might have been.

Parameter settings. We provide a general treatment into which any parameter setting may be slotted, subject to two restrictions which are utilised in the proof. Namely, we assume that $L > \ell + 1$ and $n < 2^L$, where $n = \lceil \beta/\ell \rceil$ denotes the number of output blocks produced by $\mathsf{next}^{\mathsf{H}}$ to satisfy a request for β-bits (without this latter restriction HASH-DRBG is trivially insecure, since it would lead to the same counter being hashed twice during output production). We additionally require that $L < 2^{32}$ and $m < 2^8$ where $|V| = |C| = L$ and $m = \lceil L/\ell \rceil$ is the number of blocks hashed by $\mathsf{setup}^{\mathsf{H}}$ / $\mathsf{refresh}^{\mathsf{H}}$ to produce a new counter. This is because these values have to be encoded as a 32-bit and an 8-bit string, respectively, by HASH-DRBG_df. All approved hash functions fall well within these parameters. (Indeed, for all of these $L > 2\ell$, $n < 3277 \ll 2^L$, and $m \leq 3$.)

Proof of $\overline{\textbf{Rob}}$ security. With this in place, we present the following theorem bounding the robustness of HASH-DRBG. The proof follows from a number of lemmas presented below, combined with Theorem 4.1. (When calculating the query budgets for Theorem 4.1, it is readily verified that for HASH-DRBG $q_{nxt} = n + 1$, $q_{ref} = 2m$, and $q_{\mathsf{M}} = m$.)

Theorem 4.2. *Let* PRNG *be* HASH-DRBG *with associated parameter set* $(p, \overline{p}, \alpha, \beta_{max})$*, built from a hash function* $\mathsf{H} : \{0,1\}^{\leq \omega} \to \{0,1\}^{\ell}$ *which we model as a random oracle. Let* L *denote the state length of* HASH-DRBG *where* $L > \ell + 1$, $n = \lceil \beta/\ell \rceil$, *and* $m = \lceil L/\ell \rceil$. *Let* M^{H} *denote the masking function shown in Figure 4.5 and suppose that* HASH-DRBG *is never called with additional input. Then for any* $(q_{\mathsf{H}}, q_R, q_{\mathcal{D}}, q_C, q_S)$*-attacker* $\mathcal{A}$ *in game* $\overline{\mathrm{Rob}}_{\beta}$ *against* PRNG*, and any* $(q_{\mathcal{D}}^{+}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$*, it holds that*

$$\mathbf{Adv}^{\overline{\mathrm{rob}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,\beta}(\mathcal{A}, \mathcal{D}) \leq \frac{q_R \cdot \overline{q}_{\mathsf{H}} + 2q'_{\mathsf{H}}}{2^{\gamma^*-2}} + \frac{q_R \cdot \overline{q}_{\mathsf{H}} \cdot (2n+1)}{2^{\ell-2}} + \frac{q_R \cdot ((q_C - 1)(2\overline{q}_{\mathsf{H}} + q_C) + \overline{q}_{\mathsf{H}}^2) + 2}{2^{L-2}},$$

Moreover, $q'_{\mathsf{H}} = (q_{\mathsf{H}} + 2m \cdot q_{\mathcal{D}} + (n+1) \cdot q_R)$ *and* $\overline{q}_{\mathsf{H}} = q'_{\mathsf{H}} + m \cdot q_R$.

$\overline{\textbf{Init}}$ security. We begin by bounding the $\overline{\mathrm{Init}}$ security of HASH-DRBG. The $q_{\mathsf{H}} \cdot 2^{-\gamma^*}$ term arises following the standard argument that the initial state variable V_0 will be indistinguishable from a truly random bit string unless the attacker queries the random oracle H on one of the points which was hashed to produce it. This in turn requires $\mathcal{A}$ to guess the value of the entropy sample I_1, which contains γ^*-bits of entropy. The additional 2^{-L} term arises since the queries made to compute the counter V_0 are not fully domain separated from those made to compute the constant C_0. Indeed, if it so happens that $I_1 || N = \text{0x00} || V_0$ where I_1 and N denote the entropy input and nonce (an event which — while very unlikely — is not precluded by the parameter constraints in the standard), then the derived values of V_0 and C_0 will be equal; this allows the attacker to distinguish

```
proc. main // $G_0, G_1, \boxed{G_2}$
$\sigma_0 \leftarrow \varepsilon$ ; $N \leftarrow \mathcal{N}$ ; $\mathcal{Q} \leftarrow \emptyset$
For $i = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_i, I_i, \gamma_i, z_i) \leftarrow\!\$\ \mathcal{D}(\sigma_{i-1})$
$temp_V \leftarrow \varepsilon$ ; $m \leftarrow \lceil L/\ell \rceil$
For $i = 1, \ldots, m$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup ((i)_8 \,\|\, (L)_{32} \,\|\, I_1 \,\|\, N)$
    $temp_V \leftarrow temp_V \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, I_1 \,\|\, N)$
$V_0^* \leftarrow \mathrm{left}(temp_V, L)$
$\boxed{V_0^* \leftarrow\!\$\ \{0,1\}^L}$
$temp_C \leftarrow \varepsilon$
For $i = 1, \ldots, m$
    $temp_C \leftarrow temp_C \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x00} \,\|\, V_0^*)$
$C_0^* \leftarrow \mathrm{left}(temp_C, L)$
$cnt_0^* \leftarrow 1$
$S_0^* \leftarrow (V_0^*, C_0^*, cnt_0^*)$
$b^* \leftarrow\!\$\ \mathcal{A}^{\mathsf{H}}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
Return $b^*$
```

```
proc. main // $G_3, G_4$
$\sigma_0 \leftarrow \varepsilon$ ; $N \leftarrow \mathcal{N}$ ; $\mathcal{Q} \leftarrow \emptyset$
For $i = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_i, I_i, \gamma_i, z_i) \leftarrow\!\$\ \mathcal{D}(\sigma_{i-1})$
$S_0^* \leftarrow\!\$\ \mathsf{M}^{\mathsf{H}}(\varepsilon)$
$b^* \leftarrow\!\$\ \mathcal{A}^{\mathsf{H}}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
Return $b^*$
```

```
proc. $\mathsf{H}(X)$ // $\boxed{G_0}, G_1, G_2, G_3, \boxed{G_4}$
$Y \leftarrow\!\$\ \{0,1\}^\ell$
If $\mathsf{H}[X] \neq \perp$ and $X \notin \mathcal{Q}$
    $Y \leftarrow \mathsf{H}[X]$
If $\mathsf{H}[X] \neq \perp$ and $X \in \mathcal{Q}$
    bad $\leftarrow 1$
    $\boxed{Y \leftarrow \mathsf{H}[X]}$
$\mathsf{H}[X] \leftarrow Y$
Return $Y$
```

Figure 4.6: Games for proof of Lemma 4.3 ($\overline{\text{Init}}$ security of HASH-DRBG).

the real state from $\mathsf{M}^{\mathsf{H}}(\varepsilon)$ with high probability. A small tweak to the design of setup (e.g., prepending 0x01 to $I||N$ before hashing) would have avoided this. For implementations of HASH-DRBG for which such a collision is impossible (e.g., due to length restrictions on the input I) this additional term can be removed.

Lemma 4.3. *Let* PRNG = HASH-DRBG *and masking function* M^{H} *be as specified in Theorem 4.2. Then for any* q_{H}*-adversary* $\mathcal{A}$ *in game* $\overline{\text{Init}}$ *against* PRNG*, and any* $(q_{\mathcal{D}}^{+}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$*, it holds that*

$$\mathbf{Adv}^{\overline{\text{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}, \mathcal{D}) \leq q_{\mathsf{H}} \cdot 2^{-\gamma^*} + 2^{-L} .$$

Proof. We argue by a series of game hops, shown in Figure 4.6. We begin by defining game G_0, which is easily verified to be a rewriting of $\overline{\text{Init}}$ for HASH-DRBG and M^{H} with challenge bit $b = 0$ in which we lazily sample the random oracle H. We additionally set a flag bad, although this does not affect the outcome of the game. It follows that

$$\Pr[G_0 \Rightarrow 1] = \Pr\left[\overline{\text{Init}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*} \Rightarrow 1 \mid b = 0\right] .$$

Next we define game G_1, which is identical to game G_0 except we change the way in which the random oracle H responds to queries. Namely, if H is queried more than once on one of the inputs $(i)_8 \,\|\, (L)_{32} \,\|\, I_1 \,\|\, N$ for $i \in [1, m]$ upon which it was queried to produce V_0^* (indicated in the pseudocode by the set $\mathcal{Q}$), then H responds with an independent random string rather than the value which was previously set. Notice that this event could either occur during the computation of C_0^* (if it happens that $\mathtt{0x00} \,\|\, V_0^* = I_1 \,\|\, N$), or due to a query made by $\mathcal{A}$ after receiving the challenge state. (Notice that due to the prepended counter, the queries made during the computation of V_0^* (resp. C_0^*) will never collide with each other.) These games run identically unless the flag bad is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad} = 1 \text{ in } G_1] .$$

We let Coll denote the event that $\mathtt{0x00} \,\|\, V_0^* = I_1 \,\|\, N$. Notice that if Coll occurs, then bad will be set with probability one during the computation of C_0^*. Moreover, notice that if Coll does not occur then bad will only be set if $\mathcal{A}$ queries H on a point in the set $\mathcal{Q}$, where any such query requires $\mathcal{A}$ to correctly guess the value of the entropy sample I_1. It follows that

$$\begin{aligned}
\Pr[\mathsf{bad} = 1 \text{ in } G_1] &= \Pr[\mathsf{bad} = 1 \wedge \mathsf{Coll} \text{ in } G_1] + \Pr[\mathsf{bad} = 1 \wedge \neg\mathsf{Coll} \text{ in } G_1] \\
&\leq \Pr[\mathsf{Coll} \text{ in } G_1] + \Pr[\mathcal{A} \text{ queries } \mathsf{H} \text{ on a point } X \in \mathcal{Q} \text{ in } G_1] \\
&\leq \Pr[\mathsf{Coll} \text{ in } G_1] + \Pr\left[\mathcal{A}^{\mathsf{H}}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1}) \text{ guesses the value of } I_1 \text{ in } G_1\right] .
\end{aligned}$$

Before bounding this probability we first define game G_2, which is identical to G_1 except we overwrite the string V_0^* computed by querying H with an independent random string $V_0^* \leftarrow\$ \{0,1\}^L$. In G_1, random oracle H responds to each query made to compute V_0^* with an independent random string (which is used at no other point in the game), and so it is straightforward to verify that these games are identically distributed. It follows that $\Pr[G_1 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1]$, and so:

$$\begin{aligned}
&\Pr[\mathsf{bad} = 1 \text{ in } G_1] \\
&\quad \leq \Pr[\mathsf{Coll} \text{ in } G_1] + \Pr\left[\mathcal{A}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1}) \text{ guesses the value of } I_1 \text{ in } G_1\right] \\
&\quad = \Pr[\mathsf{Coll} \text{ in } G_2] + \Pr\left[\mathcal{A}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1}) \text{ guesses the value of } I_1 \text{ in } G_2\right] \\
&\qquad \leq 2^{-L} + q_{\mathsf{H}} \cdot 2^{-\gamma^*} .
\end{aligned}$$

All probabilities are over the coins of $\mathcal{A}$, $\mathcal{D}$, M, and H. The first term in the inequality follows since $V_0^* \leftarrow\$ \{0,1\}^L$, and so the probability that Coll occurs is upper bounded by 2^{-L}. The second term follows since in G_2 the challenge state S_0^* returned to $\mathcal{A}$ is entirely independent of the input I_1. As such, by the $(q_{\mathcal{D}}^+, \gamma^*)$-legitimacy of the sampler, the probability that a single query to H made by $\mathcal{A}$ contains the correct value of I_1 conditioned on the given information is upper bounded by $2^{-\gamma^*}$. Taking a union bound over $\mathcal{A}$'s q_{H} queries then completes the argument.

Next we first define game G_3, which is the same as G_2 except we simply compute the challenge state S_0^* as $S_0^* \leftarrow_{\$} \mathsf{M}^{\mathsf{H}}(\varepsilon)$. It is straightforward to verify that S_0^* is computed identically in both games, and that the redundant H queries from G_2 which are removed in G_3 do not alter the outcome of the game. It follows that this is a syntactic change, and so $\Pr[G_2 \Rightarrow 1] = \Pr[G_3 \Rightarrow 1]$.

Next we define game G_4, which is the same as G_3 except we revert the random oracle to answer consistently on all points. However, V_0^* is chosen independently at random in both G_3 and G_4 and so the values upon which the random oracle would 'lie' are not set (e.g., $\mathcal{Q} = \emptyset$). As such, this does not alter the distribution of the game and so $\Pr[G_3 \Rightarrow 1] = \Pr[G_4 \Rightarrow 1]$. Now, G_4 is identical to a rewriting of game $\overline{\text{Init}}$ with challenge bit $b = 1$, and so

$$\Pr[G_4 \Rightarrow 1] = \Pr\left[\overline{\text{Init}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}} \Rightarrow 1 \mid b = 1\right] .$$

Putting this altogether, we conclude that

$$\begin{aligned}\mathbf{Adv}^{\overline{\text{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) &\leq |\Pr[G_0 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1] \\ &\leq q_{\mathsf{H}} \cdot 2^{-\gamma^*} + 2^{-L} .\end{aligned}$$

□

A useful lemma: Next security of HASH-DRBG. Consider the game $\text{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta}$ shown in the top left panel of Figure 4.7, in which we have assumed that PRNG is not called with additional input. In Next, the attacker $\mathcal{A}$ first chooses a state which is then masked. Depending on the challenge bit, the game either uses the next algorithm of the PRNG to generate a real output and state or chooses a random output and updated masked state; the attacker is challenged to distinguish between these two cases. The advantage of attacker $\mathcal{A}$ in game Next is defined

$$\mathbf{Adv}^{\text{next}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) = \left|\Pr\left[\text{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 0\right] - \Pr\left[\text{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 1\right]\right| .$$

This game isolates the output production portion of games $\overline{\text{Pres}}$ and $\overline{\text{Rec}}$. We first prove the Next security of $\mathsf{HASH\text{-}DRBG}$ with respect to the masking function M^{H}, which in turn allows us to treat the proofs of $\overline{\text{Pres}}$ and $\overline{\text{Rec}}$ security in a more modular manner.

Lemma 4.4. *Let* $\mathsf{PRNG} = \mathsf{HASH\text{-}DRBG}$ *and masking function* M^{H} *be as specified in*

```
$\mathrm{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta}$
$\mathsf{H} \leftarrow\!\!\$\; \mathcal{H}$ ; $b \leftarrow\!\!\$\; \{0,1\}$
$\mathsf{seed} \leftarrow\!\!\$\; \mathrm{Seed}$
$S' \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}(\mathsf{seed})$ ; $S \leftarrow\!\!\$\; \mathsf{M}^{\mathsf{H}}(S')$
If $b = 0$
  $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(\mathsf{seed}, S, \beta)$
Else $R^* \leftarrow\!\!\$\; \{0,1\}^{\beta}$ ; $S^* \leftarrow\!\!\$\; \mathsf{M}^{\mathsf{H}}(S)$
$b^* \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}(\mathsf{seed}, R^*, S^*)$
Return $(b = b^*)$
```

```
proc. main // $G_0, G_1, \boxed{G_2}$
$S' \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}$ ; $(V', C', cnt') \leftarrow S'$
$H_0 \leftarrow\!\!\$\; \{0,1\}^{\ell}$ ; $V \leftarrow V' + C' + cnt' + H_0$
$C \leftarrow C'$ ; $cnt \leftarrow cnt' + 1$
$S \leftarrow (V, C, cnt)$
$data \leftarrow V$ ; $temp_R \leftarrow \varepsilon$ ; $n \leftarrow \lceil \beta/\ell \rceil$
For $j = 1, \ldots, n$
  $r \leftarrow \mathsf{H}(data)$
  $data \leftarrow data + 1$
  $temp_R \leftarrow temp_R \,\|\, r$
$R^* \leftarrow \mathrm{left}(temp_R, \beta)$
$\boxed{R^* \leftarrow\!\!\$\; \{0,1\}^{\beta}}$
$H_1 \leftarrow \mathsf{H}(\mathtt{0x03} \,\|\, V)$
$\boxed{H_1 \leftarrow\!\!\$\; \{0,1\}^{\ell}}$
$V^* \leftarrow V + C + cnt + H_1$
$C^* \leftarrow C$ ; $cnt^* \leftarrow cnt + 1$
$S^* \leftarrow (V^*, C^*, cnt^*)$
$b^* \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. main // $G_3, G_4$
$S' \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}$ ; $(V', C', cnt') \leftarrow S'$
$H_0 \leftarrow\!\!\$\; \{0,1\}^{\ell}$ ; $V \leftarrow V' + C' + cnt' + H_0$
$C \leftarrow C'$ ; $cnt \leftarrow cnt' + 1$
$S \leftarrow (V, C, cnt)$
$R^* \leftarrow\!\!\$\; \{0,1\}^{\beta}$
$S^* \leftarrow\!\!\$\; \mathsf{M}^{\mathsf{H}}(S)$
$b^* \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. $\mathsf{H}(X)$ // $\boxed{G_0}, G_1, G_2, G_3, \boxed{G_4}$
$Y \leftarrow\!\!\$\; \{0,1\}^{\ell}$
If $\mathsf{H}[X] \neq \perp$
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $\boxed{Y \leftarrow \mathsf{H}[X]}$
$H[X] \leftarrow Y$
Return $Y$
```

```
proc. main // $G_0, G_1, \boxed{G_2}$
$\varepsilon \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}$
$V \leftarrow\!\!\$\; \{0,1\}^{L}$
$m \leftarrow \lceil L/\ell \rceil$ ; $temp_C \leftarrow \varepsilon$
For $i = 1, \ldots, m$
  $temp_C \leftarrow temp_C \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x00} \,\|\, V)$
$C \leftarrow \mathrm{left}(temp_C, L)$
$\boxed{C \leftarrow\!\!\$\; \{0,1\}^{L}}$
$cnt \leftarrow 1$
$S \leftarrow (V, C, cnt)$
$data \leftarrow V$ ; $temp_R \leftarrow \varepsilon$ ; $n \leftarrow \lceil \beta/\ell \rceil$
For $j = 1, \ldots, n$
  $r \leftarrow \mathsf{H}(data)$
  $data \leftarrow data + 1$
  $temp_R \leftarrow temp_R \,\|\, r$
$R^* \leftarrow \mathrm{left}(temp_R, \beta)$
$\boxed{R^* \leftarrow\!\!\$\; \{0,1\}^{\beta}}$
$H_1 \leftarrow \mathsf{H}(\mathtt{0x03} \,\|\, V)$
$\boxed{H_1 \leftarrow\!\!\$\; \{0,1\}^{\ell}}$
$V^* \leftarrow V + C + cnt + H_1$
$C^* \leftarrow C$ ; $cnt^* \leftarrow cnt + 1$
$S^* \leftarrow (V^*, C^*, cnt^*)$
$b^* \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. main // $\boxed{G_3}, G_4, G_5$
$\varepsilon \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}$
$V \leftarrow\!\!\$\; \{0,1\}^{L}$
$m \leftarrow \lceil L/\ell \rceil$ ; $temp_C \leftarrow \varepsilon$
For $i = 1, \ldots, m$
  $temp_C \leftarrow temp_C \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x00} \,\|\, V)$
$C \leftarrow \mathrm{left}(temp_C, L)$
$\boxed{C \leftarrow\!\!\$\; \{0,1\}^{L}}$
$cnt \leftarrow 1$
$S \leftarrow (V, C, cnt)$
$R^* \leftarrow\!\!\$\; \{0,1\}^{\beta}$
$S^* \leftarrow\!\!\$\; \mathsf{M}^{\mathsf{H}}(S)$
$b^* \leftarrow\!\!\$\; \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. $\mathsf{H}(X)$ // $\boxed{G_0}, G_1, G_2, G_3, G_4, \boxed{G_5}$
$Y \leftarrow\!\!\$\; \{0,1\}^{\ell}$
If $\mathsf{H}[X] \neq \perp$
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $\boxed{Y \leftarrow \mathsf{H}[X]}$
$H[X] \leftarrow Y$
Return $Y$
```

Figure 4.7: Game Next for a PRNG PRNG and masking function M (top left panel), and games for proof of Lemma 4.4 (Next security of HASH-DRBG). All addition is modulo 2^L.

Theorem 4.2. Let $\mathsf{H} : \{0,1\}^{\leq \omega} \to \{0,1\}^{\ell}$ *and* $n = \lceil \beta/\ell \rceil$*. Then for any adversary* $\mathcal{A}$ *in game* Next *against* PRNG *making* q_{H} *queries to the random oracle* H*, it holds that*

$$\mathbf{Adv}^{\mathrm{next}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) \leq \frac{q_{\mathsf{H}} \cdot n}{2^{\ell-1}} \ .$$

Proof. We may assume without loss of generality that $\mathcal{A}$ never repeats a query to the random oracle H. Recall that $\mathsf{M}^{\mathsf{H}}(S')$ for $S' \in \mathcal{S}$ is distributed differently from $\mathsf{M}^{\mathsf{H}}(\varepsilon)$, and so there are two cases to consider depending on which type of state $\mathcal{A}$ outputs at the start of the challenge.

We begin by proving the case in which $\mathcal{A}$ outputs $S' \neq \varepsilon$. We argue by a series of game hops, shown in the left-hand panel of Figure 4.7. We begin by defining game G_0, which is a rewriting of game Next for PRNG and M^{H} with challenge bit $b = 0$ in which the random oracle is lazily sampled. In particular, notice how the procedure for computing masked states via application of M^{H} is included in the pseudocode at the appropriate point. We additionally set a flag bad but this does not affect the outcome of the game. It follows that

$$\Pr[\, G_0 \Rightarrow 1\,] = \Pr\left[\, \mathrm{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \;\middle|\; b = 0 \,\right] \ .$$

Next we define game G_1, which is identical to G_0 except we change the way in which the random oracle H responds to queries. Now H responds to all queries with an independent random string regardless of whether the query is fresh. These games run identically until the flag bad is set. It follows that

$$|\Pr[\, G_0 \Rightarrow 1\,] - \Pr[\, G_1 \Rightarrow 1\,]| \leq \Pr[\, \mathsf{bad} = 1 \text{ in } G_1\,] \ .$$

Next we define game G_2, which is identical to G_1 except we overwrite the values of R^* and H_1 (which were previously computed by querying H) with independent random bit strings $R^* \leftarrow\!\!\$\ \{0,1\}^{\beta}$ and $H_1 \leftarrow\!\!\$\ \{0,1\}^{\ell}$. Since in G_1 the random oracle responds to each query made to compute these values with an independent random string which is used nowhere else in the game, it follows that these games are identically distributed, and so

$$\Pr[\, G_1 \Rightarrow 1\,] = \Pr[\, G_2 \Rightarrow 1\,] \ ; \text{ and}$$

$$\Pr[\, \mathsf{bad} = 1 \text{ in } G_1\,] = \Pr[\, \mathsf{bad} = 1 \text{ in } G_2\,] \ .$$

We now bound the probability of bad being set in G_2. During the computation of the challenge, H is queried on : **(1)** $data + j - 1 = V + j - 1$ for $j \in [1, n]$ during the computation of R^*; and **(2)** $\mathrm{0x03} \,\|\, V$ to generate the value H_1 (which is used to produce the final state variable V^*). Notice that since the former is shorter in length than the latter, no queries of the form **(1)** can collide with the query of form **(2)**. Moreover, since $n < 2^L$, none of the queries of form **(1)** can collide with each other. Since we have assumed that $\mathcal{A}$ never repeats a query, it follows that the only way that bad will be set is if an H query made by $\mathcal{A}$ in either the initial querying phase (i.e., before outputting S'), or after receipt of the challenge output / state pair, collides with one of the values queried by the challenger. Moreover, notice that all queries which may cause bad to be set require $\mathcal{A}$ to correctly guess the value of an element in the set $\mathcal{V} = \{V, \ldots, V + n - 1\}$. We now bound the probability that $\mathcal{A}$ submits such a query to H. Since knowledge of the challenge state S^* can only increase $\mathcal{A}$'s guessing probability, we may without loss of generality assume that $\mathcal{A}$ makes all of his q_{H} queries to H after receiving his challenge. It follows that

$$\begin{aligned}\Pr\left[\,\mathsf{bad} = 1 \text{ in } G_2\,\right] &= \Pr\left[\mathcal{A}^{\mathsf{H}}(R^*, S^*) \text{ queries a point in } \mathcal{V} \text{ to } \mathsf{H} \text{ in } G_2\right]\\ &\leq \sum_{j=1}^{n} \Pr\left[\mathcal{A}^{\mathsf{H}}(R^*, S^*) \text{ guesses the value of } V + j - 1 \text{ in } G_2\right]\\ &= n \cdot \Pr\left[\mathcal{A}^{\mathsf{H}}(R^*, S^*) \text{ guesses the value of } V \text{ in } G_2\right]. \qquad (4.1)\end{aligned}$$

Here the first inequality follows from taking a union bound over the n elements in $\mathcal{V}$. The following equality follows since guessing the value of $(V + j - 1) \bmod 2^L$ for some $j \in [1, n]$ is equivalent to guessing $V \bmod 2^L$.

In particular, notice that while R^* is chosen independently at random in G_2 and so offers $\mathcal{A}$ no assistance in guessing the value of the required state variables, $S^* = (V^*, C^*, \mathit{cnt}^*)$ *does* leak some information to $\mathcal{A}$ about the value of V. To see this, notice that V is computed as $V = V' + C' + \mathit{cnt}' + H_0$ where $H_0 \leftarrow_{\$} \{0,1\}^\ell$, $S' = (V', C', \mathit{cnt}')$ is the state output by $\mathcal{A}$ at the start of the game, and all addition is modulo 2^L. The challenge state component V^* is then computed as $V^* = V + C + \mathit{cnt} + H_1$ where $C = C'$, $\mathit{cnt} = \mathit{cnt}' + 1$, and $H_1 \leftarrow_{\$} \{0,1\}^\ell$. As such, we may rewrite V^* in the form

$$V^* = V' + 2 \cdot C' + (2 \cdot \mathit{cnt}' + 1) + (H_0 + H_1)\,,$$

where recall all variables and sums are taken modulo 2^L. Now, all the variables on the right-hand side of the above equality are known to $\mathcal{A}$ *except* H_0 and H_1; indeed, $\mathcal{A}$ selected V', C' and cnt' at the start of the game. As such, learning V^* reveals $(H_0 + H_1) \in [0, 2 \cdot (2^\ell - 1)]$ to $\mathcal{A}$. It is then straightforward to verify that guessing the value of V given $R^*, S' = (V', C', \mathit{cnt}')$, and $S^* = (V^*, C^*, \mathit{cnt}^*)$ where recall $C^* = C'$ and $\mathit{cnt}^* = \mathit{cnt}' + 2$

— that is to say, $\mathcal{A}$'s view of the experiment at this point — is equivalent to guessing the value of H_0 given $(H_0 + H_1)$, where $H_0, H_1 \leftarrow\!\!\$\, \{0,1\}^\ell$. More formally, we write e.g., S^* to denote the distribution of state S^*, and for brevity let $\alpha = V' + 2 \cdot C' + (2 \cdot cnt' + 1)$. It follows that for each $S' = (V', C', cnt')$ which may be output by $\mathcal{A}$, it holds that

$$
\begin{aligned}
& 2^{-\tilde{\mathrm{H}}_\infty(\mathrm{V}|\mathrm{R}^*,\mathrm{S}^*,\mathrm{S}'=S')} \\
&= \sum_{R^*,S^*} \max_V \Pr\left[\, \mathrm{V} = V \mid (\mathrm{R}^*,\mathrm{S}^*,\mathrm{S}') = (R^*,S^*,S') \,\right] \times \Pr\left[\, (\mathrm{R}^*,\mathrm{S}^*,\mathrm{S}') = (R^*,S^*,S') \,\right] \\
&= \sum_{R^*,S^*} \max_V \Pr\left[\, \mathrm{V} = V \wedge (\mathrm{R}^*,\mathrm{S}^*,\mathrm{S}') = (R^*,S^*,S') \,\right] \\
&= \sum_{V^* \in [\alpha, \alpha + 2\cdot(2^\ell - 1)]} \max_V \Pr\left[\, V = (V' + C' + cnt' + \mathrm{H}_0) \wedge V^* = \alpha + (\mathrm{H}_0 + \mathrm{H}_1) \,\right] \\
&= \sum_{z \in [0, 2\cdot(2^\ell - 1)]} \max_{H_0} \Pr\left[\, \mathrm{H}_0 = H_0 \wedge (\mathrm{H}_0 + \mathrm{H}_1) = z \,\right] \\
&= \sum_{z \in [0, 2\cdot(2^\ell - 1)]} \max_{H_0} \Pr\left[\, (H_0 + \mathrm{H}_1) = z \,\right] \cdot \Pr\left[\, \mathrm{H}_0 = H_0 \,\right] \\
&= \sum_{z \in [0, 2\cdot(2^\ell - 1)]} 2^{-\ell} \cdot 2^{-\ell} < 2^{-(\ell-1)} \;.
\end{aligned}
$$

All probabilities are over the random choice of $H_0, H_1 \leftarrow\!\!\$\, \{0,1\}^\ell$. The first equality follows from the definition of average-case min-entropy. The second equality follows from rearranging. The third equality follows by: **(1)** rewriting V and S^* in terms of $S' = (V', C', cnt')$ and noticing that V^* is the only element of S^* not determined by S'; and **(2)** noting that, since R^* is chosen randomly in G_2, it is independent of the distribution of state variables. The next equality follows since, with S' fixed, the values of V and V^* are completely determined by $H_0 \in [1, 2^\ell - 1]$ and $(H_0 + H_1) \in [0, 2 \cdot (2^\ell - 1)]$. The following equality is implied by rearranging. The last equality follows since for each $H_0 \in \{0,1\}^\ell$ and $z \in [0, 2 \cdot (2^\ell - 1)]$, there is at most one $H_1 \in \{0,1\}^\ell$ such that $(H_0 + H_1) = z$. (Here we use the fact that $H_0, H_1 \in [0, 2^\ell - 1]$ and by assumption $L > \ell + 1$, and so no wraparound modulo 2^L occurs when computing $(H_0 + H_1)$.) The above entropy argument implies that the probability that $\mathcal{A}$ correctly guesses the value of V given a single guess in G_2 is upper bounded by $2^{-(\ell-1)}$, and so taking a union bound over $\mathcal{A}$'s q_{H} queries and substituting into equation (4.1) yields

$$
\Pr\left[\, \mathsf{bad} \text{ in } G_2 \,\right] \le \frac{q_{\mathsf{H}} \cdot n}{2^{\ell-1}} \;.
$$

Next we define G_3, which is identical to G_2 except we compute S^* as $S^* \leftarrow\!\!\$\, \mathsf{M}^{\mathsf{H}}(S)$ and omit the now redundant random oracle queries. It is straightforward to verify that these are syntactic changes, and so $\Pr[G_2 \Rightarrow 1] = \Pr[G_3 \Rightarrow 1]$.

Next we define game G_4, which is identical to G_3 except we return H to answer truthfully on all points; by an analogous argument to that used above, these games run identically unless the flag bad is set. However, since the challenge output / state are generated randomly in both games (and so no random oracle queries are made during the computation of the challenge), it follows that bad will never be set and so this change does not alter the distribution of the game. Now G_4 is identical to game Next for PRNG with challenge bit $b = 1$, and so it follows that

$$\Pr[G_4 \Rightarrow 1] = \Pr\left[\text{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 1\right] .$$

Putting this altogether, we conclude that

$$\begin{aligned}\mathbf{Adv}^{\text{next}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) &= |\Pr[G_0 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \\ &\leq \frac{q_{\mathsf{H}} \cdot n}{2^{\ell-1}} .\end{aligned}$$

We now consider the case in which $\mathcal{A}$ outputs state $S_0' = \varepsilon$ at the start of game Next. In this case, the masking function chooses $V \leftarrow\!\!\$ \{0,1\}^L$, sets $C = \mathsf{HASH\text{-}DRBG\_df}^{\mathsf{H}}(0\text{x}00 \,\|\, V)$ and $cnt = 1$, and returns $S = (V, C, cnt)$. As we will see, the success probability of an attacker in this case is less than that in the former case, and so the previous bound holds for both cases.

We argue by a series of game hops, shown on the right-hand side of Figure 4.7. We begin by defining game G_0, which is a rewriting of game Next with challenge bit $b = 0$ for $\mathsf{HASH\text{-}DRBG}$ and M^{H} with a lazily sampled random oracle. Next we define game G_1, which is identical to G_0 except we change H to respond to each query with an independent random string regardless of whether the value has previously been set. These games run identically until the flag bad is set. An analogous argument to that used above, invoking the Fundamental Lemma of Game Playing, implies that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad} = 1 \text{ in } G_1] .$$

Next we define game G_2, which is identical to G_1 except we now overwrite the values of C, R^*, and H_1 (which were previously computed by querying H) with independent random bit strings $C \leftarrow\!\!\$ \{0,1\}^L$, $R^* \leftarrow\!\!\$ \{0,1\}^\beta$, and $H_1 \leftarrow\!\!\$ \{0,1\}^\ell$. As in the proof of the previous case, both games are identically distributed and so

$$\Pr[G_1 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1] \text{ and ;}$$

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_1\,] = \Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,]\ .$$

We now bound this latter probability. During the computation of the challenge, H is queried on: **(1)** $(i)_8 \,\|\, (L)_{32} \,\|\, 0\text{x}00 \,\|\, V$ for $i \in [1, m]$ to compute C; **(2)** $V + j - 1$ for $j \in [1, n]$ to compute the output R^*; and **(3)** $0\text{x}03 \,\|\, V$ to compute H_1. Since the queries of form **(1)** cannot collide with each other (due to the iterating counter) or with queries of forms **(2)** and **(3)** (since queries of type **(1)** are of a longer length than those of type **(2)** and **(3)**), an analogous argument to that made above implies that bad will only be set if $\mathcal{A}$ manages to query to H one of the points previously queried by the challenger. Moreover, all such queries require $\mathcal{A}$ to guess the value of a point in the set $\mathcal{V} = \{V, \ldots, V + n - 1\}$. Since knowledge of the challenge state $S^* = (V^*, C^*, cnt^*)$ can only increase $\mathcal{A}$'s guessing probability, we may without loss of generality assume that $\mathcal{A}$ makes all of his q_{H} queries to H after receiving his challenge. An analogous argument to that used above, taking a union bound, implies that

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,] \le n \cdot \Pr\left[\,\mathcal{A}^{\mathsf{H}}(R^*, S^*) \text{ guesses the value of } V \text{ in } G_2\,\right]. \qquad (4.2)$$

In particular, notice that possessing the challenge state $S^* = (V^*, C^*, cnt^*)$ leaks information to $\mathcal{A}$ about the value of the target V. Namely, $V^* = V + C + cnt + H_1$ where $H_1 \leftarrow\!\!\$\, \{0,1\}^\ell$, $C = C^*$, $cnt = 1$, and addition is modulo 2^L — notice how this differs from the previous case. As such, $\mathcal{A}$ knows that V corresponds to a random value in the interval $[V^* - C^* - 1 - (2^\ell - 1), V^* - C^* - 1]$. Given $\mathcal{A}$'s view of the experiment at this point, which consists of R^* and $S^* = (V^*, C^*, cnt^*)$, it is straightforward to verify that guessing the correct value of V is equivalent to correctly guessing the value of $H_1 \leftarrow\!\!\$\, \{0,1\}^\ell$. As such, the probability that $\mathcal{A}$ guesses the correct value of V given a single query is equal to $2^{-\ell}$. Taking a union bound over $\mathcal{A}$'s q_{H} guesses and substituting into equation (4.2) then implies that

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,] \le \frac{q_{\mathsf{H}} \cdot n}{2^\ell}\ .$$

Next we define game G_3, which is identical to G_2 except we simply compute S^* as $S^* \leftarrow\!\!\$\, \mathsf{M}^{\mathsf{H}}(S)$ and omit the redundant random oracle queries which were made to compute the variables R^* and H_1. As before, it is straightforward to verify that both games are identically distributed, and so $\Pr[\,G_2 \Rightarrow 1\,] = \Pr[\,G_3 \Rightarrow 1\,]$.

We then define game G_4, which is identical to G_3 except we no longer overwrite the string

C with a random bit string. An analogous argument to that used above implies that $\Pr[G_3 \Rightarrow 1] = \Pr[G_4 \Rightarrow 1]$.

Next we define game G_5, which is identical to G_4 except we return H to answer consistently on all points. By an analogous argument to that used above, G_4 and G_5 are identical until the flag bad is set. Since the only points queried to H during the computation of the challenge output / state in these games are during the computation of C, it follows that bad will only be set if $\mathcal{A}$ queries H on one of the points used to compute C. In turn, this may only occur if $\mathcal{A}$ correctly guesses the value of V. As such, an analogous argument to that above, invoking the Fundamental Lemma of Game Playing and the fact that the probability that $\mathcal{A}$ guesses the value of V in a single guess is upper bounded by $2^{-\ell}$, implies that

$$\begin{aligned}|\Pr[G_4 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| &\leq \Pr[\mathsf{bad} = 1 \text{ in } G_5] \\ &= \Pr[\mathsf{bad} = 1 \text{ in } G_3] \leq \frac{q_{\mathsf{H}}}{2^{\ell}} \ .\end{aligned}$$

Now G_5 is identical to game Next for PRNG with challenge bit $b = 1$, and so it follows that

$$\Pr[G_5 \Rightarrow 1] = \Pr\left[\text{Next}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 1\right] .$$

Putting this altogether, we conclude that

$$\begin{aligned}\mathbf{Adv}^{\text{next}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) &= |\Pr[G_0 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \\ &\leq \frac{q_{\mathsf{H}} \cdot (n+1)}{2^{\ell}} \leq \frac{q_{\mathsf{H}} \cdot n}{2^{\ell-1}} \ ,\end{aligned}$$

where the final inequality follows since $q_{\mathsf{H}} \geq 0$ and $n \geq 1$, thereby proving the bound in this case also. □

$\overline{\text{Pres}}$ security. We now analyse the $\overline{\text{Pres}}$ security of HASH-DRBG, during which we will invoke Lemma 4.4 proved above. At the start of game $\overline{\text{Pres}}$, the (q_{H}, q_C)-attacker $\mathcal{A}$ outputs $(S'_0, I_1, \ldots, I_d)$ where $d \leq q_C$. The game sets $(V_0, C_0, cnt_0) \leftarrow\$\ \mathsf{M}^{\mathsf{H}}(S'_0)$, and iteratively computes S_d via $S_i \leftarrow \mathsf{refresh}^{\mathsf{H}}(S_{i-1}, I_i)$ for $i \in [1, d]$. The proof begins by arguing that unless the $\mathcal{A}$ queries H on the counter V_0 or any of the counters $V_1, \ldots, V_{d-1}$ passed through during reseeding, then (barring certain accidental collisions) the updated state $S_d = (V_d, C_d, cnt_d)$ is indistinguishable from a masked state. The proof then shows that,

```
proc. main // $G_0, G_1, \boxed{G_2}$
$(S_0', I_1, \ldots, I_d) \leftarrow_{\$} \mathcal{A}^{\mathsf{H}}$ ; $(V_0', C_0', cnt_0') \leftarrow S_0'$
$H_0 \leftarrow_{\$} \{0,1\}^{\ell}$ ; $V_0 \leftarrow V_0' + C_0' + cnt_0' + H_0$
$C_0 \leftarrow C_0'$ ; $cnt_0 \leftarrow cnt_0' + 1$
$S_0 \leftarrow (V_0, C_0, cnt_0)$ ; $m \leftarrow \lceil L/\ell \rceil$ ; $\mathcal{Q} \leftarrow \emptyset$
For $j = 1, \ldots, d$
  $temp_{V_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x01} \,\|\, V_{j-1} \,\|\, I_j\}$
    $temp_{V_j} \leftarrow temp_{V_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x01} \,\|\, V_{j-1} \,\|\, I_j)$
  $V_j \leftarrow \mathrm{left}(temp_{V_j}, L)$ ; $\boxed{V_j \leftarrow_{\$} \{0,1\}^L}$
  $temp_{C_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $temp_{C_j} \leftarrow temp_{C_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x00} \,\|\, V_j)$
  $C_j \leftarrow \mathrm{left}(temp_{C_j}, L)$ ; $cnt_j \leftarrow 1$
$S_d \leftarrow (V_d, C_d, cnt_d)$
$(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$
$b^* \leftarrow_{\$} \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. main // $G_3, \boxed{G_4}$
$(S_0', I_1, \ldots, I_d) \leftarrow_{\$} \mathcal{A}^{\mathsf{H}}$ ; $(V_0', C_0', cnt_0') \leftarrow S_0'$
$H_0 \leftarrow_{\$} \{0,1\}^{\ell}$ ; $V_0 \leftarrow V_0' + C_0' + cnt_0' + H_0$
$C_0 \leftarrow C_0'$ ; $cnt_0 \leftarrow cnt_0' + 1$
$S_0 \leftarrow (V_0, C_0, cnt_0)$ ; $m \leftarrow \lceil L/\ell \rceil$ ; $\mathcal{Q} \leftarrow \emptyset$
$S_d \leftarrow_{\$} \mathsf{M}^{\mathsf{H}}(\varepsilon)$
$(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$ // $G_3$ only
$\boxed{R^* \leftarrow_{\$} \{0,1\}^{\ell}}$
$\boxed{S^* \leftarrow_{\$} \mathsf{M}^{\mathsf{H}}(S_d)}$
$b^* \leftarrow_{\$} \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. main // $\boxed{G_5}, G_6, G_7$
$(S_0', I_1, \ldots, I_d) \leftarrow_{\$} \mathcal{A}^{\mathsf{H}}$ ; $(V_0', C_0', cnt_0') \leftarrow S_0'$
$H_0 \leftarrow_{\$} \{0,1\}^{\ell}$ ; $V_0 \leftarrow V_0' + C_0' + cnt_0' + H_0$
$C_0 \leftarrow C_0'$ ; $cnt_0 \leftarrow cnt_0' + 1$
$S_0 \leftarrow (V_0, C_0, cnt_0)$ ; $m \leftarrow \lceil L/\ell \rceil$ ; $\mathcal{Q} \leftarrow \emptyset$
For $j = 1, \ldots, d$
  $temp_{V_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x01} \,\|\, V_{j-1} \,\|\, I_j\}$
    $temp_{V_j} \leftarrow temp_{V_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x01} \,\|\, V_{j-1} \,\|\, I_j)$
  $V_j \leftarrow \mathrm{left}(temp_{V_j}, L)$ ; $\boxed{V_j \leftarrow_{\$} \{0,1\}^L}$
  $temp_{C_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $temp_{C_j} \leftarrow temp_{C_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x00} \,\|\, V_j)$
  $C_j \leftarrow \mathrm{left}(temp_{C_j}, L)$ ; $cnt_j \leftarrow 1$
$S_d \leftarrow (V_d, C_d, cnt_d)$
$R^* \leftarrow_{\$} \{0,1\}^{\ell}$
$S^* \leftarrow_{\$} \mathsf{M}^{\mathsf{H}}(S_d)$
$b^* \leftarrow_{\$} \mathcal{A}^{\mathsf{H}}(R^*, S^*)$
Return $b^*$
```

```
proc. $\mathsf{H}(X)$ // $\boxed{G_0}, G_1, G_2, G_3, G_4, G_5, G_6, \boxed{G_7}$
$Y \leftarrow_{\$} \{0,1\}^{\ell}$
If $\mathsf{H}[X] \neq \bot$ and $X \notin \mathcal{Q}$
  $Y \leftarrow \mathsf{H}[X]$
If $\mathsf{H}[X] \neq \bot$ and $X \in \mathcal{Q}$
  $\mathsf{bad} \leftarrow 1$
  $\boxed{Y \leftarrow \mathsf{H}[X]}$
$H[X] \leftarrow Y$
Return $Y$
```

Figure 4.8: Games for proof of Lemma 4.5 ($\overline{\mathrm{Pres}}$ security of HASH-DRBG).

unless the attacker can guess V_d, the resulting output / state pair are indistinguishable from their idealised counterparts. We must consider a number of cases depending on whether the tuple $(S_0', I_1, \ldots, I_d)$ output by $\mathcal{A}$ is such that: **(1)** $S_0' \in \mathcal{S}$ or $S_0' = \varepsilon$; and **(2)** $d \geq 1$ or $d = 0$; since these induce different distributions on S_0 and S_d respectively.

Lemma 4.5. *Let* PRNG = HASH-DRBG *and masking function* M^{H} *be as specified in Theorem 4.2, and let* $n = \lceil \beta/\ell \rceil$. *Then for any* (q_{H}, q_C)*-adversary* $\mathcal{A}$ *in game* $\overline{\mathrm{Pres}}$ *against* PRNG*, it holds that*

$$\mathbf{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) \leq \frac{q_{\mathsf{H}} \cdot (n+1)}{2^{\ell-1}} + \frac{(q_C - 1)(2q_{\mathsf{H}} + q_C)}{2^L} .$$

Proof. Recall that in game $\overline{\mathrm{Pres}}$ the attacker $\mathcal{A}$ outputs a tuple $(S_0', I_1, \ldots, I_d)$ at the start of his challenge which is then masked to give $S_0 \leftarrow_{\$} \mathsf{M}^{\mathsf{H}}(S_0')$. Our proof will proceed by arguing that after iteratively computing $S_i \leftarrow \mathsf{refresh}^{\mathsf{H}}(S_{i-1}, I_i)$ for $i = 1, \ldots, d$, the resulting state S_d is indistinguishable from $\mathsf{M}^{\mathsf{H}}(\overline{S})$ for some state $\overline{S} \in \mathcal{S} \cup \{\varepsilon\}$ which will be made explicit during the proof. This then allows us to reduce the $\overline{\mathrm{Pres}}$ security of PRNG to the Next security of PRNG. Since the state takes a different distribution after a refresh than after an output generation request, there are a number of cases to consider.

We may assume without loss of generality that $\mathcal{A}$ never repeats a query to H. We begin by considering the case in which the tuple $(S'_0, I_1, \ldots, I_d)$ output by $\mathcal{A}$ is such that: **(1)** $d \geq 1$ and so at least one refresh occurs during the challenge computation; and **(2)** $S'_0 \neq \varepsilon$. We argue by a series of game hops, shown in Figure 4.8. We begin by defining game G_0, which is easily verified to be a rewriting of game $\overline{\text{Pres}}$ for HASH-DRBG and M^{H} with challenge bit $b = 0$ and a lazily sampled random oracle. We have explicitly written out the code of the masking function at the point at which the state is masked at the start of the game. We additionally set a flag bad in G_0, but this does not affect the outcome of the game. It follows that

$$\Pr\left[\, G_0 \Rightarrow 1 \,\right] = \Pr\left[\, \overline{\text{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \;\mid\; b = 0 \,\right] \,.$$

Next we define game G_1, which is identical to G_0 except that now if the random oracle H is queried more than once on any of the points $(i)_8 \,\|\, (L)_{32} \,\|\, \texttt{0x01} \,\|\, V_{j-1} \,\|\, I_j$ for $i \in [1, m]$, $j \in [1, d]$ upon which it was queried during the iterative reseeds to compute $V_1, \ldots, V_d$ (indicated in the pseudocode by the set $\mathcal{Q}$), then H responds with an independent random string as opposed to the value previously set. These games run identically until the flag bad is set, and so the Fundamental Lemma of Game Playing implies that

$$\left|\Pr\left[\, G_0 \Rightarrow 1 \,\right] - \Pr\left[\, G_1 \Rightarrow 1 \,\right]\right| \leq \Pr\left[\, \mathsf{bad} = 1 \text{ in } G_1 \,\right] .$$

Next we define game G_2, which is identical to G_1 except we overwrite each of the intermediate state variables V_j for $j \in [1, d]$ with an independent random bit string $V_j \leftarrow\!\!\$\, \{0,1\}^L$. Since in G_1, the strings returned in response to the H queries made to compute these variables are chosen independently at random and are used at no other point in the game, it follows that these games are identically distributed, and so

$$\Pr\left[\, G_1 \Rightarrow 1 \,\right] = \Pr\left[\, G_2 \Rightarrow 1 \,\right] ;$$

and

$$\Pr\left[\, \mathsf{bad} = 1 \text{ in } G_1 \,\right] = \Pr\left[\, \mathsf{bad} = 1 \text{ in } G_2 \,\right] .$$

We now bound this latter probability. Notice that bad will be set if any of the q_{H} H queries made by $\mathcal{A}$ collide with one of the points queried by the challenger when computing $V_1, \ldots, V_d$. The flag bad will also be set if there exist distinct $j, j' \in [0, d-1]$ such that $V_j \,\|\, I_{j+1} = V_{j'} \,\|\, I_{j'+1}$. (It is straightforward to verify that due to differing lengths and / or domain separation, none of the queries made to compute the constants C_j for $j \in [1, d]$, nor the queries made by $\mathsf{next}^{\mathsf{H}}$, will cause the flag bad to be set.)

Now, the challenge output / state pair are computed as $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$. Moreover, notice that in G_2 the state S_d is computed as a function of the randomly chosen counter $V_d \leftarrow\!\!\$\ \{0,1\}^L$ as opposed to via iterative reseeding. As such, $\mathcal{A}$'s view of the experiment is independent of the intermediate state variables V_j, C_j for $j \in [0, d-1]$ and so we can without loss of generality imagine deferring the computation of these state variables until *after* $\mathcal{A}$ has finished making all of his queries to H. We now bound the probability that bad is set during this process.

We first note that bad will be set during the first reseed if $\mathcal{A}$ has made a H query of the form $(i)_8 \,\|\, (L)_{32} \,\|\, 0\text{x}01 \,\|\, V_0 \,\|\, I_1$ where $i \in [1, m]$. Any such query requires correctly guessing the value of V_0, and since V_0 is uniformly distributed over the set $[V_0' + C_0' + cnt_0', V_0' + C_0' + cnt_0' + 2^\ell - 1]$ and independent of $\mathcal{A}$'s view of the experiment in G_2, it follows that the probability that $\mathcal{A}$ has made such a query is upper bounded by $\frac{q_{\mathsf{H}}}{2^\ell}$. Assuming that this event does not occur then bad will be set during the second reseed if either V_0 and V_1 collide, or $\mathcal{A}$ has already made a query of the correct form containing V_1. Since $V_1 \leftarrow\!\!\$\ \{0,1\}^L$ in G_2, it follows that bad is set during this reseed with probability $\frac{q_{\mathsf{H}}+1}{2^L}$ (the $q_{\mathsf{H}} \cdot 2^{-L}$ arising from $\mathcal{A}$'s queries and the additional 2^{-L} from the probability that $V_0 = V_1$ when $V_1 \leftarrow\!\!\$\ \{0,1\}^L$). Inductively applying the same argument yields that for all $j \in [2, d]$, the probability that bad is set during the j^{th} reseed is upper bounded by $\frac{q_{\mathsf{H}}+j-1}{2^L}$. Finally, summing over these terms yields

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,] \leq \frac{q_{\mathsf{H}}}{2^\ell} + \frac{(d-1)(2q_{\mathsf{H}} + d)}{2^{L+1}}\ .$$

Next we define game G_3, in which we omit the iterative $\mathsf{refresh}$ calls and instead directly set $S_d \leftarrow\!\!\$\ \mathsf{M}^{\mathsf{H}}(\varepsilon)$. It is straightforward to verify that these games are identically distributed and so $\Pr[\,G_2 \Rightarrow 1\,] = \Pr[\,G_3 \Rightarrow 1\,]$. We then define game G_4, in which instead of computing R^* and S^* via $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$, we instead set $R^* \leftarrow\!\!\$\ \{0,1\}^\beta$ and $S^* \leftarrow\!\!\$\ \mathsf{M}^{\mathsf{H}}(S_d)$. We claim there exists an adversary $\mathcal{A}'$ in game Next against HASH-DRBG such that

$$|\Pr[\,G_3 \Rightarrow 1\,] - \Pr[\,G_4 \Rightarrow 1\,] \leq \mathbf{Adv}^{\text{next}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}') \leq \frac{q_{\mathsf{H}} \cdot n}{2^{\ell-1}}\ ,$$

and moreover $\mathcal{A}'$ makes q_{H} queries to his random oracle. To see this, notice that an attacker $\mathcal{A}'$ in game Next against HASH-DRBG and M^{H} can perfectly simulate $\mathcal{A}$'s view of the game as follows. For each of $\mathcal{A}$'s initial queries, $\mathcal{A}'$ simulates $\mathcal{A}$'s random oracle by forwarding all of $\mathcal{A}$'s queries to his own random oracle and returning the response. When $\mathcal{A}$ outputs a state S_0', $\mathcal{A}'$ outputs the state ε as his challenge state, receiving (R^*, S^*) in response. $\mathcal{A}'$ passes these to $\mathcal{A}$ and continues simulating $\mathcal{A}$'s random oracle by querying his own oracle as before. At the end of the game, $\mathcal{A}'$ outputs whatever bit $\mathcal{A}$ does. If $\mathcal{A}'$'s challenge bit is equal to 0 then this perfectly simulates game G_3, otherwise it perfectly simulates G_4, and so invoking Lemma 4.4 proves the claim. Moreover, since $\mathcal{A}$ makes q_{H} queries it follows that $\mathcal{A}'$ makes q_{H} queries also.

Finally in games G_5 and G_6 we reverse the earlier transitions to return to computing S_d via the process of iterative reseeding, and then in G_7 return the random oracle to respond consistently to all queries. By analogous arguments to those used above, $G_4 - G_6$ are identically distributed and G_6 and G_7 run identically unless bad is set, and so it follows that

$$|\Pr[\,G_4 \Rightarrow 1\,] - \Pr[\,G_7 \Rightarrow 1\,]| \leq \Pr[\,\mathsf{bad} = 1 \text{ in } G_5\,] \leq \frac{q_{\mathsf{H}}}{2^{\ell}} + \frac{(d-1)(2q_{\mathsf{H}} + d)}{2^{L+1}} \;.$$

Moreover, notice that G_7 is identical to $\overline{\text{Pres}}$ for PRNG with challenge bit $b = 1$, and so

$$\Pr[\,G_7 \Rightarrow 1\,] = \Pr\left[\,\overline{\text{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 1\,\right] \;.$$

Putting this all together, we conclude that

$$\begin{aligned}\mathbf{Adv}^{\overline{\text{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) &= \left|\Pr\left[\,\overline{\text{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 0\,\right] - \Pr\left[\,\overline{\text{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 1\,\right]\right| \\ &\leq |\Pr[\,G_0 \Rightarrow 1\,] - \Pr[\,G_7 \Rightarrow 1\,]| \\ &\leq \frac{q_{\mathsf{H}} \cdot (n+1)}{2^{\ell-1}} + \frac{(d-1)(2q_{\mathsf{H}} + d)}{2^{L}} \;.\end{aligned}$$

This proves the case in which $\mathcal{A}$ outputs $S_0' \neq \varepsilon$ and $d \geq 1$. We now explain why this upper bound holds in all other cases too. Firstly for the case in which $\mathcal{A}$ outputs $S_0' = \varepsilon$ at the start of his challenge and $d \geq 1$, the proof is identical, except that when computing $S_0 \leftarrow\!\!\$\; \mathsf{M}^{\mathsf{H}}(S_0')$ we set $V_0 \leftarrow\!\!\$\; \{0,1\}^L$ (and $C_0 \leftarrow \mathsf{HASH\text{-}DRBG\_df}^{\mathsf{H}}(0\text{x}00 \,\|\, V_0)$, although this does not affect the proof). The increased entropy in the initial state can only make $\mathcal{A}$'s job harder and so the bound holds in this case also. Moreover (for both choices of initial state) if $d = 0$ and so no $\mathsf{refresh}$ calls are made, then in G_0 the challenge output / state are computed by applying $\mathsf{next}^{\mathsf{H}}$ to the masked state $S_0 \leftarrow\!\!\$\; \mathsf{M}^{\mathsf{H}}(S_0')$ where S_0' is the state output by $\mathcal{A}$ at the start of his challenge. An analogous reduction to an attacker in game Next against $\mathsf{HASH\text{-}DRBG}$ and M^{H}, who passes S_0' to his challenger and returns the response to $\mathcal{A}$, confirms the upper bound in this case also. Substituting in $d \leq q_C$ then concludes the proof. □

$\overline{\text{Rec}}$ security. Finally, we bound the $\overline{\text{Rec}}$ security of HASH-DRBG. The first stage in the proof argues that iteratively reseeding an adversarially chosen state S_0 with d entropy samples which collectively have entropy γ^* yields a state $S_d = (V_d, C_d, cnt_d)$ which is indistinguishable from $\mathsf{M}^{\mathsf{H}}(\varepsilon)$. This represents the main technical challenge in the proof, and uses Patarin's H-coefficient technique (see Section 2.2.4).

Our proof is closely based on the analogous result for sponge-based PRNGs in the IPM of Gazi and Tessaro [69], essentially making the same step-by-step argument. However, making the necessary adaptations to analyse HASH-DRBG is still non-trivial. As well as working in the ROM as opposed to the IPM, we must adapt the proof to take into account the constant C which is a state component of HASH-DRBG, as well as the more involved reseeding process, which concatenates and truncates the responses to multiple H queries to derive each updated counter V'. With this in place, an analogous argument to that made in the proof of $\overline{\text{Pres}}$ security implies that an output / state pair produced by applying $\mathsf{next}^{\mathsf{H}}$ to this masked state are indistinguishable from their idealised counterparts.

Lemma 4.6. *Let* PRNG = HASH-DRBG *and masking function* M^{H} *be as specified in Theorem 4.2, and let* $n = \lceil \beta/\ell \rceil$. *Then for any* (q_{H}, q_C)*-adversary* $\mathcal{A}$ *in game* $\overline{\text{Rec}}$ *against* PRNG, *and any* $(q^+_{\mathcal{D}}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$, *it holds that*

$$\mathbf{Adv}^{\overline{\text{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A},\mathcal{D}) \leq \frac{q_{\mathsf{H}}}{2^{\gamma^*-1}} + \frac{(q_C-1)\cdot(2q_{\mathsf{H}}+q_C)+2q_{\mathsf{H}}^2}{2^L} + \frac{q_{\mathsf{H}}\cdot n}{2^{\ell-1}} .$$

Proof. We begin by analysing the iterative refreshing process at the start of game $\overline{\text{Rec}}$. Consider the game Extract shown in the left-hand panel of Figure 4.9, in which the advantage of an attacker and sampler pair $(\mathcal{A}, \mathcal{D})$ is defined as:

$$\mathbf{Adv}^{\text{ext}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) = 2\cdot\left|\Pr\left[\text{Extract}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) \Rightarrow 1\right] - \frac{1}{2}\right| .$$

In this game, the attacker $\mathcal{A}$ is challenged to distinguish a state of his choice which is iteratively reseeded with entropy samples with a collective entropy of at least γ^*-bits from a masked state $\mathsf{M}^{\mathsf{H}}(\varepsilon)$. As before, we say that $\mathcal{A}$ is a (q_{H}, q_C)-adversary if it makes q_{H} queries to H and outputs $d \leq q_C$. In the following lemma, we upper bound the success probability of an attacker in such a game against HASH-DRBG. We will then use this result to bound the $\overline{\text{Rec}}$ security of HASH-DRBG.

Notation. We first introduce some notation. We let $\mathcal{J}$ denote the set of all H queries

```
$\underline{\text{Extract}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D})}$
$\mathsf{H} \leftarrow\$ \mathcal{H}\ ; b \leftarrow\$ \{0,1\}$
$\sigma_0 \leftarrow \varepsilon\ ; \mathsf{seed} \leftarrow\$ \text{Seed}\ ; \mu \leftarrow 0$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d) \leftarrow\$ \mathcal{A}^{\mathsf{H},\text{Sam}}(I_1, (\gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > (q_{\mathcal{D}} + 1)$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
If $b = 0$ then
    For $j = 1, \ldots, d$
        $S_j \leftarrow \mathsf{refresh}^{\mathsf{H}}(\mathsf{seed}, I_j, S_{j-1})$
Else $S_d \leftarrow\$ \mathsf{M}^{\mathsf{H}}(\varepsilon)$
$b^* \leftarrow\$ \mathcal{A}^{\mathsf{H}}(\mathsf{seed}, S_d, (I_k)_{k>\mu+d})$
Return $(b = b^*)$
---
$\underline{\text{Sam}()}$
$\mu = \mu + 1$
Return $I_\mu$
---
$\underline{\mathsf{H}(X)}$
Return $\mathsf{H}(X)$
```

```
$\underline{\text{Extract}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D})}$
$\mathsf{H} \leftarrow\$ \mathcal{H}\ ; b \leftarrow\$ \{0,1\}$
$\sigma_0 \leftarrow \varepsilon\ ; \mu \leftarrow 1$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d) \leftarrow\$ \mathcal{A}^{\mathsf{H},\text{Sam}}(\mathsf{seed}, I_1, (\gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > (q_{\mathcal{D}} + 1)$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
$S_0 \leftarrow (V_0, C_0, cnt_0)$
If $b = 0$ then
    For $j = 1, \ldots, d$
        $temp_{V_j} \leftarrow \varepsilon$
        For $i = 1, \ldots, m$
            $temp_{V_j} \leftarrow temp_{V_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \text{0x01} \,\|\, V_{j-1} \,\|\, I_{\mu+j})$
        $V_j \leftarrow \text{left}(temp_{V_j}, L)$
        $temp_{C_j} \leftarrow \varepsilon$
        For $i = 1, \ldots, m$
            $temp_{C_j} \leftarrow temp_{C_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \text{0x00} \,\|\, V_j)$
        $C_j \leftarrow \text{left}(temp_{C_j}, L)\ ; cnt_j \leftarrow 1$
    $S_d \leftarrow (V_d, C_d, cnt_d)$
Else $S_d \leftarrow\$ \mathsf{M}^{\mathsf{H}}(\varepsilon)$
$b^* \leftarrow\$ \mathcal{A}^{\mathsf{H}}(S_d, (I_k)_{k>\mu+d})$
Return $(b = b^*)$
```

Figure 4.9: Game Extract for a PRNG $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ (left) and pseudocode for proof of Lemma 4.7 (right).

made by $\mathcal{A}$ of the form

$$(i)_8 \,\|\, (L)_{32} \,\|\, \text{0x01} \,\|\, Y \,\|\, Z$$

where $i \in [1, m]$, $Y \in \{0,1\}^L$, and $Z \in \{0,1\}^{p \le i \le \overline{p}}$ (e.g., points of the form which are queried to H to derive the V values during reseeds), and notice that all such strings can be decomposed unambiguously into these component parts. To each query $x \in \mathcal{J}$, we call $\overline{x} = (i, Y, Z)$ its associated decomposition; as we will see, these will be the components of these queries that are important for the proof. With this in place, in the following lemma we analyse the Extract security of HASH-DRBG.

Lemma 4.7. *Let* PRNG = HASH-DRBG *and masking function* M^{H} *be as specified in Theorem 4.2. Then for any* (q_{H}, q_C)*-adversary* $\mathcal{A}$ *in game* Extract *against* HASH-DRBG*, for which* q_J *of* $\mathcal{A}$*'s* H *queries lie in* $\mathcal{J}$*, and any* $(q_{\mathcal{D}}^{+}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$*, it holds that*

$$\mathbf{Adv}^{\text{ext}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) \le \frac{q_J}{2^{\gamma^*}} + \frac{(q_C - 1)\cdot(2q_J + q_C) + 2q_J^2}{2^{L+1}}\,.$$

Our proof uses the H-coefficient method as defined in Section 2.2.4. We make the usual simplifying assumption that $\mathcal{A}$ is deterministic and that the sampler $\mathcal{D}$, having been initialised with coins $\omega \in \mathsf{coins}_{\mathcal{D}}$ where $\mathsf{coins}_{\mathcal{D}}$ denotes the coin space of the sampler, is deterministic also. Without loss of generality, we may assume that $\mathcal{A}$ makes precisely q_{H} queries, and never makes a redundant query. For clarity, we have written out game Extract for HASH-DRBG in the right-hand panel of Figure 4.9. We begin by introducing some notation which will simplify our definition of *bad* transcripts. Within a transcript of an execution of game Extract, we let (x_ι, y_ι) denote that $\mathcal{A}$ queried x_ι to H and received y_ι in response. As with the sponge-extraction lemma of [69], we will modify game Extract so that $\mathcal{A}$ is given some additional information before outputting its challenge bit guess. However, these extra values are provided only after $\mathcal{A}$ has finished making all of its H queries, and so this extra information cannot influence $\mathcal{A}$'s choice of queries. Namely, we will additionally give to $\mathcal{A}$: **(1)** the inputs $I_{\mu+1}, \ldots, I_{\mu+d}$ which $\mathcal{A}$ selected to use in the challenge computation; and **(2)** the coins $\omega \in \mathsf{coins}_{\mathcal{D}}$ which the sampler $\mathcal{D}$ was initialised with.

Moreover, we will further modify game Extract to change the way in which the random oracle H responds to queries. Namely, if H is queried on a point $x \in \mathcal{J}$ of the form $(*)_8 \,\|\, (L)_{32} \,\|\, \mathrm{0x01} \,\|\, Y \,\|\, Z$, the game computes $y^i = \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathrm{0x01} \,\|\, Y \,\|\, Z)$ for *all* $i \in [1, m]$ and returns $y = y^1 \,\|\, \ldots \,\|\, y^m$ to $\mathcal{A}$. (Here we use $*$ to denote an arbitrary $i \in [1, m]$.) However, this is still counted as a single query made by $\mathcal{A}$. It is straightforward to see that such a change can only increase $\mathcal{A}$'s success probability since an attacker in the modified game can perfectly simulate the original game by returning block y^i of y to $\mathcal{A}$ in response to a query prefixed with index i. Our assumption that $\mathcal{A}$ never makes a redundant query now extends to him querying both $(i)_8 \,\|\, (L)_{32} \,\|\, \mathrm{0x01} \,\|\, Y \,\|\, Z$ and $(i')_8 \,\|\, (L)_{32} \,\|\, \mathrm{0x01} \,\|\, Y \,\|\, Z$ for $i, i' \in [1, m]$ and $i \neq i'$, since in the modified game the first query provides him with the answer to the latter. This modification will simplify the subsequent proof without increasing the attacker's success probability by too much for the parameter settings we are interested in. Indeed, in the worst case it gives the attacker m times as many queries, where for all approved hash functions $m \leq 3$.

Recall that we write (x_ι, y_ι) to indicate that $\mathcal{A}$ queried x_ι to H and received y_ι in response. It is straightforward to verify from the pseudocode on the right-hand side of Figure 4.9 that each execution of game Extract for a (q_{H}, q_C)-adversary $\mathcal{A}$ defines a transcript of the form

$$\tau = ((x_1, y_1), \ldots, (x_{q_{\mathsf{H}}}, y_{q_{\mathsf{H}}}), \omega, (V_0, C_0, cnt_0), (V_d, C_d, cnt_d), (I_k, \gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}+1}, \mu, d) \ ,$$

where $d \leq q_C$. We say that a transcript τ is *compatible* if it may arise from an execution of

Extract. We let T_0 denote the distribution of compatible transcripts produced from game Extract with challenge bit $b = 0$, and let T_1 denote the distribution of all such transcripts for Extract with challenge bit $b = 1$. With this in place, we define bad transcripts below.

Definition 4.2. *Let τ be a compatible transcript with associated set of queries $(x_1, y_1), \ldots, (x_{q_\mathsf{H}}, y_{q_\mathsf{H}})$. Then τ is said to be* Bad *if among those queries there exists a subset of (not necessarily distinct) queries $\mathcal{X} = \{(u_1, v_1), \ldots, (u_d, v_d)\}$ such that:*

- *$u_\iota \in \mathcal{J}$ for all $(u_i, v_i) \in \mathcal{X}$; and*
- *For $j \in [1, d]$, it holds that u_j has decomposition $(*, X_{j-1}, I_{\mu+j})$ where $X_{j-1} = \mathrm{left}(v_{j-1}, L)$. (Here we define $V_0 = \mathrm{left}(v_0, L)$ for notational brevity.)*

In words, a compatible transcript τ is Bad if it contains a set of queries which (with respect to our modification to the random oracle) are equivalent to those made by the challenger to compute the updated state component V_d. With this in place, our result is derived from the following two lemmas.

Lemma 4.8. *Let $\tau \in$ Good be a transcript. Then*

$$\frac{\Pr[\,\mathsf{T}_0 = \tau\,]}{\Pr[\,\mathsf{T}_1 = \tau\,]} \geq 1 - \frac{(q_C - 1) \cdot (2q_J + q_C)}{2^{L+1}} .$$

Proof. We begin by noting that when $b = 1$ the challenge state S^* is computed as $\mathsf{M}^\mathsf{H}(\varepsilon)$, where recall that M^H chooses $V_d \leftarrow\$ \{0,1\}^L$, sets $C_d = \mathsf{HASH\text{-}DRBG\_df}^\mathsf{H}(0\text{x}00 \,\|\, V_d)$ and $cnt_d = 1$, and returns $S_d = (V_d, C_d, cnt_d)$. Now for a random oracle H to be compatible with transcript τ, it must hold that: **(1)** $\mathsf{H}(x_\iota) = y_\iota$ for all $\iota \in [1, q_\mathsf{H}]$; and **(2)** H is consistent with $C_d = \mathsf{HASH\text{-}DRBG\_df}^\mathsf{H}(0\text{x}00 \,\|\, V_d)$. It follows that

$$\begin{aligned} \Pr[\,\mathsf{T}_1 = \tau\,] =& \Pr[\,S_d = (V_d, \cdot, \cdot) \wedge \omega \text{ chosen} \wedge \mathsf{H} \text{ satisfies } \mathbf{(1)} \text{ and } \mathbf{(2)}\,] \\ &= 2^{-L} \cdot \Pr[\,\omega \text{ chosen} \wedge \mathsf{H} \text{ satisfies } \mathbf{(1)} \text{ and } \mathbf{(2)}\,] . \end{aligned} \tag{4.3}$$

Here the probability is over the choice of $\mathsf{H} \leftarrow\$ \mathcal{H}$, $\omega \leftarrow\$ \mathsf{coins}_\mathcal{D}$, and $V_d \leftarrow\$ \{0,1\}^L$. (Recall that by assumption $\mathcal{A}$ is deterministic.) The final inequality follows since V_d is sampled uniformly in the ideal world, and so each V_d is selected with probability 2^{-L}.

We now consider the case in which $\tau \in$ Good and $b = 0$. We write $S_d \leftarrow \mathsf{refresh}^\mathsf{H}(S_0, I_{\mu+1}, \ldots, I_{\mu+d})$ to denote the state returned by computing $S_i \leftarrow \mathsf{refresh}^\mathsf{H}(S_{i-1}, I_{\mu+i})$ for $i = 1, \ldots, d$. Parsing $(V_d, C_d, cnt_d) \leftarrow S_d$, we let $q(\tau)$ denote the probability that $S_d \leftarrow \mathsf{refresh}^{\mathsf{H}}(S_0, I_{\mu+1}, \ldots, I_{\mu+d})$ yields the required counter V_d conditioned on ω being the chosen coins and H satisfying properties **(1)** and **(2)**. It follows that

$$\begin{aligned}\Pr[\,\mathsf{T}_0 = \tau\,] &= \Pr[\,S_d = (V_d, \cdot, \cdot) \wedge \omega \text{ chosen} \wedge \mathsf{H} \text{ satisfies } \mathbf{(1)} \text{ and } \mathbf{(2)}\,]\\ &= q(\tau) \times \Pr[\,\omega \text{ chosen} \wedge \mathsf{H} \text{ satisfies } \mathbf{(1)} \text{ and } \mathbf{(2)}\,]\\ &= q(\tau) \times 2^L \times \Pr[\,\mathsf{T}_1 = \tau\,],\end{aligned}$$

where the final equality follows from substituting in the term from equation (4.3).

It remains to bound $q(\tau)$. One may imagine lazily sampling the random oracle H during the computation of V_d conditioned on it being consistent with the transcript via points **(1)** and **(2)** as defined above. Let l' be maximal such that τ contains a set of queries $\{(u_1, v_1), \ldots, (u_{l'}, v_{l'})\}$ such that $u_j \in \mathcal{J}$ for each $j \in [1, l']$ and it holds that u_j has decomposition $(*, X_{j-1}, I_{\mu+j})$ where $X_{j-1} = \text{left}(v_{j-1}, L)$. Since we have assumed that $\tau \in \mathsf{Good}$, it must be the case that $l' \in [0, d-1]$ (otherwise τ will be Bad).

For $j \in [l', d-1]$ we let $\neg\mathsf{Fresh}_j$ denote the event that either: **(a)** there exists a query $u_i \in \tau \cap \mathcal{J}$ such that u_i has decomposition $(*, V_j, Z)$ for some $Z \in \{0,1\}^{p \le i \le \overline{p}}$; or **(b)** the state component V_j collides with a state component computed in a previous reseed call, $V_j \in \{V_{l'}, \ldots, V_{j-1}\}$. (Note that if V_j collides with a state variable in the set $\{V_0, \ldots, V_{l'-1}\}$, then this state will have formed part of an earlier query in the set $\{(u_1, v_1), \ldots, (u_{l'}, v_{l'})\} \subset \tau \cap \mathcal{J}$ and so is already accounted for by **(a)**.) Notice that if Fresh_j is true for some $j \in [l', d-1]$, then the queries made to compute the following state component V_{j+1} will all be on previously unqueried points. (We note that due to the prepended counter, none of the queries made to compute V_{j+1} can collide with each other. Moreover, due to the separated domains, the queries made to compute the constants C_j for $j \in [1, d]$ can never collide with those made to compute the counters V_j.)

Now, the maximality of l' implies that the transcript τ contains no queries $u \in \mathcal{J}$ of the form $(*)_8 \,\|\, (L)_{32} \,\|\, 0\text{x}01 \,\|\, V_{l'} \,\|\, Z$. As such, it follows that the $\Pr[\,\neg\mathsf{Fresh}_{l'}\,] = 0$. This implies that the state component $V_{l'+1}$ is computed as the result of all fresh H queries, and is therefore uniformly distributed over $\{0,1\}^L$. Now conditioned on $\mathsf{Fresh}_{l'}$, it follows that the probability that $\neg\mathsf{Fresh}_{l'+1}$ occurs is upper bounded by $\frac{(q_J+1)}{2^L}$. (Here, the $\frac{q_J}{2^L}$ accounts for the probability that there exists a query in $\tau \cap \mathcal{J}$ satisfying **(a)**, and the additional $\frac{1}{2^L}$ accounts for the probability that $V_{l'+1}$ collides with $V_{l'}$ as per **(b)**. Inductively applying this argument yields that for each $k \in [1, d-1-l']$, it holds that $\Pr\left[\neg\mathsf{Fresh}_{l'+k} \mid \wedge_{j=l'}^{l'+k-1} \mathsf{Fresh}_j\right] \leq \frac{(q_J+k)}{2^L}$. Using this bound, and the fact that $\Pr[\neg\mathsf{Fresh}_{l'}] = 0$ and $d-1-l' \leq d-1$, it follows that

$$\begin{aligned}\Pr\left[\wedge_{j=l'}^{d-1}\mathsf{Fresh}_j\right] &\geq 1 - \sum_{k=0}^{d-1-l'} \Pr\left[\neg\mathsf{Fresh}_{l'+k} \mid \wedge_{j=l'}^{l'+k-1} \mathsf{Fresh}_j\right] \\ &\geq 1 - \sum_{k=1}^{d-1} \frac{(q_J+k)}{2^L} \\ &= 1 - \frac{(d-1)\cdot(2q_J+d)}{2^{L+1}} \ .\end{aligned}$$

Now, notice that if $\wedge_{j=l'}^{d-1}\mathsf{Fresh}_j$ is true then V_d is computed as a result of all fresh queries to H, and so the resulting state component is uniformly distributed over $\{0,1\}^L$. As such, the probability that the required value of V_d (as dictated by the transcript) is hit is equal to 2^{-L}. Putting this all together, we conclude that

$$\Pr[\mathsf{T}_0 = \tau] \geq \left(1 - \frac{(d-1)\cdot(2q_J+d)}{2^{L+1}}\right) \times \Pr[\mathsf{T}_1 = \tau] \ .$$

Rearranging and substituting in $d \leq q_C$ then proves Lemma 4.8. □

We now bound the probability that a compatible transcript in the ideal world is Bad.

Lemma 4.9. *Letting* Bad *denote the set of bad transcripts as defined above, it holds that*

$$\Pr[\mathsf{T}_1 \in \mathsf{Bad}] \leq \frac{q_J}{2^{\gamma^*}} + \frac{q_J^2}{2^L} \ .$$

Proof. Since we are now in the random world, the challenge state (V_d, C_d, cnt_d) is computed by choosing $V_d \leftarrow_\$ \{0,1\}^L$, setting $C_d = \mathsf{HASH\text{-}DRBG\_df}^{\mathsf{H}}(0\text{x}00 \,\|\, V_d)$ and $cnt_d = 1$, and returning $S_d = (V_d, C_d, cnt_d)$. We let Chain denote the event that a compatible ideal world transcript τ contains a set of queries such that $\tau \in \mathsf{Bad}$. We define the notion of a *potential chain* as follows:

Definition 4.3. *We say that a sequence of (not necessarily distinct) queries* $\mathcal{X}' = \{(u_1, v_1), \ldots, (u_d, v_d)\}$ *is a potential chain if*

- $(u_\iota, v_\iota) \in \mathcal{J}$ *for all* $(u_\iota, v_\iota) \in \mathcal{X}$*; and*
- *For* $j \in [1, d]$ *it holds that* u_j *has decomposition* $(*, X_{j-1}, **)$*, where* $**$ *denotes an arbitrary string in* $\{0,1\}^{p \leq i \leq \overline{p}}$ *and* $X_{j-1} = \mathrm{left}(v_{j-1}, L)$ *where* $V_0 = \mathrm{left}(v_0, L)$*.*

In words, a set of queries that form a potential chain are the same as those which form a bad transcript except we drop the condition that u_j contains the correct input $I_{\mu+j}$ in its decomposition. Moreover, notice that each potential chain defines a candidate sequence of inputs $Z = (Z_1, \ldots, Z_d)$, and a potential chain results in the transcript becoming Bad if $Z_j = I_{\mu+j}$ for $j \in [1, d]$.

We may visualise the potential chains in the form of an undirected graph as follows. We set $\omega_0 = V_0$ to be the root of the graph. We then use the queries in the transcript to construct paths of length d from the root by adding an edge between vertices ω_j, ω_k if there exists a query $(u, v) \in \tau \cap \mathcal{J}$, with decomposition $(*, \omega_j, **)$, and $\mathrm{left}(v, L) = \omega_k$. Notice that the potential chains correspond to paths of length d starting at the root V_0, and that, since $\mathcal{A}$ makes q_J queries of the required form and $d \geq 1$, the graph can contain at most q_J edges. With this in place, we say that a transcript τ induces the event Coll if:

- There exists a query $(u_i, v_i) \in \tau \cap \mathcal{J}$ such $\mathrm{left}(v_i, L) = v_k$ for some query (u_k, v_k) where $1 \leq k < i$; or
- There exists a query $(u_i, v_i) \in \tau \cap \mathcal{J}$ such that $\mathrm{left}(v_i, L) = u_k$ for some query (u_k, v_k) where $1 \leq k \leq i$.

Now notice that if Coll does not occur then no newly added edge can loop back to a previously added vertex, and so the query graph must be a tree. In this case there can be at most one path to each leaf from the root. Since each node added to the graph corresponds to a query in $\tau \cap \mathcal{J}$ of which $\mathcal{A}$ makes q_J such queries, it follows that there can be at most q_J potential chains.

We now bound the probability that Coll occurs. Notice that conditioned on Coll having *not* occurred for the first $(k-1)$ queries in the set $\tau \cap \mathcal{J}$, then the k^{th} query of this form will be computed as the result of all fresh queries to H. As such, the resulting counter / vertex is uniformly distributed over $\{0,1\}^L$. It is then straightforward to verify that the probability that the k^{th} query sets Coll is upper bounded by $\frac{(2k-1)}{2^L}$. Summing over $k \in [1, q_J]$ then yields

$$\Pr[\,\mathsf{Coll}\,] \leq \sum_{k=1}^{q_J} \frac{(2k-1)}{2^L} = \frac{q_J^2}{2^L}\,.$$

Now as mentioned above, conditioned on Coll *not* occurring, a transcript can contain at most q_J potential chains. We now bound the probability that any of these q_J potential chain forms an actual chain. Since we are in the ideal world, $\mathcal{A}$'s view of game Extract is completely independent of the inputs $I_{\mu+1}, \ldots, I_{\mu+d}$ right up until the very end of the experiment when these values are revealed to $\mathcal{A}$ and which crucially is *after* $\mathcal{A}$ has finished making its queries to H. Therefore we can without loss of generality modify the experiment so that we only compute the states $S_i \leftarrow \mathsf{refresh}^{\mathsf{H}}(S_{i-1}, I_{\mu+i})$ for $i \in [1, d]$ *after* $\mathcal{A}$ has made all of his q_{H} queries (but before the unseen inputs $I_{\mu+1}, \ldots, I_{\mu+d}$ and sampler coins ω are revealed to $\mathcal{A}$ at the end of the game). We let τ' denote the transcript information available to $\mathcal{A}$ up to and including the point in game Extract at which he makes his last guess; namely

$$\tau' = ((x_1, y_1), \ldots, (x_q, y_q), (V_0, C_0, cnt_0), (V_d, C_d, cnt_d), \\ I_1, \ldots, I_\mu, I_{\mu+d+1}, \ldots, I_{q_{\mathcal{D}}}, (\gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}+1}, \mu, d)\ .$$

Notice in particular that the inputs $I_{\mu+1}, \ldots, I_{\mu+d}$ which were used to compute $\mathcal{A}$'s challenge are *not* included, since they are still hidden from $\mathcal{A}$ at this stage. Moreover, since V_d and C_d are independent of these entropy inputs in the ideal world, these state components reveal nothing to $\mathcal{A}$ about the hidden inputs. Now letting $\mathcal{F}$ denote the set of partial transcripts τ' for which Coll is false, it follows that

$$\begin{aligned} \Pr[\mathsf{Chain}] &\leq \Pr[\mathsf{Coll}] + \Pr[\mathsf{Chain} \wedge \neg\mathsf{Coll}] \\ &\leq \frac{q_J^2}{2^L} + \sum_{\tau' \in \mathcal{F}} \Pr\left[\,\mathsf{Chain} \mid \tau'\,\right] \cdot \Pr\left[\,\tau'\,\right]\ , \end{aligned} \tag{4.4}$$

where $\Pr[\tau']$ denotes the probability that the execution of Extract produces partial transcript τ' (that is to say: $\mathcal{A}$ makes the required set of queries, $V_d \leftarrow_\$ \{0,1\}^L$, and so on and so forth). Fix $\tau' \in \mathcal{F}$, and let $Z_{\tau'}$ denote the set of potential chains within the partial transcript τ'. It follows that

$$\begin{aligned} \Pr\left[\,\mathsf{Chain} \mid \tau'\,\right] &= \Pr\left[\,(I_{\mu+1}, \ldots, I_{\mu+d}) \in Z_{\tau'} \mid \tau'\,\right] \\ &= \sum_{(Z_1,\ldots,Z_d)\in Z_{\tau'}} \Pr\left[\bigwedge_{i=\mu+1}^{\mu+d} Z_i = I_{\mu+i} \mid \tau'\right] \\ &\leq \sum_{(Z_1,\ldots,Z_d)\in Z_{\tau'}} \prod_{i=1}^{d} \Pr\left[Z_i = I_{\mu+i} \mid \bigwedge_{i=\mu+1}^{\mu+i-1} Z_i = I_{\mu+i}, \tau'\right] \\ &\leq \sum_{(Z_1,\ldots,Z_d)\in Z_{\tau'}} \prod_{i=1}^{d} 2^{-\gamma_{\mu+i}} \leq q_J \cdot 2^{-\gamma^*}\ . \end{aligned}$$

```
proc. main // $G_0$
$\mathsf{H} \leftarrow_\$ \mathcal{H}$ ; $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 1$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
  $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow_\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d) \leftarrow_\$ \mathcal{A}^{\mathsf{H},\mathrm{Sam}}(I_1, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > q_{\mathcal{D}} + 1$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
  Return $\perp$
$(V_0, C_0, cnt_0) \leftarrow S_0$
For $j = 1, \ldots, d$
  $temp_{V_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $temp_{V_j} \leftarrow temp_{V_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x01} \,\|\, V_{j-1} \,\|\, I_{\mu+j})$
  $V_j \leftarrow \mathrm{left}(temp_{V_j}, L)$
  $temp_{C_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $temp_{C_j} \leftarrow temp_{C_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x00} \,\|\, V_j)$
  $C_j \leftarrow \mathrm{left}(temp_{C_j}, L)$ ; $cnt_j \leftarrow 1$
$S_d \leftarrow (V_d, C_d, cnt_d)$
$(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(R^*, S^*, (I_k)_{k>\mu+d})$
Return $b^*$
```

```
proc. main // $G_1$, $\boxed{G_2}$
$\mathsf{H} \leftarrow_\$ \mathcal{H}$ ; $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 1$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
  $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow_\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d) \leftarrow_\$ \mathcal{A}^{\mathsf{H},\mathrm{Sam}}(I_1, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > q_{\mathcal{D}} + 1$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
  Return $\perp$
$S_d \leftarrow_\$ \mathsf{M}^{\mathsf{H}}(\varepsilon)$
$(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$ // $G_1$ only
$\boxed{R^* \leftarrow_\$ \{0,1\}^\ell}$
$\boxed{S^* \leftarrow_\$ \mathsf{M}^{\mathsf{H}}(S_d)}$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(R^*, S^*, (I_k)_{k>\mu+d})$
Return $b^*$
```

```
proc. main // $G_3$
$\mathsf{H} \leftarrow_\$ \mathcal{H}$ ; $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 1$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
  $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow_\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d) \leftarrow_\$ \mathcal{A}^{\mathsf{H},\mathrm{Sam}}(I_1, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > q_{\mathcal{D}} + 1$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
  Return $\perp$
$(V_0, C_0, cnt_0) \leftarrow S_0$
For $j = 1, \ldots, d$
  $temp_{V_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $temp_{V_j} \leftarrow temp_{V_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x01} \,\|\, V_{j-1} \,\|\, I_{\mu+j})$
  $V_j \leftarrow \mathrm{left}(temp_{V_j}, L)$
  $temp_{C_j} \leftarrow \varepsilon$
  For $i = 1, \ldots, m$
    $temp_{C_j} \leftarrow temp_{C_j} \,\|\, \mathsf{H}((i)_8 \,\|\, (L)_{32} \,\|\, \mathtt{0x00} \,\|\, V_j)$
  $C_j \leftarrow \mathrm{left}(temp_{C_j}, L)$ ; $cnt_j \leftarrow 1$
$S_d \leftarrow (V_d, C_d, cnt_d)$
$R^* \leftarrow_\$ \{0,1\}^\ell$
$S^* \leftarrow_\$ \mathsf{M}^{\mathsf{H}}(S_d)$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(R^*, S^*, (I_k)_{k>\mu+d})$
Return $b^*$
```

```
proc. Sam // $G_0, G_1, G_2, G_3$
$\mu = \mu + 1$
Return $I_\mu$
```

```
proc. $\mathsf{H}(X)$ // $G_0, G_1, G_2, G_3$
Return $\mathsf{H}(X)$
```

Figure 4.10: Games for proof of Lemma 4.6 ($\overline{\mathrm{Rec}}$ security of HASH-DRBG).

Here the second to last inequality follows since the sampler is $(q_{\mathcal{D}}^{+}, \gamma^*)$-legitimate. The last inequality follows since $\tau' \in \mathcal{F}$ implies that $Z_{\tau'}$ contains at most q_J potential chains. Substituting into equation (4.4) then proves Lemma 4.9. Combining Lemmas 4.8 and 4.9 via Theorem 2.1 then concludes the proof of Lemma 4.7 on the extraction properties of HASH-DRBG's refresh algorithm. □

With this result is place, we return to bounding the $\overline{\mathrm{Rec}}$ security of HASH-DRBG. We argue by a series of game hops, shown in Figure 4.10. We begin by defining game G_0, which is a rewriting of game $\overline{\mathrm{Rec}}$ for HASH-DRBG with challenge bit $b = 0$. It follows that

$$\Pr\left[\, G_0 \Rightarrow 1 \,\right] = \Pr\left[\, \overline{\mathrm{Rec}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}^{\mathcal{A},\mathcal{D}} \Rightarrow 1 \;\middle|\; b = 0 \,\right] .$$

Next we define game G_1, which is identical to G_0 except that rather than computing S_d via a sequence of $\mathsf{refresh}^{\mathsf{H}}$ calls we instead set $S_d \leftarrow_\$ \mathsf{M}^{\mathsf{H}}(\varepsilon)$. We claim that for any attacker / sampler pair $(\mathcal{A}, \mathcal{D})$ making q_{H} queries to H, there exists a $(q_{\mathsf{H}} + n + 1, q_C)$-adversary

$\mathcal{B}$ in game Extract against $\mathsf{HASH\text{-}DRBG}$, for which at most q_{H} of $\mathcal{B}$'s H queries lie in $\mathcal{J}$, such that

$$\begin{aligned} |\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| &\leq \mathbf{Adv}^{\mathrm{ext}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{B},\mathcal{D}) \\ &\leq \frac{q_{\mathsf{H}}}{2^{\gamma^*}} + \frac{(q_C - 1)\cdot(2q_{\mathsf{H}} + q_C) + 2q_{\mathsf{H}}^2}{2^{L+1}} \;. \end{aligned}$$

To see this, let $\mathcal{B}$ be the adversary who proceeds as follows. $\mathcal{B}$ simulates $\mathcal{A}$'s view of the game, forwarding $\mathcal{A}$'s Sam and H queries to his own oracles and returning the responses to $\mathcal{A}$. When $\mathcal{A}$ outputs a state / index pair (S_0, d), $\mathcal{B}$ forwards these to his challenger. $\mathcal{B}$ receives state S_d in response along with the remaining entropy samples. $\mathcal{B}$ computes $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$ using his own H oracle and returns these along with the entropy samples to $\mathcal{A}$, again using his oracle to answer all remaining queries. At the end of the game, $\mathcal{B}$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}$'s challenge bit is equal to 0 and so he receives the real state in his challenge then this perfectly simulates G_0; otherwise it perfectly simulates G_1. To verify the query budget, notice that $\mathcal{B}$ queries all of $\mathcal{A}$'s q_{H} queries to his own oracle and makes an additional $n+1$ queries simulating $\mathsf{next}^{\mathsf{H}}$. Noting that none of the $n+1$ queries made while simulating $\mathsf{next}^{\mathsf{H}}$ lie in $\mathcal{J}$, and substituting into the bound on the Extract security of $\mathsf{HASH\text{-}DRBG}$ from Lemma 4.7, then implies the claim.

Next we define game G_2, which is identical to G_1 except we now set $R^* \leftarrow\$ \{0,1\}^{\beta}$ and $S^* \leftarrow\$ \mathsf{M}^{\mathsf{H}}(S_d)$ rather than computing these values as $(R^*, S^*) \leftarrow \mathsf{next}^{\mathsf{H}}(S_d, \beta)$. We claim that there exists an adversary $\mathcal{C}$ in game Next, who makes the same number of H queries as $\mathcal{A}$, such that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1] \leq \mathbf{Adv}^{\mathrm{next}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{C}) \leq \frac{q_{\mathsf{H}} \cdot n}{2^{\ell-1}} \;.$$

To see this, let $\mathcal{C}$ be the adversary who proceeds as follows. $\mathcal{C}$ uses the code of the sampler to generate all entropy samples, and passes the corresponding entropy estimates / side information to $\mathcal{A}$. For each of $\mathcal{A}$'s initial queries, $\mathcal{C}$ simulates $\mathcal{A}$'s random oracle by forwarding all of $\mathcal{A}$'s queries to his own random oracle and returning the responses. $\mathcal{C}$ simulates $\mathcal{A}$'s Sam oracle by returning the appropriate entropy sample to $\mathcal{A}$. When $\mathcal{A}$ outputs a state S_0, $\mathcal{A}_2$ outputs the state ε as his challenge state, receiving (R^*, S^*) in response. $\mathcal{A}_2$ passes these to $\mathcal{A}$ along with the remaining entropy samples and continues simulating $\mathcal{A}$'s random oracle by querying his own oracle as before. At the end of the game, $\mathcal{C}$ outputs whatever bit $\mathcal{A}$ does. If $\mathcal{C}$'s challenge bit is equal to 0 then this perfectly simulates game G_1; otherwise it perfectly simulates G_2. As such, invoking Lemma 4.4 then proves the claim.

Next we define game G_3, which is identical to G_2 except we return to computing S_d via iterative reseeding as opposed to setting $S_d \leftarrow\!\!\$\ \mathsf{M}^{\mathsf{H}}(\varepsilon)$. An analogous argument to that above implies that there exists a (q_{H}, q_C)-adversary $\mathcal{B}'$ such that

$$\begin{aligned}|\Pr[\,G_2 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| &\leq \mathbf{Adv}^{\mathrm{ext}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{B}', \mathcal{D}) \\ &\leq \frac{q_{\mathsf{H}}}{2^{\gamma^*}} + \frac{(q_C - 1)\cdot(2q_{\mathsf{H}} + q_C) + 2q_{\mathsf{H}}^2}{2^{L+1}}\ .\end{aligned}$$

Here the slightly lower query budget is because $\mathcal{B}'$ no longer needs to make the $n+1$ queries to H to simulate $\mathsf{next}^{\mathsf{H}}$. Now G_3 is identical to game $\overline{\mathrm{Rec}}$ with challenge bit $b = 1$, and so

$$\Pr[\,G_3 \Rightarrow 1\,] = \Pr\left[\,\overline{\mathrm{Rec}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta} \Rightarrow 1 \mid b = 1\right]\ .$$

Finally putting this altogether, we conclude that

$$\begin{aligned}\mathbf{Adv}^{\overline{\mathrm{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A}, \mathcal{D}) &= |\Pr[\,G_0 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| \\ &\leq \frac{q_{\mathsf{H}}}{2^{\gamma^*-1}} + \frac{(q_C - 1)\cdot(2q_{\mathsf{H}} + q_C) + 2q_{\mathsf{H}}^2}{2^L} + \frac{q_{\mathsf{H}}\cdot n}{2^{\ell-1}}\ .\end{aligned}$$

□

This concludes the proof of $\overline{\mathrm{Rec}}$ security for HASH-DRBG, en-route to which we analysed the extraction properties of HASH-DRBG's refresh procedure. Using Theorem 4.1 to combine Lemmas 4.3, 4.5 and 4.6 — which bound the $\overline{\mathrm{Init}}, \overline{\mathrm{Pres}}$, and $\overline{\mathrm{Rec}}$ security of HASH-DRBG respectively — proves Theorem 4.2, and completes our analysis of the robustness of HASH-DRBG in the ROM.

4.7 Analysis of HMAC-DRBG

In this section we present our analysis of HMAC-DRBG. We give both positive and negative results, showing that the security guarantees of HMAC-DRBG differ depending on whether additional input is provided in next calls.

4.7.1 Negative Result: HMAC-DRBG Called without Additional Input is Not Forward Secure

We present an attack which breaks the forward security of HMAC-DRBG if called without additional input. This contradicts the claim in the standard that the NIST DRBGs are backtracking resistant. Since Rob security implies Fwd security, this rules out a proof of robustness in this case also.

The attack. Consider the application of update at the conclusion of a next call for HMAC-DRBG (Figure 4.1). Notice that if $addin = \varepsilon$, then the final two lines of update are not executed. In this case, the updated state $S^* = (K^*, V^*, cnt^*)$ is of the form $V^* = \mathsf{HMAC}(K^*, r^*)$ where r^* is the final output block produced in the call. An attacker $\mathcal{A}$ in game Fwd who makes a RoR query with $addin = \varepsilon$ to request ℓ-bits of output, followed immediately by a Get query to learn S^*, can easily test this relation. If it does not hold, they know the challenge output is truly random. We note that the observation that V^* depends on r^* is also implicit in the proof of pseudorandomness by Hirose [78]; however, the connection to forward security is not made in that work. To concretely bound $\mathcal{A}$'s advantage we define game Fwd$^{\$}$, which is identical to game Fwd against HMAC-DRBG except the PRNG is initialised with an 'ideally distributed' state $S_0 = (K_0, V_0, cnt_0)$ where $K_0, V_0 \leftarrow_{\$} \{0,1\}^{\ell}$ and $cnt_0 \leftarrow 1$. The attacker's job can only be harder in Fwd$^{\$}$, since they cannot exploit any flaws in the setup procedure. We put all this together in the following theorem.

Theorem 4.3. *Let* PRNG *be* HMAC-DRBG *built from the function* HMAC : $\{0,1\}^{\ell} \times \{0,1\}^{\leq \omega} \rightarrow \{0,1\}^{\ell}$, *with associated parameter set* $(p, \overline{p}, \alpha, \beta_{max})$ *for which* $\beta_{max} \geq \ell$. *We give explicit efficient adversaries* $\mathcal{A}$ *and* $\mathcal{B}$, *such that for any sampler* $\mathcal{D}$, *it holds that*

$$\mathbf{Adv}^{\text{fwd-\$}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) \geq 1 - 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{HMAC}}(\mathcal{B}, 2) - 2^{-(\ell-1)} .$$

$\mathcal{A}$ makes one RoR query in which additional input is not included, and one Get query. $\mathcal{B}$ runs in the same time as $\mathcal{A}$, and makes two queries to their RoR function oracle.

Discussion. The first negative term on the right-hand side of the above equation corresponds to the advantage of the attacker $\mathcal{B}$ who tries to break the PRF-security of HMAC given two queries to its real-or-random function oracle; since HMAC is widely understood to be a secure PRF, we expect this term to be small. This term arises from reductions to the PRF-security of HMAC used in the proof to argue that the probability that $V^* = \mathsf{HMAC}(K^*, R^*)$ for a random output $R^* \leftarrow_{\$} \{0,1\}^\ell$ is very small; since the relation holds with probability one if R^* is the real output, this allows the attacker to distinguish the two cases with high probability. Moreover ℓ denotes the output length of HMAC and so the second negative term will be small for all commonly used hash functions. This implies that $\mathcal{A}$ succeeds with probability close to one, making this an effective attack.

The theorem assumes the parameters of PRNG are such that outputs of length ℓ bits may be requested (i.e., $\beta_{max} \geq \ell$). Since ℓ is the output length of the underlying HMAC function this is not much of a restriction; however, the result can be generalised with a bounded decrease in success probability for implementations for which $\beta_{max} < \ell$, subject to $\mathcal{A}$ doing more work to brute-force the truncated output bits. We note that since the sampler $\mathcal{D}$ is not invoked during the course of the attack (recall that the initial state is sampled uniformly at random in game $\mathrm{Fwd}^{\$}$) the attack works for any choice of $\mathcal{D}$.

Proof. Let $\mathcal{A}$ be the adversary who proceeds as follows. $\mathcal{A}$ queries (ℓ, ε) to his RoR oracle, receiving R^* is response. $\mathcal{A}$ then makes a Get query to receive state $S^* = (K^*, V^*, cnt^*)$. $\mathcal{A}$ then checks if

$$V^* \stackrel{?}{=} \mathsf{HMAC}(K^*, R^*) \ .$$

If so, $\mathcal{A}$ outputs 0; else he returns 1. It is straightforward to verify from the pseudocode description of next that if $b = 0$ and $\mathcal{A}$ receives a real output in his challenge then this relation will hold with probability one; as such $\Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } \mathrm{Fwd}^{\$,\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \mid b = 0\right] = 0$.

To bound $\Pr\left[\mathcal{A} \Rightarrow 1 \text{ in } \mathrm{Fwd}^{\$,\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\gamma^*} \mid b = 1\right]$, we argue by a series of game hops. Let G_0 be identical to game $\mathrm{Fwd}^{\$}$ against PRNG for $\mathcal{A}$ with challenge bit $b = 1$. Let $S_0 = (K_0, V_0, cnt_0)$ denote the initial state of PRNG, where recall that $K_0, V_0 \leftarrow_{\$} \{0,1\}^\ell$. Notice that the RoR query made by $\mathcal{A}$ induces the challenger to update the state via $(R, S^*) \leftarrow \mathsf{next}(S_0, \ell)$. Since $b = 1$, $\mathcal{A}$ receives a random output $R^* \leftarrow_{\$} \{0,1\}^\ell$; however, $\mathcal{A}$'s subsequent Get query allows him to learn the real state $S^* = (K^*, V^*, cnt^*)$. This state is computed as $V_0' \leftarrow \mathsf{HMAC}(K_0, V_0)$, $K^* \leftarrow \mathsf{HMAC}(K_0, V_0' || 0\mathrm{x}00)$, and $V^* \leftarrow \mathsf{HMAC}(K^*, V_0')$. We now define game G_1, which is identical to G_0 except we sample V_0', $K^* \leftarrow_{\$} \{0,1\}^\ell$ instead of computing these variables via $\mathsf{HMAC}(K_0, \cdot)$. Since $K_0 \leftarrow_{\$} \{0,1\}^\ell$, it is straightforward to verify that both G_0 and G_1 can be perfectly simulated by an attacker $\mathcal{B}_2$ in the PRF game against HMAC using two RoR queries and who runs in the same time as $\mathcal{A}$. This combined with the fact that, due to their disjoint domains, the queries made to compute V_0' and K^* can never collide, implies that

$$|\Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_0] - \Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_1] \le \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HMAC}}(\mathcal{B}_1, 2) \, .$$

Now recall that $\mathcal{A}$ outputs 1 if the relation $V^* = \mathsf{HMAC}(K^*, V_0') \stackrel{?}{=} \mathsf{HMAC}(K^*, R^*)$ does *not* hold. Since K^*, R^* and V_0' are all chosen randomly from $\{0,1\}^\ell$ in G_1, it follows that $\Pr[\mathcal{A} \Rightarrow 1 \text{ in } G_1] = 1 - \epsilon_{coll}$ where

$$\epsilon_{coll} = \Pr\left[V^* = V' \;:\; V_0', R^*, K^* \leftarrow_\$ \{0,1\}^\ell \,; V^* \leftarrow \mathsf{HMAC}(K^*, V_0'), V' \leftarrow \mathsf{HMAC}(K^*, R^*)\right] .$$

We claim that there exists an attacker $\mathcal{B}_2$ in the PRF security game against HMAC such that

$$\epsilon_{coll} \le \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HMAC}}(\mathcal{B}_2, 2) + 2^{-(\ell-1)} \, .$$

To see this, consider an attacker $\mathcal{B}_2$ in the PRF security game against HMAC who proceeds as follows. $\mathcal{B}_2$ chooses $V_0', R^* \leftarrow_\$ \{0,1\}^\ell$, queries these to his RoR oracle receiving V^*, V' in response, and outputs 1 if $V^* = V'$ and 0 otherwise. Notice that in the case that $\mathcal{B}_2$'s RoR oracle implements the real HMAC function then this perfectly simulates the experiment determining ϵ_{coll} and so $\Pr[\mathcal{B}_2 \Rightarrow 1 \mid b = 0] = \epsilon_{coll}$. Moreover if $b = 1$ and so $\mathcal{B}_2$'s oracle implements a random function, it follows that

$$\begin{aligned}\Pr[\mathcal{B}_2 \Rightarrow 1 \mid b = 1] &= \Pr\left[V^* = V' \wedge V_0' = R^*\right] + \Pr\left[V^* = V' \wedge V_0' \ne R^*\right] \\ &\le 2^{-\ell} + 2^{-\ell} = 2^{-(\ell-1)} \, .\end{aligned}$$

To see this, notice that if $V_0' = R^*$ then $V^* = V'$ with probability one. Since $V_0', R^* \leftarrow_\$ \{0,1\}^\ell$, this implies the first term in the bound. Moreover, if $V_0' \ne R^*$, then V^*, V' are both the results of fresh queries to the random function and so are uniformly distributed over $\{0,1\}^\ell$, accounting for the second term in the bound. Combining these observations via a standard argument then implies the claim.

Putting this altogether, and letting $\mathcal{B}$ be the attacker who tosses a coin to decide whether to run adversary $\mathcal{B}_1$ or $\mathcal{B}_2$, we conclude that

$$\mathbf{Adv}^{\text{fwd-\$}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) \ge 1 - 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HMAC}}(\mathcal{B}, 2) - 2^{-(\ell-1)} \, .$$

□

Discussion. We only became aware of this attack while attempting to write down a formal proof of the forward security of HMAC-DRBG. This illustrates the importance of

```
$\mathsf{M}^{\mathsf{HMAC}}(S)$
If $S = \varepsilon$
    $cnt \leftarrow 0$
Else $(K, V, cnt) \leftarrow S$
$K', V' \leftarrow\$ \{0,1\}^\ell$
$cnt' \leftarrow cnt + 1$
$S' \leftarrow (K', V', cnt')$
Return $S'$
```

Figure 4.11: Masking function for proof of Theorem 4.4.

formally proving security claims, however obvious they may at first appear. This also highlights the dangers of flexibility in algorithm specifications. The option to include additional input here appears fairly inconsequential; however, as we have just seen, it turns out to change the functionality of the next algorithm in a way that impacts security. Overly flexible specifications increase the burden on designers by introducing multiple cases to consider, and, as we have just seen, the assumption that similar algorithms will have equally similar security properties is dangerous.

4.7.2 Positive Result: Robustness of HMAC-DRBG with Additional Input in the ROM

In this section, we prove that HMAC-DRBG is robust in the ROM when additional input is used with respect to a restricted (but realistic) class of samplers. We model the function HMAC : $\{0,1\}^\ell \times \{0,1\}^{\leq\omega} \rightarrow \{0,1\}^\ell$ as a keyed random oracle, whereby each fresh query of the form $(K, X) \in \{0,1\}^\ell \times \{0,1\}^{\leq\omega}$ is answered with an independent random ℓ-bit string.

Rationale. While a standard model proof of the $\overline{\text{Pres}}$ security of HMAC-DRBG is possible via a reduction to the PRF-security of HMAC, how to achieve the same for $\overline{\text{Init}}$ and $\overline{\text{Rec}}$ is unclear. These results require showing that HMAC is a good randomness extractor. In games $\overline{\text{Init}}$ and $\overline{\text{Rec}}$, the HMAC key is chosen by or known to the attacker, and so we cannot appeal to PRF-security. Entropy samples are non-uniform, so a dual-PRF assumption does not suffice either. As such, some idealised assumption on HMAC or the underlying hash / compression function seems to be inherently required. The extraction properties of HMAC (under various idealised assumptions) were studied in [58]. However, these consider a single-use version of extraction which is weaker than what is required here, and typically require inputs containing much more entropy than is required by the standard, and so are not generally applicable to real-world implementations of HMAC-DRBG.

By modelling HMAC as a keyed RO, we can analyse HMAC-DRBG with respect to the entropy levels of inputs specified in the standard (and at levels which are practical for real-world applications). This is a fairly standard assumption, having been made in other works in which HMAC is used with a known key or to extract from lower entropy sources e.g., [96, 97, 130]. In [61], HMAC was proven to be indifferentiable from a random oracle for all commonly deployed parameter settings (although since robustness is a multi-stage game, the indifferentiability result cannot be applied generically here [129]).

Discussion. A standard model proof of $\overline{\text{Rec}}$ security for HMAC-DRBG would be a stronger and more satisfying result. While idealising HMAC or the underlying hash / compression function seems inherent, a result under weaker idealised assumptions is an important open problem. Despite this, we feel our analysis is a significant forward step from existing works. Ours is the first analysis of the full specification of HMAC-DRBG; prior works [78, 168] omit reseeding and initialisation, assuming HMAC-DRBG is initialised with a state for which $K, V \leftarrow\$ \{0,1\}^{\ell}$, which is far removed from HMAC-DRBG in a real system. Our work is also the first to consider security properties stronger than pseudo-randomness. We hope our result is a valuable first step to progress the understanding of this widely deployed (yet little analysed) PRNG, and a useful starting point for further work to extend.

Sampler. Our proof is with respect to the class of samplers $\{\mathcal{D}\}_{\gamma^*}$, defined to be the set of $(q_{\mathcal{D}}^{+}, \gamma^*)$-legitimate samplers for which $\gamma_i \geq \gamma^*$ for $i \in [1, q_{\mathcal{D}} + 1]$ (i.e., each sample I contains γ^*-bits of entropy). This is a simplifying assumption, making the proof of $\overline{\text{Rec}}$ security less complex. However, we stress that this is the entropy level per sample required by the standard, and so this is precisely the restriction imposed on allowed entropy sources. An H-coefficient analysis, as in the proof of Lemma 4.6, seems likely to yield a fully general result.

Proof of robustness. With this in place, we present the following theorem bounding the robustness of HMAC-DRBG. The proof follows from a number of lemmas which we discuss below, combined with Theorem 4.1 (for which it is straightforward to verify that, for HMAC-DRBG, $q_{ref} = 4$ and $q_{nxt} = n+8$ where $n = \lceil \beta/\ell \rceil$). Our proof is with respect to the masking function $\mathsf{M}^{\mathsf{HMAC}}$ shown in Figure 4.11. We note that $\mathsf{M}^{\mathsf{HMAC}}$ does not make any calls to HMAC, and so $q_{\mathsf{M}} = 0$.

Theorem 4.4. *Let* PRNG *be* HMAC-DRBG *with associated parameter set* $(p, \overline{p}, \alpha, \beta_{max})$*, built from the function* $\mathsf{HMAC} : \{0,1\}^{\ell} \times \{0,1\}^{\leq \omega} \to \{0,1\}^{\ell}$ *which we model as a keyed random oracle. Let* $\mathsf{M}^{\mathsf{HMAC}}$ *be the masking function shown in Figure 4.11. Then for any* $(q_{\mathsf{H}}, q_R, q_{\mathcal{D}}, q_C, q_S)$*-attacker* $\mathcal{A}$ *in game* $\overline{\mathrm{Rob}}_{\beta}$ *against* HMAC-DRBG *who always outputs addin* $\neq \varepsilon$*, and any* $(q_{\mathcal{D}}^{+}, \gamma^*)$*-legitimate sampler* $\mathcal{D} \in \{\mathcal{D}\}_{\gamma^*}$*, it holds that*

$$\mathbf{Adv}^{\overline{\mathrm{rob}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,\beta}(\mathcal{A}) \leq q_R \cdot (\bar{q}_{\mathsf{H}} \cdot \epsilon_1 + \epsilon_2) \cdot 2^{-(2\ell-1)} + \bar{q}_{\mathsf{H}} \cdot 2^{-(\ell-2)} + q_R \cdot (\bar{q}_{\mathsf{H}} \cdot (n+3) + \epsilon_3) \cdot 2^{-(\ell-2)}$$
$$+ (\bar{q}_{\mathsf{H}} \cdot (2q_R + (1 + 2^{-2\ell})) \cdot 2^{-(\gamma^*-1)} + 2^{-(2\ell-1)} \, .$$

Here $\epsilon_1 = 12q_C + 10 + (4q_C - 2) \cdot 2^{-\gamma^*}$, $\epsilon_2 = (q_C \cdot (10q_C + 4n + 18 + (q_C - 1) \cdot 2^{-(\gamma^*-1)}) + 6n + 16)$, *and* $\epsilon_3 = n(n+1)$*. Moreover,* $n = \lceil \beta/\ell \rceil$ *and* $\bar{q}_{\mathsf{H}} = (q_{\mathsf{H}} + 4 \cdot q_{\mathcal{D}} + (n+8) \cdot q_R)$*.*

Concrete example. For HMAC-SHA-512, $\ell = 512$ and the bound is dominated by the $\mathcal{O}(\bar{q}_{\mathsf{H}} \cdot q_R) \cdot 2^{-(\gamma^*-1)}$ term. Supposing $q_{\mathcal{D}} \leq q_R$ (i.e., there are fewer Ref than RoR calls) and n is small, then $\bar{q}_{\mathsf{H}} \cdot q_R \leq q_R \cdot (q_{\mathsf{H}} + c \cdot q_R)$ for some small constant c. Now if HMAC-DRBG is instantiated at strength $\gamma^* = 256$, it achieves a good security margin up to fairly large q_{H}, q_R. At lower γ^* the margins are less good; however, this is likely an artefact of the proof technique.

$\overline{\text{Init}}$ security. The proof of $\overline{\mathrm{Init}}$ security argues that unless attacker $\mathcal{A}$ queries HMAC on certain points which require guessing the value of either the input I_1 with which HMAC-DRBG was seeded, or an intermediate key / counter computed during setup, then — barring an accidental collision in the inputs to the second and fourth HMAC queries made by setup, contributing $2^{-2\ell}$ to the bound — the resulting state is identically distributed to $\mathsf{M}^{\mathsf{HMAC}}(\varepsilon)$. A union bound over these guessing and collision probabilities then yields the lemma.

Lemma 4.10. *Let* PRNG = HMAC-DRBG *and masking function* $\mathsf{M}^{\mathsf{HMAC}}$ *be as specified in Theorem 4.4. Then for any* q_{H}*-adversary* $\mathcal{A}$ *in game* $\overline{\mathrm{Init}}$ *against* HMAC-DRBG*, and any* $(q_{\mathcal{D}}^{+}, \gamma^*)$*-legitimate sampler* $\mathcal{D} \in \{\mathcal{D}\}_{\gamma^*}$*, it holds that*

$$\mathbf{Adv}^{\overline{\mathrm{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}, \mathcal{D}) \leq q_{\mathsf{H}} \cdot ((1 + 2^{-2\ell}) \cdot 2^{-\gamma^*} + 2^{-(\ell-1)}) + 2^{-2\ell} \, .$$

```
proc. main // $G_0, G_1, \boxed{G_2}$
$\sigma_0 \leftarrow \varepsilon$ ; $N \leftarrow \mathcal{N}$
For $i = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_i, I_i, \gamma_i, z_i) \leftarrow\!\$\ \mathcal{D}(\sigma_{i-1})$
$K \leftarrow$ 0x00...00
$V \leftarrow$ 0x01...01
$K' \leftarrow \mathsf{HMAC}(K, V \,\|\, \texttt{0x00} \,\|\, I_1 \,\|\, N)$ ; $\boxed{K' \leftarrow\!\$\ \{0,1\}^\ell}$
$V' \leftarrow \mathsf{HMAC}(K', V)$ ;            $\boxed{V' \leftarrow\!\$\ \{0,1\}^\ell}$
$K_0^* \leftarrow \mathsf{HMAC}(K', V' \,\|\, \texttt{0x01} \,\|\, I_1 \,\|\, N)$ ; $\boxed{K_0^* \leftarrow\!\$\ \{0,1\}^\ell}$
$V_0^* \leftarrow \mathsf{HMAC}(K_0^*, V')$ ;          $\boxed{V_0^* \leftarrow\!\$\ \{0,1\}^\ell}$
$cnt_0^* \leftarrow 1$
$S_0^* \leftarrow (K_0^*, V_0^*, cnt_0^*)$
$b^* \leftarrow\!\$\ \mathcal{A}^{\mathsf{HMAC}}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
Return $b^*$
```

```
proc. main // $G_3, G_4$
$\sigma_0 \leftarrow \varepsilon$ ; $N \leftarrow \mathcal{N}$
For $i = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_i, I_i, \gamma_i, z_i) \leftarrow\!\$\ \mathcal{D}(\sigma_{i-1})$
$K \leftarrow$ 0x00...00
$V \leftarrow$ 0x01...01
$S_0^* \leftarrow\!\$\ \mathsf{M}^{\mathsf{HMAC}}(\varepsilon)$
$b^* \leftarrow\!\$\ \mathcal{A}^{\mathsf{HMAC}}(S_0^*, (I_i)_{i=2}^{q_{\mathcal{D}}+1}, (\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
Return $b^*$

proc. $\mathsf{HMAC}(X)$ // $\boxed{G_0}, G_1, G_2, G_3, \boxed{G_4}$
$Y \leftarrow\!\$\ \{0,1\}^\ell$
If $\mathsf{HMAC}[X] \neq \perp$
    $\mathsf{bad} \leftarrow 1$
    $\boxed{Y \leftarrow \mathsf{HMAC}[X]}$
$\mathsf{HMAC}[X] \leftarrow Y$
Return $Y$
```

Figure 4.12: Games for proof of Lemma 4.10 ($\overline{\text{Init}}$ security of HMAC-DRBG).

Proof. We argue by a series of game hops, shown in Figure 4.12. We assume without loss of generality that $\mathcal{A}$ never repeats a query to the random oracle HMAC. We begin by defining game G_0, which is a rewriting of game Init with $b = 0$ for HMAC-DRBG and $\mathsf{M}^{\mathsf{HMAC}}$ using a lazily sampled random oracle. We also set a flag bad, but this does not affect the outcome of the game. It holds that

$$\Pr[G_0 \Rightarrow 1] = \Pr\left[\overline{\text{Init}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}} \Rightarrow 1 \mid b = 0\right] .$$

Next we define game G_1, which is identical to G_0 except we change the way in which the random oracle HMAC responds to queries. Namely if HMAC is queried on the same value more than once in G_1 then it responds with an independent random string as opposed to the value previously set. These games run identically until the flag bad is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad} = 1 \text{ in } G_1] .$$

Next we define game G_2, which is identical to G_1 except during the challenge computation we overwrite each string returned in response to a query to HMAC with an independent random bit string drawn from $\{0,1\}^\ell$. Since in G_1 each string returned by the random oracle HMAC is chosen independently at random and used nowhere else in the game, these games are identically distributed:

$$\Pr[G_1 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1]| \text{ and } \Pr[\mathsf{bad} = 1 \text{ in } G_1] = \Pr[\mathsf{bad} = 1 \text{ in } G_2] .$$

We now bound the probability that bad is set in G_2. Such an event may occur due to one of $\mathcal{A}$'s HMAC queries, or due to a collision in the state variables during the challenge computation. We claim that

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,] \leq q_{\mathsf{H}} \cdot ((1 + 2^{-2\ell}) \cdot 2^{-\gamma^*} + 2^{-(\ell-1)}) + 2^{-2\ell}\,.$$

To see this, let Guess denote the event that $\mathcal{A}$ makes a query to HMAC on a point previously queried by the challenger during the challenge computation. Let Coll denote the probability that bad is set during the challenge computation as the result of an accidental collision. A union bound implies that

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,] = \Pr[\,\mathsf{Guess} \vee \mathsf{Coll} \text{ in } G_2\,] \leq \Pr[\,\mathsf{Guess} \text{ in } G_2\,] + \Pr[\,\mathsf{Coll} \text{ in } G_2\,]\,.$$

Now for Guess to occur, $\mathcal{A}$ must query HMAC on a point in the set:

$$\{(K, V \,\|\, \text{0x00} \,\|\, I_1 \,\|\, N), (K', V), (K', V' \,\|\, \text{0x01} \,\|\, I_1 \,\|\, N), (K_0^*, V')\}\,.$$

Now K, V, and K_0^* are all known to $\mathcal{A}$ (the former two being constant, and the latter being given to $\mathcal{A}$ as part of his challenge). However, all other variables are hidden from $\mathcal{A}$. Since $K', V' \leftarrow_\$ \{0,1\}^\ell$, and the legitimacy of the sampler guarantees that I_1 has at least γ_1 bits of entropy conditioned on $\mathcal{A}$'s view of the experiment, a union bound over the elements in this set and $\mathcal{A}$'s q_{H} HMAC queries implies that:

$$\Pr[\,\mathsf{Guess}\,] \leq q_{\mathsf{H}} \cdot ((1 + 2^{-2\ell}) \cdot 2^{-\gamma^*} + 2^{-(\ell-1)})\,.$$

Moreover, due to the domain separation of the queries, it is straightforward to verify that the only way that Coll will occur is if $K_0^* = K'$ and $V' = V$ (in which case, the final query made by the challenger will not be fresh). Since $V', K_0^* \leftarrow_\$ \{0,1\}^\ell$, it follows that this event occurs with probability at most $2^{-2\ell}$. Combining these observations then implies the claim.

Next we define G_3, which is identical to G_2 except we compute the challenge state directly as $S_0^* \leftarrow_\$ \mathsf{M}^{\mathsf{HMAC}}(\varepsilon)$ and remove the now redundant queries to the random oracle HMAC. It is straightforward to verify that S_0^* is identically distributed in both games, and that this is a syntactic change. We then define game G_4, which is identical to G_3 except we return the random oracle HMAC to answer consistently if queried more than once on the same point. However, since no HMAC queries are made by the challenger in these games and we have assumed that $\mathcal{A}$ never repeats a query, HMAC will never be queried more than once on the same point, and so these games are identically distributed. It follows that $\Pr[\,G_2 \Rightarrow 1\,] = \Pr[\,G_3 \Rightarrow 1\,] = \Pr[\,G_4 \Rightarrow 1\,]$. Moreover, notice that G_4 is equivalent to game $\overline{\text{Init}}$ against $\mathsf{HMAC\text{-}DRBG}$ with challenge bit $b = 1$, and so

$$\Pr[\,G_4 \Rightarrow 1\,] = \Pr\left[\,\overline{\text{Init}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}} \Rightarrow 1 \mid b = 1\,\right]\,.$$

Putting this altogether, yields

$$\begin{aligned}\mathbf{Adv}^{\overline{\text{init}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) =& |\Pr[G_0 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \\ \leq& q_{\mathsf{H}} \cdot ((1+2^{-2\ell}) \cdot 2^{-\gamma^*} + 2^{-(\ell-1)}) + 2^{-2\ell} .\end{aligned}$$

□

$\overline{\text{Pres}}$ and $\overline{\text{Rec}}$ security. The proofs of $\overline{\text{Pres}}$ and $\overline{\text{Rec}}$ security proceed by bounding: **(1)** the probability that two of the points queried to HMAC during the challenge computation collide; and **(2)** the probability that $\mathcal{A}$ queries HMAC on one of these points. We then argue that if neither of these events occur, then the challenge output / state are identically distributed to their idealised counterparts. However, this process is surprisingly delicate. Firstly, the domains of queries are not fully separated, so multiple collisions must be dealt with. Secondly, the guessing / collision probabilities of points from the same domain differ throughout the game. For example, queries of the form (K, V) are made during output generation and in state updates. In the former case, the attacker knows the 'secret' counter since this doubles as an output block, whereas in the latter this is unknown. This rules out a modular treatment, and complicates the bound. A small modification to separate queries would simplify analysis.

Lemma 4.11. *Let* PRNG = HMAC-DRBG *and masking function* $\mathsf{M}^{\mathsf{HMAC}}$ *be as specified in Theorem 4.4, where* $n = \lceil \beta/\ell \rceil$. *Then for any* (q_{H}, q_C)*-adversary* $\mathcal{A}$ *in game* $\overline{\text{Pres}}$ *against* HMAC-DRBG *who always outputs addin* $\neq \varepsilon$, *it holds that*

$$\mathbf{Adv}^{\overline{\text{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) \leq (q_{\mathsf{H}} \cdot (8q_C + 6) + \epsilon) \cdot 2^{-2\ell} + (q_{\mathsf{H}} \cdot (n+2) + n(n+1)) \cdot 2^{-\ell} ,$$

where $\epsilon = q_C \cdot (6q_C + 2n + 8) + 3n + 8$.

Proof. We argue by a series of game hops, shown in Figure 4.13. We assume without loss of generality that $\mathcal{A}$ never repeats a query to the random oracle HMAC. We first define game G_0, which is a rewriting of game $\overline{\text{Pres}}$ with $b = 0$ for HMAC-DRBG and $\mathsf{M}^{\mathsf{HMAC}}$ using a lazily sampled random oracle. Recall that we assume $\mathcal{A}$ outputs *addin* $\neq \varepsilon$. We also set a flag bad, but this does not affect the outcome of the game. It holds that

$$\Pr[G_0 \Rightarrow 1] = \Pr\left[\overline{\text{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 0\right] .$$

Next we define game G_1, which is identical to G_0 except we change the way in which the random oracle HMAC responds to queries. Namely in G_1 if HMAC is queried on the same

```
proc. main  // $G_0, G_1$
$(S'_0, I_1, \ldots, I_d, addin) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{HMAC}}$
If $S'_0 = \varepsilon$ then $cnt_0 \leftarrow 0$
Else $(K', V', cnt') \leftarrow S'_0$
$K_0, V_0 \leftarrow\!\!\$\ \{0,1\}^{\ell}$ ; $cnt_0 \leftarrow cnt' + 1$
$S_0 \leftarrow (K_0, V_0, cnt_0)$
For $j = 1, \ldots, d$
    $K'_{j-1} \leftarrow \mathsf{HMAC}(K_{j-1}, V_{j-1} \,\|\, \mathtt{0x00} \,\|\, I_j)$
    $V'_{j-1} \leftarrow \mathsf{HMAC}(K'_{j-1}, V_{j-1})$
    $K_j \leftarrow \mathsf{HMAC}(K'_{j-1}, V'_{j-1} \,\|\, \mathtt{0x01} \,\|\, I_j)$
    $V_j \leftarrow \mathsf{HMAC}(K_j, V'_{j-1})$
    $cnt_j \leftarrow 1$ ; $S_j \leftarrow (K_j, V_j, cnt_j)$
$K'_d \leftarrow \mathsf{HMAC}(K_d, V_d \,\|\, \mathtt{0x00} \,\|\, addin)$
$V'_d \leftarrow \mathsf{HMAC}(K'_d, V_d)$ // *
$K^0_d \leftarrow \mathsf{HMAC}(K'_d, V'_d \,\|\, \mathtt{0x01} \,\|\, addin)$
$V^0_d \leftarrow \mathsf{HMAC}(K^0_d, V'_d)$
$temp_R \leftarrow \varepsilon$
For $j = 1, \ldots, n$
    $V^j_d \leftarrow \mathsf{HMAC}(K^0_d, V^{j-1}_d)$// * *
    $temp_R \leftarrow temp_R \,\|\, V^j_d$
$R^* \leftarrow \mathrm{left}(temp_R, \beta)$
$K''_d \leftarrow \mathsf{HMAC}(K^0_d, V^n_d \,\|\, \mathtt{0x00} \,\|\, addin)$
$V''_d \leftarrow \mathsf{HMAC}(K''_d, V^n_d)$
$K^* \leftarrow \mathsf{HMAC}(K''_d, V''_d \,\|\, \mathtt{0x01} \,\|\, addin)$
$V^* \leftarrow \mathsf{HMAC}(K^*, V''_d)$
$cnt^*_d \leftarrow cnt_d + 1$
$S^* \leftarrow (K^*, V^*, cnt^*)$
$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{HMAC}}(R^*, S^*)$
Return $b^*$
```

```
proc. main  // $\boxed{G_3}, G_4, G_5$
$(S'_0, I_1, \ldots, I_d, addin) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{HMAC}}$ ; $(K', V', cnt') \leftarrow S'_0$
If $S_0 = \varepsilon$ then $cnt_0 \leftarrow 0$
Else $(K', V', cnt') \leftarrow S'_0$
$K_0, V_0 \leftarrow\!\!\$\ \{0,1\}^{\ell}$ ; $cnt_0 \leftarrow cnt' + 1$
$S_0 \leftarrow (K_0, V_0, cnt_0)$
For $j = 1, \ldots, d$
    $K'_{j-1} \leftarrow \mathsf{HMAC}(K_{j-1}, V_{j-1} \,\|\, \mathtt{0x00} \,\|\, I_j)$ ; $\boxed{K'_{j-1} \leftarrow\!\!\$\ \{0,1\}^{\ell}}$
    $V'_{j-1} \leftarrow \mathsf{HMAC}(K'_{j-1}, V_{j-1})$ ; $\boxed{V'_{j-1} \leftarrow\!\!\$\ \{0,1\}^{\ell}}$
    $K_j \leftarrow \mathsf{HMAC}(K'_{j-1}, V'_{j-1} \,\|\, \mathtt{0x01} \,\|\, I_j)$ ; $\boxed{K_j \leftarrow\!\!\$\ \{0,1\}^{\ell}}$
    $V_j \leftarrow \mathsf{HMAC}(K_j, V'_{j-1})$ ; $\boxed{V_j \leftarrow\!\!\$\ \{0,1\}^{\ell}}$
    $cnt_j \leftarrow 1$ ; $S_j \leftarrow (K_j, V_j, cnt_j)$
$R^* \leftarrow\!\!\$\ \{0,1\}^{\beta}$
$S^* \leftarrow\!\!\$\ \mathsf{M}^{\mathsf{HMAC}}(S_d)$
$b^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{HMAC}}(R^*, S^*)$
Return $b^*$

proc. HMAC($X$) // $\boxed{G_0}, G_1, G_2, G_3, G_4, \boxed{G_5}$
$Y \leftarrow\!\!\$\ \{0,1\}^{\ell}$
If $\mathsf{HMAC}[X] \neq \perp$
    $\mathsf{bad} \leftarrow 1$
    $\boxed{Y \leftarrow \mathsf{HMAC}[X]}$
$\mathsf{HMAC}[X] \leftarrow Y$
Return $Y$
```

Figure 4.13: Games for proof of Lemma 4.11 ($\overline{\text{Pres}}$ security of HMAC-DRBG).

value more than once, then it responds with an independent random string as opposed to the value previously set. These games run identically until the flag bad is set, and so the Fundamental Lemma of Game Playing implies that

$$|\Pr[G_0 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \Pr[\mathsf{bad} = 1 \text{ in } G_1] .$$

Next, we define game G_2 (not shown), which is identical to G_1 except during the challenge computation we overwrite each string returned in response to a query to HMAC with an independent random bit string drawn from $\{0,1\}^\ell$. Since in G_1 each string returned by the random oracle HMAC is chosen independently at random and used nowhere else in the game, these games are identically distributed, and so

$$\Pr[G_1 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1] \text{ and } \Pr[\mathsf{bad} = 1 \text{ in } G_1] = \Pr[\mathsf{bad} = 1 \text{ in } G_2] .$$

We now bound the probability that bad is set in G_2. Notice that bad may be set as the result of an HMAC query by $\mathcal{A}$ colliding with a query made by the challenger, or during the computation of the challenge output / state if two of the points queried to HMAC by the challenger collide.

Let Guess denote the event that one of $\mathcal{A}$'s queries coincides with one of the points queried to HMAC by the challenger. Let Coll denote the probability that bad is set during the challenge computation as the result of an accidental collision. A union bound implies that

$$\Pr[\mathsf{bad} = 1 \text{ in } G_2] = \Pr[\mathsf{Guess} \vee \mathsf{Coll} \text{ in } G_2] \leq \Pr[\mathsf{Guess} \text{ in } G_2] + \Pr[\mathsf{Coll} \text{ in } G_2] .$$

One may verify from the pseudocode in Figure 4.13 that $4(d+2)+n$ HMAC computations are made during the challenge computation. (That is: four HMAC queries for each of the d refresh calls (lines 6 - 11); four queries for the two applications of update prior and post the production of output (lines 12 - 15 and 17 - 19 respectively); and $n = \lceil \beta/\ell \rceil$ queries to produce the output blocks (lines 21 - 23)). Notice that each query made by $\mathcal{A}$ which would cause Guess to occur requires $\mathcal{A}$ to guess a key and counter pair. All such keys and counters are chosen uniformly from $\{0,1\}^\ell$ in G_2, and moreover these keys and counters are hidden from $\mathcal{A}$ *except* for: **(1)** the key K^* which forms part of the final HMAC query made by the challenger (line 24); and **(2)** the counters V_d^j for $j \in [1,n]$ (line 18) which are concatenated and truncated to form the output $R^* \leftarrow \text{left}(temp_R, \beta)$. All of these are revealed (partially in the case of V_d^n, if β is not a multiple of ℓ) to $\mathcal{A}$ as part of the challenge output / state pair. One may verify from the pseudocode that there are $n+2$ such partially known queries (i.e., the $n-1$ queries arising from the output generation loop; the queries in lines 21 and 22 for which the counter V_d^n is (at least partially) known, and the query on line 24 with known key K^*). The probability that a single HMAC query made by $\mathcal{A}$ correctly guesses the value of a given unknown key / counter pair is equal to $2^{-2\ell}$, while this probability for a partially unknown pair is upper bounded by $2^{-\ell}$. Therefore taking a union bound over the $4(d+2)+n$ pairs queried, and $\mathcal{A}$'s q_{H} guesses, implies that:

$$\Pr[\,\mathsf{Guess}\text{ in }G_2\,] \leq q_{\mathsf{H}} \cdot ((4d+6)\cdot 2^{-2\ell} + (n+2)\cdot 2^{-\ell})\ .$$

We now bound the probability that Coll occurs. We divide the queries made by the challenger into three classes as follows. For $K, V \in \{0,1\}^\ell$ and $I \in \{0,1\}^{\geq 1}$ we have: **(1)** queries of the form $(K, V \,\|\, 0\text{x}00 \,\|\, I)$; **(2)** queries of the form (K, V); and **(3)** queries of the form $(K, V \,\|\, 0\text{x}01 \,\|\, I)$. Notice that all queries made by the challenger during the challenge computation fall into one of these types. Moreover, notice that queries of different types can never collide due to their disjoint domains. Letting Coll_i for $i \in [1,3]$ denote that there is a collision amongst the type **(i)** queries, a union bound implies that

$$\Pr[\,\mathsf{Coll}\text{ in }G_2\,] \leq \Pr[\,\mathsf{Coll}_1\text{ in }G_2\,] + \Pr[\,\mathsf{Coll}_2\text{ in }G_2\,] + \Pr[\,\mathsf{Coll}_3\text{ in }G_2\,]\ .$$

We first claim that $\Pr[\,\mathsf{Coll}_1\text{ in }G_2\,] = \Pr[\,\mathsf{Coll}_3\text{ in }G_2\,] = (d(d+3)+2)\cdot 2^{-(2\ell+1)}$. To see this, notice that the challenger makes a total of $(d+2)$ type **(1)** (resp. **(3)**) queries (d queries during the iterative reseeds, and a single query in the state update before and after output generation). Each of these queries consists of a *freshly sampled* key and counter chosen from $\{0,1\}^\ell$, by which we mean that the key (resp. the counter) will not be queried as part of any other query of that type barring an accidental collision. (Looking ahead, an example of queries which are *not* freshly sampled are those made to generate output blocks in line 18, for which the same key is used for each query.) It follows that the probability that any pair of queries collide is upper bounded by $2^{-2\ell}$. Summing over the at most $\frac{1}{2}(d+1)(d+2)$ distinct pairs of queries and rearranging then proves the claim.

Next we claim that $\Pr[\,\mathsf{Coll}_2\text{ in }G_2\,] \leq (d\cdot(2d+2n+7)+3n+6)\cdot 2^{-2\ell} + n(n+1)\cdot 2^{-\ell}$. To see this, first consider the set of type **(2)** queries made by the challenger up to the point in the pseudocode indicated by $*$ inclusive. There are $2d+1$ queries in this set, and each of these queries consists of a freshly sampled key and counter chosen randomly from $\{0,1\}^\ell$. Any given pair of such queries will collide with probability $2^{-2\ell}$; therefore summing over these pairs implies that the probability of a collision up to point $*$ is upper bounded by $d\cdot(2d+1)\cdot 2^{-2\ell}$. Now consider the set of type **(2)** queries made following $*$ and up to the point indicated by $**$ inclusive. While each counter for these queries is sampled uniformly from $\{0,1\}^\ell$, the key $K_d^0 \leftarrow_{\$} \{0,1\}^\ell$ remains fixed across all queries (highlighted in red in the pseudocode). As such, each of the $n+1$ queries in this set collides with a previous query in the set with probability $2^{-\ell}$; summing over the $\frac{n(n+1)}{2}$ pairs of queries of this form (and rounding up by a factor of two for simplicity) contributes $n(n+1) \cdot 2^{-\ell}$ to the bound. Moreover, each such query collides with one of the $(2d+1)$ type **(2)** queries made up to point $*$ with probability $2^{-2\ell}$, contributing $(2d+1) \cdot (n+1) \cdot 2^{-2\ell}$ to the bound. Following this there are two more type **(2)** queries (lines 22 and 24), each consisting of a freshly sampled key and counter. Since there are $2d+n+2$ and $2d+n+3$ previous type **(2)** queries with which these points may collide, this adds a further $(4d+2n+5) \cdot 2^{-2\ell}$ to the bound. Summing over these terms and rearranging then proves the claim. Putting this altogether and simplifying the expression, we obtain:

$$\begin{aligned}\Pr[\,\mathsf{bad} = 1 \text{ in } G_2\,] \leq (q_{\mathsf{H}} \cdot (4d+6) + d \cdot (3d+2n+10) + 3n+8) \cdot 2^{-2\ell} \\ + (q_{\mathsf{H}} \cdot (n+2) + n(n+1)) \cdot 2^{-\ell}\ .\end{aligned}$$

Next we first define game G_3, which is the same as G_2 except we simply compute the challenge / output state as $R^* \leftarrow\$ \{0,1\}^{\ell}$ and $S^* \leftarrow\$ \mathsf{M}^{\mathsf{HMAC}}(S_d)$ directly, and omit the now redundant HMAC queries which were previously made during their computation. It is straightforward to verify that S^* is computed identically in both games and that this is a syntactic change, and so $\Pr[\,G_2 \Rightarrow 1\,] = \Pr[\,G_3 \Rightarrow 1\,]$. Next we define game G_4, which is identical to G_3 except we no longer overwrite the responses to HMAC queries made by the challenger during the iterative reseeds with random bit strings. Since the random oracle HMAC responds to each query with an independent random string, regardless of whether that point has previously been queried, these games are identically distributed and so $\Pr[\,G_3 \Rightarrow 1\,] = \Pr[\,G_4 \Rightarrow 1\,]$.

Finally we define game G_5, which is identical to G_4 except we revert the random oracle HMAC to answering consistently on all queries. These games run identically until the flag bad is set. In addition to being set as a result of the attacker's queries, in games $G_3 - G_5$ bad may also be set by the challenger during the iterative reseeds made to compute state S_d in the event of an accidental collision. As before, we have that $\Pr[\,\mathsf{bad} = 1 \text{ in } G_5\,] = \Pr[\,\mathsf{bad} = 1 \text{ in } G_3\,] \leq \Pr[\,\mathsf{Guess} \text{ in } G_3\,] + \sum_{i=1}^{3} \Pr[\,\mathsf{Coll}_i \text{ in } G_3\,]$.

One may verify from the pseudocode that a total of $4d$ points are queried to HMAC during the iterative reseeds, each consisting of a uniformly random key and counter which are independent of $\mathcal{A}$'s view of the experiment. An analogous argument to that used previously, taking a union bound over $\mathcal{A}$'s q_{H} queries, then implies that $\Pr[\,\mathsf{Guess} \text{ in } G_3\,] \leq q_{\mathsf{H}} \cdot 4d \cdot 2^{-2\ell}$. Moreover, there are d type **(1)** and d type **(3)** queries made during this process, each of which collides with probability at most $2^{-2\ell}$; it follows that $\Pr[\,\mathsf{Coll}_1\,] = \Pr[\,\mathsf{Coll}_3\,] \leq d \cdot (d-1) \cdot 2^{-(2\ell+1)}$. Finally, there are $2d$ type **(2)** queries made during this process, each of which collides with probability $2^{-2\ell}$. It follows that $\Pr[\,\mathsf{Coll}_2\,] \leq d \cdot (2d-1) \cdot 2^{-2\ell}$. Putting this altogether than yields

$$|\Pr[\,G_4 \Rightarrow 1\,] - \Pr[\,G_5 \Rightarrow 1\,]| \leq \Pr[\,\mathsf{bad} = 1 \text{ in } G_3\,] \leq (q_{\mathsf{H}} \cdot 4d + d \cdot (3d-2)) \cdot 2^{-2\ell}\ .$$

It is straightforward to verify that G_5 is equivalent to game $\overline{\mathrm{Pres}}$ with $b = 1$ for HMAC-DRBG. It follows that

$$\Pr[\,G_5 \Rightarrow 1\,] = \Pr\left[\,\overline{\mathrm{Pres}}^{\mathcal{A}}_{\mathsf{PRNG},\mathsf{M},\beta} \Rightarrow 1 \mid b = 1\,\right]\ .$$

Putting this altogether, rearranging, and upper bounding d with q_C implies that

$$\mathbf{Adv}^{\overline{\mathrm{pres}}}_{\mathsf{PRNG},\mathsf{M},\beta}(\mathcal{A}) \leq (q_{\mathsf{H}} \cdot (8q_C + 6) + \epsilon) \cdot 2^{-2\ell} + (q_{\mathsf{H}} \cdot (n+2) + n(n+1)) \cdot 2^{-\ell}\ ,$$

where $\epsilon = q_C \cdot (6q_C + 2n + 8) + 3n + 8$, concluding the proof.

□

This concludes our analysis of the $\overline{\mathrm{Pres}}$ security of HMAC-DRBG. All that remains is to bound its $\overline{\mathrm{Rec}}$ security, which we do in the following lemma.

Lemma 4.12. *Let* PRNG = HMAC-DRBG *and masking function* $\mathsf{M}^{\mathsf{HMAC}}$ *be as specified in Theorem 4.4, where* $n = \lceil \beta/\ell \rceil$. *Then for any* (q_{H}, q_C)*-adversary* $\mathcal{A}$ *in game* $\overline{\mathrm{Rec}}$ *against* HMAC-DRBG *who always outputs addin* $\neq \varepsilon$, *and any* $(q^+_{\mathcal{D}}, \gamma^*)$*-legitimate sampler* $\mathcal{D} \in \{\mathcal{D}\}_{\gamma^*}$, *it holds that*

$$\mathbf{Adv}^{\overline{\mathrm{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A}, \mathcal{D}) \leq (q_{\mathsf{H}} \cdot (4q_C + 4 + (4q_C - 2) \cdot 2^{-\gamma^*}) + \epsilon') \cdot 2^{-2\ell} + (q_{\mathsf{H}} \cdot (n+4) + n(n+1)) \cdot 2^{-\ell} + q_{\mathsf{H}} \cdot 2^{-(\gamma^*-1)}\ ,$$

where $\epsilon' = (q_C \cdot (4q_C + 2n + 10 + (q_C - 1) \cdot 2^{-(\gamma^*-1)}) + 3n + 8)$.

Proof. We argue by a series of game hops, shown in Figure 4.14. The proof is similar to that of Lemma 4.11; we emphasise the differences here. We assume without loss of generality that $\mathcal{A}$ never repeats a query to the random oracle HMAC and require that $\mathcal{A}$ outputs *addin* $\neq \varepsilon$. We begin by defining game G_0, which is a rewriting of game $\overline{\mathrm{Rec}}$ with

```
proc. main  // $G_0, G_1$
1. $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 1$
2. For $k = 1, \ldots, q_{\mathcal{D}} + 1$
3.     $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\$ \mathcal{D}(\sigma_{k-1})$
4. $(S_0, d, addin) \leftarrow\$ \mathcal{A}^{\mathsf{HMAC},\mathrm{Sam}}((\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
5. If $\mu + d > q_{\mathcal{D}} + 1$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
6.     Return $\perp$
7. $(K_0, V_0, cnt_0) \leftarrow S_0$
8. For $j = 1, \ldots, d$
9.     $K'_{j-1} \leftarrow \mathsf{HMAC}(K_{j-1}, V_{j-1} \,\|\, \mathrm{0x00} \,\|\, I_j)$
10.    $V'_{j-1} \leftarrow \mathsf{HMAC}(K'_{j-1}, V_{j-1})$
11.    $K_j \leftarrow \mathsf{HMAC}(K'_{j-1}, V'_{j-1} \,\|\, \mathrm{0x01} \,\|\, I_j)$
12.    $V_j \leftarrow \mathsf{HMAC}(K_j, V'_{j-1})$
13.    $cnt_j \leftarrow 1$ ; $S_j \leftarrow (K_j, V_j, cnt_j)$
14. $K'_d \leftarrow \mathsf{HMAC}(K_d, V_d \,\|\, \mathrm{0x00} \,\|\, addin)$
15. $V'_d \leftarrow \mathsf{HMAC}(K'_d, V_d)$ // *
16. $K^0_d \leftarrow \mathsf{HMAC}(K'_d, V'_d \,\|\, \mathrm{0x01} \,\|\, addin)$
17. $V^0_d \leftarrow \mathsf{HMAC}(K^0_d, V'_d)$
18. $temp_R \leftarrow \varepsilon$
19. For $j = 1, \ldots, n$
20.     $V^j_d \leftarrow \mathsf{HMAC}(K^0_d, V^{j-1}_d)$// * *
21.     $temp_R \leftarrow temp_R \,\|\, V^j_d$
22. $R^* \leftarrow \mathrm{left}(temp_R, \beta)$
23. $K''_d \leftarrow \mathsf{HMAC}(K^0_d, V^n_d \,\|\, \mathrm{0x00} \,\|\, addin)$
24. $V''_d \leftarrow \mathsf{HMAC}(K''_d, V^n_d)$
25. $K^* \leftarrow \mathsf{HMAC}(K''_d, V''_d \,\|\, \mathrm{0x01} \,\|\, addin)$
26. $V^* \leftarrow \mathsf{HMAC}(K^*, V''_d)$
27. $cnt^*_d \leftarrow cnt^*_d + 1$
28. $S^* \leftarrow (K^*, V^*, cnt^*_d)$
29. $b^* \leftarrow\$ \mathcal{A}^{\mathsf{HMAC}}(R^*, S^*, (I_k)_{k>\mu+d})$
30. Return $b^*$

proc. Sam // $G_0, G_1, G_2, G_3, G_4, G_5$
$\mu \leftarrow \mu + 1$
Return $I_\mu$
```

```
proc. main  // $\boxed{G_3}, G_4, G_5$
$\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 1$
For $k = 1, \ldots, q_{\mathcal{D}} + 1$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\$ \mathcal{D}(\sigma_{k-1})$
$(S_0, d, addin) \leftarrow\$ \mathcal{A}^{\mathsf{HMAC},\mathrm{Sam}}((\gamma_i, z_i)_{i=1}^{q_{\mathcal{D}}+1})$
If $\mu + d > q_{\mathcal{D}} + 1$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
$(K_0, V_0, cnt_0) \leftarrow S_0$
For $j = 1, \ldots, d$
    $K'_{j-1} \leftarrow \mathsf{HMAC}(K_{j-1}, V_{j-1} \,\|\, \mathrm{0x00} \,\|\, I_j)$ ; $\boxed{K'_{j-1} \leftarrow\$ \{0,1\}^\ell}$
    $V'_{j-1} \leftarrow \mathsf{HMAC}(K'_{j-1}, V_{j-1})$ ;            $\boxed{V'_{j-1} \leftarrow\$ \{0,1\}^\ell}$
    $K_j \leftarrow \mathsf{HMAC}(K'_{j-1}, V'_{j-1} \,\|\, \mathrm{0x01} \,\|\, I_j)$ ; $\boxed{K_j \leftarrow\$ \{0,1\}^\ell}$
    $V_j \leftarrow \mathsf{HMAC}(K_j, V'_{j-1})$ ;              $\boxed{V_j \leftarrow\$ \{0,1\}^\ell}$
    $cnt_j \leftarrow 1$ ; $S_j \leftarrow (K_j, V_j, cnt_j)$
$R^* \leftarrow\$ \{0,1\}^\beta$
$S^* \leftarrow\$ \mathsf{M}^{\mathsf{HMAC}}(S_d)$
$b^* \leftarrow\$ \mathcal{A}^{\mathsf{HMAC}}(R^*, S^*, (I_k)_{k>\mu+d})$
Return $b^*$

proc. HMAC$(X)$ // $\boxed{G_0}, G_1, G_2, G_3, G_4, \boxed{G_5}$
$Y \leftarrow\$ \{0,1\}^\ell$
If $\mathsf{HMAC}[X] \neq \perp$
    $\mathsf{bad} \leftarrow 1$
    $\boxed{Y \leftarrow \mathsf{HMAC}[X]}$
$\mathsf{HMAC}[X] \leftarrow Y$
Return $Y$
```

Figure 4.14: Games for proof of Lemma 4.12 ($\overline{\mathrm{Rec}}$ security of HMAC-DRBG).

$b = 0$ for HMAC-DRBG and $\mathsf{M}^{\mathsf{HMAC}}$ using a lazily sampled random oracle. We also set a flag bad, but this does not affect the outcome of the game. We have that

$$\Pr\left[\, G_0 \Rightarrow 1 \,\right] = \Pr\left[\, \overline{\mathrm{Rec}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta} \Rightarrow 1 \;|\; b = 0 \,\right] .$$

Next we define games G_1 and G_2. In the former, we modify the random oracle HMAC to respond with an independent random string to all queries, regardless of whether the value has previously been set. In G_2 (not pictured), we overwrite each string returned in response to a random oracle query made by the challenger with an independent random bit string. An analogous argument to that made in the proof of Lemma 4.11 implies that

$$\left|\Pr\left[\, G_0 \Rightarrow 1 \,\right] - \Pr\left[\, G_2 \Rightarrow 1 \,\right]\right| \leq \Pr\left[\, \mathsf{bad} = 1 \text{ in } G_2 \,\right] .$$

We now bound the probability of this event occurring. Defining the events Guess and Coll_i for $i \in [1, 3]$ as before, a union bound implies that $\Pr\left[\, \mathsf{bad} = 1 \text{ in } G_2 \,\right] \leq \Pr\left[\, \mathsf{Guess} \text{ in } G_2 \,\right] + \sum_{i=1}^{3} \Pr\left[\, \mathsf{Coll}_i \text{ in } G_2 \,\right]$.

To bound the probability that Guess occurs, consider the set of queries made by the challenger, where we define queries of types **(1)** - **(3)** as in the proof of Lemma 4.11. We first note that each of the $2(d+2)+n$ type **(2)** queries made by the challenger consists of a random ℓ-bit key and ℓ-bit counter which are hidden from the attacker, with the exception of: **(a)** the first and final type **(2)** queries (lines 10 and 26 respectively); and **(b)** queries 2 to n of the $j = 1, \ldots, n$ loop (lines 19 - 21), and the first type **(2)** query during the subsequent state update (line 24). (For the first query, the counter V_0 is chosen by the attacker. For the final query, the key K^* is given to the attacker as part of the challenge state. For all other queries, the counter is revealed to the attacker as part of the output R^*.) The probability that a single HMAC query made by $\mathcal{A}$ correctly guesses the value of a given unknown key / counter pair is equal to $2^{-2\ell}$, while this probability for a partially unknown pair is upper bounded by $2^{-\ell}$. Putting this together and taking union bounds, the probability that $\mathcal{A}$ sets Guess by querying a type **(2)** point to HMAC is upper bounded by $q_{\mathsf{H}} \cdot ((2d+2) \cdot 2^{-2\ell} + (n+2) \cdot 2^{-\ell})$.

Moreover, all of the d type **(3)** queries and all but the first of the d type **(1)** queries (line 9) made during the iterative reseeds in lines 8 - 13 requires $\mathcal{A}$ to correctly guess the value of a randomly chosen and hidden key and counter in addition to an entropy sample, which by the legitimacy of the sampler contains at least $2^{-\gamma^*}$ bits of entropy. As such, the probability that $\mathcal{A}$ correctly guesses a given one of these points in a single query to HMAC is upper bounded by $2^{-(\gamma^*+2\ell)}$. For the first type **(1)** query, the key and counter were chosen by $\mathcal{A}$; however, they must still guess the unknown entropy sample I_1 containing at least γ^* bits of entropy, and so the probability that $\mathcal{A}$ guesses this point with a single HMAC query is upper bounded by $2^{-\gamma^*}$. Finally, the remaining four type **(1)** and **(3)** queries (made as part of the state updates with *addin* before and after output generation (lines 14 - 17 and 23 - 26 respectively) each require $\mathcal{A}$ to guess a random and hidden key / counter pair, and so each pair is guessed in a single HMAC query with probability at most $2^{-2\ell}$ — *except* the first type **(1)** query after output generation (line 23), for which the counter is (fully or partially) revealed in R^*, and so is guessed with probability at most $2^{-\ell}$. Taking union bounds, the probability that $\mathcal{A}$ sets Guess by querying a type **(1)** or **(3)** point to HMAC is upper bounded by $q_{\mathsf{H}} \cdot ((3 + (2d-1) \cdot 2^{-\gamma^*}) \cdot 2^{-2\ell} + 2^{-\gamma^*} + 2^{-\ell})$. Putting this altogether yields,

$$\Pr[\,\mathsf{Guess} \text{ in } G_2\,] \leq q_{\mathsf{H}} \cdot ((2d + 5 + (2d-1) \cdot 2^{-\gamma^*}) \cdot 2^{-2\ell} + 2^{-\gamma^*} + (n+3) \cdot 2^{-\ell}) \,.$$

Next we claim that $\Pr[\,\mathsf{Coll}_1 \text{ in } G_2\,]$ and $\Pr[\,\mathsf{Coll}_3 \text{ in } G_2\,]$ are bounded above by $(4d+2+d\cdot$

$(d-1)\cdot 2^{-\gamma^*})\cdot 2^{-(2\ell+1)}$. To see this, notice that a collision between a pair of the d type **(1)** (resp. type **(3)**) queries made during the iterative reseeds requires both the entropy input, and the key and counter (which for all but the first type **(1)** query are freshly sampled random bit strings), to collide. A union bound then implies that this probability is upper bounded by $d(d-1)\cdot 2^{-(\gamma^*+2\ell+1)}$. For any of the two type **(1)** (resp. type **(3)**) queries made during state updates before and after output generation, any collision with another query of the same type requires the randomly sampled key and counter to collide, an event which occurs with probability $d\cdot 2^{-2\ell}$ for the first such query, and $(d+1)\cdot 2^{-2\ell}$ for the second, contributing the additional $(2d+1)\cdot 2^{-2\ell}$ term to the bound. For the type **(2)** queries, it is straightforward to verify that the same argument as in the proof of Lemma 4.11 applies in this case also, and so $\Pr[\,\mathsf{Coll}_2\,] \le (d\cdot(2d+2n+7)+3n+6)\cdot 2^{-2\ell} + n(n+1)\cdot 2^{-\ell}$. Putting this altogether and rearranging, yields

$$\begin{aligned}\Pr[\,\mathsf{bad}=1 \text{ in } G_2\,] \le q_{\mathsf{H}}\cdot((2d+5+(2d-1)\cdot 2^{-\gamma^*})\cdot 2^{-2\ell} \\ + (d\cdot(2d+2n+11+(d-1)\cdot 2^{-\gamma^*})+3n+8)\cdot 2^{-2\ell} + q_{\mathsf{H}}\cdot 2^{-\gamma^*} \\ + (q_{\mathsf{H}}\cdot(n+3)+n(n+1))\cdot 2^{-\ell}\,.\end{aligned}$$

Next, we define games G_3, G_4, and G_5. Game G_3 is the same as G_2, except we compute the challenge / output state pair as $R^* \leftarrow_\$ \{0,1\}^\ell$ and $S^* \leftarrow_\$ \mathsf{M}(S_d)$ and omit the now redundant HMAC queries which were previously made during their computation. In G_4, we no longer overwrite the responses to random oracle queries made by the challenger with random bit strings. Finally, in G_5 we return the random oracle HMAC to answer consistently on all queries. An analogous argument to that used in the proof of Lemma 4.11 implies that $|\Pr[\,G_2 \Rightarrow 1\,] - \Pr[\,G_5 \Rightarrow 1\,]| \le \Pr[\,\mathsf{bad}=1 \text{ in } G_3\,]$. We now bound this probability. As before, we have that $\Pr[\,\mathsf{bad}=1 \text{ in } G_3\,] \le \Pr[\,\mathsf{Guess} \text{ in } G_3\,] + \sum_{i=1}^{3}\Pr[\,\mathsf{Coll}_i \text{ in } G_3\,]$. Firstly, notice that there are $2d$ type **(2)** queries made by the challenger in G_3. Each of these consists of a freshly sampled random key and counter, except for the first query where the counter is known to $\mathcal{A}$. An analogous argument to that used previously implies that the probability that $\mathcal{A}$ queries one of these points to HMAC is upper bounded by $q_{\mathsf{H}}\cdot((2d-1)\cdot 2^{-2\ell}+2^{-\ell})$. Each of the d type **(1)** (resp. type **(3)**) queries consists of a random key / counter and entropy sample, except for the first type **(1)** query for which the key and counter are known to $\mathcal{A}$. A union bound then implies that the probability that $\mathcal{A}$ queries one of the type **(1)** or **(3)** queries to HMAC is upper bounded by $q_{\mathsf{H}}\cdot((2d-$

$1) \cdot 2^{-\gamma^*} \cdot 2^{-2\ell} + 2^{-\gamma^*})$. Putting this altogether implies that:

$$\Pr[\,\mathsf{Guess} \text{ in } G_3\,] \leq q_{\mathsf{H}} \cdot ((2d - 1 + (2d-1) \cdot 2^{-\gamma^*}) \cdot 2^{-2\ell} + 2^{-\gamma^*} + 2^{-\ell}) \,.$$

We now bound the collision probabilities. There are d type **(1)** and d type **(3)** queries made in G_3, each of which collides with another query of its type with probability at most $2^{-(2\ell+\gamma^*)}$; it follows that $\Pr[\,\mathsf{Coll}_1 \text{ in } G_3\,] = \Pr[\,\mathsf{Coll}_3 \text{ in } G_3\,] \leq d \cdot (d-1) \cdot 2^{-\gamma^*} \cdot 2^{-(2\ell+1)}$. Finally, each of the $2d$ type **(2)** queries in G_3 collides with probability $2^{-2\ell}$; taking a union bound, it follows that $\Pr[\,\mathsf{Coll}_2 \text{ in } G_3\,] \leq d \cdot (2d-1) \cdot 2^{-2\ell}$. Putting this all together then yields

$$\begin{aligned}\Pr[\,\mathsf{bad} = 1 \text{ in } G_3\,] \leq q_{\mathsf{H}} \cdot (2d - 1 + (2d-1) \cdot 2^{-\gamma^*}) \cdot 2^{-2\ell} + (d \cdot (2d - 1 + (d-1) \cdot 2^{-\gamma^*})) \cdot 2^{-2\ell} \\ + q_{\mathsf{H}} \cdot 2^{-\gamma^*} + q_{\mathsf{H}} \cdot 2^{-\ell} \,.\end{aligned}$$

Moreover, it is straightforward to verify that G_5 is equivalent to game $\overline{\text{Rec}}$ with $b = 1$ for HMAC-DRBG. It follows that

$$\Pr[\,G_5 \Rightarrow 1\,] = \Pr\left[\,\overline{\text{Rec}}^{\mathcal{A},\mathcal{D}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta} \Rightarrow 1 \;|\; b = 1\,\right] \,.$$

Putting this altogether, rearranging, and upper bounding d with q_C implies that

$$\begin{aligned}\mathbf{Adv}^{\overline{\text{rec}}}_{\mathsf{PRNG},\mathsf{M},\gamma^*,q_{\mathcal{D}},\beta}(\mathcal{A},\mathcal{D}) \leq (q_{\mathsf{H}} \cdot (4q_C + 4 + (4q_C - 2) \cdot 2^{-\gamma^*}) + \epsilon') \cdot 2^{-2\ell} \\ + (q_{\mathsf{H}} \cdot (n+4) + n(n+1)) \cdot 2^{-\ell} + q_{\mathsf{H}} \cdot 2^{-(\gamma^*-1)} \,,\end{aligned}$$

where $\epsilon' = (q_C \cdot (4q_C + 2n + 10 + (q_C - 1) \cdot 2^{-(\gamma^*-1)}) + 3n + 8)$, concluding the proof.

$\square$

This completes our analysis of the $\overline{\text{Init}}$, $\overline{\text{Pres}}$, and $\overline{\text{Rec}}$ security of HMAC-DRBG. Combining these results via Theorem 4.1 then proves Theorem 4.4, concluding our analysis of the robustness of HMAC-DRBG in the ROM.

4.8 Overlooked Attack Vectors

A note on presentation. This section stands somewhat apart from the rest of the thesis by taking a more informal and intuitive approach to analysis. The material in this

```
next(seed, $S, \beta, addin$)
If $cnt > reseed\_interval$
   Return reseed_required
$(S^0, data^0) \leftarrow$ init(seed, $S, \beta, addin$)
$\boxed{\text{If } addin \leftarrow \varepsilon \text{ then } addin \leftarrow 0^n}$
$temp_R \leftarrow \varepsilon$ ; $n \leftarrow \lceil \beta/\ell \rceil$
For $i = 1, \ldots, n$
   $(r^i, S^i, data^i) \leftarrow$ gen(seed, $S^{i-1}, data^{i-1}$)
   $temp_R \leftarrow temp_R \,\|\, r^i$
$R \leftarrow \text{left}(temp_R, \beta)$
$S' \leftarrow$ final(seed, $S^n, \beta, addin$)
Return $(R, S')$
```

Figure 4.15: Iterative next algorithm for a DRBG with associated decomposition $\mathcal{C} =$ (init, gen, final). Boxed text included for CTR-DRBG only.

section was originally written with practitioners in mind, and aims to demonstrate key potential attacks rather than being an exhaustive treatment of the problem at hand. We found that writing e.g., a strictly formal and code-based treatment of the ideas presented introduced a host of parameters and complicated notation which served mainly to obscure the fairly intuitive notions we were trying to capture, without providing further insight. Therefore, we have chosen an altogether more informal approach, but one which is still sufficiently precise to express our security model and attacks. To refer back to the title of this thesis, while this section is not entirely in keeping with the provable security aspect, we hope that this makes our findings more usable and accessible in the real world.

4.8.1 Motivation and Attack Scenario

The positive results of Sections 4.6 and 4.7 are reassuring. However, the flexibility in the standard to produce variable length and *large* outputs (of up to 2^{19} bits) in each next call means that two implementations of the same DRBG may be very different depending on how such limits are set. While this is reflected in the security bounds of the previous sections (in terms of the parameter n denoting the number of output blocks computed per request), in this section we argue that the standard security definition of robustness may overlook attack vectors against the (fairly non-standard) NIST DRBGs.

The points made in this section do not contradict the results of the previous sections; rather we argue that in certain (realistic) scenarios — namely when the DRBG is used to produce many output blocks per next call — it is worth taking a closer look at which points during output generation a state may be compromised.

Iterative next algorithms. The next algorithm of each of the NIST DRBGs has the same high-level structure (modulo slight variations which we highlight below, and which again exemplify how small design features frustrate a modular treatment). First, any additional input provided in the call is incorporated into the state, and in the case of HASH-DRBG one of the state variables is copied into an additional variable in preparation for output generation (i.e., setting $data = V$, see Figure 4.1). Output blocks are produced by iteratively applying a function to the state variables (or in the case of HASH-DRBG, the copy of the state variable). These blocks are concatenated and truncated to β-bits to form the returned output R, and a final state update is performed to produce S'.

We would like to track the evolution of the state variables during a next call relative to the production of different output blocks, in order to reason precisely about the effect of state component compromises at different points. As such, it shall be useful to formalise this structure. To this end, we say that a DRBG has an *iterative* next *algorithm* if next may be decomposed into a tuple of subroutines $\mathcal{C} = (\mathsf{init}, \mathsf{gen}, \mathsf{final})$. Here $\mathsf{init} : \text{Seed} \times \mathcal{S} \times \mathbb{N}^{\leq \beta_{max}} \times \{0,1\}^{\leq \alpha} \to \mathcal{S} \times \{0,1\}^*$ updates the state with additional input prior to output generation, and optionally sets a variable $data \in \{0,1\}^*$ to store any additional state information necessary for output generation. Algorithm $\mathsf{gen} : \text{Seed} \times \mathcal{S} \times \{0,1\}^* \to \{0,1\}^{\ell} \times \mathcal{S} \times \{0,1\}^*$ maps a state S and optional string $data$ to an output block $r \in \{0,1\}^{\ell}$, an updated state S', and string $data' \in \{0,1\}^*$. Finally $\mathsf{final} : \text{Seed} \times \mathcal{S} \times \mathbb{N}^{\leq \beta_{max}} \times \{0,1\}^{\leq \alpha} \to \mathcal{S}$ is used to update the state post output generation.

The next algorithm is constructed from these component parts as shown in Figure 4.15. The decomposition algorithms $\mathcal{C} = (\mathsf{init}, \mathsf{gen}, \mathsf{final})$ for each of the NIST DRBGs are shown in Figure 4.16. It is readily verified that substituting each of these into the framework of Figure 4.15 yields the corresponding next algorithms shown in Figures 4.1 and 4.2. (For CTR-DRBG and HMAC-DRBG, $data$ is not set during output generation (e.g., $data = \varepsilon$), and so we omit it from the discussion of these DRBGs. Similarly since none of the NIST DRBGs are specified to take a salt, we omit this parameter.) A diagrammatic depiction of output generation for each of the DRBGs is shown in Figures 4.18a–4.18c.

Variable length outputs. Within this iterative structure, the gen subroutine acts like the next algorithm of an internal PRNG, called multiple times within a single next call to produce output blocks. However, as we shall see, the state updates performed by gen

```
HASH-DRBG init
Require: $S = (V, C, cnt), \beta, addin$
Ensure: $S = (V, C, cnt), data$
If $addin \neq \varepsilon$
  $w \leftarrow \mathsf{H}(0\mathrm{x}02 \| V \| addin)$
  $V \leftarrow (V + w) \bmod 2^L$
$data \leftarrow V$
Return $(V, C, cnt), data$

HASH-DRBG gen
Require $S = (V, C, cnt), data$
Ensure: $r, S = (V, C, cnt), data$
$r \leftarrow \mathsf{H}(data)$
$data \leftarrow (data + 1) \bmod 2^L$
Return $r, (V, C, cnt), data$

HASH-DRBG final
Require: $S = (V, C, cnt), \beta, addin$
Ensure: $S = (V, C, cnt)$
$H \leftarrow \mathsf{H}(0\mathrm{x}03 \| V)$
$V \leftarrow (V + H + C + cnt) \bmod 2^L$
$cnt \leftarrow cnt + 1$
Return $(V, C, cnt)$
```

```
HMAC-DRBG init
Require : $S = (K, V, cnt), \beta, addin$
Ensure: $S = (K, V, cnt)$
If $addin \neq \varepsilon$
  $(K, V) \leftarrow \mathsf{update}(addin, K, V)$
Return $(K, V, cnt)$

HMAC-DRBG gen
Require $(K, V, cnt)$
Ensure $r, S = (K, V, cnt)$
$V \leftarrow \mathsf{HMAC}(K, V)$ ; $r \leftarrow V$
Return $r, (K, V, cnt)$

HMAC-DRBG final
Require : $S = (K, V, cnt), \beta, addin$
Ensure: $S = (K, V, cnt)$
$(K, V) \leftarrow \mathsf{update}(addin, K, V)$
$cnt \leftarrow cnt + 1$
Return $(K, V, cnt)$
```

```
CTR-DRBG init
Require: $S = (K, V, cnt), \beta, addin$
Ensure: $S = (K, V, cnt)$
If $addin \neq \varepsilon$
  If derivation function used then
    $addin \leftarrow \mathsf{CTR\text{-}DRBG\_df}(addin, (\kappa + \ell))$
  Else if $len(addin) < (\kappa + \ell)$ then
    $addin \leftarrow addin \| 0^{(\kappa + \ell - len(addin))}$
  $(K, V) \leftarrow \mathsf{update}(addin, K, V)$
Return $(K, V, cnt)$

CTR-DRBG gen
Require: $S = (K, V, cnt)$
Ensure: $r, S = (K, V, cnt)$
$V \leftarrow (V + 1) \bmod 2^\ell$ ; $r \leftarrow E(K, V)$
Return $r, (K, V, cnt)$

CTR-DRBG final
Require: $S = (K, V, cnt), \beta, addin$
Ensure: $S = (K, V, cnt)$
$(K, V) \leftarrow \mathsf{update}(addin, K, V)$
$cnt \leftarrow cnt + 1$
Return $(K, V, cnt)$
```

Figure 4.16: Algorithms $\mathcal{C} = (\mathsf{init}, \mathsf{gen}, \mathsf{final})$ for HASH-DRBG, HMAC-DRBG, and CTR-DRBG.

do *not* provide forward security after each block[2]. This may not seem unreasonable if the DRBG produces only a handful of blocks per request; however since the standard allows for up to 2^{19} bits of output to be requested in each next call, there are situations in which the possibility of a partial state compromise occurring *during* output generation is worth considering.

Attack scenario: side channels. We consider an attacker who learns some information about the state variables being processed during output generation, but who is *not* able to perform a full memory compromise by which they would learn e.g., the output blocks $r^1, \ldots, r^n$ buffered in the internal memory, thereby compromising all output in the call. The natural scenario we consider here is a *side channel attack*. Generating multiple output blocks in a single next call results in a significant amount of computation going on 'under the hood' of next — e.g., up to $2^{12} = 4096$ AES-128 computations using a fixed key K^0 for CTR-DRBG with AES-128 — which, given that AES invites leaky implementations [26, 40, 95, 104, 118, 121], is concerning. Since robustness only allows the attacker to compromise the state *after* it has 'properly' updated (via the final process) at the conclusion of a next call, it does not model side channel leakage *during* the call.

[2]This is similar to an observation by Bernstein [28] criticising the inefficiency of CTR-DRBG's update function, which appeared concurrently to the production of the first draft of this work. We stress that our modelling of the attack scenario, and systematic treatment of how the issue affects each of the NIST DRBGs, is novel.

Use case: buffering output. As pointed out by Bernstein [28], the overhead incurred by the state update at the conclusion of a CTR-DRBG next call is undesirable. As such, an appealing usage choice is to generate a large output upfront in a single request, and buffer it to later be used for different purposes[3] . Our attack model investigates the soundness of this approach for scenarios in which partial state compromise during output generation via a side channel — which can only be exacerbated by such usage — is a realistic concern. Portions of the buffered output may be used for public values such as nonces, whereas other portions of the output from the same call may be used for e.g., secret keys. As such, our model assumes an attacker learns an output block sent in the clear as e.g., a nonce, in conjunction with the partial state information gleaned via a side channel. The attacker's goal is to recover *unseen* output blocks used as security critical secrets, thereby breaking the security of the consuming application.

4.8.2 Attack Model

We now describe our attack model. We found that a more formal and / or code-based model using abstract leakage functions (in line with the literature on leakage-resilient cryptography e.g., [3, 63, 111]) introduced significant complexity, without clarifying the presentation of the attacks or providing further insight. We therefore opted for a more informal written definition of the attack model which is nonetheless sufficiently precise to capture e.g., exactly what the attacker may learn, what he is challenged to guess, and so on. We aim to demonstrate key attacks rather than providing an exhaustive treatment.

Attack set-up and goals. Consider the next call shown in Figure 4.15. Letting S denote the state input to next, then this defines a sequence of intermediate states / output blocks passed through during the course of the request:

$$(S, (S^0, data^0), r^1, (S^1, data^1), \ldots, r^n, (S^n, data^n), S') ,$$

with the algorithm finally returning $(R, S') = (r^1 \parallel \ldots \parallel r^m, S')$. (For simplicity, we assume the requested number of bits is a multiple of the block length; it is straightforward to remove this assumption.) We consider an attacker $\mathcal{A}$ who is able to compromise a given component of an arbitrary intermediate state S^i (or in the case of HASH-DRBG, the additional state information $data^i$) for $i \in [0, n]$, in addition to an arbitrary output block

[3]Indeed, NIST SP 800-90A says: "For large generate requests, CTR-DRBG produces outputs at the same speed as the underlying block cipher algorithm encrypts data", highlighting the efficiency of this approach.

Table 4.2: Table summarising our analysis. The leftmost three columns correspond to Sections 4.8.3–4.8.5. The rightmost column corresponds to Section 4.8.6. A ✓ indicates that we demonstrate an attack. A × indicates that we believe the DRBG is not vulnerable to such an attack, with justification given. A * corresponds to an attack if CTR-DRBG is implemented without a derivation function. A ** indicates an exception in the case that *cnt* = 1 at the point of compromise.

	(1) Past output within call	**(2)** Future output within call	**(3)** Updated state S'	Additional input
CTR-DRBG // compromised K	✓	✓	✓	✓*
HMAC-DRBG // compromised K	×	✓	✓	×
HASH-DRBG // compromised V	✓	✓	×**	×

r^j for $j \in [1, n]$ produced in the same call. We assume the indices (i, j) are known[4] to $\mathcal{A}$. We then assess the attacker's ability to achieve each of the following 'goals':

- **(1)** Recover unseen output blocks produced prior to the compromised block within the call $\{r^k\}_{k<j}$;
- **(2)** Recover unseen output blocks produced following the compromised block within the call $\{r^k\}_{k>j}$; and
- **(3)** Recover the state S' as updated at the conclusion of the call. This allows the attacker to run the generator forwards and recover future output.

Extensions. If $addin = \varepsilon$, then init returns the state unchanged, $S^0 = S$. As such, all attacks which succeed when S^0 is partially compromised in our model can also be executed if the relevant component of state S is compromised *prior* to the next call, creating a greater window of opportunity for the attacker.

Security analysis. We analysed each of the NIST DRBGs with respect to our attack model, and found that each DRBG exhibited vulnerabilities, with CTR-DRBG faring especially badly. We summarise our findings in Table 4.2.

[4]Here we assume the attacker learns a full block and knows its index. This seems reasonable; for example, the public nonces exchanged in a TLS handshake (i.e., the client and server random) will contain at least one whole block and 12 bytes of a second block (if 4 bytes of timestamp are used). These values would be generated early in a call to the DRBG, and so have a low index j. Both assumptions can be relaxed at the cost of the attacker performing more work to brute-force any missing bits and / or the index.

```
CTR-DRBG // $\mathcal{A}(K^i, r^j, i, j)$
$V^j \leftarrow E^{-1}(K^i, r^j)$
$V^0 \leftarrow (V^j - j) \bmod 2^\ell$
For $k = 1, \ldots, n$
    $V^k \leftarrow (V^{k-1} + 1) \bmod 2^\ell$
    $r^k \leftarrow E(K^i, V^k)$
$(K', V') \leftarrow \mathsf{update}(addin, K^i, V^n)$
$cnt' \leftarrow cnt + 1$
$S' \leftarrow (K', V', cnt')$
Return $(\{r^k\}_{k<j}, \{r^k\}_{k>j}, S')$
```

```
HMAC-DRBG // $\mathcal{A}(K^i, r^j, i, j)$
$V^j \leftarrow r^j$
For $k = j+1, \ldots, n$
    $V^k \leftarrow \mathsf{HMAC}(K^i, V^{k-1})$
    $r^k \leftarrow V^k$
$(K', V') \leftarrow \mathsf{update}(addin, K^i, V^n)$
$cnt' \leftarrow cnt + 1$
$S' \leftarrow (K', V', cnt')$
Return $(\perp, \{r^k\}_{k>j}, S')$
```

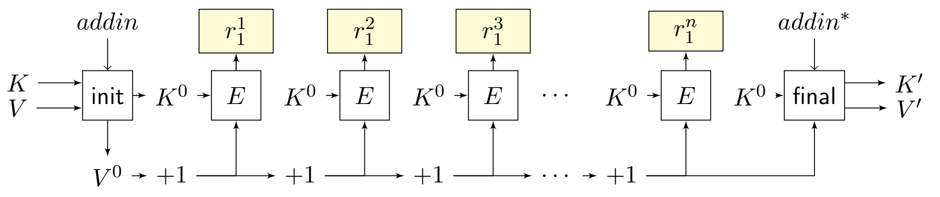
```
HASH-DRBG // $\mathcal{A}(data^i, r^j, i, j)$
$data^0 \leftarrow (data^i - i) \bmod 2^L$
For $k = 1, \ldots, n$
    $r^k \leftarrow \mathsf{H}(data^{k-1})$
    $data^k \leftarrow (data^{k-1} + 1) \bmod 2^L$
Return $(\{r^k\}_{k<j}, \{r^k\}_{k>j}, \perp)$
```

Figure 4.17: Adversaries for attacks in Sections 4.8.3-4.8.5.

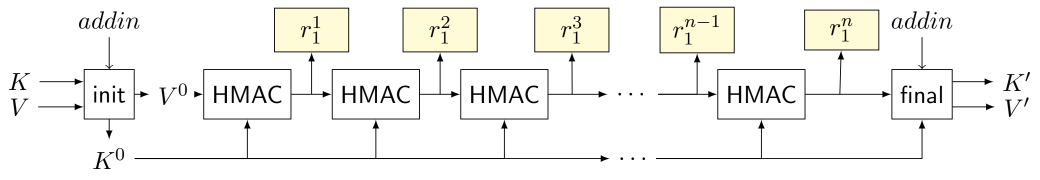
4.8.3 Security of CTR-DRBG with a Compromised Key

Since each output block encrypts the secret counter V, leakage of the key component of the CTR-DRBG state is especially damaging. Consider attacker $\mathcal{A}$ shown in the left-hand panel of Figure 4.17. We claim that for all $i \in [0, n]$ and $j \in [1, n]$, if additional input is not used ($addin = \varepsilon$) then $\mathcal{A}$ achieves goals **(1)**,**(2)** (recovery of all unseen output blocks produced in the next call) and **(3)** (recovery of the next state S') with probability one. If additional input is used ($addin \neq \varepsilon$) then the same statement holds for **(1)**, **(2)**, and the attacker's ability to satisfy **(3)** is equal to his ability to guess *addin*. To see this, notice that each block of output produced in the next call is computed as $r^k = E(K^0, V^0 + k)$ for $k \in [1, n]$, where K^0, V^0 denote the key and counter as returned by init at the start of output generation. The key does not update through this process, and so whatever intermediate key K^i attacker $\mathcal{A}$ compromises, this is the key used for output generation. It is then trivial for $\mathcal{A}$ to decrypt the output block r^j received in his challenge to recover the secret counter, thereby possessing all security critical state variables. However if $addin \neq \varepsilon$, then $\mathcal{A}$ must guess this string in order to compute S'.

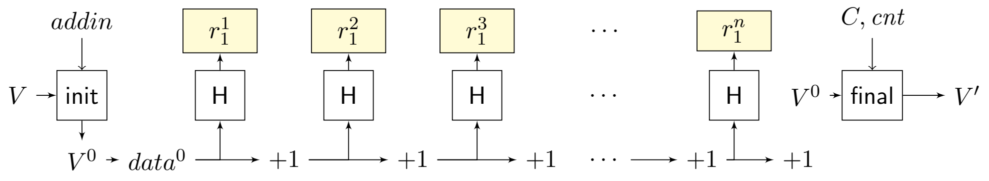
Discussion. This attack is especially damaging, since target output blocks used as e.g., secret keys will be recovered *irrespective* of their position relative to the block learnt by the attacker, increasing the exploitability of the compromised CTR-DRBG. In comparison, the infamously backdoored Dual-EC-DRBG only allowed recovery of output produced *after* the compromised block, impacting its practical exploitability [47]. (That said, the embedded backdoor in Dual-EC-DRBG means the attack itself is far easier to execute).



(a) Evolution of state $S = (K, V, cnt)$ within a next call for CTR-DRBG. Here $addin^* = addin$ if $addin \neq \varepsilon$ and $0^{\kappa+\ell}$ otherwise.



(b) Evolution of state (K, V, cnt) within a next call for HMAC-DRBG.



(c) Evolution of state $S = (V, C, cnt)$ within a next call for HASH-DRBG.

Figure 4.18: Diagrams showing output production for each of the NIST DRBGs.

4.8.4 Security of HMAC-DRBG with a Compromised Key

Consider the attacker $\mathcal{A}$ shown in the middle panel of Figure 4.17, who compromises the key component K^i of an intermediate state of HMAC-DRBG. We claim that for all $i \in [0, n]$ and $j \in [1, n]$, if $addin = \varepsilon$ then $\mathcal{A}$ achieves goals **(2)** and **(3)** with probability one. If $addin \neq \varepsilon$ then the same statement holds for **(2)**, and the attacker's ability to satisfy **(3)** is equal to his ability to guess *addin*. To see this, let K^0, V^0 denote the state variables at the beginning of output generation. Output blocks are iteratively produced by computing $V^k = \mathsf{HMAC}(K^0, V^{k-1})$ and setting $r^k = V^k$ for $k \in [1, n]$. Since the key does not update during this process, the key K^i compromised by the attacker will be equal to the key K^0 used for output generation. Since the output block r^j which $\mathcal{A}$ receives in his challenge is equal to the secret counter V^j, $\mathcal{A}$ now knows all security critical state variables of intermediate state S^j. $\mathcal{A}$ can then run HMAC-DRBG forward to recover all output produced following the compromised block in the call, as well as the updated state S' (subject to guessing *addin*).

Past output in a compromised next call. It appears that even if an attacker learns the entirety of an intermediate state S^i for $i \in [0, n]$ in addition to an output block r^j for $j \in [1, n]$, it is still infeasible to achieve goal **(1)** and recover the set of output blocks $\{r^k\}_{k<j}$ produced prior to the compromised block within the call. To see this, let V^0 denote the value of the counter at the start of output generation. For each $j \in [1, n]$, output block r^j takes the form:

$$r^j = V^j = \mathsf{HMAC}^j(K^0, V^0) \; ,$$

where $\mathsf{HMAC}^i(K, \cdot)$ denotes the i^{th} iterate of $\mathsf{HMAC}(K, \cdot)$. As such, recovering prior blocks r^k for $k < j$ given K^0 and V^j corresponds to finding preimages of $\mathsf{HMAC}(K^0, \cdot)$. Since the key is known to the attacker, we clearly cannot argue that this is difficult based on the PRF-security of HMAC. However, modelling HMAC as a random oracle (see Section 4.7) it follows that inverting HMAC for sufficiently high entropy V^0 is infeasible. Formalising this intuition under a standard model assumption remains an interesting open question.

4.8.5 Security of HASH-DRBG with a Compromised Counter.

It is straightforward to see that if $\mathcal{A}$ learns the counter value V^i in the $\mathsf{HASH\text{-}DRBG}$ state, or the iterating copy of the counter in $data^i$ for any $i \in [0, n]$, $j \in [1, n]$, then $\mathcal{A}$ achieves goals **(1)** and **(2)** with probability one. Moreover, knowledge of the counter is sufficient to execute the attack; no output block is needed. The case in which $data^i$ is compromised is shown in the rightmost panel of Figure 4.17. However, unlike $\mathsf{CTR\text{-}DRBG}$ and $\mathsf{HMAC\text{-}DRBG}$, without also learning the constant C achieving goal **(3)** does not seem possible in general. At the end of the next call, the new counter is computed as

$$V' = (V^0 + \mathsf{H}(0\text{x}03||V^0) + C + cnt) \bmod 2^L \; ;$$

however, for all but the first next call, it seems infeasible to extract the constant C from the known counter V^0 without inverting the hash function. (The exception with the first next call is because C is derived deterministically from the initial counter V during the setup process. Provided additional input is not used in the call, this is the counter $\mathcal{A}$ will be able to recover during the attack, allowing them to compute C themselves.) That said, if additional input is not used and an attacker compromises counters V^0 and $V^{0\prime}$ used for output generation from two consecutive next calls, then $\mathcal{A}$ can easily recover C by calculating $C = (V^{0\prime} - \mathsf{H}(0\text{x}03||V^0) - cnt) \bmod 2^L$, where cnt denotes the reseed counter at the point of the first next call, thus facilitating the recovery of the updated state and subsequent output. The same attack is possible if additional input is used conditional on $\mathcal{A}$ being able to guess its value.

4.8.6 Security of Additional Input

We present an additional attack against an implementation of CTR-DRBG which does not use a derivation function. This attack can (under certain conditions) allow an attacker who compromises the state of the DRBG to also recover the strings of additional input fed to the DRBG during output generation requests. This is particularly concerning given that the standard allows these strings to contain secrets and sensitive data provided they are not protected at a higher security strength than the instantiation.

Use of a derivation function. Consider the CTR-DRBG next algorithm (Section 4.4). If the derivation function is not used and additional input is included in a call, then the raw string of input is XORed directly into the CTR-DRBG state during the application of the update function (lines 8 and 15). One can verify from the pseudocode description of CTR-DRBG_df in Section 4.4 that to derive a $(\kappa+\ell)$-bit string from a T-bit input requires N_B block cipher computations where $N_B = \lceil(\kappa+\ell)/\ell\rceil \cdot (\lceil(T+72)/\ell\rceil + 2)$ and κ and ℓ denote the key and block size of the block cipher respectively. This computation is required for every next call which includes additional input, on top of each reseed and the initial state generation, and so represents a significant overhead. Indeed, a set of slides on the NIST DRBGs by Kelsey from 2004 [88] includes the comment "Block cipher derivation function is expensive and complicated... When gate count or code size is an issue, nice to be able to avoid using it!".

Recovery of additional input. We describe the attack with respect to the 'ideal' conditions. Take an implementation of CTR-DRBG built from AES-128 in which a derivation function is not used (the case for other approved block ciphers is totally analogous). Suppose that an attacker $\mathcal{A}$ has compromised the internal state $S = (K, V, cnt)$, and that the state compromise is followed by a next call in which additional input *addin* is used. Moreover, suppose *addin* has the form $addin = X_1 \parallel X_2$ where $X_1 \in \{0,1\}^{128}$ is known to the attacker and $X_2 \in \{0,1\}^{128}$ consists of 128 unknown bits. We assume X_2 includes a secret value such as a password which will be the target of the attack.

At the start of the next call, the state components K, V are updated with *addin* via $(K^0, V^0) \leftarrow \mathsf{update}(addin, K, V)$. It is straightforward to verify that

$$K^0 \,\|\, V^0 = K^* \,\|\, V^* \oplus addin = (K^* \oplus X_1) \,\|\, (V^* \oplus X_2) \ ,$$

where $K^* \,\|\, V^* = E(K, V+1) \,\|\, E(K, V+2)$. Since $\mathcal{A}$ has compromised (K, V), they can compute (K^*, V^*). Moreover, since X_1 is known to $\mathcal{A}$, it follows that the updated key $K^0 = (K^* \oplus X_1)$ is known to $\mathcal{A}$ also.

During output generation, output blocks are produced by encrypting the iterating counter under K^0. Therefore, the k^{th} block of output is of the form:

$$r^k = E(K^0, V^0 + k) = E(\mathbf{K^0}, (\mathbf{V}^* \oplus X_2) + \mathbf{k}) \ ,$$

where the variables in bold are known to $\mathcal{A}$. As such, each block of output produced is effectively an encryption of the target secret X_2 under a known key.

Given a single block of output r^k, $\mathcal{A}$ can *instantly* recover the target secret X_2 — consisting of 128-bits of unknown and secret data — as $X_2 = (E^{-1}(K^0, r^k) - k) \oplus V^*$. Moreover, it is straightforward to verify that $\mathcal{A}$ has sufficient information to compute the state as updated following the next call. As such, $\mathcal{A}$ can continue to execute the same attack against subsequent output generation requests for as long as the key component of the state evolves predictably.

Extensions. We have just described the ideal conditions for the attack. However the attack is still possible if X_1 is not known to the attacker. Supposing X_1 has γ-bits of entropy, then repeating the above process for each possible candidate for X_1 will recover the correct secret X_2 among a list of 2^γ candidates. If the data in X_2 is of a distinctive structure, or multiple output blocks can be recovered from the compromised next call, this can help $\mathcal{A}$ quickly eliminate candidates. Either way, the entropy of X_2 is reduced to at most γ-bits, a loss which — given X_2 contains up to 128-bits of unknown data — may be substantial.

Security benefit of the derivation function. The attack exploits the way in which the structure of *addin* is preserved by XOR. When the derivation function is used to condition the data, this structure is sufficiently destroyed that the above attack no longer works. Indeed, even if the attacker could compromise the raw derivation function output it is difficult to see how to recover the underlying additional input string more efficiently than by exhausting its entropy in a brute-force attack. Likewise for HASH-DRBG and HMAC-DRBG, in which additional input is hashed as it is incorporated into the state, it would appear that recovery of such strings requires a brute-force attack.

4.9 Open Source Implementation Analysis

In Section 4.8, we showed that certain implementation decisions — permitted by the overly flexible standard — may influence the security guarantees of the NIST DRBGs. To determine if these decisions are taken by implementers in the real world we investigated two open source implementations of CTR-DRBG, in OpenSSL [127] and mbed TLS [105]. We found that between the two libraries these problematic decisions have indeed been made.

Large output requests. As detailed in Section 4.8, generating many blocks of output in a single request increases both the likelihood and impact of our attacks. In OpenSSL, the `next` call of CTR-DRBG is implemented in the function `drbg_ctr_generate` in the file `drbg_ctr.c`. Interestingly — and contrary to the standard — this function does not impose *any limit* on the number of random bits which may be requested per call. As such, an arbitrarily large output may be generated using a single key, exacerbating the attacks of Section 4.8. More generally, exceeding the output generation limit increases the success probability of the well-known distinguishing attack against a block cipher in CTR-mode which uses colliding blocks to determine if an output is truly random.

By comparison, the implementation of CTR-DRBG in mbed TLS limits the number of output blocks per `next` call to 64 blocks of 128 bits. In the context of our attacks, this is much better for security than the 4,096 blocks allowed by the standard. Also this implementation forces a reseed after 10,000 calls to `next`, which is substantially lower than the 2^{48} calls that are allowed by the standard.

Derivation function. In Section 4.8.6, we showed that choosing to implement CTR-DRBG without the derivation function may allow the attacker to recover potentially sensitive data fed to the DRBG in the event of state compromise. We found that the OpenSSL implementation of CTR-DRBG allows the generator to be called simultaneously without the derivation function and with additional input. Specifically, by setting the flags field of the `RAND_DRBG_FLAG_CTR_NO_DF` structure to `RAND_DRBG_FLAG_CTR` the caller may suppress calls to the derivation function, presumably for performance purposes. As such, the attack described in Section 4.8.6 may be possible in real world implementations.

Summary. Despite the high level and theoretical nature of our analysis, we found that the problematic implementation decisions which we highlight *are* made in the real world. While none of these decisions leads to an immediate vulnerability, both the implementation and usage of the functions may exacerbate other problems such as side channel or state compromise attacks. We hope that highlighting these issues will help implementers make informed decisions about how best to use these algorithms in the context of their implementation.

4.10 Discussion and Open Problems

In this chapter, we conducted an in-depth and multi-layered analysis of NIST SP 800-90A, with a focus on investigating unproven security claims and exploring flexibilities in the standard. On the positive side, we formally verify a number of the claimed — and yet, until now unproven — security properties in the standard. However, we argue that taking certain implementation choices permitted by the overly flexible standard may lead to vulnerabilities. We conclude with a number of reflections and directions for future work.

Design and prove simultaneously. Certain design features of the NIST DRBGs complicate their analysis, and a small tweak in design would facilitate a far simpler proof. This emphasises the importance of developing cryptographic algorithms alongside security proofs, and — more importantly — not standardising algorithms with unproven security properties.

Flexibility. In Section 4.7, we saw how the option to call HMAC-DRBG without additional input changed the algorithm in a subtle way which lead to an attack. Similarly, the attacks of Section 4.8 are both facilitated, and exacerbated, by certain implementation choices allowed by the overly flexible standard. In Section 4.9, we confirmed that implementers do make these choices in the real world. These may be a warning to standard writers to avoid unnecessary flexibility which may lead to unintended vulnerabilities.

Usage recommendations. Because these vulnerabilities stem from implementation choices, we can offer recommendations to make the use of these algorithms more secure. First off, if the algorithms are being run in a setting where side channel attacks are a concern then CTR-DRBG should not be used. Additional input should be (safely) incorporated during output generation wherever possible and the DRBG should be reseeded with fresh entropy as often as is practical. While the standard allows outputs of sizeable length to be requested, users should not 'batch up' calls by making a single call for all randomness required for an application. Finally, the CTR-DRBG derivation function should always be used.

Future work. Analysing the robustness of CTR-DRBG is an important direction for further work. The key challenge with this seems to be proving that the derivation function is a good randomness extractor. Unfortunately, since the underlying CBC-MAC-based extractor (BCC in Figure 4.2) is applied multiple times to the same entropy sample, existing results on the extraction properties of CBC-MAC [58, 151] cannot be applied. A proof of robustness in the IPM [69] may be possible.

When analysing the robustness of the seedless HASH-DRBG and HMAC-DRBG, we made a strong assumption that the entropy source was independent of the random oracle in order to avoid known impossibility results on seedless extraction. Recent work by Coretti et al. [50] proposes a new robustness model for seedless PRNGs that allows positive results about such algorithms to be derived under weaker assumptions on the entropy source than that employed here. This work is an important step to address a long-standing gap between the seeded PRNGs defined in the theoretical literature, and the seedless PRNGs used in practice. Extending our results to analyse HASH-DRBG and HMAC-DRBG with respect to the model of [50] is an interesting topic for future work.

More generally, the design flexibilities we critique above are related to efficiency savings. Designing PRNGs that achieve an optimal balance between security and efficiency is a key direction for future work. For example, redesigning the CTR-DRBG derivation function to reduce computational overhead would make its use more palatable. The gap between the specification of these DRBGs, which allows for various optional inputs and implementation choices, and the far simpler manner in which PRNGs are typically modelled in the literature could indicate that theoretical models are not adequately capturing real world PRNGs. Extending these models may help understand the limits and possibilities of what can be achieved.

Chapter 5

Backdoors in Pseudorandom Generators: Possibility and Impossibility Results

Contents

5.1 Introduction and Motivation

Mass surveillance. The Snowden revelations of 2013 marked the point at which the scales collectively fell from the eyes of both the research community and society as a whole as to the extent to which government surveillance was encroaching on the privacy of civilians. The illusion that surveillance was being conducted in a targeted and proportional manner was instantly shattered, and there could be no doubt that nation states must be viewed as potential adversaries [126]. While such concerns had been raised before by privacy advocates [34], the explosive evidence of the Snowden leaks confimed that a new catalogue of threats [124] — which in the past may have seemed more at home in spy novels and conspiracy theories — were in fact a very real world concern.

A strong sense emerged that privacy, beyond being an important individual right, was vital for the functioning of democratic society [126]. Rogaway's [136] call for cryptographers to understand their work as inherently political and with the potential to be used for nefarious purposes resonated with many, and captured a feeling that such an abuse of governmental power must never again go unchecked.

Understanding Big Brother. Almost immediately, the cryptographic community sprang into action to analyse this new threat, and in the six years since Snowden pulled back the curtain a wealth of literature has been written on the topic. On the applied side, cryptographers sought to assess the impact of the Dual-EC in real world implementations [46, 47], and to extrapolate lessons from the debacle [31, 146]. On the theoretical side, foundational work has sought to understand different forms of cryptographic subversion by constructing security models and definitions to formally reason about Big Brother within the provable security framework. These lines of research have a long and rich history through topics such as kleptography [169] and subliminal channels [154], but were addressed with a new urgency post-Snowden. Such work includes the study of Algorithm Substitution Attacks (ASAs) [6, 20, 53, 68, 114, 143, 144] and backdooring cryptosystems [8, 12, 29, 47, 57]. In an ASA, the subversion is specific to a specific *implementation* of a particular algorithm or scheme, whereas in backdooring, the backdoor resides in the specification of the scheme or primitive itself and any implementation faithful to the specification will be equally vulnerable. There is a balancing act at play between these two types of attack: while ASAs are arguably easier to carry out, their impact is limited to a specific implementation, whereas the successful introduction of a backdoor into a cryptographic scheme, albeit ostensibly harder to mount and subsequently conceal, can have much wider impact.

Backdoored PRNGs. The Dual-EC-DRBG may be a particular thorn in the side of many in the cryptographic community. Languishing in plain sight in NIST SP 800-90A, it became an emblem of just how much the NSA had managed to get away with. Dual-EC-DRBG provides a particularly useful backdoor to Big Brother: given a single output from the generator, its state and therefore all future output can be recovered (with moderate computational effort). Protocols like SSL / TLS directly expose PRG outputs in protocol messages, making Dual-EC-DRBG exploitable in practice [47]. More broadly, as the previous chapter has attested, the critical reliance of cryptography on good pseudorandom generators — and its brittleness in the event of a randomness failure — makes these primitives the perfect target for subversion.

Contributions. In this chapter, we conduct a formal exploration of the extent to which a provably secure pseudorandom number generator can be backdoored. We contribute to the foundational groundwork required to understand the threat of backdoored PRNGs, and ask whether PRNGs with certain (strong) security properties may resist it.

5.1.1 Contributions

In this work, we advance understanding of backdoored pseudorandom number generators in two distinct directions. In the first part of the chapter, we build on existing work on backdoored PRGs [57] and resolve an open problem by showing that stronger forms of backdooring are possible than previously demonstrated. We then turn our attention to backdooring robust PRNGs with input and in the second part of the chapter present the first analysis of the subject, giving definitions, security models, and constructions. We begin with a brief overview of existing results on backdoored PRGs, before giving a more detailed overview of our contributions.

Related work and the Dual-EC. Dodis et al. [57] initiated the first formal study of backdoored PRGs (BPRGs), building on earlier work by Vazirani and Vazirani [163]. The authors construct backdoored PRGs using public key encryption schemes and key encapsulation mechanisms, analyse folklore immunisation techniques, and show that BPRGs are equivalent to public-key encryption (PKE) with pseudorandom ciphertexts (IND$-CPA secure PKE). The latter result is especially interesting as it suggests that PRGs built entirely from symmetric primitives are less likely to contain a backdoor.

A question that was posed (and partly answered) in [57] is: *to what extent can a PRG be backdoored while at the same time being provably secure?* This question — which motivates the problems addressed in this chapter — is especially pertinent in the context of subversion via backdooring, since the backdoor resides in the specification of the PRG which will likely be subjected to public scrutiny and analysis[1].

Stronger forms of backdooring. Prior to our work, no BPRG had been proposed that allows recovery of past output values while simultaneously being forward secure. For example, the backdoor in Dual-EC-DRBG only allows Big Brother (who holds the backdoor key) to compute unseen *future* output; pseudorandom bits produced *prior* to the compromised output remain unattainable. A construction of a 'random-seek' BPRG from [57] allows Big Brother, given any single output, to recover any past or future output with probability roughly $\frac{1}{4}$. However, this stronger form of backdooring comes at the expense of forward security. Indeed, intuitively, forward security and recovery of past output would seem to be opposing goals. As such, a natural question is whether this trade-off is inherent or if forward secure PRGs are susceptible to such strong forms of backdooring. If such a limitation were inherent, then a proof of forward security for a PRG would indicate a natural form of resistance to such backdoors.

New constructions of BPRGs. For our first contribution, we settle the above open problem in the negative by providing two constructions of forward secure BPRGs that allow recovery of past output (and more). Our constructions are a substantial strengthening of those of [57]. Firstly, both of our constructions allow Big Brother to succeed with probability 1 (rather than the $\frac{1}{4}$ attained for the random-seek BPRG construction of [57]). Secondly the backdoor is much stronger, in that for both of our BPRG constructions Big

[1]Remarkably, Dual-EC-DRBG has notable biases which rule out a proof of security — making its standardisation even more absurd and likely guided by the hand of a government agency. However, as noted in [57], these biases can be eliminated using special encodings of curve points as in [30, 116, 169], and Dual-EC-DRBG can be turned into a provably forward secure PRG under the decisional Diffie-Hellman assumption.

Brother is able to recover the initial state of the BPRG given only a single output value. This then enables all states and output values to be reconstructed.

Unsurprisingly, given the connection between BPRGs and IND\$-CPA secure PKE [57], both constructions make use of this latter primitive. This is the only chapter in this thesis to utilise public key techniques, and we provide the necessary definitions in Section 5.2.

Backdooring PRNGs with input. We then turn our attention to the study of backdoored PRNGs with input (BPRNGs). This is a natural extension of the study of BPRGs, particularly in view of the widespread deployment of PRNGs with input in real systems. As discussed in Chapter 4, robustness is generally accepted as the *de facto* security target for any new PRNG design, and so it is logical to require any BPRNG to be robust.

One might hope that with additional high entropy inputs being used to refresh the generator state, backdooring a PRNG with input might be impossible. This would certainly be a victory in the quest to defeat Big Brother. However, as we demonstrate in the second part of the chapter, this is unfortunately not the case.

Building backdoored PRNGs. As a warm-up, we first show how to adapt the robust PRNG of [60] to embed a backdoor. That construction uses an underlying PRG to produce output and our BPRNG is based on a simple trick: replace the underlying PRG with a BPRG. Given a single output, this allows Big Brother to compute *all* outputs from the last refresh to the next refresh, while the generator remains robust. However, as soon as the generator is refreshed with sufficient entropy Big Brother's advantage is lost and he will need to compromise more output in order to exploit the backdoor again.

The key challenge of backdooring a robust PRNG is therefore to construct a backdoor that endures through high entropy refreshes. We show that this can be achieved in the main result of the chapter: a construction of a robust BPRNG that allows recovery of past output values going back through the k previous refreshes (where k is a parameter of the scheme). The construction (see Section 5.4) is based on the idea of interleaving outputs of a (non-backdoored) PRNG with encryptions of snapshots of that PRNG's state, using an IND\$-CPA secure encryption scheme to ensure pseudorandomness of outputs. By taking a snapshot of the state whenever it is refreshed and storing a list of the previous k snapshots in the state, the construction allows Big Brother to recover, with some probability, output values that were computed as many as k refreshes previously. The actual construction is considerably more complex than this sketch hints, since achieving robustness is challenging when the state has this additional structure. We additionally sketch variants of the construction that trade state and output size for strength of backdooring.

Open problems. While we demonstrate that even robust PRNGs succumb to strong forms of backdooring, our construction has the limitation that the state grows linearly with the number of high entropy refreshes we wish to simultaneously backdoor. The work upon which this chapter is based [54] included an impossibility result that claimed to demonstrate that this limitation is inherent. Regrettably, during the production of this thesis we discovered an error in the proof of [54] that invalidates the result. We therefore finish our study of backdoored PRNGs with a discussion of the claimed result and the error, and outline some possible directions for future work.

5.2 Definitions and Preliminaries

This thesis has so far focused on symmetric cryptography. In this chapter, we will extend our cryptographic tool-kit to include a number of public key primitives. We then specify the PRG and PRNG definitions that shall be used throughout the chapter.

5.2.1 Public Key Encryption Schemes

A public key encryption (PKE) scheme allows two parties who do not share a secret key to exchange encrypted messages. We define our PKE syntax below, and then discuss the security properties we require of such schemes.

Definition 5.1. *A public key encryption (PKE) scheme is a triple of algorithms* $\mathsf{PKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$*, with associated public key space* $\mathcal{PK}$*, secret key space* $\mathcal{SK}$*, message space* $\mathcal{M}$*, ciphertext space* $\mathcal{C}$*, and coin space* $\mathcal{R}$*, defined as follows:*

- $\mathsf{Key} : \to \mathcal{PK} \times \mathcal{SK}$ *is a randomised algorithm which takes no input, and returns a public / secret key pair* $(pk, sk) \in \mathcal{PK} \times \mathcal{SK}$*.*
- $\mathsf{Enc} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C}$ *is a randomised algorithm which takes as input a public key*

$pk \in \mathcal{PK}$ and a message $M \in \mathcal{M}$, and returns a ciphertext $C \in \mathcal{C}$. For $r \in \mathcal{R}$, we write $\mathsf{Enc}(\cdot, \cdot; r)$ to denote executing Enc *with coins r fixed.*

- Dec $: \mathcal{SK} \times \mathcal{C} \rightarrow \mathcal{M}$ *is a deterministic algorithm which takes as input a ciphertext $C \in \mathcal{C}$ and secret key $sk \in \mathcal{SK}$, and returns a string $M' \in \mathcal{M} \cup \{\perp\}$.*

Correctness and length regularity. We require a PKE scheme $\mathsf{PKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ to be *perfectly correct*, by which we mean that for all $pk \in \mathcal{PK}$ and all messages $M \in \mathcal{M}$, it holds that $\Pr[\,\mathsf{Dec}(sk, (\mathsf{Enc}(pk, M)) = M\,] = 1$. We also require that encryption is *length-regular*, by which we mean that there exists a function $\mathsf{len}_{\mathsf{PKE}} : \mathbb{N} \rightarrow \mathbb{N}$ such that for all $(pk, M) \in \mathcal{PK} \times \mathcal{M}$ it holds that $\Pr[\,|C| = \mathsf{len}_{\mathsf{PKE}}(|M|) \,:\, C \leftarrow\!\!\$\, \mathsf{Enc}(pk, M)\,] = 1$. In both cases, the probability is over the coins of Enc.

Cyclic groups and the DDH assumption. PKE schemes such as ElGamal [64] (introduced later in this section) are frequently defined with respect to a cyclic group of order q; that is to say, a group $\mathbb{G}$ for which there exists an element $g \in \mathbb{G}$ such that $\mathbb{G} = \{g^0, \ldots, g^{q-1}\}$. To use cyclic groups for cryptography, we need an algorithm to efficiently sample them. We define the group generation algorithm GGen to be a randomised algorithm which takes no input and outputs a tuple $(\mathbb{G}, q, g)$, where $\mathbb{G}$ is a description of a cyclic group of order q for which g is a generator.

A commonly used cryptographic assumption for cyclic groups is the *decisional Diffie-Hellman (DDH) assumption.* The DDH assumption requires that for a cyclic group $(\mathbb{G}, g, q)$, the tuples (g^x, g^y, g^{xy}) and (g^x, g^y, g^z) for $x, y, z \leftarrow\!\!\$\, \mathbb{Z}_q$ are indistinguishable. More formally, we define the advantage of an attacker $\mathcal{A}$ in the DDH game with respect to group generation algorithm GGen to be:

$$\mathbf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{A}) = \Big| \Pr[\,\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^{xy}) \Rightarrow 1 \,:\, (\mathbb{G}, g, q) \leftarrow\!\!\$\, \mathsf{GGen}\,; x, y \leftarrow\!\!\$\, \mathbb{Z}_q\,] \\ - \Pr[\,\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) \Rightarrow 1 \,:\, (\mathbb{G}, g, q) \leftarrow\!\!\$\, \mathsf{GGen}\,; x, y, z \leftarrow\!\!\$\, \mathbb{Z}_q\,] \Big| \,.$$

The DDH assumption is widely used, underpinning the security of the Diffie-Hellman key exchange protocol [55] and the ElGamal PKE scheme, among many others.

The ElGamal PKE scheme. A well-known PKE scheme that we will use as an illustrating example throughout this chapter is ElGamal [64]. The scheme $\mathsf{ElG.PKE} =$

```
$\mathsf{ElG.Key}$
$(\mathbb{G}, q, g) \leftarrow\!\!\$\ \mathsf{GGen}$
$x \leftarrow\!\!\$\ \{0, \ldots, q-1\}$
$sk \leftarrow (\mathbb{G}, q, g, x)$
$pk \leftarrow (\mathbb{G}, q, g, g^x)$
Return $(pk, sk)$
```

```
$\mathsf{ElG.Enc}(pk, M)$
$(\mathbb{G}, q, g, h) \leftarrow pk$
$y \leftarrow\!\!\$\ \mathbb{Z}_q$
$C_1 \leftarrow g^y$
$C_2 \leftarrow M \cdot h^y$
Return $(C_1, C_2)$
```

```
$\mathsf{ElG.Dec}(sk, (C_1, C_2))$
$(\mathbb{G}, q, g, x) \leftarrow sk$
$M \leftarrow C_2 \cdot (C_1^x)^{-1}$
Return $M$
```

```
$\mathsf{ElG.Rand}(pk, (C_1, C_2))$
$(\mathbb{G}, q, g, h) \leftarrow pk$
$z \leftarrow\!\!\$\ \mathbb{Z}_q$
$C_1 \leftarrow C_1 \cdot g^z$
$C_2 \leftarrow C_2 \cdot h^z$
Return $(C_1, C_2)$
```

```
$\mathsf{ElG.Rand}^{-1}(pk, (C_1, C_2); z)$
$(\mathbb{G}, q, g, h) \leftarrow pk$
$C_1 \leftarrow C_1 \cdot g^{-z}$
$C_2 \leftarrow C_2 \cdot h^{-z}$
Return $(C_1, C_2)$
```

Figure 5.1: The ElGamal re-randomisable PKE Scheme $\mathsf{ElG.PKE} = (\mathsf{ElG.Key}, \mathsf{ElG.Enc}, \mathsf{ElG.Dec}, \mathsf{ElG.Rand}, \mathsf{ElG.Rand}^{-1})$, where GGen is a group generation algorithm.

$(\mathsf{ElG.Key}, \mathsf{ElG.Enc}, \mathsf{ElG.Dec})$ is depicted in the two leftmost panels of Figure 5.1. The scheme has associated message space $\mathcal{M} = \mathbb{G}$ and ciphertext space $\mathcal{C} = \mathbb{G} \times \mathbb{G}$. We will frequently abuse notation to express the message space and messages encrypted by ElGamal as bit strings rather than group elements. In such cases we implicitly assume that a suitable encoding scheme is used to map the bit strings into the appropriate group.

Real-or-random PKE security. The usual security requirement for PKE schemes is that of IND-CPA security, which requires that encryptions of equal length messages are indistinguishable. It is straightforward to see that ElGamal is an IND-CPA secure PKE scheme under the assumption that the DDH problem is hard with respect to the group generation algorithm GGen. In this work, we shall require the stronger property of IND\$-CPA security, which requires that ciphertexts are indistinguishable from random bit strings. This security property is more challenging to achieve than the analogous ROR notion for AEAD schemes (see Section 3.2), since the ciphertexts of PKE schemes typically have mathematical structure (for example, pairs of group elements in the case of ElGamal) which allow them to be distinguished from uniform strings. Nonetheless, concrete and efficient examples of such schemes can be obtained by applying carefully constructed encoding schemes to the group elements of ciphertexts in the ElGamal encryption scheme. For example, the Elligator scheme introduced in [30] gives a pseudorandom encoding of elliptic curve points, thereby allowing an Elligator-encoded variant of elliptic curve ElGamal to be IND\$-CPA. Further examples of encodings are given in [116, 169]. We define both security properties formally below.

Definition 5.2. *Let* $\mathsf{PKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme. The* IND-CPA *advantage* *of an attacker $\mathcal{A}$ against* PKE*, who makes q queries to their LoR oracle, is defined:*

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{PKE}}(\mathcal{A},q) = \Big|\Pr\left[\mathcal{A}^{\text{LoR}_0(pk,\cdot,\cdot)}(pk) \Rightarrow 1 \ : \ (pk,sk) \leftarrow\!\!\$\ \mathsf{Key}\right] \\ - \Pr\left[\mathcal{A}^{\text{LoR}_1(pk,\cdot,\cdot)}(pk) \Rightarrow 1 \ : \ (pk,sk) \leftarrow\!\!\$\ \mathsf{Key}\right]\Big| \ ,$$

where $\text{LoR}_b(pk,\cdot,\cdot)$*, on input a pair of messages $M_0, M_1 \in \mathcal{M}$, returns $\perp$ if $|M_0| \neq |M_1|$, and $C \leftarrow\!\!\$\ \mathsf{Enc}(pk, M_b)$ otherwise. The* IND\$-CPA *advantage of an attacker $\mathcal{A}$ against* PKE*, who makes q queries to their real-or-random (RoR) oracle, is defined:*

$$\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{A},q) = \Big|\Pr\left[\mathcal{A}^{\mathsf{Enc}(pk,\cdot)}(pk) \Rightarrow 1 \ : \ (pk,sk) \leftarrow\!\!\$\ \mathsf{Key}\right] \\ - \Pr\left[\mathcal{A}^{\$(\cdot)}(pk) \Rightarrow 1 \ \ (pk,sk) \leftarrow\!\!\$\ \mathsf{Key}\right]\Big| \ ,$$

where $\$(\cdot)$, on input a message $M \in \mathcal{M}$, returns a random string of length $\mathsf{len}_{\mathsf{PKE}}(|M|)$.

It is straightforward to show that for any PKE scheme PKE, and any attacker $\mathcal{A}$ in game IND-CPA against PKE, there exists an attacker $\mathcal{B}$ in game IND\$-CPA, running in the same time as $\mathcal{A}$ and making the same number of oracle queries, such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{PKE}}(\mathcal{A},q) \leq 2 \cdot \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{B},q) \ .$$

As such, any PKE scheme which is IND\$-CPA secure is also IND-CPA.

As mentioned in Section 5.1, it was shown in [57] that backdoored PRGs are equivalent to IND\$-CPA-encryption, and so it is unsurprising that these schemes feature heavily in our constructions.

Re-randomisable PKE schemes. For certain results in this chapter, we utilise PKE schemes which have the additional property of being *statistically re-randomisable* [77]. A re-randomisable PKE scheme is a regular PKE scheme with an additional procedure Rand which is used to update (or 're-randomise') a ciphertext with fresh random coins. We require that the re-randomised ciphertext is statistically indistinguishable from a fresh encryption of the underlying message. More formally, we define a re-randomisable encryption scheme to be a tuple of algorithms $\mathsf{rPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ where $(\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ are defined identically to the analogous algorithms in Definition 5.1, and $\mathsf{Rand} \ : \ \mathcal{PK} \times \mathcal{C} \to \mathcal{C}$ is a randomised algorithm which takes as input a public key $pk \in \mathcal{PK}$ and a ciphertext $C \in \mathcal{C}$, and outputs a re-randomised ciphertext $C' \in \mathcal{C}$. We define the *re-randomisation* distance of rPKE, denoted Δ_{rPKE}, to be

$$\Delta_{\mathsf{rPKE}} = \max_{pk,M,r_0'} = \Delta(\{C \ : \ C \leftarrow\!\$\ \mathsf{Enc}(pk, M)\}, \{C \ : \ C \leftarrow\!\$\ \mathsf{Rand}(pk, \mathsf{Enc}(pk, M; r_0'))\}) \ .$$

In words, we take Δ_{rPKE} to be the maximum statistical distance (over $pk \in \mathcal{PK}$, $M \in \mathcal{M}$, and $r_0' \in \mathcal{R}$) between the distribution of an honest encryption of M under pk, and that of a ciphertext obtained by applying Rand to an encryption of M under pk and coins r_0'.

Correctness. We require that decrypting a re-randomised ciphertext returns the correct underlying message with probability one. More formally, we require that for all $pk \in \mathcal{PK}$, and all messages $M \in \mathcal{M}$, it holds that:

$$\Pr\left[\, M \leftarrow \mathsf{Dec}(sk, C^*) \ : \ C \leftarrow\!\$\ \mathsf{Enc}(pk, M) \ ; C^* \leftarrow\!\$\ \mathsf{Rand}(pk, C) \,\right] = 1 \ ,$$

where the probability is over the coins of Enc and Rand.

Letting $\mathcal{Z}$ denote the coin space of Rand, we write $\mathsf{Rand}(pk, C; z)$ to denote executing Rand with coins $z \in \mathcal{Z}$. We write $\mathsf{Rand}(pk, C_0; z_1, \ldots, z_q)$ to denote the ciphertext C_q obtained by computing $C_j \leftarrow \mathsf{Rand}(pk, C_{j-1}; z_j)$ for $j = 1, \ldots, q$ and $z_1, \ldots, z_q \in \mathcal{Z}$.

Reverse re-randomisable PKE schemes. We may extend the above definition to introduce *reverse re-randomisable* PKE schemes. These are re-randomisable PKE schemes with an additional algorithm Rand^{-1} which, given a public key, a re-randomised ciphertext, and the coins with which it was re-randomised, can recover the original ciphertext. We formalise this in the following definition

Definition 5.3. *A reverse re-randomisable PKE scheme is a tuple of algorithms* $\mathsf{rrPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$ *where:*

- $(\mathsf{Key}, \mathsf{Enc}, \mathsf{Rand}, \mathsf{Dec})$ *are defined identically to the analogous algorithms for re-randomisable PKE schemes; and*
- $\mathsf{Rand}^{-1} \ : \ \mathcal{PK} \times \mathcal{C} \times \mathcal{Z} \to \mathcal{C}$ *is a deterministic algorithm that takes as input a public key* $pk \in \mathcal{PK}$*, a ciphertext* $C' \in \mathcal{C}$*, and coins* $\mathcal{R} \in \mathcal{Z}$*, and outputs a ciphertext* $C \in \mathcal{C}$*.*
- *We require that for all* $pk \in \mathcal{PK}$*, all* $C \in \mathcal{C}$*, and all* $z \in \mathcal{Z}$*, it holds that:*

$$\mathsf{Rand}^{-1}(pk, \mathsf{Rand}(pk, C; z), z) = C \ .$$

Suppose $C_q = \mathsf{Rand}(pk, C_0; z_1, \ldots, z_q)$, so $C_j = \mathsf{Rand}(pk, C_{j-1}; z_j)$ for $j = 1, \ldots, q$. Then, from the above, we know that $C_{j-1} = \mathsf{Rand}^{-1}(pk, C_j; z_j)$ for $j = 1, \ldots, q$. To denote C_0, we write $\mathsf{Rand}^{-1}(pk, C_q; z_1, \ldots, z_q)$.

It is straightforward to see that ElGamal encryption and its encoded variants are reverse re-randomisable. The Rand and Rand^{-1} algorithms for ElGamal are shown in the right-hand panel of Figure 5.1.

5.2.2 Trapdoor Permutations

Trapdoor permutations are easy to compute in the forward direction but infeasible to invert without knowledge of a secret parameter called a *trapdoor*. We recall their formal definition below.

Definition 5.4. *A family of trapdoor permutations is a tuple of algorithms* $\mathsf{TDP} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$ *defined as follows:*

- $\mathsf{Gen} :\to \{0,1\}^* \times \{0,1\}^*$ *is a randomised algorithm which takes no input, and outputs a public parameter and trapdoor* $(\mathsf{PK}, \mathsf{SK}) \in \{0,1\}^* \times \{0,1\}^*$*. Each* PK *implicitly defines a permutation over* $\{0,1\}^n$*.*
- $\mathsf{F} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^n$ *takes as input* $\mathsf{PK} \in \{0,1\}^*$ *and* $x \in \{0,1\}^n$*, and outputs* $y \in \{0,1\}^n$*. We write* $y \leftarrow \mathsf{F}_{\mathsf{PK}}(x)$ *to denote applying* F *to input* (PK, x) *to produce output* y*.*
- $\mathsf{F}^{-1} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^n$ *takes as input* $\mathsf{SK} \in \{0,1\}^*$ *and* $y \in \{0,1\}^n$ *and outputs* $z \in \{0,1\}^n$ *such that* $\mathsf{F}_{\mathsf{PK}}(z) = y$*. We write* $z \leftarrow \mathsf{F}^{-1}_{\mathsf{SK}}(y)$ *to denote applying* F^{-1} *to input* (SK, y) *to produce output* z*.*

We require that TDP is infeasible to invert without knowledge of the trapdoor SK. We model this via game TDP-INV, shown in Figure 5.2, where the advantage of attacker $\mathcal{A}$ is defined

$$\mathbf{Adv}^{\mathrm{inv}}_{\mathsf{TDP}}(\mathcal{A}) = \Pr\left[\,\text{TDP-INV}^{\mathcal{A}}_{\mathsf{TDP}} \Rightarrow \mathsf{true}\,\right] .$$

While requiring that trapdoor permutations have domain $\{0,1\}^n$ as opposed to e.g., $\mathbb{Z}_N^*$ is a restriction, standard trapdoor permutations such as RSA [131] and Blum, Blum, Shub [39], modified as described in [18] to have domain $\{0,1\}^n$, satisfy the above definition.

```
TDP-INV$^{\mathcal{A}}_{\mathsf{TDP}}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow\!\!\$\ \mathsf{Gen}$
$x \leftarrow\!\!\$\ \{0,1\}^n$
$y \leftarrow \mathsf{F}_{\mathsf{PK}}(x)$
$z \leftarrow\!\!\$\ \mathcal{A}(\mathsf{PK}, y)$
Return $(z = x)$
```

Figure 5.2: Trapdoor permutation security game.

5.2.3 Pseudorandom Number Generators

In this chapter, we will consider both PRNGs with input (the focus of Chapter 4) as well as deterministic pseudorandom generators (PRGs). We define the variants of these primitives that will be used in this chapter below.

Parameter generation. Throughout this chapter, we define PRNGs as in Definition 2.3 with one slight modification. Namely, we define $\mathsf{init}\ :\rightarrow \{0,1\}^* \times \{0,1\}^*$ to be a randomised algorithm which takes no input and outputs a *pair* of public and backdoor parameters $(pp, bp) \in \{0,1\}^* \times \{0,1\}^*$. The public parameter pp corresponds to the seed, seed, returned by init in Definition 2.3. Looking ahead, when modelling backdoored PRNGs, we will assume that Big Brother knows the backdoor parameter bp. To capture a non-backdoored PRNG in this syntax, we simply take $bp = \perp$.

Deterministic PRGs. A PRG is a deterministic pseudorandom number generator which has no access to an entropy source (and so has no $\mathsf{refresh}$ procedure). More formally, we define a deterministic PRG with output length $\ell \in \mathbb{N}$ to be a tuple of algorithms $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$. Algorithms setup and next are defined identically to those in Definition 2.3, and init is as defined as above.

Notation. Let $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$ be a PRG. Given an initial state S_0, let $(R_i, S_i) \leftarrow \mathsf{next}(pp, S_{i-1})$ for $i = 1, \ldots, q$. We write $(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$ and $(S_1, \ldots, S_q) \leftarrow \mathsf{state}^q(\mathsf{next}(pp, S_0))$ to denote, respectively, the sequence of outputs and states produced by this process.

PRG security. Security for PRGs can be defined as a special case of game Rob (Section 2.5.1) in which an attacker may make no Ref queries. In this chapter we will use an equivalent formulation which dispenses with the oracles which are available to the attacker

```
$\text{PRG-DIST}^{\mathcal{A},q}_{\mathsf{PRG}}$
$(pp, bp) \leftarrow\!\$\ \mathsf{init}$
$S_0 \leftarrow\!\$\ \mathsf{setup}(pp)$
$b \leftarrow\!\$\ \{0,1\}$
If $b = 0$
    $(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
Else $(R_1, \ldots, R_q) \leftarrow\!\$\ (\{0,1\}^\ell)^q$
$b' \leftarrow \mathcal{A}(pp, R_1, \ldots, R_q)$
Return $(b = b')$
```

```
$\text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}}$
$(pp, bp) \leftarrow\!\$\ \mathsf{init}$
$S_0 \leftarrow\!\$\ \mathsf{setup}(pp)$
$b \leftarrow\!\$\ \{0,1\}$
If $b = 0$
    $(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
Else $(R_1, \ldots, R_q) \leftarrow\!\$\ (\{0,1\}^\ell)^q$
$(S_1, \ldots, S_q) \leftarrow \mathsf{state}^q(\mathsf{next}(pp, S_0))$
$b' \leftarrow \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $(b = b')$
```

```
$\overline{\text{PRG-FWD}}^{\mathcal{A},q}_{\mathsf{PRG}}$
$(pp, bp) \leftarrow\!\$\ \mathsf{init}$
$S_0 \leftarrow\!\$\ \mathsf{setup}(pp)$
$b \leftarrow\!\$\ \{0,1\}$
If $b = 0$
    $(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
    $(S_1, \ldots, S_q) \leftarrow \mathsf{state}^q(\mathsf{next}(pp, S_0))$
Else $(R_1, \ldots, R_q) \leftarrow\!\$\ (\{0,1\}^\ell)^q$
    $S_q \leftarrow\!\$\ \mathsf{setup}(pp)$
$b' \leftarrow \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $(b = b')$
```

Figure 5.3: PRG security games.

in the robustness game. This simplifies the presentation of our results and remains in line with prior work on backdoored PRGs [57].

We consider three variants of PRG security, shown in Figure 5.3. In each game, the PRG is initialised and used to compute q outputs. Game PRG-DIST captures the basic pseudorandomness requirement that an attacker should not be able to distinguish these q outputs from random bit strings of appropriate length. The forward security game PRG-FWD is identical to PRG-DIST, except the attacker is also given the (real) final PRG state S_q as part of his challenge. Finally, $\overline{\text{PRG-FWD}}$ is a strengthening of PRG-FWD in which we additionally require the challenge state S_q to be indistinguishable from a freshly generated state $S_q \leftarrow\!\$\ \mathsf{setup}(pp)$. Looking ahead, this final game shall be used to simplify the proof of security for the simple backdoored PRNG given in Section 5.4. We define the advantage of an attacker $\mathcal{A}$ in $\text{Gm}^x_y \in \{\text{PRG-DIST}^{\mathcal{A},q}_{\mathsf{PRG}}, \text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}}, \overline{\text{PRG-FWD}}^{\mathcal{A},q}_{\mathsf{PRG}}\}$ to be:

$$\mathbf{Adv}^{\text{gm}}_y(x) = 2 \cdot \left| \Pr\left[\, \text{Gm}^x_y \Rightarrow 1 \,\right] - \frac{1}{2} \right| .$$

Stateless PRGs. We will occasionally utilise a simple, stateless variant of a deterministic PRG. We define a stateless PRG with parameters $(u, \ell) \in \mathbb{N}^2$ to be an algorithm G which takes as input a seed $\zeta \in \{0,1\}^u$ and returns an output $R \in \{0,1\}^\ell$. We require that if the seed ζ is chosen uniformly, then the resulting output is pseudorandom. More formally, we define the stateless PRG distinguishing advantage of an attacker $\mathcal{A}$ to be:

$$\mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{A}) = \Big| \Pr\left[\, \mathcal{A}(R) \Rightarrow 1 \ :\ \zeta \leftarrow\!\$\ \{0,1\}^u \,; R \leftarrow \mathsf{G}(\zeta) \,\right] - \Pr\left[\, \mathcal{A}(R) \Rightarrow 1 \ :\ R \leftarrow\!\$\ \{0,1\}^\ell \,\right] \Big| .$$

5.3 Backdooring Pseudorandom Generators

We begin our investigation into backdoored PRNGs by considering the simpler case of backdoored deterministic PRGs (BPRGs). Given that a PRG is a restricted version of a PRNG, this is a natural starting point at which to develop the techniques that we will ultimately use to backdoor PRNGs with input.

5.3.1 Modelling Backdoored PRGs

Intuitively, a backdoored cryptosystem is a scheme coupled with some secret backdoor information. In the view of an adversary who does not know the backdoor information, the scheme fulfils its usual security definition. However, an adversary in possession of the backdoor information will gain some advantage in breaking the security of the scheme.

Big Brother. The backdoor attacker is modelled as an algorithm which we call BB (for 'Big Brother') to distinguish it from an attacker $\mathcal{A}$ whose goal is to break the usual security of the scheme *without* access to the backdoor. Whilst the backdoor attacker BB will be external in the sense that it will only be able to observe public outputs and parameters, the attack is also internalised as the backdoor algorithm is designed alongside, and incorporated into, the scheme. We formalise this below, following [57].

Backdoored PRG syntax. Formally, a BPRG is defined to be a tuple of algorithms $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$. Algorithms $\mathsf{init}, \mathsf{setup}$, and next are defined identically to those for (non-backdoored) PRGs (Section 5.2.3). The input / output behaviour of Big Brother will vary depending on the BPRG in question. Crucially, BB will always take as input a backdoor parameter $bp \in \{0,1\}^*$, as well as the public parameter $pp \in \{0,1\}^*$.

5.3.2 Backdoored PRG Security

Let $\mathsf{BPRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next}, \mathsf{BB})$ be a BPRG. There are two metrics by which we assess the effectiveness and security of BPRG. Firstly, we require that the PRG defined by

```
$\text{BPRG-DIST}^{\mathsf{BB},q}_{\mathsf{BPRG}}$
$(pp, bp) \leftarrow\$ \mathsf{init}$
$S_0 \leftarrow\$ \mathsf{setup}(pp)$
$b \leftarrow\$ \{0,1\}$
If $b = 0$
   $(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
Else $(R_1, \ldots, R_q) \leftarrow\$ (\{0,1\}^\ell)^q$
$b^* \leftarrow\$ \mathsf{BB}(pp, bp, R_1, \ldots, R_q)$
Return $(b = b^*)$
```

```
$\text{BPRG-NEXT}^{\mathsf{BB},q}_{\mathsf{BPRG}}$
$(pp, bp) \leftarrow\$ \mathsf{init}$
$S_0 \leftarrow\$ \mathsf{setup}(pp)$
$(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
$(S_1, \ldots, S_q) \leftarrow \mathsf{state}^q(\mathsf{next}(pp, S_0))$
$S_q^* \leftarrow\$ \mathsf{BB}(pp, bp, R_1, \ldots, R_q)$
Return $(S_q = S_q^*)$
```

```
$\text{BPRG-RSEEK}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i, j)$
$(pp, bp) \leftarrow\$ \mathsf{init}$
$S_0 \leftarrow\$ \mathsf{setup}(pp)$
$(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
$R_j^* \leftarrow\$ \mathsf{BB}(pp, bp, i, j, R_i)$
Return $(R_j = R_j^*)$
```

Figure 5.4: Security games for backdoored PRGs from [57].

$\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$[2] is secure in the face of a regular (non-backdoor) attacker. This is important, since for Big Brother to be able to utilise his backdoor the BPRG itself must be used in honest implementations. Algorithms which are clearly insecure are likely to be rejected by implementers, and certainly would not achieve the widespread adoption enjoyed by the standardised Dual-EC-DRBG. Dodis et al. [57] measure the PRG security of a BPRG with respect to PRG-DIST; looking ahead, we will target the stronger notion of PRG-FWD.

For our second metric, we measure the effectiveness of the backdoor embedded in BPRG by assessing the probability that Big Brother, given bp, can achieve some backdooring goal. Our results on BPRGs will state the advantage terms of both a regular PRG attacker $\mathcal{A}$ and Big Brother BB in their respective games.

Backdooring games. To capture a given backdooring goal, we define a game $\text{BPRNG-TYPE}^{\mathsf{BB},q}_{\mathsf{BPRG}}$ with corresponding advantage term. The three games considered in [57] are defined in Figure 5.4. In the left-hand panel of Figure 5.4, game BPRG-DIST challenges Big Brother to break the PRG-DIST security of the PRG in the most basic sense of distinguishing real from random outputs. The advantage term of Big Brother in this game is defined

$$\mathbf{Adv}^{\text{bprg-dist}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = 2 \cdot \left| \Pr\left[\text{BPRG-DIST}^{\mathsf{BB},q}_{\mathsf{BPRG}} \Rightarrow \mathsf{true} \right] - \frac{1}{2} \right| .$$

In game $\text{BPRG-NEXT}^{\mathsf{BB},q}_{\mathsf{BPRG}}$ (middle panel), BB aims to recover the current state of the PRG given q consecutive outputs from the generator; the advantage term is defined

$$\mathbf{Adv}^{\text{bprg-next}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \Pr\left[\text{BPRG-NEXT}^{\mathsf{BB},q}_{\mathsf{BPRG}} \Rightarrow \mathsf{true} \right] .$$

This is a far more powerful compromise, since achieving it allows BB to predict all of the generator's future outputs. Finally, in game $\text{BPRG-RSEEK}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i, j)$ (right-hand panel),

[2]We sometimes use $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ to refer to the associated PRG $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$ when clear from the context.

BB is given the i^{th} output and index j, and tries to recover the j^{th} output. We define

$$\mathbf{Adv}^{\text{bprg-rseek}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \min_{1 \le i,j, \le q} \Pr\left[\text{BPRG-RSEEK}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i, j) \Rightarrow \mathsf{true}\right] .$$

In [57], Dodis et al. present constructions of BPRGs that are backdoored in the BPRG-NEXT and BPRG-RSEEK senses. However, their construction of a BPRG of the latter type is *not* forward secure.

A question. Notice that for certain parameter settings (namely the case in which $j < i$), game BPRG-RSEEK requires Big Brother to recover an output produced *prior* to that which was compromised. Intuitively, this seems somewhat at odds with forward security, which requires that past outputs remains pseudorandom in the event of a state compromise. This raises the question of whether forward security is inherently at odds with forms of backdooring which allow recovery of past outputs / states. As well as being of intrinsic interest, if such a relation were *true* it would give assurance that forward secure PRGs — and thereby robust PRNGs — withstand certain forms of backdooring.

We explore this question in the remainder of the chapter — and ultimately answer it in the negative — by presenting two forward secure BPRGs which contain a backdoor allowing Big Brother to recover the initial state of the PRG.

5.3.3 Stronger Models and New Constructions for Backdoored PRGs

We begin by defining two new backdooring models for PRGs. As we will see, the stronger of the two notions implies all the backdooring goals introduced in [57].

Backdoored PRG security models. Consider games BPRG-FIRST and BPRG-OUT shown in Figure 5.5. In both games, the BPRG is run with initial state S_0 to produce q outputs $R_1, \ldots, R_q$, and the Big Brother adversary BB is given a particular output R_i along with the backdoor parameter bp. In game BPRG-FIRST, BB is challenged to recover the initial state of the BPRG, S_0, with advantage defined

$$\mathbf{Adv}^{\text{first}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \min_{1 \le i \le q} \Pr\left[\text{BPRG-FIRST}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i) \Rightarrow \mathsf{true}\right] .$$

In game BPRG-OUT, BB is recovered to recover all output values (including the $q - 1$

```
$\text{BPRG-FIRST}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i)$
$(pp, bp) \leftarrow\$ \mathsf{init}$
$S_0 \leftarrow\$ \mathsf{setup}(pp)$
$(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
$S_0^* \leftarrow\$ \mathsf{BB}(pp, bp, i, R_i)$
Return $(S_0 = S_0^*)$
```

```
$\text{BPRG-OUT}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i)$
$(pp, bp) \leftarrow\$ \mathsf{init}$
$S_0 \leftarrow\$ \mathsf{setup}(pp)$
$(R_1, \ldots, R_q) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0))$
$(R_1^*, \ldots, R_q^*) \leftarrow\$ \mathsf{BB}(pp, bp, i, R_i)$
Return $((R_1, \ldots R_q) = (R_1^*, \ldots, R_q^*))$
```

Figure 5.5: Backdoored PRG security games BPRG-FIRST and BPRG-OUT.

unseen outputs), with corresponding advantage term

$$\mathbf{Adv}^{\text{out}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \min_{1 \le i \le q} \Pr\left[\text{BPRG-OUT}^{\mathsf{BB},q}_{\mathsf{BPRG}}(i) \Rightarrow \mathsf{true}\right].$$

5.3.4 Relations and Separations

Before presenting our BPRG constructions, we briefly discuss the relations and separations between the backdooring notions in [57] and our new definitions, BPRG-FIRST and BPRG-OUT.

Relations. We say that one BPRG backdooring notion, BPRG-TYPE, implies another, BPRG-TYPE′, if given any BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ for which $\mathbf{Adv}^{\text{type}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \delta$ we can define a new BPRG $\mathsf{BPRG}' = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB}')$ with similar advantage in game BPRG-TYPE′. Notice that the regular PRG security of BPRG' is precisely that of BPRG, since the induced PRG $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$ is identical in both cases.

Let $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ be a BPRG for which $\mathbf{Adv}^{\text{first}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \delta$. We will show that this implies the existence of BPRGs $\mathsf{BPRG}_i = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB}_i)$ for $i \in [1, 3]$ with similar advantage in BPRG-TYPE for TYPE $\in \{\text{DIST}, \text{NEXT}, \text{RSEEK}\}$ respectively. For each BPRG_i, the Big Brother algorithm BB_i perfectly simulates BB's view of game BPRG-FIRST against BPRG. If BB wins that game by successfully recovering the initial state of the generator, then BB_i exploits this to succeed in his own challenge. This means that BPRG-FIRST implies each of the backdooring notions given in [57] (and is in fact strictly stronger, as we will show later in the section). We formalise this in the following theorem.

Theorem 5.1. *Let* $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ *be a BPRG with parameters* $(n, \ell) \in \mathbb{N}^2$ *for which* $\mathbf{Adv}^{\text{first}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \delta$. *Then there exist BPRGs* $\mathsf{BPRG}_i = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB}_i)$

```
$\mathsf{BB}_1(pp, bp, R_1, \ldots, R_q)$
$i \leftarrow_{\$} [1, q]$
$S_0^* \leftarrow_{\$} \mathsf{BB}(pp, bp, i, R_i)$
$(S_1^*, \ldots, S_q^*) \leftarrow \mathsf{state}^q(\mathsf{next}(pp, S_0^*))$
Return $S_q^*$
```

```
$\mathsf{BB}_2(pp, bp, i, j, R_i)$
$S_0^* \leftarrow_{\$} \mathsf{BB}(pp, bp, i, R_i)$
$(R_1^*, \ldots, R_q^*) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0^*))$
Return $R_j^*$
```

```
$\mathsf{BB}_3(pp, bp, R_1, \ldots, R_q)$
$i \leftarrow_{\$} [1, q]$
$S_0^* \leftarrow_{\$} \mathsf{BB}(pp, bp, i, R_i)$
$(R_1^*, \ldots, R_q^*) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0^*))$
For $i = 1, \ldots, q$
  If $(R_i \neq R_i^*)$
    Return 1
Return 0
```

Figure 5.6: Big Brother algorithms for Theorem 5.1.

for $i \in [1, 3]$ such that: **(1)** $\mathbf{Adv}^{\text{bprg-next}}_{\mathsf{BPRG}_1}(\mathsf{BB}_1, q) \geq \delta$*;* **(2)** $\mathbf{Adv}^{\text{bprg-rseek}}_{\mathsf{BPRG}_2}(\mathsf{BB}_2, q) \geq \delta$*; and* **(3)** $\mathbf{Adv}^{\text{bprg-dist}}_{\mathsf{BPRG}_3}(\mathsf{BB}_3, q) \geq \delta - 2^{-(q-1)\cdot\ell}$.

Proof. For **(1)**, consider the Big Brother algorithm BB_1 shown in the left-hand panel of Figure 5.6. Notice that in game BPRG-NEXT against BPRG_1, BB_1 perfectly simulates BB's view of the experiment in game BPRG-FIRST(i) against BPRG. Moreover, notice that if BB returns the correct initial state S_0^*, then BB_1 will in turn output the correct state S_q^* with probability one. The BPRG-FIRST success probability of BPRG then implies that

$$\mathbf{Adv}^{\text{bprg-next}}_{\mathsf{BPRG}_1}(\mathsf{BB}_1, q) \geq \Pr\left[\text{BPRG-FIRST}(i)^{\mathsf{BB},q}_{\mathsf{BPRG}} \Rightarrow \mathsf{true}\right] \geq \delta\ .$$

For **(2)**, consider BB_2 shown in the middle panel of Figure 5.6. Again it is the case that BB_2 in game BPRG-RSEEK against BPRG_2 perfectly simulates BPRG-FIRST(i) for BB. As before, conditioned on BB returning the correct value of S_0^*, it follows that BB_2 will return the correct output R_j with probability one, and so

$$\mathbf{Adv}^{\text{bprg-rseek}}_{\mathsf{BPRG}_2}(\mathsf{BB}_2, q) \geq \Pr\left[\text{BPRG-FIRST}(i)^{\mathsf{BB},q}_{\mathsf{BPRG}} \Rightarrow \mathsf{true}\right] \geq \delta\ .$$

Finally consider BB_3, shown in the left-hand panel of Figure 5.6. In the case that the challenge bit for BB_3 in game BPRG-DIST is $b = 0$ (and so BB_3 receives real outputs in their challenge) an analogous argument to those used above implies that BB_3 perfectly simulates BPRG-FIRST(i) for BB. Moreover, BB_3 will output one with probability zero if BB succeeds in his game, and so it follows that $\Pr[\,\mathsf{BB}_3 \Rightarrow 1 \mid b = 0\,] \leq (1 - \delta)$. On the other hand, if $b = 1$ then the outputs $(R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_q)$ which must match those generated using S_0^* in order for BB_3 to *not* return 1 are random and independent of BB's view of the experiment, implying that $\Pr[\,\mathsf{BB}_3 \Rightarrow 1 \mid b = 1\,] \geq 1 - 2^{-(q-1)\cdot\ell}$. Putting this together gives

$$\mathbf{Adv}^{\text{bprg-dist}}_{\mathsf{BPRG}_3}(\mathsf{BB}_3, q) \geq (1 - 2^{-(q-1)\cdot\ell}) - (1 - \delta) = \delta - 2^{-(q-1)\cdot\ell}\ .$$

□

This proves that BPRG-FIRST implies each of the backdooring notions in [57]. It is also the case that BPRG-FIRST implies our second new backdooring notion, BPRG-OUT. To see this, notice that since the initial state of a PRG determines all of its output, it is clear that for any BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ for which $\mathbf{Adv}^{\text{first}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \delta$, we can define a BPRG $\mathsf{BPRG}' = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB}')$ where BB' passes his input to BB, computes $(R_1^*, \ldots, R_q^*) \leftarrow \mathsf{out}^q(\mathsf{next}(pp, S_0^*))$ where S_0^* is the state output by BB, and returns these outputs to his challenger. It is straightforward to verify that $\mathbf{Adv}^{\text{bprg-out}}_{\mathsf{BPRG}'}(\mathsf{BB}', q) \geq \delta$.

Separations. We will now construct separating examples to show that BPRG-FIRST is in fact strictly stronger than all the other PRG backdooring models considered in this thesis. We describe the separating example for the BPRG-OUT case; the other cases are similar. Let BPRG be a BPRG with state space $\mathcal{S}$ such that $\mathbf{Adv}^{\text{bprg-out}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = \delta$. We define a modified BRPG $\mathsf{BPRG}' = (\mathsf{init}', \mathsf{setup}', \mathsf{next}', \mathsf{BB})$ with state space $\mathcal{S}' = \mathcal{S} \cup \{0,1\}^n$ such that $\mathsf{init}' = \mathsf{init}$, setup' computes $S_0 \leftarrow\!\!\$\ \mathsf{setup}(pp)$, chooses $d \leftarrow\!\!\$\ \{0,1\}^n$, and returns $S_0' = S_0 \,\|\, d$, and next' on input $(pp, S \,\|\, d)$ simply computes $(R, S') \leftarrow \mathsf{next}(pp, S)$ and returns $(R, S' \,\|\, d)$. In particular, notice that the output produced by BPRG' is identical to that of BPRG, and so it is easy to see that the modified BPRG' has $\mathbf{Adv}^{\text{bprg-out}}_{\mathsf{BPRG}'}(\mathsf{BB}, q) = \delta$ also. However, since the output produced by BPRG' is entirely independent of the random state component d, for any candidate BPRG-FIRST BPRG, $\mathsf{BPRG}'' = (\mathsf{init}', \mathsf{setup}', \mathsf{next}', \mathsf{BB}'')$, it must hold that $\mathbf{Adv}^{\text{bprg-first}}_{\mathsf{BPRG}''}(\mathsf{BB}'', q) \leq 2^{-n}$ since BB'' can do no better than guess the n unknown bits of state d.

Comparison to BPRG-OUT. The above arguments readily extend to show that any BPRG-OUT BPRG is also backdoored in a BPRG-RSEEK and BPRG-DIST sense. However, the separating example used to imply that BPRG-OUT does not imply BPRG-FIRST additionally implies that BPRG-OUT does not imply BPRG-NEXT (where recall that the latter notion requires recovery of the current state of the PRG rather than the first).

Discussion. From Big Brother's perspective, in most practical attack scenarios the ability to compute all unseen output (as in BPRG-OUT) is as useful as being able to compute the initial state (as in BPRG-FIRST), since it is the output values of the BPRG that will be consumed in applications. This makes the BPRG-OUT notion a natural and powerful target for constructions of BPRGs. That said, in the following sections we will go one

```
init
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow\$ \mathsf{Gen}$
$pp \leftarrow (pk, \mathsf{PK})$
$bp \leftarrow (sk, \mathsf{SK})$
Return $(pp, bp)$

setup$(pp)$
$(pk, \mathsf{PK}) \leftarrow pp$
$S_0 \leftarrow\$ \{0,1\}^n$
Return $S_0$
```

```
next$(pp, S)$
$(pk, \mathsf{PK}) \leftarrow pp$
$r \leftarrow \mathsf{H}(S)$
$R \leftarrow \mathsf{Enc}(pk, S; r)$
$S' \leftarrow \mathsf{F}_{\mathsf{PK}}(S)$
Return $(R, S')$
```

```
BB$(pp, bp, i, R_i)$
$(sk, \mathsf{SK}) \leftarrow bp$
$S^*_{i-1} \leftarrow \mathsf{Dec}(sk, R_i)$
$S^*_0 \leftarrow \mathsf{F}_{\mathsf{SK}}^{-(i-1)}(S^*_{i-1})$
Return $S^*_0$
```

Figure 5.7: Construction of a forward secure BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ from a TDP family $\mathsf{TDP} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$, an IND\$-CPA-secure PKE scheme $\mathsf{PKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$, and a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{R}$.

better and present two constructions of forward secure BPRGs which achieve the strongest BPRG-FIRST backdooring goal with probability one.

5.3.5 Forward Secure BPRGs in the Random Oracle Model

We now present the first of our two forward secure BPRG-FIRST backdoored PRGs. Consider the construction $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ shown in Figure 5.7. The construction uses as ingredients an IND\$-CPA secure PKE scheme PKE, a family of TDPs TDP, and a hash function H with range equal to the coin space of PKE. For its security analysis, we model H as a random oracle.

Construction overview. The scheme is reminiscent of the 'Encrypt-with-Hash'(EwH) transform for constructing deterministic encryption schemes from [14]. Parameter generation via init generates a public / secret key pair (pk, sk) for the PKE scheme, and public / trapdoor parameters for the TDP $(\mathsf{PK}, \mathsf{SK})$. The public portions of these form the public parameters of the BPRG, while their secret counterparts form Big Brother's backdoor parameter. An initial BPRG state is generated by sampling from the domain of the TDP, $S_0 \leftarrow\$ \{0,1\}^n$.

With this in place, the next algorithm of the BPRG produces each output by encrypting the current state S under pk using randomness produced by hashing S, $r_i \leftarrow \mathsf{H}(S_{i-1})$, $C \leftarrow \mathsf{Enc}(pk, S_{i-1}; r_i)$. It then updates the state by applying TDP, $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$. The IND\$-CPA security of PKE ensures the resulting outputs are pseudorandom, while the one-wayness of TDP implies that a regular forward security attacker is unable to recover past states. However, given $bp = (sk, \mathsf{SK})$ and an arbitrary output R_i, BB can simply decrypt R_i using $\mathsf{Dec}(sk, \cdot)$ to recover the underlying state S_{i-1}. Using the trapdoor SK to compute the $(i-1)^{\text{st}}$ pre-image of S_{i-1} then allows BB to recover the initial state $S_0 = \mathsf{F}_{\mathsf{SK}}^{-(i-1)}(S_{i-1})$ with probability one. We formalise this in the theorem below.

Analysis. Our analysis will be in the ROM, in which all adversaries and the challenger are given access to the random oracle H. However, we do not give the algorithms of TDP and PKE access to H. To see why this restriction is necessary, let PKE be an IND\$-CPA PKE scheme and H a hash function, and define PKE' to be identical to PKE except, on input $(pk, M; r)$, Enc checks if $r = \mathsf{H}(M)$, outputs M if the relation holds and $C = \mathsf{Enc}(pk, M; r)$ otherwise. It is straightforward to verify that PKE' is still IND\$-CPA and that Dec can be modified to ensure correctness provided ciphertexts and messages can clearly be distinguished by their length. However, the scheme cannot be used to instantiate the EwH transform nor our construction. It seems possible that techniques similar to those used in [16] could be used to develop a standard model variant of the construction, thereby avoiding this idealised and admittedly unnatural restriction; we leave formalising this to future work.

Theorem 5.2. *Let* $\mathsf{TDP} = (\mathsf{Gen}, \mathsf{F}, \mathsf{F}^{-1})$ *be a family of TDPs with domain* $\{0,1\}^n$*. Let* $\mathsf{PKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ *be a PKE scheme with coin space* $\mathcal{R}$*, message space* $\mathcal{M} \supseteq \{0,1\}^n$*, and ciphertext space* $\mathcal{C} \subseteq \{0,1\}^*$*, for which* $\mathsf{len}_{\mathsf{PKE}}(n) = \ell$*. Let* $\mathsf{H} : \{0,1\}^* \to \mathcal{R}$ *be a hash function which we model as a random oracle. Consider the BPRG* $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ *built from these components as shown in Figure 5.7, with output length* ℓ *and state space* $\mathcal{S} = \{0,1\}^n$*. Then* BPRG *is such that*

$$\mathbf{Adv}_{\mathsf{BPRG}}^{\text{game-first}}(\mathsf{BB}, q) = 1 \ .$$

Moreover, for any attacker $\mathcal{A}$ *in game* $\text{PRG-FWD}^* \in \{\text{PRG-FWD}, \overline{\text{PRG-FWD}}\}$ *against* $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$*, who runs in time* T *and makes* q_{H} *queries to the random oracle* H*, there exist attackers* $\mathcal{B}_1, \mathcal{B}_2$*, and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}_{\mathsf{PRG}}^{\text{prg-fwd}^*}(\mathcal{A}, q) \leq \mathbf{Adv}_{\mathsf{TDP}}^{\text{tdp-inv}}(\mathcal{B}_1) + \mathbf{Adv}_{\mathsf{TDP}}^{\text{tdp-inv}}(\mathcal{B}_2) + 2 \cdot \mathbf{Adv}_{\mathsf{PKE}}^{\text{ind\$-cpa}}(\mathcal{C}, q) \ .$$

$\mathcal{B}_1$ *runs in time* $\mathcal{O}(q)$*, and* $\mathcal{B}_2$ *runs in time* T *plus an* $\mathcal{O}((q_{\mathsf{H}} + 1) \cdot q + q_{\mathsf{H}})$ *overhead.* $\mathcal{C}$ *runs in time* T *plus an* $\mathcal{O}(q + q_{\mathsf{H}})$ *overhead.*

Proof. The correctness of PKE immediately implies that $\mathbf{Adv}_{\mathsf{BPRG}}^{\text{first}}(\mathsf{BB}, q) = 1$. It remains to bound the security of $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$ against a standard PRG adversary $\mathcal{A}$ in game PRG-FWD. We argue by a series of game hops, shown in Figure 5.8. Our main line of argument is given in games $G_0 - G_5$. Games G_1^* and G_3^* lie perpendicular to the other games and are used to bound certain bad events.

We may without loss of generality assume that $\mathcal{A}$ never repeats a query to the random oracle H. We begin by defining game G_0, which is a re-writing of game PRG-FWD for PRG with challenge bit $b = 0$ using a lazy-sampled random oracle. We also set a number of flags, but these do not affect the outcome of the game. It follows that

$$\Pr\left[\, G_0 \Rightarrow 1 \,\right] = \Pr\left[\, \text{PRG-FWD}_{\mathsf{PRG}}^{\mathcal{A},q} \Rightarrow 1 \mid b = 0 \,\right] .$$

Next we define game G_1, which is identical to G_0 except we change the way in which the random oracle H responds to queries. Namely, if H is queried more than once on a state $S_0, \ldots, S_{q-1}$ (indicated by the set $\mathcal{Q}$) in G_1, then H responds with an independent random string instead of the value previously set. These games run identically until the flag bad is set, and so the Fundamental Lemma of Game Playing implies that:

$$\left|\Pr\left[\, G_0 \Rightarrow 1 \,\right] - \Pr\left[\, G_1 \Rightarrow 1 \,\right]\right| \le \Pr\left[\, \mathsf{bad} = 1 \text{ in } G_1 \,\right] .$$

To bound this probability, we make a distinction between: **(1)** bad being set during the challenge computation if $S_i = S_j$ for some $0 \le i < j \le q-1$, an event indicated by setting the flag coll; and **(2)** bad being set after the challenge computation by $\mathcal{A}$ submitting a query of the form S_i for $i \in [0, q-1]$ to H, an event indicated by setting the flag guess. Moreover, notice that bad is set if and only if at least one of coll and guess is set. A union bound then implies that:

$$\begin{aligned} \Pr\left[\, \mathsf{bad} = 1 \text{ in } G_1 \,\right] &= \Pr\left[\, \mathsf{coll} \vee \mathsf{guess} = 1 \text{ in } G_1 \,\right] \\ &\le \Pr\left[\, \mathsf{coll} = 1 \text{ in } G_1 \,\right] + \Pr\left[\, \mathsf{guess} = 1 \text{ in } G_1 \,\right] . \end{aligned} \tag{5.1}$$

To bound the probability of coll being set we define game G_1^*, which is identical to G_1 except we change the way in which the initial state S_0 is computed. In more detail, instead of setting $S_0 \leftarrow_\$ \{0,1\}^n$ as in G_1, we now choose $S_0' \leftarrow_\$ \{0,1\}^n$ and set $S_0 \leftarrow \mathsf{F}_{\mathsf{PK}}(S_0')$. Since F_{PK} is a permutation, applying it to a point sampled uniformly from domain $\{0,1\}^n$ is equivalent to sampling directly from $\{0,1\}^n$, and so S_0 is identically distributed in both games. As such, this is a purely syntactic change, and so $\Pr\left[\, G_1 \Rightarrow 1 \,\right] = \Pr\left[\, G_1^* \Rightarrow 1 \,\right]$ and $\Pr\left[\, \mathsf{coll} = 1 \text{ in } G_1 \,\right] = \Pr\left[\, \mathsf{coll} = 1 \text{ in } G_1^* \,\right]$.

We now bound the probability that coll is set in G_1^*. We claim that there exists an attacker

$\mathcal{B}_1$ in game TDP-INV against TDP such that

$$\Pr[\,\mathsf{coll} = 1 \text{ in } G_1^*\,] \le \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_1)\ , \tag{5.2}$$

and moreover, $\mathcal{B}_1$ runs in time $\mathcal{O}(q)$. To see this, let $\mathcal{B}_1$ proceed as follows. On input (PK, y), $\mathcal{B}$ sets $S_0 \leftarrow y$. $\mathcal{B}_1$ then generates states by setting $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$ for $i \in [1, q]$. Notice that this perfectly simulates the generation of states in G_1^*. If during this process $\mathcal{B}$ finds states $S_i = S_j$ for $0 \le i < j \le q - 1$ — precisely the event that would set the flag coll in G_1^* — then $\mathcal{B}$ halts and outputs S_{j-i-1} in his game. If no such collision is found, $\mathcal{B}_1$ aborts the simulation and outputs a random point from $\{0,1\}^n$. Notice that, since F is a permutation, it must be the case that S_{j-i-1} is the (unique) pre-image of $\mathcal{B}$'s challenge $y = S_0$, and so conditioned on such a collision occurring $\mathcal{B}$ will win his game with probability one. Since $\mathcal{B}_1$ invokes F at most q times it is straightforward to verify that $\mathcal{B}_1$ runs in time $\mathcal{O}(q)$, and since games G_1 and G_1^* are equivalent this in turn implies that $\Pr[\,\mathsf{coll} = 1 \text{ in } G_1\,] \le \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_1)$. We will return to bound $\Pr[\,\mathsf{guess} = 1 \text{ in } G_1\,]$ and thereby complete our analysis of equation (5.1) in a later game.

We now return to our main series of game hops. We next define game G_2, which is identical to G_1 except rather than generating the coins used for encryptions via $r_i \leftarrow \mathsf{H}(S_{i-1})$ for $i = 1, \ldots, q$, we instead simply sample coins randomly from the coin space of the scheme $r_i \leftarrow_{\$} \mathcal{R}$. Since in G_1 each H query on $S_i \in \mathcal{Q}$ is answered with an independent random string drawn from $\mathcal{R}$ (recall the modification to H in G_1), this is a purely syntactic change and so $\Pr[\,G_1 \Rightarrow 1\,] = \Pr[\,G_2 \Rightarrow 1\,]$ and $\Pr[\,\mathsf{guess} = 1 \text{ in } G_1\,] = \Pr[\,\mathsf{guess} = 1 \text{ in } G_2\,]$.

Next we define G_3, which is identical to G_2 except instead of computing outputs via $R_i \leftarrow \mathsf{Enc}(pk, S_{i-1}; r_i)$ for $r_i \leftarrow_{\$} \mathcal{R}$, we instead set $R_i \leftarrow_{\$} \{0,1\}^\ell$ for $i \in [1, q]$. We claim that there exists an attacker $\mathcal{C}_1$ in game IND\$-CPA against PKE, running in time T plus an $\mathcal{O}(q + q_{\mathsf{H}})$ overhead and who makes q queries to his real-or-random oracle, such that

$$|\Pr[\,G_2 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| = \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}_1, q)\ .$$

To see this, let $\mathcal{C}_1$ be the attacker who proceeds as follows. On input pk, $\mathcal{C}_1$ generates a public parameter / trapdoor pair for TDP via $(\mathsf{PK}, \mathsf{SK}) \leftarrow_{\$} \mathsf{Gen}$, and sets $pp \leftarrow (pk, \mathsf{PK})$. $\mathcal{C}_1$ chooses an initial state $S_0 \leftarrow_{\$} \{0,1\}^n$. For $i \in [1, q]$, $\mathcal{C}_1$ generates a challenge output / state by querying S_{i-1} to his real-or-random encryption oracle, setting R_i equal to the response, and updating the state via $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$. Finally, $\mathcal{C}_1$ passes $(pp, R_1, \ldots, R_q, S_q)$ to $\mathcal{A}$, and simulates H for $\mathcal{A}$ by returning an independent random string drawn from $\mathcal{R}$ in response to each query. When $\mathcal{A}$ outputs a bit, $\mathcal{C}_1$ outputs the same bit to his challenger. If $\mathcal{C}_1$ receives real encryptions from his real-or-random oracle then this perfectly simulates G_2; otherwise it perfectly simulates G_3. It follows that

$$\begin{aligned}\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}_1, q) &= \Big|\Pr\Big[\mathcal{C}_1^{\mathsf{Enc}(pk,\cdot)}(pk) \Rightarrow 1 \;:\; (pk, sk) \leftarrow\!\!\$\, \mathsf{Key}\Big] \\ &\qquad\qquad - \Pr\Big[\mathcal{C}_1^{\$(\cdot)}(pk) \Rightarrow 1 \;:\; (pk, sk) \leftarrow\!\!\$\, \mathsf{Key}\Big]\Big| \\ &= |\Pr[\, G_2 \Rightarrow 1\,] - \Pr[\, G_3 \Rightarrow 1\,]\,.\end{aligned}$$

It is straightforward to verify that $\mathcal{C}_1$ makes q oracle queries, and runs in the same time as $\mathcal{A}$ plus an $\mathcal{O}(q + q_{\mathsf{H}})$ overhead to account for computing the challenge and simulating the H queries, thereby proving the claim.

Moreover, we claim that there exists an attacker $\mathcal{C}_2$ in game IND\$-CPA against PKE who makes q queries to his real-or-random oracle and runs in the same time as $\mathcal{C}_1$, such that

$$\Pr[\,\mathsf{guess} = 1 \text{ in } G_2\,] \leq \Pr[\,\mathsf{guess} = 1 \;\text{ in } G_3\,] + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}_2, q)\,. \qquad (5.3)$$

We define $\mathcal{C}_2$ to proceed identically to $\mathcal{C}_1$ up to and including the point $\mathcal{C}_1$ passes the challenge to $\mathcal{A}$. At this point, $\mathcal{C}_2$ starts to simulate H for $\mathcal{A}$ by returning an independent random string in response to each of $\mathcal{A}$'s queries as before. However, if one of $\mathcal{A}$'s q_{H} queries causes guess to be set — that is to say, $\mathcal{A}$ queries one of the states in $\mathcal{Q} = \{S_i \;:\; i \in [0, q{-}1]\}$ to H, then $\mathcal{A}$ halts and outputs 1 to his own challenger. If this event does not occur, then at the end of the simulation he returns 0. Notice that $\mathcal{C}_2$ outputs 1 if and only if the flag guess is set. Moreover, notice that if $\mathcal{C}_2$ receives real encryptions from his oracle then this perfectly simulates G_2; otherwise it perfectly simulates G_3. It follows that

$$\begin{aligned}\mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}_2, q) &= \Big|\Pr\Big[\mathcal{C}_2^{\mathsf{Enc}(pk,\cdot)}(pk) \Rightarrow 1 \;:\; (pk, sk) \leftarrow\!\!\$\, \mathsf{Key}\Big] \\ &\qquad\qquad - \Pr\Big[\mathcal{C}_2^{\$(\cdot)}(pk) \Rightarrow 1 \;:\; (pk, sk) \leftarrow\!\!\$\, \mathsf{Key}\Big]\Big| \\ &= |\Pr[\,\mathsf{guess} = 1 \;\text{ in } G_2\,] - \Pr[\,\mathsf{guess} = 1 \;\text{ in } G_3\,]|\,.\end{aligned}$$

An analogous argument to that used above verifies $\mathcal{C}_2$'s run time and query budget. Rearranging and invoking the triangle inequality then proves the claim.

We now bound $\Pr[\,\mathsf{guess} = 1 \text{ in } G_3\,]$. We first define another perpendicular game G_3^* in which we modify the way in which the sequence of states $S_0, \ldots, S_q$ are computed. Namely, instead of choosing $S_0 \leftarrow\!\!\$\, \{0,1\}^n$ and setting $S_i \leftarrow \mathsf{F}^i_{\mathsf{PK}}(S_0)$ for $i \in [1,q]$, we instead choose $S_{q-1} \leftarrow\!\!\$\, \{0,1\}^n$, set $S_q = \mathsf{F}_{\mathsf{PK}}(S_{q-1})$, and $S_i = \mathsf{F}^{-q+i}_{\mathsf{PK}}(S_q)$ for $i = 0, \ldots, q-2$. (Note that the challenger knows the trapdoor SK for the TDP, and so can compute the required pre-images). Again, since TDP is a permutation, the sequence of states $S_0, \ldots, S_q$ is identically distributed in both games. As such, this is a purely syntactic change and so $\Pr[\, G_3 \Rightarrow 1\,] = \Pr[\, G_3^* \Rightarrow 1\,]$ and $\Pr[\, \mathsf{guess} = 1 \text{ in } G_3\,] = \Pr[\, \mathsf{guess} = 1 \text{ in } G_3^*\,]$.

With this in place, we claim that there exists an attacker $\mathcal{B}_2$ in game TDP-INV against TDP such that

$$\Pr[\, \mathsf{guess} = 1 \text{ in } G_3^*\,] \leq \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_2)\;,$$

where $\mathcal{B}_2$ runs in the same time as $\mathcal{A}$ plus an $\mathcal{O}((q_{\mathsf{H}}+1)\cdot q + q_{\mathsf{H}})$ overhead. Let $\mathcal{B}_2$ be the attacker who proceeds as follows. On input (PK, y), $\mathcal{B}_2$ begins by generating a public / secret key pair for the PKE scheme via $(pk, sk) \leftarrow\!\!\$\, \mathsf{Key}$ and sets $pp \leftarrow (pk, \mathsf{PK})$. $\mathcal{B}_2$ chooses $R_i \leftarrow\!\!\$\, \{0,1\}^\ell$ for $i \in [1,q]$, sets $S_q \leftarrow y$, and passes $(pp, R_1, \ldots, R_q, S_q)$ to $\mathcal{A}$. $\mathcal{B}_2$ begins simulating H for $\mathcal{A}$ by answering each query with an independent random string. Since F_{PK} is a permutation, there is a unique sequence of states / pre-images $S_0, \ldots, S_{q-1}$ which correspond to $S_q = y$, and these make up the set $\mathcal{Q} = \{S_{q-i} = \mathsf{F}^{-i}_{\mathsf{PK}}(S_q) \;:\; i \in [1,q]\}$. The flag guess is set if and only if $\mathcal{A}$ queries $S_i \in \mathcal{Q}$ to H. Now for a good TDP, $\mathcal{Q}$ will not be immediately known to $\mathcal{B}_2$ (since computing any point in $\mathcal{Q}$ corresponds to inverting y). However, letting $X_1, \ldots, X_{q_{\mathsf{H}}}$ denote the set of H queries made by $\mathcal{A}$, $\mathcal{B}_2$ can test whether each X_j lies in $\mathcal{Q}$ as follows. For each H query X_j, $\mathcal{B}_2$ uses PK to compute $\nu^i_j = \mathsf{F}^i_{\mathsf{PK}}(X_j)$ for $i \in [1,q]$, and checks if $\nu^i_j = S_q$ (where recall $S_q = y$ is the value $\mathcal{B}_2$ is trying to invert). If such a relation holds, then this implies that $X_j = \mathsf{F}^{-i}_{\mathsf{SK}}(S_q) = S_{q-i} \in \mathcal{Q}$, and so such a query causes guess to be set with probability one. If such a query is made, $\mathcal{B}_2$ then halts the simulation and outputs $\nu^{i-1}_j = S_{q-1}$ to his own challenger. Since F_{PK} is a permutation, this will be equal to the unique preimage of $\mathcal{B}_2$'s challenge $S_q = y$ with probability one, and so $\mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_2) \geq \Pr[\, \mathsf{guess} = 1 \text{ in } G_3^*\,]$. Moreover, $\mathcal{B}_2$ runs in the same time as $\mathcal{A}$ with an $\mathcal{O}((q_{\mathsf{H}}+1)\cdot q + q_{\mathsf{H}})$ overhead, which accounts for the q F_{PK} computations $\mathcal{B}_2$ makes for each of $\mathcal{A}$'s q_{H} queries, plus the $\mathcal{O}(q + q_{\mathsf{H}})$ overhead required to compute the challenge and simulate the oracle queries.

We may now put this altogether to finally bound equation (5.1). From (5.3) and the previous line of argument we have that $\Pr[\, \mathsf{guess} = 1 \text{ in } G_1\,] \leq \Pr[\, \mathsf{guess} = 1 \text{ in } G_3^*\,] + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}_2, q)$, and by (5.2) it holds that $\Pr[\, \mathsf{coll} = 1 \text{ in } G_1\,] \leq \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_1)$. Substituting into (5.1) then yields

$$\Pr[\,\mathsf{bad} = 1 \text{ in } G_1\,] \leq \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_1) + \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_2) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}_2, q)\ .$$

With this in place, we again return to our main line of game hops, and define G_4 to be identical to G_3 except we remove the now redundant coins $r_i \leftarrow\!\!\$\ \mathcal{R}$ sampled during the challenge computation. Since these coins are not used in either game, this is clearly a syntactic change and so $\Pr[\,G_3 \Rightarrow 1\,] = \Pr[\,G_4 \Rightarrow 1\,]$. We then define G_5 to be identical to G_4, except we return the random oracle to answer consistently on the points in $\mathcal{Q}$. Since these points are not queried to H by the challenger in either game, and by assumption $\mathcal{A}$ never repeats a query, in neither game will these points be queried to H more than once. As such the games are equivalent, and $\Pr[\,G_4 \Rightarrow 1\,] = \Pr[\,G_5 \Rightarrow 1\,]$.

Moreover, it is straightforward to verify that G_5 is identical to PRG-FWD for PRG with challenge bit $b = 1$, and so

$$\Pr[\,G_5 \Rightarrow 1\,] = \Pr\left[\,\text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}} \Rightarrow 1 \mid b = 1\,\right]\ .$$

Combining the above observations via a standard argument implies that there exist adversaries $\mathcal{B}_1, \mathcal{B}_2$, and $\mathcal{C}$ with the claimed query budgets such that

$$\begin{aligned}\mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}}(\mathcal{A}, q) &= \left|\Pr\left[\,\text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}} \Rightarrow 1 \mid b = 0\,\right] - \Pr\left[\,\text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}} \Rightarrow 1 \mid b = 1\,\right]\right| \\ &\leq \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_1) + \mathbf{Adv}^{\text{tdp-inv}}_{\mathsf{TDP}}(\mathcal{B}_2) + 2 \cdot \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{PKE}}(\mathcal{C}, q)\ .\end{aligned}$$

This concludes the proof of PRG-FWD security for PRG. We now claim that the same bound holds for $\overline{\text{PRG-FWD}}$ security. Looking ahead, we shall utilise this security property in Section 5.4 to show that BPRG can be used to instantiate a construction of a simple backdoored PRNG. To this end, we define game G_6 to be identical to G_5 except we overwrite the final state $S_q \leftarrow \mathsf{F}^q_{\mathsf{PK}}(S_0)$ for $S_0 \leftarrow\!\!\$\ \{0,1\}^n$ with a freshly sampled state $S_q \leftarrow\!\!\$\ \{0,1\}^n$. Since F is a permutation and the intermediate states $S_i \leftarrow \mathsf{F}^i_{\mathsf{PK}}(S_0)$ for $i \in [0, q-1]$ are not used in either game other than to compute S_q, it follows that both games are identically distributed. Moreover, it is straightforward to verify that game G_6 is equivalent to game $\overline{\text{PRG-FWD}}$ against PRG with challenge bit $b = 1$. Since G_0 was equivalent to game $\overline{\text{PRG-FWD}}$ with challenge bit $b = 0$, extending the above argument to include the hop to G_6 implies the $\overline{\text{PRG-FWD}}$ security of PRG also, thereby concluding the proof. $\square$

```
proc. // main $G_0, G_1$
$\mathsf{chal\text{-}set} \leftarrow 0$ ; $\mathcal{Q} \leftarrow \emptyset$
$(pk, sk) \leftarrow_\$ \mathsf{Key}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow_\$ \mathsf{Gen}$
$pp \leftarrow (pk, \mathsf{PK})$
$bp \leftarrow (sk, \mathsf{SK})$
$S_0 \leftarrow_\$ \{0,1\}^n$
For $i = 1, \ldots, q$
    $r_i \leftarrow \mathsf{H}(S_{i-1})$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{S_{i-1}\}$
    $R_i \leftarrow \mathsf{Enc}(pk, S_{i-1}; r_i)$
    $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$
$\mathsf{chal\text{-}set} \leftarrow 1$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. // main $G_1^*$
$\mathsf{chal\text{-}set} \leftarrow 0$ ; $\mathcal{Q} \leftarrow \emptyset$
$(pk, sk) \leftarrow_\$ \mathsf{Key}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow_\$ \mathsf{Gen}$
$pp \leftarrow (pk, \mathsf{PK})$
$bp \leftarrow (sk, \mathsf{SK})$
$S_0' \leftarrow_\$ \{0,1\}^n$
$S_0 \leftarrow \mathsf{F}_{\mathsf{PK}}(S_0')$
For $i = 1, \ldots, q$
    $r_i \leftarrow \mathsf{H}(S_{i-1})$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{S_{i-1}\}$
    $R_i \leftarrow \mathsf{Enc}(pk, S_{i-1}; r_i)$
    $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$
$\mathsf{chal\text{-}set} \leftarrow 1$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. // main $G_2, \boxed{G_3}$
$\mathsf{chal\text{-}set} \leftarrow 0$ ; $\mathcal{Q} \leftarrow \emptyset$
$(pk, sk) \leftarrow_\$ \mathsf{Key}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow_\$ \mathsf{Gen}$
$pp \leftarrow (pk, \mathsf{PK})$
$bp \leftarrow (sk, \mathsf{SK})$
$S_0 \leftarrow_\$ \{0,1\}^n$
For $i = 1, \ldots, q$
    $r_i \leftarrow_\$ \mathcal{R}$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{S_{i-1}\}$
    $R_i \leftarrow \mathsf{Enc}(pk, S_{i-1}; r_i)$
    $\boxed{R_i \leftarrow_\$ \{0,1\}^\ell}$
    $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$
$\mathsf{chal\text{-}set} \leftarrow 1$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. // main $G_3^*$
$\mathsf{chal\text{-}set} \leftarrow 0$ ; $\mathcal{Q} \leftarrow \emptyset$
$(pk, sk) \leftarrow_\$ \mathsf{Key}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow_\$ \mathsf{Gen}$
$pp \leftarrow (pk, \mathsf{PK})$
$bp \leftarrow (sk, \mathsf{SK})$
$S_{q-1} \leftarrow_\$ \{0,1\}^n$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup S_{q-1}$
$S_q \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{q-1})$
For $i = 0, \ldots, q-2$
    $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}^{-q+i}(S_q)$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{S_i\}$
For $i = 1, \ldots, q$
    $r_i \leftarrow_\$ \mathcal{R}$
    $R_i \leftarrow_\$ \{0,1\}^\ell$
$\mathsf{chal\text{-}set} \leftarrow 1$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. // main $G_4, G_5, \boxed{G_6}$
$\mathsf{chal\text{-}set} \leftarrow 0$ ; $\mathcal{Q} \leftarrow \emptyset$
$(pk, sk) \leftarrow_\$ \mathsf{Key}$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow_\$ \mathsf{Gen}$
$pp \leftarrow (pk, \mathsf{PK})$
$bp \leftarrow (sk, \mathsf{SK})$
$S_0 \leftarrow_\$ \{0,1\}^n$
For $i = 1, \ldots, q$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{S_{i-1}\}$
    $R_i \leftarrow_\$ \{0,1\}^\ell$
    $S_i \leftarrow \mathsf{F}_{\mathsf{PK}}(S_{i-1})$
$\mathsf{chal\text{-}set} \leftarrow 1$
$\boxed{S_q \leftarrow_\$ \{0,1\}^n}$
$b^* \leftarrow_\$ \mathcal{A}^{\mathsf{H}}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. $\mathsf{H}(X)$ // $\boxed{G_0}, G_1, \ldots, G_4, \boxed{G_5}$
$Y \leftarrow_\$ \mathcal{R}$
If $X \in \mathcal{Q}$
    $\mathsf{bad} \leftarrow \mathsf{true}$
    If $\mathsf{chal\text{-}set} = 0$ then $\mathsf{coll} \leftarrow \mathsf{true}$
    Else $\mathsf{guess} \leftarrow \mathsf{true}$
    $\boxed{Y \leftarrow \mathsf{H}[X]}$
$\mathsf{H}[X] \leftarrow Y$
Return $Y$
```

Figure 5.8: Games for proof of Theorem 5.2.

5.3.6 Standard Model Forward Secure BPRGs from Reverse Re-Randomisable Encryption

We now present our second BPRG construction, in which we dispense with the ROM and the use of trapdoor permutations at the expense of requiring the IND\$-CPA-secure PKE scheme to be reverse re-randomisable (see Definition 5.3). Such PKE schemes can be instantiated in the standard model using the encoded variant of ElGamal sketched in Section 5.2.

Consider the BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ shown in Figure 5.9. The scheme is based on a (non-backdoored) forward secure PRG, $\mathsf{PRG}' = (\mathsf{init}', \mathsf{setup}', \mathsf{next}')$, which we will augment to embed a backdoor using a reverse re-randomisable PKE scheme $\mathsf{rrPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$. Looking ahead, we will take a similar approach of modifying a robust PRNG using a re-randomisable PKE scheme to construct a robust backdoored

PRNG in Section 5.4. To simplify the presentation of the construction, we write (t, s) to denote output / state pairs produced by PRG', to distinguish these from the outputs and states produced by BPRG.

Construction overview. The public and backdoor parameters of BPRG are defined to be $(pp, bp) = ((pk, pp'), sk)$, where pp' is a public parameter for the underlying PRG PRG', and (pk, sk) are a public / secret key pair for rrPKE. The initial state of BPRG is set to (s_0, C_0), where s_0 is an initial state for PRG' and $C_0 \leftarrow_{\$} \mathsf{Enc}(pk, s_0)$ is an encryption of that state under pk.

On input $(pp, (s, C))$, next first applies next' to s to generate an output / state pair $(t, s') \leftarrow \mathsf{next}'(pp', s)$. The output t is then used to re-randomise the ciphertext stored in the state via $C' \leftarrow \mathsf{Rand}(pk, C; t)$. The algorithm then returns an output and state pair of the form $(R, S) = (C, (s', C'))$. Since rrPKE is re-randomisable and IND\$-CPA, the output C will appear random and independent from the re-randomised ciphertext C' stored in the state. Combined with the forward security of PRG', this implies the forward security of BPRG.

By construction, it holds that whichever output, R_i, Big Brother receives in his challenge, this corresponds to a ciphertext C_{i-1} encrypting the initial underlying PRG state s_0 which has been re-randomised $(i-1)$ times. BB can simply decrypt $R_i = C_{i-1}$ to recover s_0, then run PRG' forward to recover $t_1, \ldots, t_q$ and use these to iteratively reverse the re-randomisations of C_{i-1} to recover C_0. Setting $S_0 = (s_0, C_0)$ thereby reconstructs the initial state of BPRG. We formalise this in the following theorem.

Theorem 5.3. *Let* $\mathsf{PRG}' = (\mathsf{init}', \mathsf{setup}', \mathsf{next}')$ *be a PRG with output length* ℓ' *and state space* $\mathcal{S} = \{0,1\}^n$. *Let* $\mathsf{rrPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$ *be a reverse re-randomisable PKE scheme with message space* $\mathcal{M} \supseteq \{0,1\}^n$ *and ciphertext space* $\mathcal{C} \subseteq \{0,1\}^*$*, for which* Rand *has coin space* $\mathcal{Z} = \{0,1\}^{\ell'}$ *and* $\mathsf{len}_{\mathsf{PKE}}(n) = \ell$. *Consider the BPRG* $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ *built from these components as shown in Figure 5.9, with output length* ℓ *and state space* $\mathcal{S} = \{0,1\}^{n+\ell}$. *Then* BPRG *is such that*

$$\mathbf{Adv}^{\text{game-first}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = 1 \ .$$

Moreover, for any attacker $\mathcal{A}$ *in game* PRG-FWD *against* $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$ *who runs in time* T*, there exist attackers* $\mathcal{B}, \mathcal{B}'_1, \mathcal{B}'_2$*, and* $\mathcal{C}$ *running in time* T *plus an* $\mathcal{O}(q)$

```
$\underline{\mathsf{init}}$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
Return $(pp, bp)$

$\underline{\mathsf{setup}(pp)}$
$(pk, pp') \leftarrow pp$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \mathsf{Enc}(pk, s_0)$
$S_0 \leftarrow (s_0, C_0)$
Return $S_0$
```

```
$\underline{\mathsf{next}(pp, S)}$
$(pk, pp') \leftarrow pp$
$(s, C) \leftarrow S$
$(t, s') \leftarrow \mathsf{next}'(pp', s)$
$C' \leftarrow \mathsf{Rand}(pk, C; t)$
$R \leftarrow C$ ; $S \leftarrow (s', C')$
Return $(R, S)$
```

```
$\underline{\mathsf{BB}(pp, bp, i, R_i)}$
$(pk, pp') \leftarrow pp$ ; $sk \leftarrow bp$
$C^*_{i-1} \leftarrow R_i$
$s^*_0 \leftarrow \mathsf{Dec}(sk, C^*_{i-1})$
$(t^*_1, \ldots, t^*_q) \leftarrow \mathsf{out}^q(\mathsf{next}'(pp', s^*_0))$
For $j = i-1, \ldots, 1$
    $C^*_{j-1} \leftarrow \mathsf{Rand}^{-1}(pk, C^*_j, t^*_j)$
$S^*_0 \leftarrow (s^*_0, C^*_0)$
Return $S^*_0$
```

Figure 5.9: Construction of a forward secure BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ from a reverse re-randomisable PKE scheme $\mathsf{rrPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand}, \mathsf{Rand}^{-1})$ and a forward secure PRG $\mathsf{PRG}' = (\mathsf{init}', \mathsf{setup}', \mathsf{next}')$.

overhead, such that

$$\mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}}(\mathcal{A}, q) \leq 2 \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}, 1) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}'_1, q) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}'_2, 1) + 2 \cdot \mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}'}(\mathcal{C}, q) + 2q \cdot \Delta_{\mathsf{rrPKE}} .$$

Furthermore, we can define attackers $\mathcal{B}, \mathcal{B}'$, and $\mathcal{C}$, running in time T plus an $\mathcal{O}(q)$ overhead, such that

$$\mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{PRG}}(\mathcal{A}, q) \leq 2 \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}, 1) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}', q) + \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{PRG}'}(\mathcal{C}, q) + q \cdot \Delta_{\mathsf{rrPKE}} .$$

Proof. The correctness of rrPKE immediately implies that $\mathbf{Adv}^{\text{first}}_{\mathsf{BPRG}}(\mathsf{BB}, q) = 1$. To prove the PRG-FWD security of $\mathsf{PRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next})$ we argue by a series of game hops, shown in Figure 5.10. We begin by defining game G_0, which is identical to PRG-FWD against PRG with challenge bit $b = 0$. It follows that

$$\Pr\left[\text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}} \Rightarrow 1 \mid b = 0\right] = \Pr\left[G_0 \Rightarrow 1\right] .$$

Next we define game G_1, which is identical to G_0 except we change the way in which the initial state $S_0 = (s_0, C_0)$ is computed. Namely, instead of setting $C_0 \leftarrow\$ \mathsf{Enc}(pk, S_0)$ as in G_0, we instead sample an independent initial state for the underlying PRG PRG' via $M \leftarrow\$ \mathsf{setup}(pp')$ and set $C_0 \leftarrow\$ \mathsf{Enc}(pk, M)$. We claim that there exists an attacker $\mathcal{B}_1$ in game IND-CPA against rrPKE, who runs in time T plus an $\mathcal{O}(q)$ overhead and makes a single query to their left-or-right oracle, such that

$$\left|\Pr\left[G_1 \Rightarrow 1\right] - \Pr\left[G_0 \Rightarrow 1\right]\right| \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}_1, 1) .$$

To see this, let $\mathcal{B}_1$ be the attacker who proceeds as follows. On input pk, $\mathcal{B}_1$ begins by generating public parameters for PRG' via $(pp', \perp) \leftarrow\$ \mathsf{init}'$. $\mathcal{B}_1$ then generates states

$s_0, M \leftarrow\$ \mathsf{init}(pp')$. $\mathcal{B}_1$ queries (s_0, M) to his left-or-right oracle, receiving C_0 in response. $\mathcal{B}_1$ sets $S_0 = (s_0, C_0)$ and computes the q challenge outputs and corresponding states following the pseudocode description in G_1 (this accounts for the $\mathcal{O}(q)$ overhead in $\mathcal{B}_1$'s run time). At the end of the game, $\mathcal{B}_1$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}_1$ receives real outputs in his game then this perfectly simulates G_0, otherwise it perfectly simulates G_1. Combining these observations via a standard argument then implies the claim.

Next we define game G_2, which is identical to G_1 except we sample the strings used for re-randomisations uniformly at random, $t_i \leftarrow\$ \{0,1\}^{\ell'}$ for $i \in [1, q]$, as opposed to generating these using PRG'. We claim that there exists an attacker $\mathcal{C}_1$ in game PRG-FWD against PRG', who runs in time T plus an $\mathcal{O}(q)$ overhead, such that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}'}(\mathcal{C}_1, q) .$$

To see this, let $\mathcal{C}_1$ be the adversary who proceeds as follows. On input $(pp', t_1, \ldots, t_q, s_q)$, $\mathcal{C}_1$ begins by generating a public / secret key pair for rrPKE, $(pk, sk) \leftarrow\$ \mathsf{Key}$, and sets $pp = (pk, pp')$. $\mathcal{C}_1$ then generates $M \leftarrow\$ \mathsf{setup}(pp')$, and encrypts M under pk to yield C_0. $\mathcal{C}_1$ uses the outputs received in his challenge to iteratively compute $C_i \leftarrow \mathsf{Rand}(pk, C_{i-1}; t_i)$ and $R_i \leftarrow C_{i-1}$ for $i \in [1, q]$. Finally, $\mathcal{C}_1$ sets $S_q = (s_q, C_q)$ where recall that s_q is the state received in his challenge, and passes $(pp, R_1, \ldots, R_q, S_q)$ to $\mathcal{A}$. When $\mathcal{A}$ outputs a bit, $\mathcal{C}_1$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{C}_1$ receives real PRG outputs in his challenge then this perfectly simulates G_1; otherwise it perfectly simulates G_2, thereby proving the claim.

Next we define game G_3, which is identical to G_2 except we set $C_i \leftarrow\$ \mathsf{Enc}(pk, M)$ for $i \in [1, q]$, as opposed to generating these by re-randomising the previous ciphertext state component, $C_i \leftarrow \mathsf{Rand}(pk, C_{i-1}; t_i)$. We claim that

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq q \cdot \Delta_{\mathsf{rrPKE}} .$$

We first define a series of hybrid games H_i, where H_0 is equivalent to G_2, H_q is equivalent to G_3, and for each $i \in [1, q]$, H_i is identical to H_{i-1} except we replace $C_i \leftarrow \mathsf{Rand}(pk, C_{i-1}; t_i)$ with $C_i \leftarrow\$ \mathsf{Enc}(pk, M)$. The triangle inequality implies that

$$\begin{aligned}|\Pr[G_3 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| &= |\Pr[H_q \Rightarrow 1] - \Pr[H_0 \Rightarrow 1]| \\ &\leq \sum_{i=1}^{q} |\Pr[H_i \Rightarrow 1] - \Pr[H_{i-1} \Rightarrow 1]| .\end{aligned}$$

Notice that for each $i \in [1, q]$, $\mathcal{A}$'s view of game H_i can be computed as a deterministic function of $((pk, pp'), s_0, C_0, \ldots, C_i)$ (i.e., by running PRG' forward using (pp', s_0) to generate s_q and t_i for $i \in [1, q]$, setting $C_j \leftarrow \mathsf{Rand}(pk, C_{j-1}; t_j)$ for $j \in [i+1, q]$, and assembling the outputs and states accordingly). Letting g_i denote this function, we may write:

$$\begin{aligned}|\Pr[H_i \Rightarrow 1] - \Pr[H_{i-1} \Rightarrow 1]| = |&\Pr\left[\mathcal{A}(g_i((pk, pp'), s_0, C_0, \ldots, C_{i-1}, \mathsf{Enc}(pk, M)) \Rightarrow 1\right] \\ &- \Pr\left[\mathcal{A}(g_i((pk, pp'), s_0, C_0, \ldots, C_{i-1}, \mathsf{Rand}(pk, C_{i-1}; t_i)) \Rightarrow 1\right]| ,\end{aligned}$$

where $C_j \leftarrow\!\!\$\ \mathsf{Enc}(pk, M)$ for $j \in [0, i-1]$. By the definition of a re-randomisable PKE scheme, we have that:

$$\Delta(\{C_i \ : \ C_i \leftarrow\!\!\$\ \mathsf{Enc}(pk, M)\}, \{C_i \ : \ t_i \leftarrow\!\!\$\ \{0,1\}^{\ell'} ; C_i \leftarrow \mathsf{Rand}(pk, C_{i-1}; t_i)\}) \leq \Delta_{\mathsf{rrPKE}} .$$

Noting that applying a function to two random variables cannot increase their statistical distance, this implies that $|\Pr[H_i \Rightarrow 1] - \Pr[H_{i-1} \Rightarrow 1]| \leq \Delta_{\mathsf{rrPKE}}$ for each $i \in [1, q]$. Summing over the q pairs of games then implies the claim.

Next we define game G_4, which is identical to G_3 except we replace all the ciphertexts $C_{i-1} \leftarrow\!\!\$\ \mathsf{Enc}(pk, M)$ for $i \in [1, q]$ which form the outputs of PRG with random bit strings $C_{i-1} \leftarrow\!\!\$\ \{0,1\}^{\ell}$. Notice that we do not replace C_q (which does not form a PRG output) since looking ahead this will form part of the 'real' state that is given to the attacker regardless of the challenge bit. We claim there exists an adversary $\mathcal{B}'_1$ in game IND\$-CPA against rrPKE, who runs in time T plus an $\mathcal{O}(q)$ overhead and makes q queries to his real-or-random oracle, such that

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}'_1, q) .$$

To see this, let $\mathcal{B}'_1$ be the attacker who proceeds as follows. On input pk, $\mathcal{B}'_1$ generates public parameters for PRG' via $(pp', \perp) \leftarrow\!\!\$\ \mathsf{init}'$, sets $pp = (pk, pp')$, and samples states $s_0, M \leftarrow\!\!\$\ \mathsf{setup}'(pp')$. To generate $\mathcal{A}$'s challenge, $\mathcal{B}'_1$ first runs PRG' forward to generate $(s_1, \ldots, s_q) \leftarrow \mathsf{state}^q(\mathsf{next}'(pp', s_0))$. $\mathcal{B}'_1$ then queries M to his real-or-random oracle q times, receiving $C_0, \ldots, C_{q-1}$ in response. $\mathcal{B}'_1$ sets $R_i = C_{i-1}$ for $i \in [1, q]$, uses pk to compute $C_q \leftarrow\!\!\$\ \mathsf{Enc}(pk, M)$, and passes $(pp, R_1, \ldots, R_q, (s_q, C_q))$ to $\mathcal{A}$. At the end of the game, $\mathcal{B}'_1$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{B}'_1$ receives real ciphertexts from his oracle then this perfectly simulates G_3; otherwise it perfectly simulates G_4, thereby implying the claim.

Next we define game G_5, which is identical to G_4 except we now: **(1)** explicitly sample outputs as $R_i \leftarrow\!\!\$\ \{0,1\}^{\ell}$ for $i \in [1, q]$, rather than choosing $C_{i-1} \leftarrow\!\!\$\ \{0,1\}^{\ell}$ and setting

$R_i = C_{i-1}$; and **(2)** return to computing $C_0 \leftarrow\$ \mathsf{Enc}(pk, M)$ when generating the initial PRG state, rather than sampling $C_0 \leftarrow\$ \{0,1\}^\ell$. Since $R_1, \ldots, R_q$ are already equivalent to uniform bit strings in G_4, **(1)** is a purely syntactic change. To bound **(2)**, we claim that there exists an attacker $\mathcal{B}_2'$ in game IND\$-CPA against rrPKE, who runs in time T plus an $\mathcal{O}(q)$ overhead and makes a single query to his real-or-random oracle, such that

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}_2', 1) .$$

We define $\mathcal{B}_2'$ to proceed identically to $\mathcal{B}_1'$ in the previous reduction, except during the challenge computation $\mathcal{B}_2'$ makes only a single query M to his real-or-random oracle (setting C_0 equal to the response) and samples random outputs $R_i \leftarrow\$ \{0,1\}^\ell$ for $i \in [1, q]$. As before, if $\mathcal{B}_2'$ receives a real encryption in his challenge then this perfectly simulates G_4; otherwise it perfectly simulates G_5, thereby implying the claim.

Next we define game G_6, which is identical to G_5 except we return to computing C_q as the q^{th} re-randomisation of C_0, $C_q \leftarrow \mathsf{Rand}(pk, C_0; t_1, \ldots, t_q)$, rather than as a fresh ciphertext. An analogous argument to that made above, invoking the re-randomisability of rrPKE, implies that

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq q \cdot \Delta_{\mathsf{rrPKE}} .$$

Next we define game G_7, which is identical to G_6 except we return to computing $t_1, \ldots, t_q$ via PRG′ rather than sampling these randomly. Since both games can be perfectly simulated by an attacker in PRG-FWD against PRG′, an analogous argument to that made above implies that there exists an attacker $\mathcal{C}_2$ with the claimed run time such that

$$|\Pr[G_7 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}'}(\mathcal{C}_2, q) .$$

Finally we define game G_8, which is identical to G_7 except we return to setting $C_0 \leftarrow\$ \mathsf{Enc}(pk, s_0)$ when generating the initial state. By an analogous argument to that made previously, both games can be perfectly simulated by an adversary $\mathcal{B}_2$ in game IND-CPA against rrPKE with the claimed run time and query budget, and so

$$|\Pr[G_8 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}_2, 1) .$$

Moreover, notice that game G_8 is identical to PRG-FWD against PRG with challenge bit $b = 1$, and so

$$\Pr\left[\text{PRG-FWD}^{\mathcal{A},q}_{\mathsf{PRG}} \Rightarrow 1 \mid b = 1\right] = \Pr[G_8 \Rightarrow 1] .$$

Putting this altogether via a standard argument, it follows that

$$\begin{aligned}\mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}}(\mathcal{A},q) \leq 2 \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B},1) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}'_1,q) \\ + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}'_2,1) + 2 \cdot \mathbf{Adv}^{\text{prg-fwd}}_{\mathsf{PRG}'}(\mathcal{C},q) + 2q \cdot \Delta_{\mathsf{rrPKE}} \, .\end{aligned}$$

This concludes the proof of PRG-FWD security for PRG. We now prove that PRG also achieves $\overline{\text{PRG-FWD}}$ security as a straightforward extension of the above result. We argue by a series of game hops, also shown in Figure 5.10, where we define games G'_0 and G'_1 to be identical to G_0 and G_1 respectively. An identical argument to that used previously implies that there exists an attacker $\mathcal{B}_1$ in game IND-CPA against rrPKE with the claimed run time such that

$$|\Pr\left[\, G'_1 \Rightarrow 1 \,\right] - \Pr\left[\, G'_0 \Rightarrow 1 \,\right]| \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}_1,1) \, .$$

In game G'_2, we replace the PRG' outputs $t_1,\ldots,t_q$ with uniform bit strings as in G_2, but now also replace the final PRG' state with a freshly generated state $s_q \leftarrow\!\!\$\; \mathsf{setup}'(pp')$. A straightforward reduction to the $\overline{\text{PRG-FWD}}$ security of PRG' implies that there exists an adversary $\mathcal{C}$, running in time T plus an $\mathcal{O}(q)$ overhead, such that

$$|\Pr\left[\, G'_2 \Rightarrow 1 \,\right] - \Pr\left[\, G'_1 \Rightarrow 1 \,\right]| \leq \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{PRG}'}(\mathcal{C},q) \, .$$

We define games G'_3, G'_4 to be identical to games G_3, G_4 respectively, but with the extra line of code setting $s_q \leftarrow\!\!\$\; \mathsf{setup}'(pp')$ (introduced in the previous game hop) included. An entirely analogous argument to that made previously implies that there exists an adversary $\mathcal{B}'$ with the claimed run time such that

$$|\Pr\left[\, G'_4 \Rightarrow 1 \,\right] - \Pr\left[\, G'_2 \Rightarrow 1 \,\right]| \leq q \cdot \Delta_{\mathsf{rrPKE}} + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}',q) \, .$$

In game G'_5, we replace the ciphertext $C_q \leftarrow\!\!\$\; \mathsf{Enc}(pk,M)$ in the challenge state with $C_q \leftarrow\!\!\$\; \mathsf{Enc}(pk,s_q)$. (We also remove some redundant lines of code; it is straightforward to verify that this does not alter the distribution of the game.) A reduction to the IND\$-CPA security of rrPKE implies the existence of an attacker $\mathcal{B}_2$ with the claimed run time such that

$$|\Pr\left[\, G'_5 \Rightarrow 1 \,\right] - \Pr\left[\, G'_4 \Rightarrow 1 \,\right]| \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}_2,1) \, .$$

Moreover, notice that the challenge computation in G'_5 is equivalent to that in $\overline{\text{PRG-FWD}}$ with challenge bit $b = 1$. Combining the above via a standard argument, implies that there exists attackers $\mathcal{B},\mathcal{B}',\mathcal{C}$ with the claimed run times and query budgets, such that

$$\mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{PRG}}(\mathcal{A},q) \leq 2 \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rrPKE}}(\mathcal{B},1) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{B}',q) + \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{PRG}'}(\mathcal{C},q) + q \cdot \Delta_{\mathsf{rrPKE}} \, .$$

```
proc. main // $G_0, \boxed{G_1}$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \mathsf{Enc}(pk, s_0)$
$\boxed{M \leftarrow\$ \mathsf{setup}'(pp')}$
$\boxed{C_0 \leftarrow\$ \mathsf{Enc}(pk, M)}$
$S_0 \leftarrow (s_0, C_0)$
For $i = 1, \ldots, q$
    $(t_i, s_i) \leftarrow \mathsf{next}'(pp', s_{i-1})$
    $C_i \leftarrow \mathsf{Rand}(pk, C_{i-1}; t_i)$
    $R_i \leftarrow C_{i-1}$
$S_q \leftarrow (s_q, C_q)$
$b^* \leftarrow\$ \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. main // $G_2, G_2', \boxed{G_3, G_3'}$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$M \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \mathsf{Enc}(pk, M)$
$S_0 \leftarrow (s_0, C_0)$
For $i = 1, \ldots, q$
    $(t_i, s_i) \leftarrow \mathsf{next}'(pp', s_{i-1})$
    $t_i \leftarrow\$ \{0,1\}^{\ell'}$
    $C_i \leftarrow \mathsf{Rand}(pk, C_{i-1}; t_i)$
    $\boxed{C_i \leftarrow\$ \mathsf{Enc}(pk, M)}$
    $R_i \leftarrow C_{i-1}$
$s_q \leftarrow\$ \mathsf{init}'(pp')$ // $G_2', G_3'$ only
$S_q \leftarrow (s_q, C_q)$
$b^* \leftarrow\$ \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. main // $G_4$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$M \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \{0,1\}^{\ell}$
$S_0 \leftarrow (s_0, C_0)$
For $i = 1, \ldots, q$
    $(t_i, s_i) \leftarrow \mathsf{next}'(pp', s_{i-1})$
    $t_i \leftarrow\$ \{0,1\}^{\ell'}$
    $C_i \leftarrow\$ \{0,1\}^{\ell}$
    $R_i \leftarrow C_{i-1}$
$s_q \leftarrow\$ \mathsf{init}'(pp')$ // $G_4'$ only
$C_q \leftarrow\$ \mathsf{Enc}(pk, M)$
$S_q \leftarrow (s_q, C_q)$
$b^* \leftarrow\$ \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. main // $G_5, \boxed{G_6}$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$M \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \mathsf{Enc}(pk, M)$
$S_0 \leftarrow (s_0, C_0)$
For $i = 1, \ldots, q$
    $(t_i, s_i) \leftarrow \mathsf{next}'(pp', s_{i-1})$
    $t_i \leftarrow\$ \{0,1\}^{\ell'}$
    $R_i \leftarrow\$ \{0,1\}^{\ell}$
$C_q \leftarrow\$ \mathsf{Enc}(pk, M)$
$\boxed{C_q \leftarrow \mathsf{Rand}(pk, C_0; t_1, \ldots, t_q)}$
$S_q \leftarrow (s_q, C_q)$
$b^* \leftarrow\$ \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. main // $G_7, \boxed{G_8}$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$M \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \mathsf{Enc}(pk, M)$
$\boxed{C_0 \leftarrow\$ \mathsf{Enc}(pk, s_0)}$
$S_0 \leftarrow (s_0, C_0)$
For $i = 1, \ldots, q$
    $(t_i, s_i) \leftarrow \mathsf{next}'(pp', s_{i-1})$
    $R_i \leftarrow\$ \{0,1\}^{\ell}$
$C_q \leftarrow \mathsf{Rand}(pk, C_0; t_1, \ldots, t_q)$
$S_q \leftarrow (s_q, C_0)$
$b^* \leftarrow\$ \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

```
proc. main // $G_5'$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$(pp', \perp) \leftarrow\$ \mathsf{init}'$
$pp \leftarrow (pk, pp')$
$bp \leftarrow sk$
$s_0 \leftarrow\$ \mathsf{setup}'(pp')$
$M \leftarrow\$ \mathsf{setup}'(pp')$
$C_0 \leftarrow\$ \{0,1\}^{\ell}$
$S_0 \leftarrow (s_0, C_0)$
For $i = 1, \ldots, q$
    $R_i \leftarrow\$ \{0,1\}^{\ell}$
$s_q \leftarrow\$ \mathsf{init}'(pp')$
$C_q \leftarrow\$ \mathsf{Enc}(pk, s_q)$
$S_q \leftarrow (s_q, C_q)$
$b^* \leftarrow\$ \mathcal{A}(pp, R_1, \ldots, R_q, S_q)$
Return $b^*$
```

Figure 5.10: Games for proof of Theorem 5.3.

□

This concludes our analysis of the two forward secure BPRG-FIRST backdoored PRGs presented in this section. We now build on these results to address the main theme of the chapter: backdooring robust PRNGs with input.

5.4 Backdooring Pseudorandom Number Generators with Input

Having demonstrated that forward security for PRGs offers no protection against backdooring, we turn our attention to subverting a primitive with far stronger security properties: robust PRNGs with input. In this section we present the first provable security treatment of backdoored PRNGs (BPRNGs), giving definitions and concrete constructions. We begin by specifying our BPRNG syntax.

BPRNG syntax. BPRNGs are defined analogously to BPRGs (see Section 5.3) as a PRNG with an associated Big Brother algorithm BB which represents the backdoor attacker. More formally, a BPRNG $\mathsf{BPRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next}, \mathsf{BB})$ with parameter set (ℓ, p) is a tuple of algorithms where $\mathsf{init}, \mathsf{setup}, \mathsf{refresh}$, and next are as defined in Section 5.2. The input / output behaviour of BB will be specified for each BPRNG — however, BB always takes as input a public parameter $pp \in \{0,1\}^*$ and a backdoor parameter $bp \in \{0,1\}^*$.

5.4.1 A Simple Backdoored PRNG

Before introducing backdooring models for PRNGs, and to motivate our definitions, we show that the forward secure BPRGs presented in Section 5.3 immediately allow us to construct a robust BPRNG satisfying a weak form of backdooring.

Intuition. Let $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$ be a robust PRNG with input. By considering the special case of game Rob in which the adversary $\mathcal{A}$ makes no Set or Ref queries, and one Get query at the conclusion of the game, it is straightforward to see that $\mathsf{PRG} = (\mathsf{setup}, \mathsf{init}, \mathsf{next})$ must be a forward secure PRG. This suggests that we might be able to replace PRG with a backdoored variant BPRG, as defined in Section 5.3. The effect of this is that during the periods of output production between $\mathsf{refresh}$ calls — when the PRNG is effectively operating as a deterministic PRG — Big Brother should be able to exploit the backdoor embedded in BPRG.

Construction. Consider the BPRNG $\mathsf{BPRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}})$ shown in Figure 5.11. The construction is built from an extractor $\mathsf{Ext} : \{0,1\}^* \times \{0,1\}^v \to \{0,1\}^n$ which is online-computable on inputs of length p with associated algorithms $\mathsf{iterate} : \{0,1\}^p \times \{0,1\}^p \times \{0,1\}^v \to \{0,1\}^p$ and $\mathsf{finalise} : \{0,1\}^p \times \{0,1\}^v \to \{0,1\}^n$, and a BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ with output length $\ell \in \mathbb{N}$ and state space $\mathcal{S} = \{0,1\}^n$. We have written the BPRNG algorithms and states with an overline to distinguish these from the algorithms of the underlying BPRG. Looking ahead, we will require that BPRG

```
$\overline{\mathsf{init}}$
$(pp', bp) \leftarrow_\$ \mathsf{init}$
$A \leftarrow_\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
Return $(pp, bp)$

$\overline{\mathsf{setup}}(pp)$
$S_1 \leftarrow_\$ \mathsf{setup}(pp)$
$S_2 \leftarrow 0^p$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S} \leftarrow (S_1, S_2, \mathsf{flag\text{-}ref})$
Return $\overline{S}$
```

```
$\overline{\mathsf{refresh}}(pp, \overline{S}, I)$
$(pp', A) \leftarrow pp$
$(S_1, S_2, \mathsf{flag\text{-}ref}) \leftarrow \overline{S}$
$S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
$\mathsf{flag\text{-}ref} \leftarrow 1$
$\overline{S} \leftarrow (S_1, S_2, \mathsf{flag\text{-}ref})$
Return $\overline{S}$
```

```
$\overline{\mathsf{next}}(pp, \overline{S})$
1. $(pp', A) \leftarrow pp$
2. $(S_1, S_2, \mathsf{flag\text{-}ref}) \leftarrow \overline{S}$
3. If $\mathsf{flag\text{-}ref} = 1$
4.     $U \leftarrow \mathsf{finalise}(S_2; A)$
5.     $S_1 \leftarrow U \oplus S_1$
6.     $S_2 \leftarrow 0^p$
7. $(R, S_1) \leftarrow \mathsf{next}(pp', S_1)$
8. $\mathsf{flag\text{-}ref} \leftarrow 0$
9. $\overline{S} \leftarrow (S_1, S_2, \mathsf{flag\text{-}ref})$
10. Return $(R, \overline{S})$
```

Figure 5.11: Construction of a robust BPRNG $\mathsf{BPRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}}, \mathsf{BB})$ from a $\overline{\text{PRG-FWD}}$ secure BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{BB})$, and an extractor Ext which is online-computable with respect to p, with associated algorithms $\mathsf{iterate}$ and $\mathsf{finalise}$.

is $\overline{\text{PRG-FWD}}$ secure and has the additional property that the states output by setup are pseudorandom. More formally, for a BPRG $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ with state space $\mathcal{S} = \{0,1\}^n$, we define the IND-STATE advantage of an attacker $\mathcal{A}$ to be

$$\mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{A}) = \Big| \Pr\left[\, \mathcal{A}(pp, S_0) \Rightarrow 1 \ : \ (pp, bp) \leftarrow_\$ \mathsf{init}\,; S_0 \leftarrow_\$ \mathsf{setup}(pp) \,\right] - \Pr\left[\, \mathcal{A}(pp, S_0) \Rightarrow 1 \ : \ (pp, bp) \leftarrow_\$ \mathsf{init}\,; S_0 \leftarrow_\$ \{0,1\}^n \,\right] \Big| \,.$$

Overview. Our backdoored PRNG BPRNG is based on the PRNG with input from [60]. The key differences are that: **(1)** we produce outputs via a stateful and forward secure (backdoored) PRG as opposed to the stateless PRG used in [60]); and **(2)** the construction uses an abstract online-computable extractor rather than the concrete online extractor-like function in the original.

The state of BPRNG is of the form $\overline{S} = (S_1, S_2, \mathsf{flag\text{-}ref}) \in \{0,1\}^n \times \{0,1\}^p \times \{0,1\}$, where S_1 is a state for BPRG, S_2 is a state for the online-computable extractor (initialised to 0^p), and $\mathsf{flag\text{-}ref}$ is used to record whether $\overline{\mathsf{refresh}}$ has been invoked since the previous $\overline{\mathsf{next}}$ call. The public parameters are of the form $pp = (pp', A)$, where pp' are parameters for BPRG and $A \leftarrow_\$ \{0,1\}^v$ is an extractor seed.

On input $(pp, (S_1, S_2, \mathsf{flag\text{-}ref}), I)$, $\overline{\mathsf{refresh}}$ incorporates I into the state of the online-computable extractor via $S_2 \leftarrow \mathsf{iterate}(S_2, I; A)$ and sets $\mathsf{flag\text{-}ref}$ to true. On input $(pp, (S_1, S_2, \mathsf{flag\text{-}ref}))$, algorithm $\overline{\mathsf{next}}$ first checks if $\mathsf{flag\text{-}ref} = 1$. If so, it finalises the extraction process by computing $U \leftarrow \mathsf{finalise}(S_2; A)$ and uses this to update state component S_1 via $S_1 \leftarrow S_1 \oplus U$. Algorithm $\overline{\mathsf{next}}$ then uses the next algorithm of the underlying

```
$\underline{\text{on-EXT}^{\mathcal{A},\mathcal{D}}_{\mathsf{Ext},\gamma^*,q_{\mathcal{D}}}}$
$b \leftarrow\!\!\$ \{0,1\}$
$\sigma_0 \leftarrow \varepsilon$ ; $A \leftarrow\!\!\$ \{0,1\}^v$ ; $\mu \leftarrow 0$
For $k = 1, \ldots, q_{\mathcal{D}}$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$ \mathcal{D}(\sigma_{k-1})$
$d \leftarrow\!\!\$ \mathcal{A}^{\text{Sam}}(A, (\gamma_k, z_k)_{k=1}^{q_{\mathcal{D}}})$
If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
If $b = 0$ then
    $y_0 \leftarrow 0^p$
    For $j = 1, \ldots, d$
        $y_j \leftarrow \mathsf{iterate}(y_{j-1}, I_j; A)$
    $U \leftarrow \mathsf{finalise}(y_d; A)$
Else $U \leftarrow\!\!\$ \{0,1\}^n$
$b^* \leftarrow\!\!\$ \mathcal{A}(A, U, (I_k)_{k>\mu+d})$
Return $(b = b^*)$
---
$\underline{\text{Sam}}$
$\mu = \mu + 1$
Return $I_\mu$
```

Figure 5.12: Security game for an online-computable extractor.

PRG to compute $(R, S_1) \leftarrow \mathsf{next}(pp, S_1)$, returning the output R and updating the S_1 component of $\overline{S}$ accordingly. The Big Brother algorithm BB is simply taken to be that of BPRG.

We will now prove that BPRNG is robust. We begin by defining a variant of security for online-computable extractors that we will utilise in the proof.

Online-computable extractors. Consider the game on-EXT shown in Figure 5.12, in which an attacker is challenged to distinguish the output of the online-computable extractor Ext — applied to a set of inputs, adaptively chosen by $\mathcal{A}$ and which collectively contain γ^* bits of entropy — from a random string. The game is very similar to game Extract used in the proof of Rec security for $\mathsf{HASH\text{-}DRBG}$ (see Section 4.6), and isolates the required security of an online-computable extractor with respect to a distribution sampler $\mathcal{D}$. For an attacker / sampler pair $(\mathcal{A}, \mathcal{D})$, we define

$$\mathbf{Adv}^{\text{on-ext}}_{\mathsf{Ext},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}, \mathcal{D}) = 2 \cdot \left| \Pr\left[\text{on-EXT}^{\mathcal{A},\mathcal{D}}_{\mathsf{Ext},\gamma^*,q_{\mathcal{D}}} \Rightarrow 1 \right] - \frac{1}{2} \right| .$$

We say that $\mathcal{A}$ is a q_C-adversary if they output $d \leq q_C$ in their challenge.

Security analysis of BPRNG. It is fairly straightforward to verify that BPRNG is robust; the proof is similar to that of the original construction in [60]. Notice that $\overline{\mathsf{setup}}$ returns states $\overline{S}_1 = (S_1, S_2, \mathsf{flag\text{-}ref})$ for which S_1 is pseudorandom. This ensures that in game Pres, the updated state $S_1 \leftarrow U \oplus S_1$, where U is the output of the online extractor (line 5 of $\overline{\mathsf{next}}$), is a pseudorandom n-bit string too. Similarly in game Rec, the properties of Ext imply that U and thereby the updated S_1 are statistically close to uniform. In both cases, applying next to S_1 yields a pseudorandom output and state by reductions to the $\overline{\text{PRG-FWD}}$ and IND-STATE security of BPRG. From this, the robustness of BPRNG follows from Theorem 2.2. We formalise this in the following theorem.

Theorem 5.4. *Let* $\mathsf{BPRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}}, \mathsf{BB})$ *be the BPRNG with associated parameters* (ℓ, p) *and state space* $\mathcal{S} = \{0,1\}^n \times \{0,1\}^p \times \{0,1\}$ *shown in Figure 5.11, built from an extractor* $\mathsf{Ext} : \{0,1\}^* \times \{0,1\}^v \to \{0,1\}^n$ *which is online-computable with respect to* p *and a BPRG* $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ *with output length* ℓ *and state space* $\mathcal{S} = \{0,1\}^n$. *Let* $\mathsf{PRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}})$. *Then for any* $(q_R, q_{\mathcal{D}}, q_C, q_S)$*-attacker* $\mathcal{A}$ *running in time* T *and any* $(q_{\mathcal{D}}, \gamma^*)$*-legitimate sampler* $\mathcal{D}$, *there exist adversaries* $\mathcal{B}$, $\mathcal{C}$, *and a* q_C*-adversary* $\mathcal{E}$, *each running in time* $T' \approx T$, *such that*

$$\mathbf{Adv}^{\text{rob}}_{\mathsf{PRNG},\gamma^*}(\mathcal{A}, \mathcal{D}) \leq 2q_R \cdot \left(3 \cdot \mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{B}) + 2 \cdot \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{BPRG}}(\mathcal{C}, 1) + \mathbf{Adv}^{\text{ext}}_{\mathsf{Ext},\gamma^*,q_{\mathcal{D}}}(\mathcal{E}, \mathcal{D})\right) .$$

Proof. We prove the Rob security of PRNG by bounding its Pres and Rec security, then combining these results via Theorem 2.2. We begin by considering Pres security.

Pres security. We argue by a series of game hops, shown in Figure 5.13. Let G_0 be identical to game Pres for PRNG with challenge bit $b = 0$. We begin by defining game G_1, which is identical to G_0 except we replace the initial underlying PRG state $S_1 \leftarrow\$\, \mathsf{setup}(pp)$ with a truly random state $S_1 \leftarrow\$\, \{0,1\}^n$. A straightforward reduction to the IND-STATE security of BPRG implies that there exists an attacker $\mathcal{B}_1$ running in time $T' \approx T$ such that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_0 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{B}_1) .$$

Next we define game G_2, which is identical to G_1 except in the steps in the 'If $\mathsf{flag\text{-}ref} = 1$' conditional we replace the updated state $S_1' \leftarrow S_1 \oplus U$ with a uniformly random state $S_1' \leftarrow\$\, \{0,1\}^n$. Since $S_1 \leftarrow\$\, \{0,1\}^n$ is uniformly distributed in G_1 by the previous transition, it follows that S_1' is uniformly distributed in G_1 also. As such, the games are identical and $\Pr[G_2 \Rightarrow 1] = \Pr[G_1 \Rightarrow 1]$.

Next in game G_3, we replace the randomly sampled state $S_1' \leftarrow\$\, \{0,1\}^n$ with a freshly generated state $S_1' \leftarrow\$\, \mathsf{setup}(pp)$. As before, an analogous reduction to the IND-STATE security of PRG implies that there exists an attacker $\mathcal{B}_2$, running in time $T' \approx T$, such that

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{B}_2) \ .$$

Next we define game G_4, which is identical to G_3 except we replace the output / state pair $(R^*, S_1^*) \leftarrow \mathsf{next}(pp, S_1')$ with a random output $R^* \leftarrow\!\!\$\ \{0,1\}^\ell$ and a freshly generated state $S_1^* \leftarrow\!\!\$\ \mathsf{setup}(pp)$. Since both games can be perfectly simulated by an attacker $\mathcal{C}$ in game $\overline{\text{PRG-FWD}}^{\mathcal{C},1}_{\mathsf{BPRG}}$ against BPRG (who given challenge output / state pair (R^*, S_1^*) constructs $\mathcal{A}$'s challenge as $\overline{S}^* = (R^*, (S_1^*, 0^p, 0))$ and at the end of the game outputs whatever bit $\mathcal{A}$ does), it follows that

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{BPRG}}(\mathcal{C}, 1) \ .$$

In game G_5, G_6, and G_7 (not shown), we revert the modifications introduced in games G_3, G_2, and G_1 respectively to return to computing states S_1' and S_1 according to the specification of $\overline{\mathsf{next}}$. However, since $\mathcal{A}$'s view of the experiment is now independent of these intermediate states, the games are identically distributed and so $\Pr[G_7 \Rightarrow 1] = \Pr[G_4 \Rightarrow 1]$. Moreover, notice that G_7 is identical to game Pres against PRNG with challenge bit $b = 1$. (To see this, notice that $\mathcal{A}$ receives in their challenge a random output R^* and a state $\overline{S}^* = (S_1^*, 0^p, 0)$ for $S_1^* \leftarrow\!\!\$\ \mathsf{setup}(pp)$, which is precisely the distribution of states returned by $\overline{\mathsf{setup}}$.) Combining the above observations via a standard argument implies that there exist attackers $\mathcal{B}$ and $\mathcal{C}$, running in time $T' \approx T$, such that

$$\mathbf{Adv}^{\text{pres}}_{\mathsf{PRNG}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{B}) + \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{BPRG}}(\mathcal{C}, 1) \ .$$

Rec security. The proof of Rec security is very similar; we emphasise the key differences here. It is straightforward to verify that game G_0' shown in Figure 5.14 is equivalent to game Rec against BPRNG with challenge bit $b = 0$. Next we define game G_1' to be identical to G_0', except we replace $U \leftarrow \mathsf{finalise}(S_2; A)$ with a uniform string $U \leftarrow\!\!\$\ \{0,1\}^n$. It is straightforward to verify that both games can be perfectly simulated by an attacker $\mathcal{E}$ in game Ext against Ext (who uses his own oracle to simulate $\mathcal{A}$'s Sam oracle, outputs the same index to his challenger as $\mathcal{A}$ does, sets U equal to the returned string, and uses this plus the remaining entropy samples to simulate the rest of the game). It follows that

$$|\Pr[G_1' \Rightarrow 1] - \Pr[G_0' \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ext}}_{\mathsf{Ext},\gamma^*,q_{\mathcal{D}}}(\mathcal{E}, \mathcal{D}) \ .$$

In games G_2' and G_3' we replace S_1' in the 'If $\mathsf{flag\text{-}ref} = 1$' conditional with first a uniform string, and then with a freshly generated state $S_1' \leftarrow\!\!\$\ \mathsf{setup}(pp)$. An entirely analogous argument to that made in the transitions to games G_2 and G_3 in the proof of Pres security implies there exists an attacker $\mathcal{B}$, running in time $T' \approx T$, such that

$$|\Pr\left[\, G_3' \Rightarrow 1 \,\right] - \Pr\left[\, G_1' \Rightarrow 1 \,\right]| \leq \mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{B}) \;.$$

In game G_4', we replace the output / state pair $(R^*, S_1^*) \leftarrow \mathsf{next}(pp, S_1')$ with a random output and freshly generated state $R^* \leftarrow\!\!\$\, \{0,1\}^\ell$, $S_1^* \leftarrow\!\!\$\, \mathsf{setup}(pp)$. An analogous reduction to the $\overline{\text{PRG-FWD}}$ security of BPRG implies that there exists an attacker $\mathcal{C}$, running in time $T' \approx T$, such that

$$|\Pr\left[\, G_4' \Rightarrow 1 \,\right] - \Pr\left[\, G_3' \Rightarrow 1 \,\right]| \leq \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{BPRG}}(\mathcal{C}, 1) \;.$$

Finally, in games G_5', G_6', and G_7', we reverse the modifications made in games G_3', G_2', and G_1', respectively, to return to computing the state S_1' according to the protocol. Again since the attacker's view is independent of intermediate states in G_4', it follows that these games are identically distributed. Moreover, game G_7' is equivalent to game Rec against PRNG with challenge bit $b = 1$. Putting this all together via a standard argument implies that

$$\mathbf{Adv}^{\text{rec}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}, \mathcal{D}) \leq \mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{B}) + \mathbf{Adv}^{\overline{\text{prg-fwd}}}_{\mathsf{BPRG}}(\mathcal{C}, 1) + \mathbf{Adv}^{\text{ext}}_{\mathsf{Ext},\gamma^*,q_{\mathcal{D}}}(\mathcal{E}, \mathcal{D}) \;.$$

Combining the statements on Pres and Rec security via Theorem 2.2 completes the proof.

□

Instantiation. Recall that BPRNG requires as a component a $\overline{\text{PRG-FWD}}$ BPRG which additionally has pseudorandom initial states. Consider $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ of Theorem 5.2, built from a TDP family TDP, PKE scheme PKE, and hash function H. Since the initial states for BPRG are of the form $S_0 \leftarrow\!\!\$\, \{0,1\}^n$, it is straightforward to verify that $\mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{A}) = 0$. Combined with the $\overline{\text{PRG-FWD}}$ security of BPRG proved in Theorem 5.2, this implies that BPRG may be used to instantiate BPRNG.

For $\mathsf{BPRG} = (\mathsf{init}, \mathsf{setup}, \mathsf{next}, \mathsf{BB})$ of Theorem 5.3, built from a PKE scheme rrPKE and PRG PRG', the initial states returned by setup are of the form $S_0 = (s_0, C_0)$ where s_0 is an initial state for PRG' and $C_0 \leftarrow\!\!\$\, \mathsf{Enc}(pk, s_0)$. Since rrPKE is assumed to be IND\$-CPA, it follows that BPRG will have pseudorandom initial states provided PRG' does. More formally, a simple game hopping argument replacing first s_0, and then C_0, in state S_0 with random bit strings of appropriate length implies that for any attacker $\mathcal{A}$ running in time T, there exist attackers $\mathcal{B}, \mathcal{C}$, running in time $T' \approx T$, such that

$$\mathbf{Adv}^{\text{ind-state}}_{\mathsf{BPRG}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ind-state}}_{\mathsf{PRG}'}(\mathcal{B}) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rrPKE}}(\mathcal{C}, 1) \;.$$

Moreover, BPRG was shown in Theorem 5.3 to meet $\overline{\text{PRG-FWD}}$ security and so BPRG

```
proc. main // $G_0$, $\boxed{G_1}$
$(pp', bp) \leftarrow\$ \mathsf{init}$ ; $A \leftarrow\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
$S_1 \leftarrow\$ \mathsf{setup}(pp)$ ; $\boxed{S_1 \leftarrow\$ \{0,1\}^n}$
$S_2 \leftarrow 0^p$ ; $\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S} \leftarrow (S_1, S_2, \mathsf{flag\text{-}ref})$
$(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp)$
For $i = 1, \ldots, d$
    $S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
    $\mathsf{flag\text{-}ref} \leftarrow 1$
If $\mathsf{flag\text{-}ref} = 1$
    $U \leftarrow \mathsf{finalise}(S_2; A)$
    $S_1' \leftarrow U \oplus S_1$
    $S_2 \leftarrow 0^p$
$(R^*, S_1^*) \leftarrow \mathsf{next}(pp', S_1')$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S}^* \leftarrow (S_1^*, S_2, \mathsf{flag\text{-}ref})$
$b^* \leftarrow\$ \mathcal{A}(pp, R^*, \overline{S^*})$
Return $b^*$
```

```
proc. main // $G_2$, $\boxed{G_3}$
$(pp', bp) \leftarrow\$ \mathsf{init}$ ; $A \leftarrow\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
$S_1 \leftarrow\$ \{0,1\}^n$
$S_2 \leftarrow 0^p$ ; $\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S} \leftarrow (S_1, S_2, \mathsf{flag\text{-}ref})$
$(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp)$
For $i = 1, \ldots, d$
    $S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
    $\mathsf{flag\text{-}ref} \leftarrow 1$
If $\mathsf{flag\text{-}ref} = 1$
    $U \leftarrow \mathsf{finalise}(S_2; A)$
    $S_1' \leftarrow\$ \{0,1\}^n$
    $\boxed{S_1' \leftarrow\$ \mathsf{setup}(pp)}$
    $S_2 \leftarrow 0^p$
$(R^*, S_1^*) \leftarrow \mathsf{next}(pp', S_1')$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S}^* \leftarrow (S_1^*, S_2, \mathsf{flag\text{-}ref})$
$b^* \leftarrow\$ \mathcal{A}(pp, R^*, \overline{S^*})$
Return $b^*$
```

```
proc. main // $G_4$
$(pp', bp) \leftarrow\$ \mathsf{init}$ ; $A \leftarrow\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
$S_1 \leftarrow\$ \{0,1\}^n$
$S_2 \leftarrow 0^p$ ; $\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S} \leftarrow (S_1, S_2, \mathsf{flag\text{-}ref})$
$(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp)$
For $i = 1, \ldots, d$
    $S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
    $\mathsf{flag\text{-}ref} \leftarrow 1$
If $\mathsf{flag\text{-}ref} = 1$
    $U \leftarrow \mathsf{finalise}(S_2; A)$
    $S_1' \leftarrow\$ \mathsf{setup}(pp)$
    $S_2 \leftarrow 0^p$
$R^* \leftarrow \{0,1\}^\ell$ ; $S_1^* \leftarrow\$ \mathsf{setup}(pp)$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S}^* \leftarrow\$ (S_1^*, S_2, \mathsf{flag\text{-}ref})$
$b^* \leftarrow\$ \mathcal{A}(pp, R^*, \overline{S^*})$
Return $b^*$
```

Figure 5.13: Games for proof of Theorem 5.4.

may also be used to instantiate BPRNG.

For both choices of BPRG, this form of backdooring would then allow BB, given an arbitrary output R_i, to recover the state which immediately followed the last refresh call prior to the production of R_i. BB can then use this to recover all output produced between this point and the next refresh. Of course, following a high entropy refresh the state will become unpredictable again, and Big Brother will need to compromise another output in order to regain his advantage.

Towards stronger backdooring. Since BB's advantage is lost following a high entropy refresh, the form of backdooring achieved by the BPRNG presented in Figure 5.11 is highly limited. One implication of this construction is that when considering stronger forms of backdooring we must turn our attention to subverting refresh calls in some way.

5.4.2 Security Models for Backdoored PRNGs

We now define security models for BPRNGs. As we shall see, specifying such models requires some care in the more complicated BPRNG setting. We discuss the rationale for our choice of models below.

```
proc. main // $G'_0$, $\boxed{G'_1}$
$\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 0$
$(pp', bp) \leftarrow_\$ \mathsf{init}$ ; $A \leftarrow_\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
For $k = 1, \ldots, q_{\mathcal{D}}$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow_\$ \mathcal{D}(\sigma_{k-1})$
$(\overline{S_0}, d) \leftarrow_\$ \mathcal{A}^{\mathrm{Sam}}(pp, (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
$(S_1, S_2, \mathsf{flag\text{-}ref}) \leftarrow \overline{S_0}$
For $i = 1, \ldots, d$
    $S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
    $\mathsf{flag\text{-}ref} \leftarrow 1$
If $\mathsf{flag\text{-}ref} = 1$
    $U \leftarrow \mathsf{finalise}(S_2; A)$
    $\boxed{U \leftarrow_\$ \{0,1\}^n}$
    $S'_1 \leftarrow U \oplus S_1$
    $S_2 \leftarrow 0^p$
$(R^*, S_1^*) \leftarrow \mathsf{next}(pp'; S'_1)$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S}^* \leftarrow (S_1^*, S_2, \mathsf{flag\text{-}ref})$
$b^* \leftarrow_\$ \mathcal{A}(pp, R^*, \overline{S^*}, (I_k)_{k>\mu+d})$
Return $b^*$

proc. Sam // $G'_0 - G'_7$
$\mu = \mu + 1$
Return $I_\mu$
```

```
proc. main // $G'_2$, $\boxed{G'_3}$
$\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 0$
$(pp', bp) \leftarrow_\$ \mathsf{init}$ ; $A \leftarrow_\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
For $k = 1, \ldots, q_{\mathcal{D}}$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow_\$ \mathcal{D}(\sigma_{k-1})$
$(\overline{S_0}, d) \leftarrow_\$ \mathcal{A}^{\mathrm{Sam}}(pp, (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
$(S_1, S_2, \mathsf{flag\text{-}ref}) \leftarrow \overline{S_0}$
For $i = 1, \ldots, d$
    $S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
    $\mathsf{flag\text{-}ref} \leftarrow 1$
If $\mathsf{flag\text{-}ref} = 1$
    $U \leftarrow_\$ \{0,1\}^n$
    $S'_1 \leftarrow_\$ \{0,1\}^n$
    $\boxed{S'_1 \leftarrow_\$ \mathsf{setup}(pp)}$
    $S_2 \leftarrow 0^p$
$(R^*, S_1^*) \leftarrow \mathsf{next}(pp'; S'_1)$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S}^* \leftarrow (S_1^*, S_2, \mathsf{flag\text{-}ref})$
$b^* \leftarrow_\$ \mathcal{A}(pp, R^*, \overline{S^*}, (I_k)_{k>\mu+d})$
Return $b^*$
```

```
proc. main // $G'_4$
$\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 0$
$(pp', bp) \leftarrow_\$ \mathsf{init}$ ; $A \leftarrow_\$ \{0,1\}^v$
$pp \leftarrow (pp', A)$
For $k = 1, \ldots, q_{\mathcal{D}}$
    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow_\$ \mathcal{D}(\sigma_{k-1})$
$(\overline{S_0}, d) \leftarrow_\$ \mathcal{A}^{\mathrm{Sam}}(pp, (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
    Return $\perp$
$(S_1, S_2, \mathsf{flag\text{-}ref}) \leftarrow \overline{S_0}$
For $i = 1, \ldots, d$
    $S_2 \leftarrow \mathsf{iterate}(S_2, I_i; A)$
    $\mathsf{flag\text{-}ref} \leftarrow 1$
If $\mathsf{flag\text{-}ref} = 1$
    $U \leftarrow_\$ \{0,1\}^n$
    $S'_1 \leftarrow_\$ \{0,1\}^n$
    $S'_1 \leftarrow_\$ \mathsf{setup}(pp)$
    $S_2 \leftarrow 0^p$
$R^* \leftarrow_\$ \{0,1\}^\ell$ ; $S_1^* \leftarrow_\$ \mathsf{setup}(pp)$
$\mathsf{flag\text{-}ref} \leftarrow 0$
$\overline{S}^* \leftarrow (S_1^*, S_2, \mathsf{flag\text{-}ref})$
$b^* \leftarrow_\$ \mathcal{A}(pp, R^*, \overline{S^*}, (I_k)_{k>\mu+d})$
Return $b^*$
```

Figure 5.14: Games for proof of Theorem 5.4.

BPRNG security. Analogously to our treatment of BPRGs (Section 5.3), we require that the PRNG $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ associated to a BPRNG $\mathsf{BPRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next}, \mathsf{BB})$ is secure in the face of an attacker who does not know the backdoor. Simultaneously, Big Brother should be able to exploit the backdoor to undermine security in some way. To make our models as strong as possible, we make minimal assumptions about Big Brother's influence, while allowing the non-backdoored attacker $\mathcal{A}$ — to whom the backdoored scheme must still appear secure — maximum power to compromise the scheme. To this end, we model BB as a passive observer who is able to capture just one BPRNG output which he is then challenged to exploit. At the same time, we demand that the BPRNG is provably secure in the face of a robustness adversary $\mathcal{A}$, with all the capabilities this allows. Notably, the latter condition allows us to explore the extent to which a guarantee of robustness may act as an immuniser against backdooring. When analysing BPRNGs, we present advantage terms for both regular robustness and backdoor attackers. As in the BPRG setting, we sometimes abuse notation to let BPRNG refer to the associated PRNG PRNG when the meaning is clear from the context.

Distribution samplers. When defining backdooring games, we do not allow BB any degree of compromise over the distribution sampler $\mathcal{D}$ from which the BPRNG draws

```
evolve(PRNG, $pp$, $S$, rp, $\mathcal{D}$)
$(a_1, b_1, \ldots, a_\rho, b_\rho) \leftarrow$ rp
$\phi \leftarrow (); \sigma \leftarrow \varepsilon$
For $i = 1 \ldots \rho$
  For $j = 1 \ldots a_i$
    $(R, S) \leftarrow \mathsf{next}(pp, S)$
    $\phi \leftarrow \phi \,\|\, (R, S)$
  For $k = 1 \ldots b_i$
    $(\sigma, I, \gamma, z) \leftarrow\$ \mathcal{D}(\sigma)$
    $S \leftarrow \mathsf{refresh}(pp, S, I)$
Return $\phi$
```

Figure 5.15: The evolve algorithm.

entropy samples. This is again to fit with our ethos of making minimal assumptions on BB's capabilities. Moreover, it strengthens the backdooring model by demanding that the backdoor be effective against *all* $(q_{\mathcal{D}}, \gamma^*)$-legitimate samplers $\mathcal{D}$, including those not under the control of BB. We also note that in the extreme case where BB has complete knowledge of all the inputs used in refresh calls, then BB's view of the evolution of the state is deterministic and the PRNG is reduced to a forward secure PRG which is periodically reseeded with correlated values. As such, this restriction on Big Brother's power ensures a clear separation between the results of Section 5.3 and those which follow.

Refresh patterns. Notice that the evolution of a PRNG state and the set of outputs produced is dictated by the sequence of refresh and next calls made. We call this sequence of calls a *refresh pattern*, and express it in the form $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$ where, for each i, a_i denotes a number of consecutive calls to next and b_i denotes a subsequent number of consecutive calls to refresh. We refer to the sequence of next calls indicated by each a_i as the i^{th} *refresh period.*

We define evolve to be the algorithm which takes as input a PRNG with input $\mathsf{PRNG} = (\mathsf{setup}, \mathsf{init}, \mathsf{refresh}, \mathsf{next})$, public parameter pp, an initial state S, a refresh pattern $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$, and a distribution sampler $\mathcal{D}$, and returns the set of outputs and states generated by executing the refresh pattern with the initial state and public parameters indicated. We give a formal pseudocode description of evolve in Figure 5.15. The output of evolve is a sequence $(R_1, S_1, \ldots, R_{q_R}, S_{q_R})$ of PRNG output and state pairs where $q_R = \sum_{i=1}^{\rho} a_i$. Based on evolve, we define an additional algorithm out, which takes the same input, runs evolve, and returns only the output values $(R_1, \ldots, R_{q_R})$.

Backdooring Models for BPRNGs. We present two new backdooring models for

```
$\text{BPRNG-STATE}^{\mathsf{BB}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{rp}, i, j)$
$(pp, bp) \leftarrow\!\!\$\ \mathsf{init}$
$S_0 \leftarrow\!\!\$\ \mathsf{setup}(pp)$
$(R_1, S_1, \ldots, R_{q_R}, S_{q_R}) \leftarrow \mathsf{evolve}(\mathsf{BPRNG}, pp, S_0, \mathsf{rp}, \mathcal{D})$
$S_j^* \leftarrow \mathsf{BB}(pp, bp, R_i, i, j, \mathsf{rp})$
Return $(S_j^* = S_j)$
```

```
$\text{BPRNG-OUT}^{\mathsf{BB}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{rp}, i, j)$
$(pp, bp) \leftarrow\!\!\$\ \mathsf{init}$
$S_0 \leftarrow\!\!\$\ \mathsf{setup}(pp)$
$(R_0, \ldots, R_{q_R}) \leftarrow \mathsf{out}(\mathsf{BPRNG}, pp, S_0, \mathsf{rp}, \mathcal{D})$
$R_j^* \leftarrow \mathsf{BB}(pp, bp, R_i, i, j, \mathsf{rp})$
Return $(R_j^* = R_j)$
```

Figure 5.16: Security games for backdoored PRNGs.

PRNGs with input in Figure 5.16. As we saw in Section 5.3, Big Brother's ability to exploit a backdoor in a PRNG may be affected by the sequence of $\mathsf{refresh}$ and next calls made. To reflect this, each backdooring game and corresponding advantage term is parameterised by a refresh pattern rp.

In the first game, the BPRNG is evolved according to the specified refresh pattern. Big Brother is given an output R_i, and challenged to recover state S_j. In the second game, Big Brother is again given output R_i, but now we ask him to recover an output value R_j. In both games, Big Brother is additionally given the refresh pattern. For a BPRNG $\mathsf{BPRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next}, \mathsf{BB})$, sampler $\mathcal{D}$, and refresh pattern rp, the advantage of BB in game BPRNG-TYPE $\in$ {BPRNG-STATE, BPRNG-OUT} against BPRNG is defined

$$\mathbf{Adv}^{\text{type}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{BB}, \mathsf{rp}, i, j) = \Pr\left[\text{BPRNG-TYPE}^{\mathsf{BB}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{rp}, i, j) \Rightarrow \mathsf{true}\right].$$

As with the corresponding BPRG definitions in Section 5.3.3, it is straightforward to verify that a BPRNG backdoored in the BPRNG-STATE sense is strictly stronger than one backdoored in the BPRNG-OUT sense.

Extensions. Stronger backdooring models can be achieved by considering games in which Big Brother is not given the refresh pattern. In Section 5.4.5, we will discuss how to extend our BPRNG construction (Section 5.4.3) to handle this case.

5.4.3 Construction of a Robust Backdoored PRNG

We now present our main result of the chapter: a construction of a provably robust BPRNG achieving our BPRNG-OUT backdooring notion.

Components. Consider $\mathsf{BPRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}}, \mathsf{BB})$ as shown in Figure 5.17. The construction is built from the following components:

- A (non-backdoored) PRNG with input $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ with associated parameters (ℓ, p) and state space $\mathcal{S} = \{0,1\}^n$. We require that PRNG is Pres and Rec secure.
- A pair of stateless PRGs (see Section 5.2): $\mathsf{G} : \{0,1\}^\ell \to \{0,1\}^{2k \cdot u + 1}$ and $\mathsf{G}' : \{0,1\}^u \to \{0,1\}^{k \cdot m}$.
- A re-randomisable PKE scheme $\mathsf{rPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ with public / secret key spaces $\mathcal{PK}$ / $\mathcal{SK}$, message space $\mathcal{M} \supseteq \{0,1\}^n$, coin space $\mathcal{R} = \{0,1\}^\ell$, ciphertext space $\mathcal{C} \subseteq \{0,1\}^*$, and Rand coin space $\mathcal{Z} = \{0,1\}^u$. We let $m = \mathsf{len}_{\mathsf{PKE}}(n)$, which by the length regularity of rPKE implies that $m = |\mathsf{Enc}(pk, M)|$ for all $pk \in \mathcal{PK}$ and $M \in \{0,1\}^n$. We say that a ciphertext C is *invalid* for a public key $pk \in \mathcal{PK}$ if for all $M \in \mathcal{M}$ and coins $r \in \mathcal{R}$ it holds that $C \neq \mathsf{Enc}(pk, M; r)$. We require that rPKE has an associated procedure $\mathsf{invalid}$, which takes as input a public key $pk \in \mathcal{PK}$ and a ciphertext C and outputs 1 if C is invalid for pk and 0 otherwise. It is straightforward to see that this can be instantiated for ElGamal and its encoded variants by checking that both ciphertext components are valid group elements.

Combining these components yields a BPRNG with associated parameters $(k \cdot m, p)$ and state space $\mathcal{S} = \{0,1\}^{k \cdot m + n + 1}$. We write the algorithms, outputs, and states of the backdoored PRNG with an overline to distinguish these from those of the underlying PRNG.

Construction overview. The construction is based on the simple idea of interleaving outputs of the underlying PRNG PRNG with encrypted snapshots of its state. By taking a snapshot of this PRNG state whenever it is refreshed and storing a list of the previous k snapshots, the construction will enable BB to recover, with reasonable probability, output values that were computed in the previous k refresh periods. Of course, this means that the state of the final construction is large compared to that of the PRNG used as a component in its construction.

Algorithms. We now give an overview of each of the BPRNG component algorithms shown in Figure 5.17. Parameter generation via $\overline{\mathsf{init}}$ generates public parameters for the

```
$\overline{\mathsf{init}}$
$(pp, \perp) \leftarrow\$ \mathsf{init}$
$(pk, sk) \leftarrow\$ \mathsf{Key}$
$\overline{pp} \leftarrow (pp, pk)$
$bp \leftarrow sk$
Return $(\overline{pp}, bp)$

$\overline{\mathsf{setup}}(\overline{pp})$
$(pp, pk) \leftarrow \overline{pp}$
$S_0 \leftarrow\$ \mathsf{setup}(pp)$
$C[1] \leftarrow\$ \mathsf{Enc}(pk, S_0)$
For $i = 2, \ldots, k$
    $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
$\psi \leftarrow 0$
$\overline{S}_0 \leftarrow (S_0, C[1], \ldots, C[k], \psi)$
Return $\overline{S}_0$

$\overline{\mathsf{refresh}}(\overline{pp}, \overline{S}, I)$
$(pp, pk) \leftarrow \overline{pp}$
$(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}$
$S \leftarrow \mathsf{refresh}(pp, S, I)$
$\psi \leftarrow 1$
$\overline{S} \leftarrow (S, C[1], \ldots, C[k], \psi)$
Return $\overline{S}$
```

```
$\overline{\mathsf{next}}(\overline{pp}, \overline{S})$
1. $(pp, pk) \leftarrow \overline{pp}$
2. $(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}$
3. For $i = 1, \ldots, k$
4.     If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
5.         $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
6. If $\psi = 1$
7.     $(R, S) \leftarrow \mathsf{next}(pp, S)$
8.     $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$
9.     $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
10. $(R, S) \leftarrow \mathsf{next}(pp, S)$
11. $(\eta, R_1, \ldots, R_{2k}) \leftarrow \mathsf{G}(R)$
12. If $\eta = 0$
13.     $\overline{R} \leftarrow \mathsf{G}'(R_1)$
14. Else
15.     For $i = 1, \ldots, k$
16.         $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
17.     $\overline{R} \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
18. For $i = 1, \ldots, k$
19.     $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
20. $\psi \leftarrow 0$
21. $\overline{S} \leftarrow (S, C[1], \ldots, C[k], \psi)$
22. Return $(\overline{R}, \overline{S})$
```

```
$\mathsf{BB}(\overline{pp}, bp, \overline{R}_i, i, j, \mathsf{rp})$
If $i = j$
    Return $\overline{R}_i$
$(pp, pk) \leftarrow \overline{pp}$ ; $sk \leftarrow bp$
$(a_1, b_1, \ldots, a_\rho, b_\rho) \leftarrow \mathsf{rp}$
$(C[1], \ldots, C[k]) \leftarrow R_i$
$i_{ref} \leftarrow \min_\sigma \left[\sum_{\nu=1}^{\sigma} a_\nu \geq i\right]$
$j_{ref} \leftarrow \min_\sigma \left[\sum_{\nu=1}^{\sigma} a_\nu \geq j\right]$
If $j_{ref} > i_{ref}$ or $i_{ref} - j_{ref} \geq k$
    Return $\perp$
$S \leftarrow \mathsf{Dec}(sk, C[(i_{ref} - j_{ref} + 1)])$
$(R, S_0) \leftarrow \mathsf{next}(pp, S)$
$j_{it} \leftarrow j - \sum_{\nu=1}^{j_{ref}-1} a_\nu$
For $z = 1, \ldots, j_{it}$
    $(R_z, S_z) \leftarrow \mathsf{next}(pp, S_{z-1})$
$(\eta, R'_1, \ldots, R'_{2k}) \leftarrow \mathsf{G}(R_{j_{it}})$
$\overline{R}^* \leftarrow \mathsf{G}'(R'_1)$
Return $\overline{R}^*$
```

Figure 5.17: Construction of a robust BPRNG $\mathsf{BPRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}}, \mathsf{BB})$, using as components a re-randomisable PKE scheme $\mathsf{rPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$, a PRNG with input $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$, and stateless PRGs G and G'.

underlying PRNG PRNG and a public / secret key pair (pk, sk) for the re-randomisable PKE scheme rPKE. The backdoor parameter of BPRNG is set equal to the secret key for PKE; the other parameters form the public parameters of the scheme. To construct the initial state, $\overline{\mathsf{setup}}$ first generates a state $S_0 \leftarrow\$ \mathsf{setup}(pp)$ for the underlying PRNG, and outputs $\overline{S}_0 = (S_0, C[1], \ldots, C[k], \psi)$, where $C[1]$ is an encryption of S_0, and $C[2], \ldots, C[k]$ are encryptions of 0^n, under $\mathsf{Enc}(pk, \cdot)$. Here ψ is a flag which is initialised to 0. Looking ahead, the state components $C[1], \ldots, C[k]$ will correspond to encrypted snapshots of the state of PRNG during the previous k refresh periods. The flag ψ will be used to indicate to $\overline{\mathsf{next}}$ that a new refresh period has been entered and a new encrypted snapshot should be taken. On input $(\overline{pp}, \overline{S}, I)$ where $\overline{S} = (S, C[1], \ldots, C[k], \psi)$, $\overline{\mathsf{refresh}}$ simply uses the $\mathsf{refresh}$ algorithm of the underlying PRNG to update state component S with input I, and sets the flag ψ to 1.

Output generation. The specification of $\overline{\mathsf{next}}$ is more complex. On input $\overline{pp} = (pp, pk)$ and $\overline{S} = (S, C[1], \ldots, C[k], \psi)$, $\overline{\mathsf{next}}$ first uses $\mathsf{invalid}$ to check if any of the encrypted snapshots correspond to an invalid ciphertext (lines 3-5). If so, the ciphertext is replaced by an encryption of 0^n using $0^\ell \in \mathcal{R}$ as coins (recall that $\overline{\mathsf{next}}$ is deterministic and so cannot generate randomised encryptions). Looking ahead, this is required for the proof of recovering security in which the attacker may set the value of the BPRNG state; since there is no guarantee that the re-randomisation properties of rPKE hold for invalid ciphertexts, we remove these from the state. With this in place, $\overline{\mathsf{next}}$ then uses ψ to check if a new refresh period has been entered (line 6). If so, a new encrypted snapshot of the underlying PRNG state is produced by generating an output / state pair $(R, S) \leftarrow \mathsf{next}(pp, S)$ and setting $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$ and $C[i] \leftarrow C[i-1]$ for $i \in [1, k]$ (lines 7-9). Notice how the k^{th} encrypted snapshot is discarded during this process.

Outputs for BPRNG are produced in two different ways; one of which leaks the encrypted snapshots stored in the state to Big Brother, and the other in a way which can be replicated by Big Brother given knowledge of the appropriate encrypted snapshot. In more detail, $\overline{\mathsf{next}}$ uses PRNG to generate an output / state pair (R, S), and expands R using the stateless PRG G, parsing the resulting string as $(\eta, R_1, \ldots, R_{2k})$ where $\eta \in \{0,1\}$ and $R_i \in \{0,1\}^u$ for $i \in [1, 2k]$ (lines 10-11). The bit η is used to determine which method of output production will be taken. If $\eta = 0$, then the BPRNG output is simply computed as $\overline{R} \leftarrow \mathsf{G}'(R_1)$ (lines 12-13). If $\eta = 1$, $\overline{R}$ is constructed using encrypted snapshots as follows. Firstly, each $C[i]$ is re-randomised using R_i as coins via $C[i] \leftarrow \mathsf{Rand}(pk, C[i]; R_i)$ for $i \in [1, k]$. The BPRNG output is set to the concatenation of these ciphertexts, $\overline{R} = C[1] \,\|\, \ldots \,\|\, C[k]$ (lines 14-17). To compute the updated state, the remaining random strings R_{k+i} are used to further re-randomise each $C[i]$ for $i \in [1, k]$, and the flag ψ is set to 0 (lines 18-21). This second re-randomisation ensures that the ciphertexts in the state appear independent from those used to produce the output $\overline{R}$.

Big Brother. For the Big Brother algorithm BB in the construction to be successful, it is required that: **(1)** the output value $\overline{R}_i$ given to BB corresponds to $(C[1], \ldots, C[k])$; and **(2)** the output value $\overline{R}_j$ that BB is required to recover corresponds to a value computed via G'. Since the type of output produced is decided from the output of PRNG and G which are both assumed to be good generators, this will happen with probability close to 1/4. Furthermore, it is required that the number of refresh periods between $\overline{R}_j$ and $\overline{R}_i$ is less than k. More precisely, for a refresh pattern $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$, Big Brother first computes the refresh periods i_{ref} and j_{ref} in which, respectively, the challenge and target outputs R_i and R_j were generated via $i_{ref} = \min_\sigma[\sum_{\nu=1}^{\sigma} a_\nu \geq i]$ and $j_{ref} = \min_\sigma[\sum_{\nu=1}^{\sigma} a_\nu \geq j]$. If $i_{ref} - j_{ref} \in [0, k-1]$, then the initial refreshed state used to compute $\overline{R}_j$ will be encrypted in $C[i_{ref} - j_{ref} + 1]$. Hence, all BB has to do is to decrypt and iterate this state $j_{it} + 1$ times where $j_{it} = j - \sum_{\nu=1}^{j_{ref}-1} a_\nu$ to obtain the seed used to compute $\overline{R}_j$. If the condition on i_{ref} and j_{ref} is not met then BB aborts and returns $\perp$.

5.4.4 Security Analysis

In this section we analyse the security properties of BPRNG. To prove Rob security, we first prove the Pres and Rec security of BPRNG and then invoke Theorem 2.2. We require that rPKE satisfies a stronger re-randomisation property than that introduced in Section 5.2: namely, that the re-randomisation of an adversarially chosen ciphertext is indistinguishable from the encryption of an adversarially chosen message. In particular, there is no requirement that the ciphertext is generated using e.g., good randomness, or that the message output by $\mathcal{A}$ matches that encrypted in C^*. We formalise this property in the following definition.

Definition 5.5. *Let* $\mathsf{rPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ *be a re-randomisable PKE scheme with message space* $\mathcal{M}$*. For an attacker* $\mathcal{A}$*, the strong re-randomisability advantage of* rPKE *is defined*

$$\mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE}}(\mathcal{A}) = \Big| \Pr\big[b = b' \ : \ (pk, sk) \leftarrow_{\$} \mathsf{Key}; b \leftarrow_{\$} \{0,1\}; (C^*, M^*) \leftarrow_{\$} \mathcal{A}(pk); \\ C^0 \leftarrow_{\$} \mathsf{Rand}(pk, C^*); C^1 \leftarrow_{\$} \mathsf{Enc}(pk, M^*); b' \leftarrow_{\$} \mathcal{A}(C^b)\big] - 1/2 \Big| \, .$$

We require that $M^* \in \mathcal{M}$ *and that the ciphertext* C^* *output by* $\mathcal{A}$ *is valid (i.e.,* $\mathsf{invalid}(pk, C^*) = 0$*).*

Discussion and instantiation. We note that this property is not immediately comparable to the re-randomisation definition for PKE given in Section 5.2. That was a statistical notion concerning encryptions of the same message, while in contrast the following is a computational notion regarding possibly different messages. However, we will show in the following lemma that ElGamal is strongly re-randomisable provided the DDH problem is hard with respect to ElGamal's group generation algorithm GGen. As such, an appropriately encoded variant of ElGamal may be used to instantiate the scheme.

Lemma 5.1. *Let* $\mathsf{ElG.rPKE} = (\mathsf{ElG.Key}, \mathsf{ElG.Enc}, \mathsf{ElG.Dec}, \mathsf{ElG.Rand})$ *be the ElGamal re-randomisable PKE scheme with associated group generation algorithm* GGen*. Then for any attacker* $\mathcal{A}$ *running in time* T*, there exists an attacker* $\mathcal{B}$ *running in time* $T' \approx T$

such that

$$\mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE.ElG}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ddh}}_{\mathsf{GGen}}(\mathcal{B}) \ .$$

Proof. We argue by a series of game hops. Let G_0 be equivalent to the strong re-randomisability game against $\mathsf{ElG.rPKE}$ for $\mathcal{A}$ with challenge bit $b = 0$. Since by assumption the ciphertext C^* output by $\mathcal{A}$ is valid for $pk = (\mathbb{G}, q, g, g^x)$, it must be the case that there exists a message $M \in \mathbb{G}$ and $y' \in \mathbb{Z}_q$ such that $C^* = (C_1^*, C_2^*) = (g^{y'}, M \cdot g^{xy'})$, and so $(C_1^*, C_2^*) \in \mathbb{G} \times \mathbb{G}$. Now with this notation the ciphertext $C^0 \leftarrow\$ \ \mathsf{ElG.Rand}(pk, C^*)$ which is given to $\mathcal{A}$ for his challenge is of the form $C^0 = (C_1^0, C_2^0) = (C_1^* \cdot g^y, C_2^* \cdot g^{xy})$ for $y \leftarrow\$ \ \mathbb{Z}_q$.

Next we define game G_1, which is identical to G_0 except we generate the C_2^0 component of C^0 by multiplying C_2^* by a random group element, $C_2^0 \leftarrow C_2^* \cdot g^z$ where $z \leftarrow\$ \ \mathbb{Z}_q$. It is straightforward to verify that both G_0 and G_1 can be perfectly simulated by an attacker $\mathcal{B}_1$ in the DDH game against GGen running in time $T' \approx T$, who on input $(\mathbb{G}, q, g, g^x, g^y, Z)$ proceeds as follows. $\mathcal{B}_1$ first passes $pk = (\mathbb{G}, q, g, g^x)$ to $\mathcal{A}$. When $\mathcal{A}$ outputs $((C_1^*, C_2^*), M)$, $\mathcal{B}_1$ sets $C^0 = (C_1^* \cdot g^y, C_2^* \cdot Z)$ and passes this to $\mathcal{A}$. At the end of the game, $\mathcal{B}$ outputs whatever bit $\mathcal{A}$ does. It is straightforward to verify that if $\mathcal{A}$ receives $Z = g^{xy}$ in his challenge then this perfectly simulates G_0, whereas if $Z = g^z$ for $z \leftarrow\$ \mathbb{Z}_q$ then this perfectly simulates G_1. It follows that

$$|\Pr[\, G_0 \Rightarrow 1\,] - \Pr[\, G_1 \Rightarrow 1\,]| \leq \mathbf{Adv}^{\text{ddh}}_{\mathsf{GGen}}(\mathcal{B}_1) \ .$$

Next we define game G_2 to be identical to G_1, except we replace C^0 with a pair of random group elements, $C^0 = (g^y, g^z)$ for $y, z \leftarrow\$ \ \mathbb{G}$. Recall that in G_1, $C^0 = (C_1^* \cdot g^y, C_2^* \cdot g^z)$ where $y, z \leftarrow\$ \ \mathbb{Z}_q$ and $(C_1^*, C_2^*) \in \mathbb{G} \times \mathbb{G}$. Since multiplying an arbitrary element of a cyclic group by a random group element is equivalent to sampling uniformly from the group, it follows that these games are identically distributed, and

$$\Pr[\, G_1 \Rightarrow 1\,] = \Pr[\, G_2 \Rightarrow 1\,] \ .$$

Next we define game G_3, which is identical to G_2 except we now set C_2^0 equal to $M^* \cdot g^z$ for $z \leftarrow\$ \mathbb{Z}_q$. Since by definition $M^* \in \mathcal{M} = \mathbb{G}$, an analogous argument to that made above implies that the games are identically distributed, and

$$\Pr[\, G_2 \Rightarrow 1\,] = \Pr[\, G_3 \Rightarrow 1\,] \ .$$

Next we define game G_4, which is identical to G_3 except we set $C^0 = (g^y, M^* \cdot g^{xy})$ for $y \leftarrow\$ \mathbb{Z}_q$, where recall that $pk = (\mathbb{G}, q, g, g^x)$. An entirely analogous argument to that made previously implies that both games can be perfectly simulated by an attacker $\mathcal{B}_2$ in the DDH game against GGen, implying that

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \mathbf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}_2) .$$

Moreover, notice that G_4 is equivalent to the strong re-randomisability game for $\mathsf{ElG.rPKE}$ with challenge bit $b = 1$. Combining the above observations via a standard argument implies that

$$\mathbf{Adv}^{\mathrm{s\text{-}rerand}}_{\mathsf{rPKE.ElG}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{ddh}}_{\mathsf{GGen}}(\mathcal{B}) ,$$

thereby concluding the proof.

□

Security of BPRNG. With this in place, the security of BPRNG is captured in the following theorem. The proof follows from a number of lemmas, presented below, combined with Theorem 2.2.

Theorem 5.5. *Let* $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ *be a PRNG with associated parameters* (ℓ, p) *and state space* $\mathcal{S} = \{0,1\}^n$. *Let* $\mathsf{rPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ *be a re-randomisable PKE scheme* $\mathsf{rPKE} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Rand})$ *with public / secret key spaces* $\mathcal{PK}$ / $\mathcal{SK}$, *message space* $\mathcal{M} \supseteq \{0,1\}^n$, *coin space* $\mathcal{R} = \{0,1\}^\ell$, *ciphertext space* $\mathcal{C} \subseteq \{0,1\}^*$, *and* Rand *coin space* $\mathcal{Z} = \{0,1\}^u$. *We let* $\mathsf{len}_{\mathsf{PKE}}(n) = m$, *and require that* rPKE *has an associated ciphertext validity procedure* $\mathsf{invalid}$. *Let* $\mathsf{G} : \{0,1\}^\ell \to \{0,1\}^{2k\cdot u+1}$ *and* $\mathsf{G}' : \{0,1\}^u \to \{0,1\}^{k\cdot m}$ *be stateless PRGs.*

Let $\mathsf{BPRNG} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}}, \mathsf{BB})$ *be as shown in Figure 5.17, with associated parameters* $(k \cdot m, p)$ *and state space* $\overline{\mathcal{S}} = \{0,1\}^{n+k\cdot m+1}$. *Then for all refresh patterns* $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$, *all distribution samplers* $\mathcal{D}$, *and for all* $1 \leq i, j \leq \sum_{\nu=1}^{\rho} a_\nu = q_R$, *there exist efficient adversaries* $\mathcal{B}$, $\mathcal{C}$ *such that* $\mathbf{Adv}^{\mathrm{bprng\text{-}out}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{BB}, \mathsf{rp}, i, j) \geq \delta(\mathsf{rp}, i, j)$, *where*

$$\delta(\mathsf{rp}, i, j) = \begin{cases} 1 & \textit{if } j = i \text{ ; } \textit{else} \\ \frac{1}{4} - \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{B}, \mathcal{D}) - 2 \cdot \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}}(\mathcal{C}) & \textit{if } i_{ref} - j_{ref} \in [0, k-1] \\ 0 & \textit{otherwise} \end{cases}$$

Here $i_{ref} \leftarrow \min_\sigma [\sum_{\nu=1}^{\sigma} a_\nu \geq i]$ *and* $j_{ref} \leftarrow \min_\sigma [\sum_{\nu=1}^{\sigma} a_\nu \geq j]$. $\mathcal{B}$ *is a* $(q'_R, q_{\mathcal{D}}, q_C, q_S)$*-adversary where* $q_{\mathcal{D}} = \sum_{i=1}^{\rho} b_i$, $q_C = \max_{1 \leq i \leq \rho} b_i$, $q'_R = q_R + \rho - 1$, *and* $q_S = 0$.

Moreover, for any $(q_R, q_\mathcal{D}, q_C, q_S)$-attacker $\mathcal{A}$ running in time T and any $(q_\mathcal{D}, \gamma^*)$-legitimate sampler $\mathcal{D}$ in game Rob against $\overline{\mathsf{PRNG}} = (\overline{\mathsf{init}}, \overline{\mathsf{setup}}, \overline{\mathsf{refresh}}, \overline{\mathsf{next}})$, there exist attackers $\mathcal{B}$, $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{E}$, $\mathcal{E}'$, $\mathcal{F}$, $\mathcal{G}$, running in time $T' \approx T$, such that

$$\begin{aligned}\mathbf{Adv}^{\mathrm{rob}}_{\overline{\mathsf{PRNG}},\gamma^*,q_\mathcal{D}}(\mathcal{A},\mathcal{D}) \le 2q_R \cdot \Big(&3 \cdot \mathbf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{B},1) + \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{C}_1) + 2 \cdot \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2) \\ &+ \mathbf{Adv}^{\mathrm{rec}}_{\mathsf{PRNG},\gamma^*,q_\mathcal{D}}(\mathcal{C}_3,\mathcal{D}) + 2 \cdot \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}}(\mathcal{E}) + 2 \cdot \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}}(\mathcal{E}') \\ &+ (3k-1) \cdot \Delta_{\mathsf{rPKE}} + (k+1) \cdot \mathbf{Adv}^{\mathrm{srerand}}_{\mathsf{rPKE}}(\mathcal{F}) + 2 \cdot \mathbf{Adv}^{\mathrm{ind\$\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{G},k)\Big) \,.\end{aligned}$$

Here, $\mathcal{C}_1$ and $\mathcal{C}_3$ are q_C-attackers and $\mathcal{C}_2$ is a 0-attacker.

Preserving security. We begin by analysing the Pres security of BPRNG. The proof uses a reduction to the Pres security of the underlying PRNG PRNG to argue that the output / state pair (R, S) on line 7 of $\overline{\mathsf{next}}$ can be replaced with a random output $R \leftarrow_\$ \{0,1\}^\ell$ and fresh state $S \leftarrow_\$ \mathsf{setup}(pp)$. In order for an attacker $\mathcal{C}$ in game Pres against PRNG to be able to simulate the games required for the reduction, we must first use the IND-CPA security of rPKE to argue that the encryption of the initial state of PRNG — unknown to $\mathcal{C}$ and encrypted in component $C[0]$ of the state of BPRNG — can be replaced with an encryption of 0^n. With this in place, a series of game hops sequentially argues that each output produced by PRNG or the PRGs G, G' during output production in $\overline{\mathsf{next}}$ can be replaced with a random string. This then allows us to argue, via properties of rPKE, that the output / state pair received by $\mathcal{A}$ in their challenge may be replaced by their idealised counterparts.

Lemma 5.2. *Let $\overline{\mathsf{PRNG}}$ be as in Theorem 5.5. Then for any q_C-attacker $\mathcal{A}$ running in time T in game Pres against $\overline{\mathsf{PRNG}}$, there exist attackers $\mathcal{B}, \mathcal{C}_1, \mathcal{C}_2, \mathcal{E}, \mathcal{E}', \mathcal{F}$, and $\mathcal{G}$ running in time $T' \approx T$ such that*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{pres}}_{\overline{\mathsf{PRNG}}}(\mathcal{A}) \le 2 \cdot \mathbf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{B},1) + \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{C}_1) + \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2) + \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}}(\mathcal{E}) \\ + \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}'}(\mathcal{E}') + (2k-1) \cdot \Delta_{\mathsf{rPKE}} + \mathbf{Adv}^{\mathrm{s\text{-}rerand}}_{\mathsf{rPKE}}(\mathcal{F}) + \mathbf{Adv}^{\mathrm{ind\$\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{G},k) \,.\end{aligned}$$

Here, $\mathcal{C}_1$ is a q_C-attacker and $\mathcal{C}_2$ is a 0-attacker.

Proof. We argue by a series of game hops, shown in Figures 5.18 and 5.19. We begin by defining game G_0, which is readily verified to be identical to game Pres against $\overline{\mathsf{PRNG}}$

which challenge bit $b = 0$. Next we define game G_1, which is identical to G_0 except we change the way in which the initial state $\overline{S}$ is generated. Namely, rather than setting state component $C[1] \leftarrow\$ \ \mathsf{Enc}(pk, S)$ in line 3, where $S \leftarrow\$ \ \mathsf{setup}(pp)$ is the initial state of the underlying PRNG, we instead set $C[1] \leftarrow\$ \ \mathsf{Enc}(pk, 0^n)$. We claim that there exists an attacker $\mathcal{B}_1$ running in time $T' \approx T$ such that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_0 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rPKE}}(\mathcal{B}_1, 1) .$$

To see this, notice that both games can be perfectly simulated by an attacker in game IND-CPA against rPKE who proceeds as follows. On input pk, $\mathcal{B}_1$ generates public parameters $pp \leftarrow\$ \ \mathsf{init}$ and passes $\overline{pp} = (pp, pk)$ to $\mathcal{A}$. $\mathcal{B}_1$ generates initial state $S \leftarrow\$ \ \mathsf{setup}(pp)$ for PRNG, and makes a single query of $(S, 0^n)$ to his LoR oracle, setting $C[1]$ equal to the returned ciphertext. $\mathcal{B}_1$ sets the initial state of BPRNG equal to $\overline{S}_0 = (S, C[1], \mathsf{Enc}(pk, 0^n), \ldots, \mathsf{Enc}(pk, 0^n), 0)$ and uses this to simulate the rest of the game for $\mathcal{A}$ as per the pseudocode description, finally outputting whatever bit $\mathcal{A}$ does. A straightforward argument then implies the claim.

Next we define game G_2, which is identical to G_1 except we replace $(R, S) \leftarrow \mathsf{next}(pp, S)$ (line 15 in G_1) with a random output and freshly generated state, $R \leftarrow\$ \ \{0,1\}^\ell$ and $S \leftarrow\$ \ \mathsf{setup}(pp)$ (line 14 in G_2). We claim that there exists a q_C-attacker $\mathcal{C}_1$ running in time $T' \approx T$ such that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{pres}}_{\mathsf{PRNG}}(\mathcal{C}_1) .$$

Notice that if $\mathcal{A}$ outputs ε at the start of their challenge then the modified 'If $\psi = 1$' loop is not executed and so the games are identical. Therefore, we may without loss of generality assume that $\mathcal{A}$ outputs $(I_1, \ldots, I_d)$ for $d \geq 1$. Let $\mathcal{C}_1$ be the attacker who proceeds as follows. On input pp, $\mathcal{C}_1$ generates a public / secret key pair via $(pk, sk) \leftarrow\$ \ \mathsf{Key}$ and passes (pp, pk) to $\mathcal{A}$. $\mathcal{C}_1$ constructs a partial initial state for BPRNG as $(\cdot, C[1], \ldots, C[k], 0)$ where $C[i] \leftarrow\$ \ \mathsf{Enc}(pk, 0^n)$ for $i \in [1, k]$ and the missing first component is implicitly taken to be the unknown initial state S in $\mathcal{C}_1$'s challenge. When $\mathcal{A}$ outputs a tuple $(I_1, \ldots, I_d)$, $\mathcal{C}_1$ outputs the same tuple to his own challenger. $\mathcal{C}_1$ receives (R, S) in response, inserts these at line 15 in G_1, and continues running the game, finally outputting whatever bit $\mathcal{A}$ does. It is straightforward to verify that if $\mathcal{C}_1$ receives a real output / state in his challenge then this perfectly simulates G_1; otherwise it perfectly simulates G_2, thereby implying the claim.

Next we define game G_3, which is identical to G_2 except we replace $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$

(line 15) with $C[0] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$. Recalling that $R \leftarrow\$ \{0,1\}^\ell$ in G_2 where rPKE's coin space is $\mathcal{R} = \{0,1\}^\ell$, an analogous argument to that made above implies that there exists an attacker $\mathcal{B}_2$ running in time $T' \approx T$ and who makes a single query of $(S, 0^n)$ to his LoR oracle, such that

$$|\Pr[G_3 \Rightarrow 1] - \Pr[G_2 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rPKE}}(\mathcal{B}_2, 1) .$$

Next we define game G_4, which is identical to G_3 except we replace the line $(R, S) \leftarrow \mathsf{next}(pp, S)$ (line 18 in G_3) with $R \leftarrow\$ \{0,1\}^\ell$; $S \leftarrow\$ \mathsf{setup}(pp)$ (line 17 in G_4). We claim that there exists a 0-attacker $\mathcal{C}_2$ in game Pres against PRNG running in time $T' \approx T$ such that

$$|\Pr[G_4 \Rightarrow 1] - \Pr[G_3 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2) .$$

To see this, notice that due to a previous transition the output / state pair in this line are computed by applying next to pp and a freshly generated state $S \leftarrow\$ \mathsf{setup}(pp)$ (i.e., from line 14 in G_3 if the 'If $\psi = 1$' loop is executed, and line 2 otherwise). Let $\mathcal{C}_2$ proceed as follows. $\mathcal{C}_2$ generates parameters and initial state as described in the previous reduction to Pres security and passes (pp, pk) to $\mathcal{A}$. If $\mathcal{A}$ outputs $(I_1, \ldots, I_d)$ with $d \geq 1$, then $\mathcal{C}_2$ ignores the inputs that $\mathcal{A}$ outputs, and sets $C[0] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$, followed by $C[k] \leftarrow C[k-1]; \ldots; C[1] \leftarrow C[0]$. $\mathcal{C}_2$ then outputs ε to their challenger, receiving (R, S) in response. $\mathcal{C}_2$ substitutes these values into line 18 of G_3 and continues running the game. At the end of the game, $\mathcal{C}_2$ outputs whatever bit $\mathcal{A}$ does. It is straightforward to verify that if $\mathcal{C}_2$ receives a real output / state in his challenge this perfectly simulates G_3; otherwise it perfectly simulates G_4, thereby implying the claim.

Next we define game G_5, which is identical to G_4 except we replace the output of the stateless PRG G on line 18 with a truly random string of appropriate length. Since the PRG seed R is chosen uniformly at random in both games (line 17), a straightforward reduction to the stateless PRG security of G implies that there exists an attacker $\mathcal{E}$ running in time $T' \approx T$ such that

$$|\Pr[G_5 \Rightarrow 1] - \Pr[G_4 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{E}) .$$

Next we define G_6 to be identical to G_5, except we replace the output of the stateless PRG G' (line 20 in G_5) with a random string of appropriate length. Due to the previous transition, G' takes as input as truly random seed R_1 (line 18) in G_5, and so an analogous reduction to the PRG security of G' implies that there exists an attacker $\mathcal{E}'$ running in time $T' \approx T$ such that

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}'}(\mathcal{E}') .$$

Notice that in G_6, $\mathcal{A}$ will receive a random output $\overline{R}^* \leftarrow_\$ \{0,1\}^{k \cdot m}$ in his challenge if $\eta = 0$. Next we define game G_7, which is identical to G_6 except we replace the re-randomised ciphertexts $C[i] \leftarrow \mathsf{Rand}(pk, C[i]; R_i)$ for $i \in [1, k]$ (line 22) with fresh encryptions $C[i] \leftarrow_\$ \mathsf{Enc}(pk, 0^n)$. We claim that

$$|\Pr[G_7 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq k \cdot \Delta_{\mathsf{rPKE}} .$$

To see this, notice that if $\eta = 0$ then the modified code is not executed, and so we may assume without loss of generality that $\eta = 1$. Now in both games each of the ciphertexts to be re-randomised corresponds to a fresh randomised encryption of 0^n, and each R_i corresponds to a random string drawn from the coin space of Rand. As such, invoking the re-randomisability of rPKE and taking a hybrid argument over the k ciphertexts implies the claim.

Next we define game G_8, which is identical to G_7 except we replace the re-randomised ciphertexts $C[i] \leftarrow \mathsf{Rand}(pk, C[i]; R_{k+i})$ for $i \in [1, k]$ in line 26 of G_7 with fresh encryptions: $C[i] \leftarrow_\$ \mathsf{Enc}(pk, 0^n)$ for $i \in [2, k]$, and $C[1] \leftarrow_\$ \mathsf{Enc}(pk, S)$ where $S \leftarrow_\$ \mathsf{setup}(pp)$ in line 17. We claim that there exists an attacker $\mathcal{F}$ running in $T' \approx T$ such that

$$|\Pr[G_8 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq (k-1) \cdot \Delta_{\mathsf{rPKE}} + \mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE}}(\mathcal{F}) .$$

To see this, notice that by the previous game hop each $C[i]$ which is re-randomised is an honestly generated encryption of 0^n and so a valid ciphertext. A straightforward reduction to the strong re-randomisability of rPKE then implies that we can replace $C[1] \leftarrow \mathsf{Rand}(pk, C[1]; R_{k+i})$ with $C[1] \leftarrow_\$ \mathsf{Enc}(pk, S)$, with the transition upper bounded by $\mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE}}(\mathcal{F})$. (To see this, let $\mathcal{F}$ be the attacker who simulates the game as in G_7, but generates $C[1]$ on line 26 by submitting (C^*, S) to their challenger, where $C^* \leftarrow_\$ \mathsf{Enc}(pk, 0^n)$ and S is as generated in line 17, and inserts the response as $C[1]$.) Moreover, the re-randomisability of rPKE implies that each $C[i]$ for $i \in [2, k]$ is Δ_{rPKE} close to a fresh encryption of 0^n; taking a hybrid argument over each of the $(k-1)$ ciphertexts then implies the claim. Moreover, notice that the structure of the challenge state $\overline{S}^*$ is now identical to that of a fresh state output by $\overline{\mathsf{setup}}(pp)$.

Next we define game G_9, which is identical to G_8 except we replace each ciphertext $C[i] \leftarrow_\$ \mathsf{Enc}(pk, 0^n)$ for $i \in [1, k]$ (line 22) with a random bit string $C[i] \leftarrow_\$ \{0,1\}^m$. A straightforward reduction to the IND\$-CPA security of rPKE implies that there exists an attacker $\mathcal{G}$ running in time $T' \approx T$ and who makes k queries of 0^n to his RoR oracle such that

$$|\Pr[G_9 \Rightarrow 1] - \Pr[G_8 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rPKE}}(\mathcal{G}, k) .$$

In particular, notice that in G_9 the output $\overline{R}^*$ given to $\mathcal{A}$ in his challenge is equivalent to a string drawn uniformly from $\{0,1\}^{k \cdot m}$ regardless of whether $\eta = 0$ or $\eta = 1$. Finally, we define game G_{10} (not shown), which reverses the modifications made in G_1 to return to computing $C[1]$ on line 3 as per the protocol. However, since $\mathcal{A}$'s view of the game is now independent of this value, it follows that these games are identically distributed and

$$\Pr[G_{10} \Rightarrow 1] = \Pr[G_9 \Rightarrow 1] .$$

Moreover, since $\mathcal{A}$ always receives a random output $\overline{R}^* \leftarrow_\$ \{0,1\}^{k \cdot m}$ and freshly generated state $\overline{S}^* \leftarrow_\$ \overline{\mathsf{setup}}(pp)$ in his challenge, it is straightforward to verify that G_{10} is equivalent to game Pres against $\overline{\mathsf{PRNG}}$ with challenge bit $b = 1$. Combining the above via a standard argument implies that there exist attackers $\mathcal{B}, \mathcal{C}_1, \mathcal{C}_2, \mathcal{E}, \mathcal{E}', \mathcal{F}, \mathcal{G}$, with the claimed run times and query budgets, such that

$$\begin{aligned}\mathbf{Adv}^{\text{pres}}_{\overline{\mathsf{PRNG}}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rPKE}}(\mathcal{B}, 1) + \mathbf{Adv}^{\text{pres}}_{\mathsf{PRNG}}(\mathcal{C}_1) + \mathbf{Adv}^{\text{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2) + \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{E}) \\ + \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}'}(\mathcal{E}') + (2k-1) \cdot \Delta_{\mathsf{rPKE}} + \mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE}}(\mathcal{F}) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rPKE}}(\mathcal{G}, k) ,\end{aligned}$$

thereby concluding the proof of Pres security.

□

Recovering security. In the following lemma, we analyse the Rec security of BPRNG. The proof is similar to that of Pres security. A natural difference is that we appeal to the Rec security of PRNG to argue that (R, S) on line 7 of $\overline{\mathsf{next}}$ can be replaced with a random output and freshly generated state. Interestingly, despite working in the Rec setting, we still require a reduction to the Pres security of PRNG to argue that (R, S) on line 10 can be replaced with their idealised counterparts. This is because this pair is produced by applying $\mathsf{next}(pp, \cdot)$ to a state which, depending on the value of state component ψ, is either freshly generated or has been output by next; since the state does not follow a period of refreshing, we cannot appeal to Rec security. Instead we show that (modulo previous game hops) the relevant games can be simulated by an attacker in game Pres who outputs ε at the start of their challenge.

Lemma 5.3. *Let $\overline{\mathsf{PRNG}}$ be as in Theorem 5.5. Then for any q_C-attacker $\mathcal{A}$ running in time T in game Rec against $\overline{\mathsf{PRNG}}$, and any $(q_{\mathcal{D}}, \gamma^*)$-legitimate sampler $\mathcal{D}$, there exist attackers $\mathcal{B}, \mathcal{C}_2, \mathcal{C}_3, \mathcal{E}, \mathcal{E}', \mathcal{F}$, and $\mathcal{G}$ running in time $T' \approx T$, such that*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{rec}}_{\overline{\mathsf{PRNG}},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) \leq \mathbf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{B},1) + \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2)\\ + \mathbf{Adv}^{\mathrm{rec}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{C}_3,\mathcal{D}) + \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}}(\mathcal{E}) + \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}'}(\mathcal{E}')\\ + k \cdot (\Delta_{\mathsf{rPKE}} + \mathbf{Adv}^{\mathrm{s\text{-}rerand}}_{\mathsf{rPKE}}(\mathcal{F})) + \mathbf{Adv}^{\mathrm{ind\$\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{G},k)\ .\end{aligned}$$

Here, $\mathcal{C}_2$ is a 0-attacker and $\mathcal{C}_3$ is a q_C-attacker.

Proof. We argue by a series of game hops, shown in Figures 5.20 and 5.21. The proof is similar to that of Lemma 5.2; we emphasise the differences here. We begin by defining game G_0, which is readily verified to be equivalent to game Rec against $\overline{\mathsf{PRNG}}$ with challenge bit $b = 0$. Next we define game G_1, which is identical to G_0 except we replace $(R, S) \leftarrow \mathsf{next}(pp, S)$ on line 15 with $R \leftarrow_\$ \{0,1\}^\ell, S \leftarrow_\$ \mathsf{setup}(pp)$. We claim that there exists a q_C-attacker $\mathcal{C}_3$ in game Rec against PRNG, running in time $T' \approx T$, such that

$$|\Pr[G_1 \Rightarrow 1] - \Pr[G_0 \Rightarrow 1]| \leq \mathbf{Adv}^{\mathrm{rec}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{C}_3,\mathcal{D})\ .$$

Let $\mathcal{C}_3$ be the adversary who proceeds as follows. On input pp and entropy estimates / side information $(\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}}$, $\mathcal{C}_3$ generates $(pk, sk) \leftarrow_\$ \mathsf{Key}$, and passes $((pp, pk), (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$ to $\mathcal{A}$. $\mathcal{C}_3$ simulates $\mathcal{A}$'s Sam oracle by querying his own oracle and passing the response to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs an initial state $\overline{S} = (S, C[1], \ldots, C[k], \psi)$ and index d. $\mathcal{C}_3$ passes S, d to his own challenger, receiving back unused inputs $(I_k)_{k>\mu+d}$ and (R, S). $\mathcal{C}_3$ inserts (R, S) at line 15 in G_0 and continues to simulate the game, finally passing the resulting output / state $(\overline{R}^*, \overline{S}^*)$ and inputs $(I_k)_{k>\mu+d}$ to $\mathcal{A}$. At the end of the game, $\mathcal{C}_3$ outputs whatever bit $\mathcal{A}$ does. Notice that if $\mathcal{C}_3$ receives a real output / state in his challenge then this perfectly simulates G_0, otherwise it perfectly simulates G_1; the claim follows.

Next we define game G_2, which is identical to G_1 except we replace $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$ on line 17 of G_1 with $C[0] \leftarrow_\$ \mathsf{Enc}(pk, 0^n)$. An entirely analogous argument to that made in the proof of Lemma 5.2 implies that there exists an attacker $\mathcal{B}$ running in time $T' \approx T$, and who makes a single query of $(S, 0^n)$ to his LoR oracle, such that

$$|\Pr[G_2 \Rightarrow 1] - \Pr[G_1 \Rightarrow 1]| \leq \mathbf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathsf{rPKE}}(\mathcal{B},1)\ .$$

Next we define game G_3, which is identical to G_2 except we replace $(R, S) \leftarrow \mathsf{next}(pp, S)$

on line 18 with $R \leftarrow\!\!\$\ \{0,1\}^{\ell}, S \leftarrow\!\!\$\ \mathsf{setup}(pp)$. Notice that since we are in the Rec setting, at least one $\mathsf{refresh}$ must have previously occurred (i.e., $\mathcal{A}$ must output $d \geq 1$ at the start of the challenge, and so the 'If $\psi = 1$' loop is executed). As such, the state S input to next will be that generated freshly in line 15. Therefore, this is again simulatable by a 0-adversary $\mathcal{C}_2$ in game Pres against the underlying PRNG PRNG. $\mathcal{C}_2$ uses the code of the sampler to generate the required entropy samples, estimates, and side information. $\mathcal{C}_2$ generates parameters as before, and passes these along with the entropy estimates / side information to $\mathcal{A}$. $\mathcal{C}_2$ uses the generated inputs to simulate $\mathcal{A}$'s Sam oracle. Eventually, $\mathcal{A}$ outputs state $(S, C[1], \ldots, C[k], \psi)$ and some value d. $\mathcal{C}_2$ discards S and d, and keeps the remaining ciphertexts to simulate the rest of the challenge, first replacing any invalid ciphertexts with encryptions of 0^n as per the protocol. We view $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$ in line 15 as the unknown initial state for the computation of $\mathcal{C}_2$'s Pres challenge against PRNG. As described in the proof of Lemma 5.2, $\mathcal{C}_2$ returns the empty string to its challenger, and receives back (R, S), which $\mathcal{C}_2$ inserts at line 18 of G_2. $\mathcal{C}_2$ then simulates the remainder of the challenge, again using the code of the sampler $\mathcal{D}$ to produce the unused inputs which are given to $\mathcal{A}$ along with the challenge output / state. At the end of the game, $\mathcal{C}_2$ outputs whatever bit $\mathcal{A}$ does. It is straightforward to verify that if $\mathcal{C}_2$ receives a real output / state in his challenge then this perfectly simulates G_2; otherwise it perfectly simulates G_3. It follows that

$$|\Pr[\,G_3 \Rightarrow 1\,] - \Pr[\,G_2 \Rightarrow 1\,]| \leq \mathbf{Adv}^{\mathrm{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2)\ ,$$

where $\mathcal{C}_2$ runs in time $T' \approx T$.

In games G_4 and G_5, we first replace the output of G on line 19 and then G' on line 20 with random bit strings. As before, by construction each PRG is seeded with a truly random seed in the game in which their output is modified, and so a straightforward reduction implies the existence of attackers $\mathcal{E}$ and $\mathcal{E}'$ with the claimed run times such that

$$|\Pr[\,G_5 \Rightarrow 1\,] - \Pr[\,G_3 \Rightarrow 1\,]| \leq \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}}(\mathcal{E}) + \mathbf{Adv}^{\mathrm{sprg\text{-}dist}}_{\mathsf{G}'}(\mathcal{E}')\ .$$

In game G_6, we replace the re-randomised ciphertexts $C[i] \leftarrow \mathsf{Rand}(pk, C[i]; R_i)$ in line 23 of G_5 with fresh encryptions $C[i] \leftarrow\!\!\$\ \mathsf{Enc}(pk, M_i)$, where $M_1 = S$ from line 18 of G_5 and $M_i = 0^n$ for $i \in [2, k]$. Notice that while $\mathcal{A}$ may output invalid ciphertexts as part of his challenge state, the loop in lines 11-13 ensures that each of these is replaced with $\mathsf{Enc}(pk, 0^n; 0^{\ell})$ and so each ciphertext which is re-randomised is valid. As such, an analogous reduction to that made in the proof of Lemma 5.2, invoking the strong re-randomisability of rPKE, plus a hybrid argument over the k ciphertexts to be replaced, implies that there exists an attacker $\mathcal{F}$ running in time $T' \approx T$ such that

$$|\Pr[G_6 \Rightarrow 1] - \Pr[G_5 \Rightarrow 1]| \leq k \cdot \mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE}}(\mathcal{F}) .$$

In game G_7, we replace the re-randomised ciphertexts on line 27 of G_6 with fresh encryptions of their underlying messages. Since, by the previous transition, all ciphertexts which are re-randomised at this point in G_6 are fresh encryptions, a straightforward hybrid argument invoking the re-randomisability of rPKE implies that

$$|\Pr[G_7 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1]| \leq k \cdot \Delta_{\mathsf{rPKE}} .$$

In game G_8, we replace each ciphertext $C[i]$ on lines 22, 24 of G_7 with a random bit string $C[i] \leftarrow_\$ \{0,1\}^m$ for $i \in [1,k]$. As before, a reduction to the IND\$-CPA security of rPKE implies that there exists an attacker $\mathcal{G}$ with the claimed run time and query budget, such that

$$|\Pr[G_8 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1]| \leq \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rPKE}}(\mathcal{G}, k) .$$

Notice that $\mathcal{A}$ always receives a random output $\overline{R}^* \leftarrow_\$ \{0,1\}^{k \cdot m}$ and freshly generated state $\overline{S}^* \leftarrow_\$ \overline{\mathsf{setup}}(pp)$ in his challenge in G_8. As such, it is straightforward to verify that G_8 is equivalent to game Rec against $\overline{\mathsf{PRNG}}$ with challenge bit $b = 1$. Combining the above via a standard argument implies that there exist attackers $\mathcal{B}, \mathcal{C}, \mathcal{E}, \mathcal{E}', \mathcal{F}, \mathcal{G}$ with the claimed run times, such that

$$\begin{aligned} \mathbf{Adv}^{\text{rec}}_{\overline{\mathsf{PRNG}},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}, \mathcal{D}) \leq{} & \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{rPKE}}(\mathcal{B}, 1) + \mathbf{Adv}^{\text{pres}}_{\mathsf{PRNG}}(\mathcal{C}_2) \\ & + \mathbf{Adv}^{\text{rec}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{C}_3, \mathcal{D}) + \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{E}) + \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}'}(\mathcal{E}') \\ & + k \cdot (\Delta_{\mathsf{rPKE}} + \mathbf{Adv}^{\text{s-rerand}}_{\mathsf{rPKE}}(\mathcal{F})) + \mathbf{Adv}^{\text{ind\$-cpa}}_{\mathsf{rPKE}}(\mathcal{G}, k) , \end{aligned}$$

thereby concluding the proof of Rec security.

□

Big Brother's success probability. It remains to bound the probability that Big Brother succeeds in game BPRNG-OUT for BPRNG. Let i_{ref}, j_{ref} denote, respectively, the refresh periods in which the challenge output $\overline{R}_i$ and target output $\overline{R}_j$ were computed. The key observation is that for Big Brother to succeed, two conditions must be met. Firstly, we require that $i_{ref} - j_{ref} \in [0, k-1]$; this ensures that the encrypted snapshot of the PRNG state used in refresh period j_{ref} is stored in the state of BPRNG. Secondly, letting η_i, η_j denote the values of η computed in line 11 of the $\overline{\mathsf{next}}$ algorithm when generating the

i-th and j-th output values $\overline{R}_i$ and $\overline{R}_j$, respectively, we require that $\eta_i = 1$ (so challenge output $\overline{R}_i$ consists of encrypted shapshots), and $\eta_j = 0$ (and hence can be reconstructed by BB given the appropriate encrypted snapshot). If $\eta_i, \eta_j \leftarrow\$ \{0,1\}$ then this relation will occur with probability $\frac{1}{4}$; a brief series of game hops to transition from their production in $\overline{\mathsf{next}}$ to this idealised variant then implies the claim.

Lemma 5.4. *Let* BPRNG *be as in Theorem 5.5. Then for all refresh patterns* $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$*, all distribution samplers* $\mathcal{D}$*, and for all* $1 \leq i, j \leq \sum_{\nu=1}^{\rho} a_\nu = q_R$*, there exist efficient adversaries* $\mathcal{B}$*,* $\mathcal{C}$ *such that* $\mathbf{Adv}^{\text{bprng-out}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{BB}, \mathsf{rp}, i, j) \geq \delta(\mathsf{rp}, i, j)$*, where*

$$\delta(\mathsf{rp}, i, j) = \begin{cases} 1 & \textit{if } j = i \text{ ; } \textit{else} \\ \frac{1}{4} - \mathbf{Adv}^{\text{rob}}_{\mathsf{PRNG},\gamma^*,q_\mathcal{D}}(\mathcal{B}, \mathcal{D}) - 2 \cdot \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{C}) & \textit{if } i_{ref} - j_{ref} \in [0, k-1] \\ 0 & \textit{otherwise} \end{cases}$$

Here $i_{ref} \leftarrow \min_\sigma \left[\sum_{\nu=1}^{\sigma} a_\nu \geq i\right]$ *and* $j_{ref} \leftarrow \min_\sigma \left[\sum_{\nu=1}^{\sigma} a_\nu \geq j\right]$*.* $\mathcal{B}$ *is a* $(q'_R, q_\mathcal{D}, q_C, q_S)$*-adversary where* $q_\mathcal{D} = \sum_{i=1}^{\rho} b_i$*,* $q_C = \max_{1 \leq i \leq \rho} b_i$*,* $q'_R = q_R + \rho - 1$*, and* $q_S = 0$*.*

Proof. If $i = j$, then the target output is given to BB as part of his challenge and BB trivially succeeds with probability one. Similarly, if $j_{ref} > i_{ref}$ or $i_{ref} - j_{ref} \geq k$ then BB aborts and output $\perp$, justifying this case also.

For the remaining case, let η_i, η_j denote the values of η computed in line 11 of the $\overline{\mathsf{next}}$ algorithm when generating the i-th and j-th output values $\overline{R}_i$ and $\overline{R}_j$, respectively. As discussed above, for BB to successfully recover the target output $\overline{R}_j$ we must have that $\eta_i = 1$ and $\eta_j = 0$. If this is the case, and assuming that $i_{ref} - j_{ref} \in [0, k-1]$, then it is straightforward to verify that BB will output the correct value with probability one. As such, for $j \neq i$ such that $i_{ref} - j_{ref} \in [0, k-1]$, it holds that

$$\mathbf{Adv}^{\text{bprng-out}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{BB}, \mathsf{rp}, i, j) \geq \Pr\left[\eta_i = 1 \wedge \eta_j = 0\right] .$$

We now bound this probability. We let G_0 be identical to game BPRNG-OUT against BPRNG. Consider the evolution of the state of the underlying PRNG PRNG within an execution of BPRNG with refresh pattern $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$ and sampler $\mathcal{D}$. It is straightforward to verify that the state of PRNG evolves via a refresh pattern $\mathsf{rp}' = (a_1, b_1, a_2 + 1, b_2, \ldots, a_\rho + 1, b_\rho)$, where the additional plus one for a_i with $i \in [2, \rho]$ is due to the extra application of next on line 7 of $\overline{\mathsf{next}}$ which occurs at the start of all but the first refresh period. Letting $(R_1, \ldots, R_{q_R+\rho-1}) \leftarrow \mathsf{out}(\mathsf{PRNG}, S_0, pp, \mathsf{rp}', \mathcal{D})$ where $q_R = \sum_{i=1}^{\rho} a_i$

(the extra $\rho - 1$ outputs accounting for the additional next calls mentioned above), then we may write

$$\Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_0\,\right] = \Pr\left[\,\mathrm{lsb}(\mathsf{G}(R_{i+i_{ref}-1})) = 1 \wedge \mathrm{lsb}(\mathsf{G}(R_{j+j_{ref}-1})) = 0 \text{ in } G_0\,\right] .$$

To bound this probability, we argue by a series of game hops. We let G_1 be identical to G_0 except each output R_i for $i \in [1, q_R + \rho - 1]$ is replaced with a random string $R_i \leftarrow_{\$} \{0,1\}^{\ell}$. Notice that both games can be perfectly simulated by a $(q'_R, q_{\mathcal{D}}, q_C, q_S)$-adversary $\mathcal{B}$ in game Rob against PRNG, where $q_{\mathcal{D}} = \sum_{i=1}^{\rho} b_i$, $q_C = \max_{1 \le i \le \rho} b_i$, $q'_R = q_R + \rho - 1$, and $q_S = 0$, who uses his Ref and RoR oracles to evolve the state of PRNG according to rp'. At the end of this process, $\mathcal{B}$ computes $(\eta_i, R_{1,i}, \ldots, R_{2k,i}) \leftarrow \mathsf{G}(R_{i+i_{ref}-1})$ and $(\eta_j, R_{1,j} \ldots, R_{2k,j}) \leftarrow \mathsf{G}(R_{j+j_{ref}-1})$, and outputs 1 if $\eta_i = 1$ and $\eta_j = 0$ and 0 otherwise. It follows that

$$\left|\Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_1\,\right] - \Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_0\,\right]\right| \le \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{B}, \mathcal{D}) .$$

In games G_2 and G_3 we replace, respectively, the outputs of $\mathsf{G}(R_{i+i_{ref}-1})$ and $\mathsf{G}(R_{j+j_{ref}-1})$ with random bit strings. In both games, by the previous transition, the PRGs are seeded with an independent random string, and so a pair of straightforward reductions to the stateless PRG security of G, combined via a standard argument, implies that there exists an attacker $\mathcal{C}$ such that

$$\left|\Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_3\,\right] - \Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_1\,\right]\right| \le 2 \cdot \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{C}) .$$

Now in G_3 both η_i and η_j are chosen as random bits, and so

$$\Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_3\,\right] = \frac{1}{4} .$$

Putting this altogether, and rearranging via a standard argument, we conclude that

$$\begin{aligned}\mathbf{Adv}^{\text{bprng-out}}_{\mathsf{BPRNG},\mathcal{D}}(\mathsf{BB}, \mathsf{rp}, i, j) &\ge \Pr\left[\,\eta_i = 1 \wedge \eta_j = 0 \text{ in } G_0\,\right] \\ &\ge \frac{1}{4} - \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{B}, \mathcal{D}) - 2 \cdot \mathbf{Adv}^{\text{sprg-dist}}_{\mathsf{G}}(\mathcal{C}) ,\end{aligned}$$

thereby completing the proof. □

Summary. Combining Lemmas 5.2 and 5.3 (Pres and Rec security of BPRNG) via Theorem 2.2 implies the upper bound on the robustness of BPRNG. The analysis of Big Brother's success probability (Lemma 5.4) then completes the proof of Theorem 5.5.

5.4.5 Extensions and Modifications

We conclude our analysis of BPRNG with a discussion of the extensions and modifications which can be made to the above construction to give it slightly different properties. Each of the following modifications can be shown to be secure using almost identical arguments to the existing security analysis.

Alternative snapshot storage. An alternative to storing a snapshot of a refreshed state by rotating the ciphertexts $(C[1], \ldots, C[k])$ as done in line 9 of $\overline{\mathsf{next}}$ would be to choose a random ciphertext to replace. More specifically, the PRNG output R computed in line 7 of $\overline{\mathsf{next}}$ could be stretched to produce a $\log k$ bit value t, and ciphertext $C[t]$ would then be replaced with $C[0]$. Note, however, that BB would no longer be able to tell which ciphertext corresponds to which snapshot of the state. This can be addressed if the encryption scheme, in addition to the aforementioned properties, can be used in a way which is additively homomorphic (for example, ElGamal using its lifted variant [64]). In this case, the construction would be able to maintain an encrypted counter of the number of refresh periods, and, for each snapshot, store an encrypted value corresponding to the number of refresh periods PRNG has undergone before the snapshot was taken. If the ciphertexts containing these values are concatenated with $(C[1], \ldots, C[k])$ to produce the output value $\overline{R}$, then BB obtains sufficient information to derive which state to use to recover a given output value. This yields a construction with different advantage function $\delta(\mathsf{rp}, i, j)$ compared to the above construction; instead of a sharp drop to 0 when i and j are separated by k refresh periods, the advantage gradually declines as the distance (in terms of the number of refresh periods) between i and j increases.

Shorter outputs. The above construction can further be modified to produce shorter output values. Specifically, instead of setting $\overline{R} \leftarrow (C[1], \ldots, C[k])$ in line 17 of $\overline{\mathsf{next}}$, a random ciphertext $C[t]$ could be chosen as $\overline{R}$ by stretching the output of G in line 11 with an additional $\log k$ bits to produce t. This will reduce the output length from $k \cdot m$ bits to m bits. However, a similar problem to the above occurs: BB will not be able to tell which snapshot $C[t]$ represents. Using a similar solution to the above will increase the output length to $2m$ bits. This modification will essentially reduce the backdooring advantage by a factor of $1/k$ compared to the above construction.

Refresh pattern. Lastly, we note that the above construction assumes BB receives as input the refresh pattern rp. Again, by maintaining encrypted counters for both the number of refresh periods and the number of produced output values for each snapshot, we can obtain an algorithm BB which does not require rp as input, but at the cost of increasing the output size.

```
proc. main // $G_0$, $\boxed{G_1}$
1. $pp \leftarrow\$ \mathsf{init}$ ; $(pk, sk) \leftarrow\$ \mathsf{Key}$
2. $S \leftarrow\$ \mathsf{setup}(pp)$
3. $C[1] \leftarrow\$ \mathsf{Enc}(pk, S)$
4. $\boxed{C[1] \leftarrow\$ \mathsf{Enc}(pk, 0^n)}$
5. For $i = 2 \ldots k$
6.    $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
7. $\psi \leftarrow 0$ ; $\overline{S}_0 \leftarrow (S, C[1] \ldots C[k], \psi)$
8. $(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp, pk)$
9. For $j = 1, \ldots, d$
10.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
11. For $i = 1 \ldots k$
12.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
13.       $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
14. If $\psi = 1$
15.    $(R, S) \leftarrow \mathsf{next}(pp, S)$
16.    $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$
17.    $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
18. $(R, S) \leftarrow \mathsf{next}(pp, S)$
19. $(\eta, R_1, \ldots, R_{2k}) \leftarrow \mathsf{G}(R)$
20. If $\eta = 0$ then $\overline{R}^* \leftarrow \mathsf{G}'(R_1)$
21. Else
22.    For $i = 1 \ldots k$
23.       $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
24.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
25. For $i = 1 \ldots k$
26.    $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
27. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
28. $b^* \leftarrow\$ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*)$
29. Return $b^*$
```

```
proc. main // $G_2$, $\boxed{G_3}$
1. $pp \leftarrow\$ \mathsf{init}$ ; $(pk, sk) \leftarrow\$ \mathsf{Key}$
2. $S \leftarrow\$ \mathsf{setup}(pp)$
3. $C[1] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
4. For $i = 2 \ldots k$
5.    $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
6. $\psi \leftarrow 0$ ; $\overline{S}_0 \leftarrow (S, C[1] \ldots C[k], \psi)$
7. $(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp, pk)$
8. For $j = 1, \ldots, d$
9.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
10. For $i = 1 \ldots k$
11.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
12.       $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
13. If $\psi = 1$
14.    $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
15.    $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$
16.    $\boxed{C[0] \leftarrow\$ \mathsf{Enc}(pk, 0^n)}$
17.    $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
18. $(R, S) \leftarrow \mathsf{next}(pp, S)$
19. $(\eta, R_1, \ldots, R_{2k}) \leftarrow \mathsf{G}(R)$
20. If $\eta = 0$ then $\overline{R}^* \leftarrow \mathsf{G}'(R_1)$
21. Else
22.    For $i = 1 \ldots k$
23.       $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
24.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
25. For $i = 1 \ldots k$
26.    $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
27. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
28. $b^* \leftarrow\$ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*)$
29. Return $b^*$
```

```
proc. main // $G_4$, $\boxed{G_5}$
1. $pp \leftarrow\$ \mathsf{init}$ ; $(pk, sk) \leftarrow\$ \mathsf{Key}$
2. $S \leftarrow\$ \mathsf{setup}(pp)$
3. $C[1] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
4. For $i = 2 \ldots k$
5.    $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
6. $\psi \leftarrow 0$ ; $\overline{S}_0 \leftarrow (S, C[1] \ldots C[k], \psi)$
7. $(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp, pk)$
8. For $j = 1, \ldots, d$
9.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
10. For $i = 1 \ldots k$
11.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
12.       $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
13. If $\psi = 1$
14.    $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
15.    $C[0] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
16.    $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
17. $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
18. $(\eta, R_1, \ldots, R_{2k}) \leftarrow \mathsf{G}(R)$
19. $\boxed{(\eta, R_1, \ldots, R_{2k}) \leftarrow\$ \{0,1\}^{2k \cdot u+1}}$
20. If $\eta = 0$ then $\overline{R}^* \leftarrow \mathsf{G}'(R_1)$
21. Else
22.    For $i = 1 \ldots k$
23.       $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
24.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
25. For $i = 1 \ldots k$
26.    $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
27. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
28. $b^* \leftarrow\$ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*)$
29. Return $b^*$
```

```
proc. main // $G_6$, $\boxed{G_7}$
1. $pp \leftarrow\$ \mathsf{init}$ ; $(pk, sk) \leftarrow\$ \mathsf{Key}$
2. $S \leftarrow\$ \mathsf{setup}(pp)$
3. $C[1] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
4. For $i = 2 \ldots k$
5.    $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
6. $\psi \leftarrow 0$ ; $\overline{S}_0 \leftarrow (S, C[1] \ldots C[k], \psi)$
7. $(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp, pk)$
8. For $j = 1, \ldots, d$
9.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
10. For $i = 1 \ldots k$
11.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
12.       $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
13. If $\psi = 1$
14.    $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
15.    $C[0] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
16.    $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
17. $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
18. $(\eta, R_1, \ldots, R_{2k}) \leftarrow\$ \{0,1\}^{2k \cdot u+1}$
19. If $\eta = 0$ then $\overline{R}^* \leftarrow\$ \{0,1\}^{k \cdot m}$
20. Else
21.    For $i = 1 \ldots k$
22.       $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
23.       $\boxed{C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)}$
24.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
25. For $i = 1 \ldots k$
26.    $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
27. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
28. $b^* \leftarrow\$ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*)$
29. Return $b^*$
```

Figure 5.18: Games for proof of Lemma 5.2.

```
proc. main // $G_8$, $\boxed{G_9}$
1. $pp \leftarrow\$ \mathsf{init}$ ; $(pk, sk) \leftarrow\$ \mathsf{Key}$
2. $S \leftarrow\$ \mathsf{setup}(pp)$
3. $C[1] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
4. For $i = 2 \ldots k$
5.     $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
6. $\psi \leftarrow 0$ ; $\overline{S}_0 \leftarrow (S, C[1] \ldots C[k], \psi)$
7. $(I_1, \ldots, I_d) \leftarrow\$ \mathcal{A}(pp, pk)$
8. For $j = 1, \ldots, d$
9.     $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
10. For $i = 1 \ldots k$
11.     If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
12.         $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
13. If $\psi = 1$
14.     $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
15.     $C[0] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
16.     $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
17. $R \leftarrow\$ \{0,1\}^\ell$ ; $S \leftarrow\$ \mathsf{setup}(pp)$
18. $(\eta, R_1, \ldots, R_{2k}) \leftarrow\$ \{0,1\}^{2k \cdot u' + 1}$
19. If $\eta = 0$ then $\overline{R}^* \leftarrow\$ \{0,1\}^{k \cdot m}$
20. Else
21.     For $i = 1 \ldots k$
22.         $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
23.         $\boxed{C[i] \leftarrow\$ \{0,1\}^m}$
24.     $\overline{R}^* \leftarrow C[1] \parallel \ldots \parallel C[k]$
25. For $i = 2, \ldots, k$
26.     $C[i] \leftarrow\$ \mathsf{Enc}(pk, 0^n)$
27. $C[1] \leftarrow\$ \mathsf{Enc}(pk, S)$
28. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
29. $b^* \leftarrow\$ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*)$
30. Return $b^*$
```

Figure 5.19: Further games for proof of Lemma 5.2.

```
proc. main // $G_0$, $\boxed{G_1}$
1. $\sigma \leftarrow \varepsilon \,;\, \mu \leftarrow 0$
2. $pp \leftarrow\!\!\$\ \mathsf{init} \,;\, (pk, sk) \leftarrow\!\!\$\ \mathsf{Key}$
3. For $k = 1, \ldots, q_{\mathcal{D}}$
4.     $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{k-1})$
5. $(\overline{S}_0, d) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Sam}}((pp, pk), (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
6. If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
7.     Return $\perp$
8. $(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}_0$
9. For $j = 1, \ldots, d$
10.     $S \leftarrow \mathsf{refresh}(pp, S, I_j) \,;\, \psi \leftarrow 1$
11. For $i = 1 \ldots k$
12.     If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
13.         $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
14. If $\psi = 1$
15.     $(R, S) \leftarrow \mathsf{next}(pp, S)$
16.     $\boxed{R \leftarrow\!\!\$\ \{0,1\}^\ell \,;\, S \leftarrow\!\!\$\ \mathsf{setup}(pp)}$
17.     $C[0] \leftarrow \mathsf{Enc}(pk, S; R)$
18.     $C[k] \leftarrow C[k-1]; \ldots; C[1] \leftarrow C[0]$
19. $(R, S) \leftarrow \mathsf{next}(pp, S)$
20. $(\eta, R_1, \ldots, R_{2k}) \leftarrow \mathsf{G}(R)$
21. If $\eta = 0$ then $\overline{R}^* \leftarrow \mathsf{G}'(R_1)$
22. Else
23.     For $i = 1 \ldots k$
24.         $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
25.     $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
26. For $i = 1 \ldots k$
27.     $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
28. $\psi \leftarrow 0 \,;\, \overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
29. $b^* \leftarrow\!\!\$\ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*, (I_k)_{k>\mu+d})$

Sam // $G_0 - G_3$
$\mu = \mu + 1$
Return $I_\mu$
```

```
proc. main // $G_2$, $\boxed{G_3}$
1. $\sigma \leftarrow \varepsilon \,;\, \mu \leftarrow 0$
2. $pp \leftarrow\!\!\$\ \mathsf{init} \,;\, (pk, sk) \leftarrow\!\!\$\ \mathsf{Key}$
3. For $k = 1, \ldots, q_{\mathcal{D}}$
4.     $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{k-1})$
5. $(\overline{S}_0, d) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Sam}}((pp, pk), (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
6. If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
7.     Return $\perp$
8. $(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}_0$
9. For $j = 1, \ldots, d$
10.     $S \leftarrow \mathsf{refresh}(pp, S, I_j) \,;\, \psi \leftarrow 1$
11. For $i = 1 \ldots k$
12.     If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
13.         $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
14. If $\psi = 1$
15.     $R \leftarrow\!\!\$\ \{0,1\}^\ell \,;\, S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
16.     $C[0] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)$
17.     $C[k] \leftarrow C[k-1]; \ldots; C[1] \leftarrow C[0]$
18. $(R, S) \leftarrow \mathsf{next}(pp, S)$
19. $\boxed{R \leftarrow\!\!\$\ \{0,1\}^\ell \,;\, S \leftarrow\!\!\$\ \mathsf{setup}(pp)}$
20. $(\eta, R_1, \ldots, R_{2k}) \leftarrow \mathsf{G}(R)$
21. If $\eta = 0$ then $\overline{R}^* \leftarrow \mathsf{G}'(R_1)$
22. Else
23.     For $i = 1 \ldots k$
24.         $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
25.     $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
26. For $i = 1 \ldots k$
27.     $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
28. $\psi \leftarrow 0 \,;\, \overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
29. $b^* \leftarrow\!\!\$\ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*, (I_k)_{k>\mu+d})$
```

Figure 5.20: Games for proof of Lemma 5.3.

```
proc. main // $G_4$, $\boxed{G_5}$
1. $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 0$
2. $pp \leftarrow\!\!\$\ \mathsf{init}$ ; $(pk, sk) \leftarrow\!\!\$\ \mathsf{Key}$
3. For $k = 1, \ldots, q_{\mathcal{D}}$
4.     $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{k-1})$
5. $(\overline{S}_0, d) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Sam}}((pp, pk), (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
6. If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
7.     Return $\perp$
8. $(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}_0$
9. For $j = 1, \ldots, d$
10.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
11. For $i = 1 \ldots k$
12.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
13.        $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
14. If $\psi = 1$
15.    $R \leftarrow\!\!\$\ \{0,1\}^\ell$ ; $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
16.    $C[0] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)$
17.    $C[k] \leftarrow C[k-1]; \ldots; C[1] \leftarrow C[0]$
18. $R \leftarrow\!\!\$\ \{0,1\}^\ell$ ; $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
19. $(\eta, R_1, \ldots, R_{2k}) \leftarrow\!\!\$\ \{0,1\}^{2k \cdot u+1}$
20. If $\eta = 0$ then $\overline{R}^* \leftarrow \mathsf{G}'(R_1)$ ; $\boxed{\overline{R}^* \leftarrow \{0,1\}^{k \cdot m}}$
21. Else
22.    For $i = 1 \ldots k$
23.        $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_i)$
24.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
25. For $i = 1 \ldots k$
26.    $C[i] \leftarrow \mathsf{Rand}(pk, C[i], R_{k+i})$
27. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
28. $b^* \leftarrow\!\!\$\ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*, (I_k)_{k > \mu+d})$

Sam // $G_4 - G_7$
$\mu = \mu + 1$
Return $I_\mu$
```

```
proc. main // $G_6$, $\boxed{G_7}$
1. $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 0$
2. $pp \leftarrow\!\!\$\ \mathsf{init}$ ; $(pk, sk) \leftarrow\!\!\$\ \mathsf{Key}$
3. For $k = 1, \ldots, q_{\mathcal{D}}$
4.     $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{k-1})$
5. $(\overline{S}_0, d) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Sam}}((pp, pk), (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
6. If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
7.     Return $\perp$
8. $(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}_0$
9. For $j = 1, \ldots, d$
10.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
11. For $i = 1 \ldots k$
12.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
13.        $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
14. If $\psi = 1$
15.    $R \leftarrow\!\!\$\ \{0,1\}^\ell$ ; $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
16.    $C[0] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)$
17.    $C[k] \leftarrow C[k-1]; \ldots; C[1] \leftarrow C[0]$
18. $R \leftarrow\!\!\$\ \{0,1\}^\ell$ ; $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
19. $(\eta, R_1, \ldots, R_{2k}) \leftarrow\!\!\$\ \{0,1\}^{2k \cdot u+1}$
20. If $\eta = 0$ then $\overline{R}^* \leftarrow \{0,1\}^{k \cdot m}$
21. Else
22.    $C[1] \leftarrow\!\!\$\ \mathsf{Enc}(pk, S)$
23.    For $i = 2 \ldots k$
24.        $C[i] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)$
25.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
26. For $i = 1 \ldots k$
27.    $C[i] \leftarrow \mathsf{Rand}(pk, C[i]; R_{k+i})$
28. $\boxed{\text{For } i = 2, \ldots, k}$
29.    $\boxed{C[i] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)}$
30. $\boxed{C[1] \leftarrow\!\!\$\ \mathsf{Enc}(pk, S)}$
31. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
32. $b^* \leftarrow\!\!\$\ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*, (I_k)_{k > \mu+d})$
```

Figure 5.21: Further games for proof of Lemma 5.3.

```
proc. main // $G_8$
1. $\sigma \leftarrow \varepsilon$ ; $\mu \leftarrow 0$
2. $pp \leftarrow\!\!\$\ \mathsf{init}$ ; $(pk, sk) \leftarrow\!\!\$\ \mathsf{Key}$
3. For $k = 1, \ldots, q_{\mathcal{D}}$
4.    $(\sigma_k, I_k, \gamma_k, z_k) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{k-1})$
5. $(\overline{S}_0, d) \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Sam}}((pp, pk), (\gamma_k, z_k)_{i=1}^{q_{\mathcal{D}}})$
6. If $\mu + d > q_{\mathcal{D}}$ or $\sum_{i=\mu+1}^{\mu+d} \gamma_i < \gamma^*$
7.    Return $\perp$
8. $(S, C[1], \ldots, C[k], \psi) \leftarrow \overline{S}_0$
9. For $j = 1, \ldots, d$
10.    $S \leftarrow \mathsf{refresh}(pp, S, I_j)$ ; $\psi \leftarrow 1$
11. For $i = 1 \ldots k$
12.    If $1 \leftarrow \mathsf{invalid}(pk, C[i])$
13.       $C[i] \leftarrow \mathsf{Enc}(pk, 0^n; 0^\ell)$
14. If $\psi = 1$
15.    $R \leftarrow\!\!\$\ \{0,1\}^\ell$ ; $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
16.    $C[0] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)$
17.    $C[k] \leftarrow C[k-1]; \ldots ; C[1] \leftarrow C[0]$
18. $R \leftarrow\!\!\$\ \{0,1\}^\ell$ ; $S \leftarrow\!\!\$\ \mathsf{setup}(pp)$
19. $(\eta, R_1, \ldots, R_{2k}) \leftarrow\!\!\$\ \{0,1\}^{2k \cdot u + 1}$
20. If $\eta = 0$ then $\overline{R}^* \leftarrow \{0,1\}^{k \cdot m}$
21. Else
22.    For $i = 1 \ldots k$
23.       $C[i] \leftarrow\!\!\$\ \{0,1\}^m$
24.    $\overline{R}^* \leftarrow C[1] \,\|\, \ldots \,\|\, C[k]$
25. For $i = 2 \ldots k$
26.    $C[i] \leftarrow\!\!\$\ \mathsf{Enc}(pk, 0^n)$
27. $C[1] \leftarrow\!\!\$\ \mathsf{Enc}(pk, S)$
28. $\psi \leftarrow 0$ ; $\overline{S}^* \leftarrow (S, C[1] \ldots C[k], \psi)$
29. $b^* \leftarrow\!\!\$\ \mathcal{A}((pp, pk), \overline{R}^*, \overline{S}^*, (I_k)_{k > \mu+d})$

Sam // $G_8$
$\mu = \mu + 1$
Return $I_\mu$
```

Figure 5.22: Further games for proof of Lemma 5.3.

5.4.6 On the Inherent Resistance of PRNGs with Input to Backdooring

We have just presented and analysed a construction of a PRNG with input that is backdoored in a powerful sense: from a given output, Big Brother can recover prior output values through a number of past refreshes. However, our construction is limited in that the number of past refresh periods that Big Brother can simultaneously compromise is proportional to the state size.

The paper upon which this chapter is based [54] included a result purporting to formalise this intuition by bounding the number of refresh periods that a backdoor attacker can simultaneously compromise in terms of the state size of the PRNG. Unfortunately during the production of this thesis we discovered that the previously stated result is incorrect and have been unable to recover a similar result. For the remainder of the section, we give an overview of the problem with the result of [54] and of the challenges associated with recovering a similar result.

The result from [54]. We define a ‘high entropy refresh’ to be a sequence of consecutive refresh calls such that the entropy samples used in those calls collectively contain at least γ^* bits of entropy. The claimed result of [54] considers a backdoored PRNG that allowed Big Brother to recover a sequence of past PRNG states, each separated by a high entropy refresh, given a single output and the backdoor parameter.

The key observation that motivated the proof approach of [54] is that since all PRNG outputs are a deterministic function of the PRNG state and public parameters, it follows that any ‘useful’ information that can be leaked to Big Brother via the output must be ‘stored’ in these values. The proof of [54] aimed to lower bound the collision entropy of sequences of states separated by high entropy refreshes. If the expected collision entropy (over the choice of public parameters) of such a sequence can be shown to decrease as the number of states in the sequence increases, then this can be used to argue that the state must grow in order to hold enough information for the sequence of states to be recovered. While this intuition does seem sound, we shall see that proving a relationship between collision probability and compromised refresh periods does not seem as straightforward as erroneously assumed in [54].

Collision probabilities. We now describe in more detail the collision probabilities that the proof of [54] aims to bound, and discuss where the problems arise. We have intentionally kept our description fairly informal to avoid introducing additional notation; the reader is referred to the original work for a more rigorous treatment.

The result of [54] considers a restricted class of $(q_{\mathcal{D}}, \gamma^*)$-legitimate samplers. The only restriction necessary to recall for our discussion here is that we assume that the sampler $\mathcal{D}$ always returns a fixed sequence of entropy estimates, although the actual entropy samples which these estimates accompany may differ. More formally, we assume that there exists a sequence of entropy estimates $(\gamma_1^*, \gamma_2^*, \ldots, \gamma_{q_{\mathcal{D}}}^*)$ such that for all $j \geq 1$ it holds that $\Pr\left[\gamma_j = \gamma_j^*\right] = 1$, where γ_i is obtained by setting $\sigma_0 = \varepsilon$ and iteratively computing $(\sigma_i, I_i, \gamma_i, z_i) \leftarrow\!\!\$\ \mathcal{D}(\sigma_{i-1})$ for $i \in [1, j]$. We say that such samplers are *well-behaved.*

With this in place, consider the following experiment for a PRNG $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$, a well-behaved $(q_{\mathcal{D}}, \gamma^*)$-legitimate sampler $\mathcal{D}$, and a given refresh pattern $\mathsf{rp} = (a_1, b_1, \ldots, a_\rho, b_\rho)$ with $q_R = \sum_{i=1}^{\rho} a_i$. Sample public parameters $pp \leftarrow\!\!\$\ \mathsf{init}$ and a pair of initial states $S_0, S_0' \leftarrow\!\!\$\ \mathsf{setup}(pp)$. Evolve both states according to rp, producing $(R_1, S_1, \ldots, R_{q_R}, S_{q_R}) \leftarrow \mathsf{evolve}(\mathsf{PRNG}, pp, S_0, \mathsf{rp}, \mathcal{D})$ and $(R_1', S_1', \ldots, R_{q_R}', S_{q_R}')$ $\leftarrow \mathsf{evolve}(\mathsf{PRNG}, pp, S_0', \mathsf{rp}, \mathcal{D})$. For some $j \leq \rho$, pick out a sequence of states from the first run $\overline{S_j} = (S_{i_0}, S_{i_1}, \ldots, S_{i_j})$, where $i_0 = 0$ and $i_k \in [1, q_R]$ for $k \in [1, j]$, such that each pair of states is separated by a high entropy refresh. Since the sampler is well-behaved, it follows that the corresponding sequence of states from the second run, $\overline{S_j'} = (S_{i_0}', S_{i_1}', \ldots, S_{i_j}')$, are each separated by a high entropy refresh also. We let $\mathrm{CP}(\overline{S_j} \mid pp)$ denote the probability that these sequences of states collide. More formally, we define

$$\mathrm{CP}(\overline{S_j} \mid pp) = \sum_z \Pr\left[\overline{S_j} = \overline{S_j'} \mid pp = z\right] \cdot \Pr\left[pp = z\right],$$

where the probability is over the coins of $\mathsf{init}, \mathsf{setup}$, and $\mathcal{D}$.

The false claim. Let PRNG be a PRNG with associated parameters (ℓ, p). Let $\delta = \max_{\mathcal{A},\mathcal{D}} \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{A}, \mathcal{D})$, where the maximum is over all $(q_R, q_{\mathcal{D}}, q_C, q_S)$-attackers running in time T and all well-behaved $(q_{\mathcal{D}}, \gamma^*)$-legitimate-samplers $\mathcal{D}$[3]. We define $\epsilon = \delta + 2^{-\ell}$, where recall that ℓ is the output length of PRNG. The result of [54] hinged on proving that for PRNG as described, it holds that $\mathrm{CP}(\overline{S_j} \mid pp) \leq \epsilon^{j+1}$ for all $j \geq 0$.

[3]In [54], a scheme is defined to be (T, ϵ)-secure with respect to a given security game if the maximum advantage of an attacker running in time T is ϵ, and we have stated the result here in this way for ease of comparison. It is straightforward to verify that the same argument holds when rephrasing the claimed result of [54] in line with the concrete security style advantage terms given in this work.

The claim is correct in the case that $j = 0$. To see this, notice that if $\mathrm{CP}(\overline{S_0} \mid pp)$ is large, then an attacker $\mathcal{A}$ can exploit this to win the robustness game against PRNG as follows. Upon input pp, $\mathcal{A}$ makes a RoR query, receiving R_1 in response. $\mathcal{A}$ then generates a fresh state $S_0' \leftarrow\$\, \mathsf{setup}(pp)$, computes $R_1' \leftarrow \mathsf{next}(pp, S_0')$, and returns 1 if $R_1 = R_1'$ and 0 otherwise. It is straightforward to verify that $\mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) \geq \mathrm{CP}(\overline{S_0} \mid pp) - 2^{-\ell}$. Since by definition $\delta \geq \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D})$, this implies the claim in the case that $j = 0$. However, unfortunately the result does not extend to $j \geq 1$ in generality, and we construct as a counterexample a PRNG for which the claim does not hold.

A counterexample. For the counterexample, consider a PRNG $\mathsf{PRNG} = (\mathsf{init}, \mathsf{setup}, \mathsf{refresh}, \mathsf{next})$ with parameters (ℓ, p) and for which $\mathcal{S} = \{0,1\}^n$. Let $\mathcal{PP} \subseteq \{0,1\}^*$ denote the public parameter space of PRNG. We let $\delta = \max_{\mathcal{A},\mathcal{D}} \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG},\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D})$ as defined above. We require that $\delta < 1 - 2^{-(\ell-1)}$ (note that any cryptographically useful PRNG will satisfy this property).

We now define a modified PRNG $\mathsf{PRNG}' = (\mathsf{init}', \mathsf{setup}', \mathsf{refresh}', \mathsf{next}')$ as follows. We set the public parameter space $\mathcal{PP}'$ of PRNG' to be $\mathcal{PP}' = \{0,1\} \times \mathcal{PP}$. We define modified parameter generation via init' to toss a coin $c \leftarrow\$\, \{0,1\}$, run init', and output parameters $pp' = c \,\|\, pp$. When setup', $\mathsf{refresh}'$, or next' are called on an input tuple containing $pp' = c \,\|\, pp$, they remove the prepended c from the public parameters and apply the corresponding algorithm of the original scheme to the resulting input tuple. If $c = 0$, they simply return the result. However, if $c = 1$ then the algorithm overwrites the state component with 0^n prior to returning the result. For example, for all $pp \in \mathcal{PP}$, $0^n \leftarrow\$\, \mathsf{setup}'(1 \,\|\, pp)$ and $(R, 0^n) \leftarrow \mathsf{next}(1 \,\|\, pp, S)$ where $(R, S') \leftarrow \mathsf{next}(pp, S)$. Crucially, notice that if PRNG' is run with public parameters $pp' \in \{1\} \times \mathcal{PP}$ then, for any refresh pattern rp and sampler $\mathcal{D}$, the sequence of states generated by running PRNG' according to rp will be constant at 0^n.

It is straightforward to verify that for any $(q_R, q_{\mathcal{D}}, q_C, q_S)$-attacker $\mathcal{A}$ running in time T, and well-behaved $(q_{\mathcal{D}}, \gamma^*)$-legitimate sampler $\mathcal{D}$, it holds that $\max_{\mathcal{A},\mathcal{D}} \mathbf{Adv}^{\mathrm{rob}}_{\mathsf{PRNG}',\gamma^*,q_{\mathcal{D}}}(\mathcal{A},\mathcal{D}) \leq \frac{1}{2} \cdot (1 + \delta)$. (To see this, notice that game Rob against PRNG' is equivalent to that against PRNG in the case that $pp' \in \{0\} \times \mathcal{PP}$, in which case the advantage of such an attacker is upper bounded by δ. Upper bounding the attacker advantage in the case that $pp' \in \{1\} \times \mathcal{PP}$ with 1, and noting that parameters for the different cases are chosen with probability $\frac{1}{2}$, justifies the claim.) This implies that

$\epsilon \leq \frac{1}{2} \cdot (1+\delta) + 2^{-\ell}$ for PRNG', and since by assumption $\delta < 1 - 2^{-(\ell-1)}$, this implies that $\epsilon < 1$.

We now consider $\mathrm{CP}(\overline{S_j} \mid pp)$ for PRNG'. Due to the pathological behaviour of PRNG' in the case the $pp' \in \{1\} \times \mathcal{PP}$, it follows that $\mathrm{CP}(\overline{S_j} \mid pp) \geq \frac{1}{2}$ for all $j \geq 0$. This is because conditioned on a 'bad' parameter $pp' \in \{1\} \times \mathcal{PP}$ being sampled — an event that occurs with probability $\frac{1}{2}$ — then *any* pair of state sequences produced by PRNG' will collide with probability one. However, the claimed result in [54] would imply that $\mathrm{CP}(\overline{S_j} \mid pp) \leq \epsilon^{j+1}$. Since $\epsilon < 1$ by our assumption on δ, this implies that ϵ^{j+1} — and thereby $\mathrm{CP}(\overline{S_j} \mid pp)$ — tends to 0 as j tends to infinity. However, since we know that $\mathrm{CP}(\overline{S_j} \mid pp) \geq \frac{1}{2}$ for all $j \geq 0$, this is clearly a contradiction.

Challenges and future work. Despite this considerable setback, we believe that the intuition that simultaneously backdooring more refresh periods for a robust PRNG requires the backdoored PRNG to have a larger state is likely to be correct, although the counterexample suggests that the increase in state size compared to refresh periods compromised might be smaller than previously believed.

If we could correctly upper bound $\mathrm{CP}(\overline{S_j} \mid pp)$ for $j \geq 0$, we could likely recover a variant of the result. Unfortunately, it is not clear how to construct this bound via a direct reduction to the Rob security of the PRNG, since intuitively an attacker in the robustness game can win if e.g., the states output by setup collide, and gains no greater advantage if a *sequence* of j states collides.

A possible approach. It may be possible to circumvent this roadblock by considering a restricted set of PRNGs and introducing a new robustness definition. Namely, suppose that the PRNG in question is Pres secure. Then this implies that the states following high entropy refreshes are indistinguishable from initial states generated by setup. As such, we can hop to a game in which these refreshed states are replaced by freshly generated states (with the gap between the games bounded via a series of reductions to the Pres security of the PRNG). Thus in the modified game the desired collision probability is equivalent to the following:

$$\mathrm{CP}(\overline{S_j} \mid pp) = \sum_z \Pr\left[\, \overline{S_j} = \overline{S_j'} \mid pp \leftarrow\!\!\$\; \mathsf{init}\, ; \overline{S_j}, \overline{S_j'} \leftarrow\!\!\$\; \mathsf{setup}^j(pp) \,\right] \cdot \Pr\left[\, pp = z\,\right],$$

where $\overline{S_j} \leftarrow\!\!\$\, \mathsf{setup}^j(pp)$ corresponds to computing $S_i \leftarrow\!\!\$\, \mathsf{setup}(pp)$ for $i \in [0,j]$, and returning

$\overline{S_j} = (S_0, \ldots, S_j)$.

As such, in this modified game the problem reduces to bounding the probability that two sequences of freshly sampled states collide. It seems likely that this conceptually simpler setting may in turn simplify the analysis. That said, the probability still does not seem possible to bound via a reduction to the robustness of the PRNG for the reasons discussed above. An alternative approach might be to define a 'multi-instance' variant of robustness, in which public parameters are sampled and an attacker must tackle j independent instances of the PRNG using these public parameters and break the robustness of *all* of them. If the j initial states corresponding to each challenge instance collide with a sequence of j states sampled by the attacker (an event that occurs with probability $\mathrm{CP}(\overline{S_j} \mid pp)$) then the attacker can exploit this to win their game. However, it has been shown that pinning down the 'right' multi-instance definition for indistinguishability games such as robustness — in the sense of requiring the attacker to succeed against each instance, and not e.g., succeed against the majority and guess the remainder — can be very challenging [21]. More broadly, modifying robustness in this artificial way (and restricting the set of PRNGs considered to those that achieve Pres security) makes for a much less satisfying result. For these reasons we leave formalising this intuition, and hopefully overcoming its limitations, to future work.

5.5 Conclusion

In this chapter we presented a provable security treatment of mass surveillance in the context of backdoored pseudorandom number generators, which post-Snowden and the Dual-EC-DRBG scandal must be considered a very real world problem. We set out to explore the extent to which a secure pseudorandom number generator can be backdoored while simultaneously being provably secure.

In the first part of the chapter, we resolved an open problem from [57] by demonstrating via two novel BPRG constructions that forward secure PRGs can be backdoored in a way that allows Big Brother to recover past output. This unfortunately disproves the intuition that forward secure PRGs — which are designed to protect past output in the event of state compromise by a non-backdoor attacker — may offer some protection against such a form of backdooring.

We built on these results in the second part of the chapter to tackle a more challenging problem: backdooring robust PRNGs with input. This setting adds a considerable layer of complexity since the state of a robust PRNG is periodically updated with fresh entropy. We used a simple BPRNG construction to highlight the key challenge in building powerful robust backdoored PRNGs: constructing a backdoor that persists through multiple high entropy refreshes. We defined a new security model to capture this goal, and the chapter culminated with a construction of a robust BPRNG that provably achieves it, possessing a backdoor that allows Big Brother to recover past outputs stretching back through a bounded number of refresh periods. The fact that a provably robust PRNG — which can withstand all manner of compromise by a non-backdoor attacker — succumbs to a powerful form of backdooring underscores that Big Brother represents a formidable threat, quite distinct from the adversaries typically considered in the cryptographic literature pre-2013, and motivates the need for further study to understand and prevent backdoor attackers.

We conclude by highlighting some open problems and directions for future work.

Impossibility result. The main problem left open by our work is understanding the extent to which robust PRNGs possess an innate resistance to backdooring. A new result proving that there is a limit to how many refresh periods can be simultaneously backdoored would be a positive step to counteract surveillance, suggesting that use of robust PRNGs which are frequently refreshed with sufficient entropy may frustrate Big Brother's attempts at subversion. On the other hand, disproving the intuition sketched in Section 5.4.6 via a construction that allows Big Brother to compromise many refresh periods without a significant increase in state size would certainly be interesting (if disappointing), and nonetheless useful to increase our understanding of the dangers and limitations of backdoored PRNGs.

Immunisers. Regardless of whether this limitation is inherent, we have shown that robust PRNGs succumb to powerful forms of backdooring. As such, developing immunisers — post-processing techniques which aim to destroy the subliminal channel opened by a backdoor — emerges as an important direction for future work. Naively the immunisation techniques for BPRGs of [57] should work equally well for PRNGs with input, since a

PRNG collapses to a PRG when no refresh calls are made. However, it seems plausible that robust PRNGs, being seemingly ‘harder’ to backdoor, can be immunised with less intrusive or idealised cryptographic techniques than for PRGs (which require a hash function modelled as either a random oracle or a universal computational extractor [57]). This motivates the development of immunisers designed specifically for PRNGs.

Chapter 6

Conclusion

In this thesis we have used three real world concerns — EtE encrypted messaging, standardised PRNGs, and the threat of backdoored algorithms — as a jumping-off point for a provable security analysis of a number of practical problems. In each setting, we were motivated by uncovering and modelling new attacks, and developing provably secure solutions.

In **Chapter 3**, we considered message franking (formalised as ccAEAD [73]), a proposed solution to the emerging problem of verifiable abuse reporting in encrypted messaging applications. We began by showing that Facebook's attachment franking scheme succumbs to an attack caused by the non-robustness [67] of GCM colliding with a server side bug. We used the ad hoc and ultimately insecure scheme deployed by Facebook to motivate the need for fast, provably secure ccAEAD. To this end, we dived into the wealth of literature on collision-resistant hashing to both explain why secure franking schemes cannot match the efficiency (in terms of rate) of the fastest AEAD schemes, and to inspire our construction of the first secure single-pass ccAEAD scheme. Problems left open by our work include developing a scheme that matches the efficiency of ours under weaker assumptions than the RKA-PRF-security required of the compression function here, and developing new security models for message franking as its usage increases and other desirable properties for franking schemes emerge.

Chapters 4 and 5 took as their starting point the NIST SP 800-90A standard, which was uncomfortably thrust into the spotlight with the Snowden revelations regarding the backdoored Dual-EC.

In **Chapter 4** we set out to fill some of the significant gaps in analysis that surround the three PRNG constructions that remain in the widely deployed standard. Prior to our work, these PRNGs — HASH-DRBG, HMAC-DRBG, and CTR-DRBG — had received minimal formal analysis, likely having been comparatively overlooked in light of the Dual-EC-DRBG controversy. On the positive side we formally proved the robustness of HASH-DRBG in the ROM, along with a similar but more restricted result for HMAC-DRBG. However, on a less positive note we presented a new attack that breaks the forward security of a permitted variant of HMAC-DRBG, directly contradicting security claims made in the standard and underscoring the need to formally prove all such claims. Finally, we stepped outside the conventional PRNG security models to consider state compromise during output production, and highlighted a number of vulnerabilities that may arise from implementation choices allowed by the overly flexible standard. Directions for future work include extending our analysis to the recently proposed seed-dependent sampler setting [50], and developing new PRNGs that achieve an effective balance between provable security and practicality that might be candidates to replace those in NIST SP 800-90A in the future.

For **Chapter 5**, we took inspiration from both the infamously backdoored Dual-EC-DRBG and earlier work on backdoored deterministic PRGs [57] to explore the extent to which a provably secure pseudorandom number generator can be backdoored. This question is interesting from a practical, as well as theoretical, perspective since any new standardised PRNG will likely have to stand up to rigorous security analysis. We showed that, unfortunately, strong security properties such as forward security for PRGs and robustness for PRNGs with input do not offer the protection against backdooring one might hope. To this end, we presented novel constructions of generators that are provably secure with respect to these definitions and yet nonetheless conceal powerful backdoors. In particular, the main result of the chapter was a construction of a provably robust PRNG that allows recovery of past output values through a bounded number of refresh periods — a backdoor with the potential to completely undermine the security of real world protocols such as SSL / TLS. En route, we strengthened existing definitions for backdoored PRGs, and developed the first security model for backdoored PRNGs with input. Open problems from this section include either formalising or disproving the intuition sketched in the latter part of the chapter that the fresh entropy which periodically flows into the state of robust PRNGs makes them more challenging to backdoor, and developing immuniser functions that are tailored to robust PRNGs.

Provable security in the real world. The results in this thesis illustrate the productive interplay that can exist between provable security and practice. As we've seen, a provable security analysis is a highly useful component in the design of real world cryptosystems, with the potential to pre-empt and prevent attacks, produce practical solutions grounded in decades of research, and provide a framework within which to formally reason about the security required for an application.

On the other hand, the results presented illustrate the extent to which the wealth of problems that exist in practice, and the solutions to these already deployed by practitioners, are a great source of inspiration for provable security research. In each setting considered, we have shown how analysing a real world problem through the lens of provable security can act as a catalyst to define new security models, develop new constructions, and unearth unexpected connections to existing literature.

Rather than provable security being unhelpfully academic — or worse, directly at odds with practice, as some critics have suggested [94] — we strongly believe that there is a natural and mutually beneficial relationship between provable security and the real world. This is exemplified by the success — both on paper and in practice — of practice-oriented provable security. Encouraging an exchange of ideas and culture of collaboration between practitioners and the academic community is likely to yield further successes on both sides in the future.

Bibliography

[1] FIPS 140-3. Security requirements for cryptographic modules. *National Institute of Standards and Technology*, March 2019. Online: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf`.

[2] Josh Aas. Looking forward to 2019. *Let's Encrypt*, December 31 2018. Online: `https://letsencrypt.org/2018/12/31/looking-forward-to-2019.html`.

[3] Michel Abdalla, Sonia Belaïd, David Pointcheval, Sylvain Ruhault, and Damien Vergnaud. Robust pseudo-random number generators with input secure against side-channel attacks. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, pages 635–654, 2015.

[4] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 480–497, 2010.

[5] Advanced Micro Devices (AMD). The ZEN microarchitecture. `https://www.amd.com/en/technologies/zen-core` (viewed September 1 2019).

[6] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 364–375, 2015.

[7] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, pages 119–135, 2013.

[8] Thomas Baignères, Cécile Delerablée, Matthieu Finiasz, Louis Goubin, Tancrède Lepoint, and Matthieu Rivain. Trap me if you can – million dollar curve. IACR Cryptology ePrint Archive, Report 2015/1249, 2015. `http://eprint.iacr.org/2015/1249`.

[9] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security. *The Guardian*, September 6 2013. Online: `https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security`.

[10] Elaine Barker and John Kelsey. NIST special publication 800-90A. Recommendation for random number generation using deterministic random bit generators revision 1. *National Institute of Standards and Technology (NIST)*, June 2015. Online: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf`.

[11] Elaine Barker and John Kelsey. (Second draft) NIST special publication 800-90C. Recommendation for random bit generator (RBG) constructions. *National Institute of Standards and Technology (NIST)*, April 2016. Online: `https://csrc.nist.gov/CSRC/media/Publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf` Year = 2012.

[12] Balthazar Bauer, Pooya Farshim, and Sogol Mazaheri. Combiners for backdoored random oracles. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 272–302, 2018.

[13] Mihir Bellare. Practice-oriented provable-security. In *Information Security, First International Workshop, ISW '97, Tatsunokuchi, Japan, September 17-19, 1997, Proceedings*, pages 221–231, 1997.

[14] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 535–552, 2007.

[15] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 1–15, 1996.

[16] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 360–378, 2008.

[17] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 491–506, 2003.

[18] Mihir Bellare and Silvio Micali. How to sign given any trapdoor permutation. *Journal of the ACM*, **39(1)** (1992), pp. 214–233.

[19] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 531–545, 2000.

[20] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 1–19, 2014.

[21] Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. Multi-instance security and its application to password-based cryptography. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 312–329. 2012.

[22] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[23] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of*

Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings, pages 92–111, 1994.

[24] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 317–330, 2000.

[25] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[26] Daniel J Bernstein. Cache-timing attacks on AES. *Preprint*, April 2005. Online: `https://cr.yp.to/antiforgery/cachetiming-20050414.pdf`.

[27] Daniel J. Bernstein. ChaCha, a variant of Salsa20. *Workshop Record of SASC 2008: The State of the Art of Stream Ciphers*, January 2008. Online: `https://cr.yp.to/chacha/chacha-20080128.pdf`.

[28] Daniel J. Bernstein. Fast-key-erasure random-number-generators. *The cr.yp.to blog*, July 23 2017. `https://blog.cr.yp.to/20170723-random.html`.

[29] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: A white paper for the black hat. In *Security Standardisation Research - Second International Conference, SSR 2015, Tokyo, Japan, December 15-16, 2015, Proceedings*, pages 109–139, 2015.

[30] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980, 2013.

[31] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A standardized back door. In *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, pages 256–281. 2016.

[32] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, pages 320–337, 2011.

[33] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document, version 2.1. June 19 2010. Online: `https://keccak.noekeon.org`.

[34] Didier Bigo, Gertjan Boulet, Caspar Bowden, Sergio Carrera, Julien Jeandesboz, and Amandine Scherrer. Fighting cyber crime and protecting privacy in the cloud. *Brussels: European Parliament*, 2012.

[35] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 1–18, 2009.

[36] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and related-key attack on the full AES-256. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 231–249. 2009.

[37] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 526–541, 2005.

[38] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 320–335, 2002.

[39] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, **15(2)** (1986), pp. 364–383.

[40] Andrey Bogdanov. Improved side-channel collision attacks on AES. In *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, pages 84–95, 2007.

[41] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, version 0.4. *Unpublished manuscript*, 2017. Online: `https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf`.

[42] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach. *The Guardian*, March 16 2018. Online: `https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election`.

[43] Matthew J. Campagna. Security bounds for the NIST codebook-based deterministic random bit generator. Cryptology ePrint Archive, Report 2006/379, 2006. `https://eprint.iacr.org/2006/379`.

[44] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 209–218, 1998.

[45] Rahul Chatterjee, Joanne Woodage, Yuval Pnueli, Anusha Chowdhury, and Thomas Ristenpart. The TypTop system: Personalized typo-tolerant password checking. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 329–346, 2017.

[46] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the Juniper Dual EC incident. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 468–479, 2016.

[47] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of Dual EC in TLS implementations. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 319–335, 2014.

[48] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 327–350, 2014.

[49] Benny Chor and Ronald L. Rivest. A knapsack type public key cryptosystem based on arithmetic in finite fields. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 54–65, 1984.

[50] Sandro Coretti, Yevgeniy Dodis, Harish Karthikeyan, and Stefano Tessaro. Seedless fruit is the sweetest: Random number generation, revisited. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, pages 205–234, 2019.

[51] Mario Cornejo and Sylvain Ruhault. Characterization of real-life PRNGs under partial state corruption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1004–1015, 2014.

[52] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 416–427, 1989.

[53] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 579–598, 2015.

[54] Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 403–432, 2016.

[55] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, **22(6)** (1976), pp. 644–654.

[56] Yevgeniy Dodis and Jee Hea An. Concealment and its applications to authenticated encryption. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 312–329, 2003.

[57] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In *Advances*

in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, pages 101–126, 2015.

[58] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, Cascade and HMAC modes. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 494–510, 2004.

[59] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 155–186, 2018.

[60] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 647–658, 2013.

[61] Yevgeniy Dodis, Thomas Ristenpart, John P. Steinberger, and Stefano Tessaro. To hash or not to hash again? (In)Differentiability results for H^2 and HMAC. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 348–366, 2012.

[62] Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too - optimal recovery strategies for compromised RNGs. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 37–54, 2014.

[63] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 293–302, 2008.

[64] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 10–18, 1984.

[65] Facebook. Messenger Secret conversations. *Technical Whitepaper*, July 8 2016. Online: `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`.

[66] Pooya Farshim, Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust encryption, revisited. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, pages 352–368, 2013.

[67] Pooya Farshim, Claudio Orlandi, and Razvan Rosie. Security of symmetric primitives under incorrect usage of keys. *IACR Transactions on Symmetric Cryptology*, **2017(1)** (2017), pp. 449–473.

[68] Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 76–90, 2018.

[69] Peter Gaži and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 87–116, 2016.

[70] Kristian Gjøsteen. Comments on Dual-EC-DRBG/NIST SP 800-90, draft December 2005, 2006. Online: `http://www.math.ntnu.no/~kristiag/drafts/dual-ec-drbg-comments.pdf`.

[71] Oded Goldreich. On post-modern cryptography. Cryptology ePrint Archive, Report 2006/461, 2006. `https://eprint.iacr.org/2006/461`.

[72] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377, 1982.

[73] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 66–97, 2017.

[74] Sean Gulley, Vinodh Gopal, Kirk Yap, Wajdi Feghali, and Jim Guilford. Intel SHA extensions. *Technical Whitepaper*, July 2013. Online: `https://software.intel.com/sites/default/files/article/402097/intel-sha-extensions-white-paper.pdf`.

[75] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 56–75, 2010.

[76] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, **28(4)** (1999), pp. 1364–1396.

[77] Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 70–88, 2011.

[78] Shoichi Hirose. Security analysis of DRBG using HMAC in NIST SP 800-90. In *Information Security Applications, 9th International Workshop, WISA 2008, Jeju Island, Korea, September 23-25, 2008, Revised Selected Papers*, pages 278–291, 2008.

[79] Seokhie Hong, Jongsung Kim, Sangjin Lee, and Bart Preneel. Related-key rectangle attacks on reduced versions of SHACAL-1 and AES-192. In *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, pages 368–383, 2005.

[80] Loïs Huguenin-Dumittan and Iraklis Leontiadis. A message franking channel. Cryptology ePrint Archive, Report 2018/920, 2018. `https://eprint.iacr.org/2018/920`.

[81] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of OCB2: Attacks on authenticity and confidentiality. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, pages 3–31, 2019.

[82] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012, Proceedings*, pages 273–293, 2012.

[83] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part I*, pages 85–104, 2014.

[84] Charanjit S. Jutla. Encryption modes with almost free message integrity. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceedings*, pages 529–544, 2001.

[85] Wilson Kan. Analysis of underlying assumptions in NIST DRBGs. Cryptology ePrint Archive, Report 2007/345, 2007. `https://eprint.iacr.org/2007/345`.

[86] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* 2nd ed. CRC Press, 2014.

[87] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, pages 284–299, 2000.

[88] John Kelsey. Five DRBG algorithms based on hash functions & block ciphers. *Presentation at NIST Random Number Generation Workshop 2004*, July 2004. Online: `https://csrc.nist.gov/CSRC/media/Events/Random-Number-Generation-Workshop-2004/documents/HashBlockCipherDRBG.pdf`.

[89] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, pages 244–263, 2012.

[90] Jongsung Kim, Guil Kim, Seokhie Hong, Sangjin Lee, and Dowon Hong. The related-key rectangle attack - application to SHACAL-1. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, pages 123–136, 2004.

[91] Jongsung Kim, Guil Kim, Sangjin Lee, Jongin Lim, and Jung Hwan Song. Related-key attacks on reduced rounds of SHACAL-2. In *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, pages 175–190, 2004.

[92] Neal Koblitz and Alfred Menezes. Another look at provable security. `http://cacr.uwaterloo.ca/~ajmeneze/anotherlook/ps.shtml` (viewed September 1 2019).

[93] Neal Koblitz and Alfred Menezes. Another look at "provable security". *Journal of Cryptology*, **20(1)** (2007), pp. 3–37.

[94] Neal Koblitz and Alfred Menezes. Critical perspectives on provable security: Fifteen years of "another look" papers. *Advances in Mathematics of Communications*, **13(4)** (2019), pp. 517–558.

[95] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering* **1(1)** (2011), pp. 5–27.

[96] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 631–648, 2010.

[97] Hugo Krawczyk and Pasi Eronen. HMAC-based extract-and-expand key derivation function (HKDF). *RFC 5869 (Proposed Standard)*, May 2010.

[98] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 429–448, 2013.

[99] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 306–327, 2011.

[100] Mario Lamberger and Florian Mendel. Higher-order differential attack on reduced SHA-256. Cryptology ePrint Archive, Report 2011/037, 2011. `https://eprint.iacr.org/2011/037`.

[101] Iraklis Leontiadis and Serge Vaudenay. Private message franking with after opening privacy. Cryptology ePrint Archive, Report 2018/938, 2018. `https://eprint.iacr.org/2018/938`.

[102] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 31–46, 2002.

[103] Jiqiang Lu, Jongsung Kim, Nathan Keller, and Orr Dunkelman. Related-key rectangle attack on 42-round SHACAL-2. In *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, pages 85–100, 2006.

[104] Stefan Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, pages 343–358, 2002.

[105] ARM MBED. mbed TLS. *Software library.* Online: `https://tls.mbed.org/` (viewed September 1 2019).

[106] Cara McGoogan. FBI director says companies should ditch encryption. *Wired*, December 10 2015. Online: `https://www.wired.co.uk/article/fbi-director-calls-for-encryption-end`.

[107] David McGrew and John Viega. The Galois/counter mode of operation (GCM). *Submission to NIST Modes of Operation Process*, 2004.

[108] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, pages 343–355, 2004.

[109] Ralph Merkle. Secrecy, authentication, and public key systems. *Ph. D. Thesis, Stanford University*, 1979.

[110] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 218–238, 1989.

[111] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, pages 278–296, 2004.

[112] Jon Millican. Challenges of E2E Encryption in Facebook Messenger. *Presentation at Real World Crypto Symposium, New York City, NY, USA*, January 4-6, 2017.

[113] Jon Millican. Personal communication.

[114] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 657–686, 2015.

[115] Payman Mohassel. A closer look at anonymity and robustness in encryption schemes. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 501–518, 2010.

[116] Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In *Computer Security - ESORICS 2004, 9th European Symposium on Research Computer Security, Sophia Antipolis, France, September 13-15, 2004, Proceedings*, pages 335–351, 2004.

[117] Nathaniel Mott. FBI: End-to-end encryption is an infectious problem. *Tom's Hardware*, February 27 2019. Online: `https://www.tomshardware.co.uk/fbi-end-to-end-encryption-infects-law-enforcement,news-60039.html`.

[118] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, pages 1–20, 2006.

[119] Jacques Patarin. The "Coefficients H" technique. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, pages 328–345, 2008.

[120] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application*

of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings, pages 372–389, 2011.

[121] Colin Percival. Cache missing for fun and profit. *BSDCan 2005*, Ottawa, 2005. Online: `https://www.daemonology.net/papers/htt.pdf`.

[122] Nicole Perlroth. Government announces steps to restore confidence on encryption standards. *The New York Times*, September 10 2013. Online: `https://bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards/`.

[123] Nicole Perlroth, Jeff Larson, and Scott Shane. NSA able to foil basic safeguards of privacy on web. *The New York Times*, September 5 2013. Online: `https://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html`.

[124] Bart Preneel. New threat models for cryptography. *Presentation at CryptoDay 2015, Bar-Ilan University, Ramat-Gan, Israel*, December 28 2015. Online: `http://www.cs.technion.ac.il/~biham/Workshops/Cryptoday/2015/Slides/preneel_technion_2015v1_print.pdf`.

[125] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, pages 368–378, 1993.

[126] Bart Preneel, Phillip Rogaway, Mark Dermot Ryan, and Peter Y. A. Ryan. Privacy and security in an age of surveillance (Dagstuhl Perspectives Workshop 14401). *Dagstuhl Reports*, **4(9)** (2014), pp.106–123.

[127] The OpenSSL Project. OpenSSL cryptography and SSL/TLS toolkit. *Software library*. Online: `https://www.openssl.org/` (viewed September 1 2019).

[128] D R. Stinson. Universal hash families and the leftover hash lemma, and applications to cryptography and computing. *Journal of Combinatorial Mathematics and Combinatorial Computing*, **42** (2002), pp. 3–31.

[129] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and*

Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings, pages 487–506, 2011.

[130] Thomas Ristenpart and Scott Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*, 2010.

[131] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, **21(2)** (1978), pp. 120–126.

[132] Phillip Rogaway. Practice-oriented provable security and the social construction of cryptography. *IEEE Security & Privacy*, **14(6)** (2016), pp. 10–17.

[133] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107, 2002.

[134] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pages 16–31, 2004.

[135] Phillip Rogaway. Formalizing human ignorance. In *Progress in Cryptology - VIETCRYPT 2006, First International Conference on Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, pages 211–228. 2006.

[136] Phillip Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015. `https://eprint.iacr.org/2015/1162`.

[137] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 196–205, 2001.

[138] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual Inter-*

national Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings, pages 373–390, 2006.

[139] Phillip Rogaway and John P. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 433–450, 2008.

[140] Phillip Rogaway and John P. Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 220–236, 2008.

[141] Amber Rudd. We don't want to ban encryption, but our inability to see what terrorists are plotting undermines our security. *The Telegraph*, July 31 2017. Online: `https://www.telegraph.co.uk/news/2017/07/31/dont-want-ban-encryption-inability-see-terrorists-plotting-online`.

[142] Sylvain Ruhault. SoK: Security models for pseudo-random number generators. *IACR Transactions on Symmetric Cryptology*, **2017(1)** (2017), pp. 506–544.

[143] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, pages 34–64, 2016.

[144] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 907–922, 2017.

[145] Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24-step SHA-2. In *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, pages 91–103, 2008.

[146] Bruce Schneier, Matthew Fredrikson, Tadayoshi Kohno, and Thomas Ristenpart. Surreptitiously weakening cryptographic systems. Cryptology ePrint Archive, Report 2015/097, 2015. `https://eprint.iacr.org/2015/097`.

[147] Berry Schoenmakers and Andrey Sidorenko. Cryptanalysis of the Dual Elliptic Curve pseudorandom generator. Cryptology ePrint Archive, Report 2006/190, 2006. `https://eprint.iacr.org/2006/190`.

[148] Victor Shoup. OAEP reconsidered. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 239–259, 2001.

[149] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `https://eprint.iacr.org/2004/332`.

[150] Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, pages 643–654, 2008.

[151] Thomas Shrimpton and R. Seth Terashima. A provable-security analysis of Intel's secure key RNG. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 77–100, 2015.

[152] Thomas Shrimpton and R. Seth Terashima. Salvaging weak security bounds for blockcipher-based constructions. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 429–454, 2016.

[153] Dan Shumow and Nils Ferguson. On the possibility of a back door in the NIST SP800-90 Dual EC PRNG. *Presentation at CRYPTO 2007 rump session*, Santa Barbara, CA, USA, August 19–23, 2007. Online: `https://rump2007.cr.yp.to/15-shumow.pdf`.

[154] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983.*, pages 51–67, 1983.

[155] Douglas R Stinson. *Cryptography: theory and practice.* 3$^{\text{rd}}$ ed. Chapman and Hall/CRC, 2005.

[156] Richard Taylor and Andrew Wiles. Ring-theoretic properties of certain Hecke algebras. *Annals of Mathematics*, **141** (1995), pp. 553–572.

[157] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. NIST special publication 800-90B. Recommendation for the entropy sources used for random bit generation. *National Institute of Standards and Technology (NIST)*, January 2018. Online: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf`.

[158] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 222–250, 2019.

[159] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, **7(1-3)** (2012), pp. 1–336.

[160] Wouter van der Linde, Peter Schwabe, and Lejla Batina. Parallel SHA-256 in NEON for use in hash-based signatures. *B.Sc thesis, Radboud University*, 2016.

[161] Apostol Vassilev and Willy May. Annex C: Approved random number generators for FIPS PUB 140-2, security requirements for cryptographic modules. *National Institute of Standards and Technology (NIST)*, June 10 2019. Online: `https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402annexc.pdf`.

[162] Serge Vaudenay. Cryptanalysis of the Chor-Rivest cryptosystem. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 243–256, 1998.

[163] Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 23–30, 1983.

[164] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, **22(3)** (1981), pp. 265–279.

[165] Andrew Wiles. Modular elliptic curves and Fermat's last theorem. *Annals of Mathematics*, **141** (1995), pp. 443–551.

[166] Joanne Woodage, Rahul Chatterjee, Yevgeniy Dodis, Ari Juels, and Thomas Ristenpart. A new distribution-sensitive secure sketch and popularity-proportional hashing. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pages 682–710, 2017.

[167] Joanne Woodage and Dan Shumow. An analysis of NIST SP 800-90A. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 151–180, 2019.

[168] Katherine Q. Ye, Matthew Green, Naphat Sanguansin, Lennart Beringer, Adam Petcher, and Andrew W. Appel. Verified correctness and security of mbedTLS HMAC-DRBG. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2007–2020, 2017.

[169] Adam L. Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 62–74, 1997.