Modelling Cryptographic Attacks by Powerful Adversaries

Marcel Armour

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

Information Security Group
Department of Mathematics
Royal Holloway, University of London

2023

Declaration

These doctoral studies were conducted under the supervision of Dr. Bertram Poettering, Dr. Elizabeth Quaglia and Prof. Carlos Cid.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of 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.

Marcel Armour

October, 2022

Abstract

In this work we consider the ability of powerful adversaries to conduct attacks targeting cryptography. The archetype for a powerful adversary is a nation state actor that has access to resources (in terms of funding, computation, expertise and state apparatus) of a magnitude greater than even well-organised, well-funded and technically proficient "professional" adversaries. The attacks we consider in this work arise from a consideration of motives particular to such powerful adversaries (modelling the interests and capabilities of nation state agencies): firstly, to conduct mass surveillance against populations; and secondly, to restrict access to the wider internet.

The main part of this work considers Algorithm Substitution Attacks (ASAs) against encrypted communication. In light of revelations concerning mass surveillance, ASAs were initially introduced by Bellare, Paterson and Rogaway (Crypto'14) as a novel attack class against the confidentiality of encryption schemes. Such an attack replaces one or more of the regular scheme algorithms with a subverted version that aims to reveal information to an adversary (engaged in mass surveillance), while remaining undetected by users. In this work, we begin by describing a unified framework that provides a generic syntax for ASAs targeting a communication channel between sender and receiver. Our generic syntax allows for formal definitions of the adversary's aims (successful subversion and undetectability). Our framework applies to message authentication schemes (MACs), authenticated encryption (AEAD) and

Public-Key Encryption (PKE).

To introduce some intuition around ASAs, we consider the case study of subverting deniable encryption. Deniable Public Key Encryption (DPKE) is a cryptographic primitive that allows the sender of an encrypted plaintext message to later claim that a different faked plaintext was sent – useful for evading censorship and as a tool for coercion free elections. Discussing deniable encryption allows us to introduce notions of subversion with an interesting example, although the practical implications are limited as no practical DPKE schemes have been proposed to date.

Our main contribution is to present a new class of attack that targets the receiver of a communication between two parties. Our work provides a generic attack that applies to any scheme where a secret key is held by the receiver. Our results rely on the adversary having access to a 'receiver oracle', an interface that allows them to observe whether the receiver algorithm outputs success or failure. By exploiting this receiver oracle, a subverter is able to create a subliminal channel which can leak secret keys. Our generic framework applies to authenticated encryption with associated data, message authentication schemes, public key encryption and Key/Data Encapsulation Mechanism (KEM/DEM) constructions.

We end with a consideration of a class of attack, relying on so-called "partitioning oracles", that continues the theme of powerful adversaries undermining the privacy of users. This class of attack furthermore relies on a similar receiver oracle to the ASAs we consider earlier. Our interest in partitioning oracle attacks arises through the fact that they can be used to prevent users bypassing censorship, by attacking proxy servers (offering users the ability to bypass filtered or censored internet connections). Partitioning oracles were introduced by Len et al. [89] in exactly such a scenario and conceptually take a ciphertext as input and output whether or not the decryption key belongs to some known subset of keys, allowing an adversary to query multiple keys simultaneously. This leads to practical attacks against low entropy keys (e. g., those

derived from passwords), and we discuss some practical scenarios. We show how so-called weak key forgeries against polynomial hash based Authenticated Encryption (AE) schemes, such as AES-GCM, can be leveraged to launch partitioning oracle attacks. A weak key forgery is essentially a MAC forgery that effectively tests whether the key is in some arbitrary set of ('weak') keys.

Our work examines the capability of powerful attackers to undermine cryptography. We are interested in strong adversarial models that extend the "traditional" security model to take into account stronger adversaries. The main contribution of our work is to refine the ASA model and contribute to an ongoing investigation raising awareness and understanding about what is possible with ASAs.

Acknowledgements

Thank you to my supervisors for your support, guidance and inspiration. In chronological order: first year supervisor Kenny Paterson and PhD supervisors Bertram Poettering, Liz Quaglia, Carlos Cid and Liz Quaglia. Thank you to my PhD colleagues, in particular those in the 2017 cohort and those doing cryptography (especially everyone who attended and/or organised the cryptography reading groups, ACRG and YACRG). Thank you also to the CDT for all the opportunities and the many conferences; SPOTNIQ, RWC and Crypto'18 were particular highlights. Thank you to all the ISG staff, who make the ISG the place it is. A special thank you to Claire Hudson for keeping everything running smoothly.

Thank you to the EPSRC for the funding. Thank you to Jean-Paul Degabriele, Christian Janson and Sogol Mazaheri for inviting me to speak at the Darmstadt cryptography seminar. Thank you to Shahram Mossayebi and my colleagues at Crypto Quantique for the internship. Thank you to all the anonymous reviewers.

Doing a PhD in a global pandemic hasn't been easy, but I got there in the end. Thank you to my friends and family for your support. Thank you to my wife Rachel for putting up with me and keeping me going. Thank you to my son Arthur for always cheering me up. Thank you to my parents and my brother for always being there to encourage me. Thank you to my in-laws for all your help and support. Thanks to all my friends for enriching my life.

Thank you to my examiners, Ben Dowling and Darren Hurley-Smith, and the independent chair Keith Martin.

Contents

List of Figures

Acronyms

This thesis repeatedly uses a number of terms which are explained in this section.

AE Authenticated Encryption

AEAD Authenticated Encryption with Associated Data

API Application Programming Interface

ASA Algorithm Substitution Attack

ASIC Application Specific Integrated Circuit

CBC Chained Block Cipher

CCA Chosen-Ciphertext Attack

CPA Chosen-Plaintext Attack

DEM Data Encapsulation Mechanism

DPKE Deniable Public-Key Encryption

EtM Encrypt-then-MAC

FHE Fully Homomorphic Encryption

FPGA Field-Programmable Gate Array

iO Indistinguishability Obfuscation

IV Initialisation Vector

KEM Key Encapsulation Mechanism

MAC Message Authentication Code

NIST National Institute of Standards and Technology

OS Operating System

PAKE Password-Authenticated Key Exchange

PC Personal Computer

PKE Public-Key Encryption

PQC Post-Quantum Cryptography

PRF Pseudo-Random Function

PRNG Pseudo-Random Number Generator

PRP Pseudo-Random Permutation

TPM Trusted Platform Module

Chapter 1

Introduction

Contents

This chapter lays out the motivation behind the thesis, and describes the structure of the document and its contributions. In this work, we consider cryptographic primitives designed to protect the communication of two parties – arguably, the fundamental task of cryptography. We are interested in notions of privacy and confidentiality in the presence of well-resourced adversaries who are interested in surveillance and censorship, such as a state level actor.

1.1 Motivation

Our work is motivated by a number of recent incidents demonstrating the difficulty in securing communications against highly motivated and powerful adversaries. In particular, our adversary definitions model 'state level actors' who present the most sophisticated capabilities and who are able to access resources at a level of magnitude greater than non-state actors. Our main contributions centre around a new class of cryptographic attack that was missed by prior work, and which demonstrate the need for definitions that better model adversarial capabilities. Our work advances

knowledge by refining the definitional framework modelling mass surveillance adversaries, and hopefully serves to raise awareness of the need for cryptography that is resilient against such threats. Our work follows a line of enquiry within cryptography that began with a series of work by Young and Yung [120, 119] and was reignited by the Snowden revelations [14, 73, 77]. The Snowden revelations presented evidence of widespread mass surveillance by the national security agencies of several allied governments. However, the motivation for this thesis is much broader and we consider powerful state-level actors in an abstract sense.

There are many documented examples of states (mis-)using their power to abuse the rights of citizens – for example, the recent case of Pegasus spyware being used to target journalists, activists, opposition politicians and state officials, among others [79]. This illustrates the fact that the greater resources available to state-level actors compared to "regular" adversaries results in a qualitative change in the adversarial strategies that they can employ. These resources are not simply more computational ability. Powerful adversaries have the means to insert unreliability into cryptography via external ("real-world") infrastructure: whether by requesting encryption keys from corporations, influencing standards bodies to adopt "backdoored" parameters, inserting exploitable errors into software implementations, or compromising supply chains to interfere with hardware. The Snowden revelations showed that this is indeed the case, and that large and powerful adversaries (interested in mass surveillance) have sought to circumvent cryptography in an attempt to undermine the security and privacy of users.

Two interesting case studies for subverted encryption are the Dual-EC Pseudo-Random Number Generator (PRNG) incident and the recently surfaced story of Crypto AG.

Dual-EC PRNG Incident. A PRNG should provide a source of random looking

numbers, but the Dual-EC PRNG, a pseudo-random generator that was approved in the NIST SP 800-90A standard [117], had a built in weakness with the choice of elliptic curve points that parametrise the algorithm. The weakness was such that it is possible to choose points in a way that establishes a backdoor – allowing outputs of the PRNG to be predicted. Given a single output from the generator, it is possible (with some computational effort) to recover the algorithm's state, and therefore all future output. The backdoor was practically exploitable due to the fact that protocols like SSL / TLS directly expose PRNG outputs in protocol messages [44].

Crypto AG. The Swiss company Crypto AG built encryption devices that were supplied to more than 120 countries and used to encrypt governmental communications. However, it turns out that Crypto AG was secretly owned by the United States Central Intelligence Agency (CIA) and West German Federal Intelligence Service (Bundesnachrichtendienst, BND). The encryption devices were backdoored, allowing the CIA and BND to read messages [61, 92]. Both of these examples illustrate that it is easier to subvert cryptography than to break it; in particular, one of the silver linings of the Snowden revelations was that there is no evidence that cryptographic hardness assumptions do not hold.

In this thesis, we consider a security model that extends the traditional cryptographic model by considering strong adversaries who are interested in undermining users' security. The traditional model in cryptography, following Kerckhoffs' principle[1], allows an adversary to play a game in which they try to break security (informally: do or learn something that they shouldn't be able to) whilst having full knowledge about the specification of the scheme or protocol. The adversary is usually allowed to observe all outputs, and may even be allowed query access to an oracle. The scheme or protocol is considered secure if the adversary is unable to break the security

[1] That the security of a cryptosystem should depend only on the secret key(s) used.

without knowing the key. However, the assumption that an adversary will play by the rules of a security game does not hold in the real world, given the evidence of the scale of attempts to subvert cryptography. Our work continues a line of study – which we refer to as the ASA framework – that considers adversaries that attack the assumption of trusted cryptographic implementations. The ASA framework removes the clearly delineated boundary between what the adversary is and isn't allowed to do.

1.2 Thesis Structure and Contributions

The thesis is organised as follows.

We begin with the requisite background material in Chapter 2: Standard definitions, including notation, standard primitives and cryptographic security notions.

Chapter 3 introduces Algorithm Substitution Attacks. After discussing related literature, we provide formal definitions for subversion attacks against generic cryptographic primitives consisting of a sender and receiver – including notions of undetectability and adversarial goals. Our syntax allows for both the sending and receiving party to be subverted, in contrast to previous work in this area which considered only subversion of the sender. As such, our definitions are a subtle extension of prior work which allow for a class of attack that was previously missed. We discuss a generic subversion method (rejection sampling) and show that our syntax applies to the primitives considered in this work: symmetric encryption, MACs and PKE schemes. By extending the definitions to capture a broader class of attack, we add to the understanding of the adversarial model.

Chapter 4 provides an introductory case study demonstrating what is possible with

ASAs and illustrating the techniques and framework. We consider subverting Deniable Public-Key Encryption (DPKE) schemes, a primitive that allows the sender of an encrypted plaintext message to later claim that a different plaintext was sent. We work up to the main result gently, beginning with a simpler case study before recalling the definition of DPKE and showing how ASAs against DPKE schemes can subvert deniability. This chapter, beyond providing an introduction to the power of ASAs, demonstrates that subversion attacks should be considered part of the adversarial model for deniability and should be considered in the design of deniable schemes.

Chapter 5 discusses our subversion attacks targeting the receiver. Our subversion attack allows an adversary to learn the user's secret key by observing their communication; once the adversary has learnt the key, the user's cryptography is completely undermined. We consider a passive adversary, following prior work which considers a mass surveillance adversary to be engaged in eavesdropping on a huge scale, but we also consider an active variant. The active variant allows an adversary to target users far more effectively, which makes the attack attractive from the point of view of an adversary. Our attacks work by altering the behaviour of the receiver's algorithm to leak information through (artificially induced) decryption error events – the subverted algorithm either rejects (particular, "trigger") valid ciphertexts or accepts (particular, "trigger") bogus ciphertexts. An adversary observing the receiver who is able to determine whether a ciphertext has been accepted or rejected learns some information; this subliminal channel can be used to exfiltrate the user's key. Our attack, targeting the receiver, is a new class of attack that was missed by previous work.

Chapter 6 continues the theme of powerful adversaries undermining privacy of users. Our attacks in Chapter 5 rely on the assumption that an adversary has access to a decryption oracle – that is, can observe whether a receiver's algorithm implementa-

tion accepts or rejects a ciphertext. In Chapter 6, we discuss an attack class that gives a practical illustration of decryption oracles, which gives some support to the assumption that an adversary can utilise decryption oracles. We show how algebraic properties of polynomial hash-based Authenticated Encryption (AE) schemes can be leveraged to launch so-called partitioning oracle attacks. Partitioning oracle attacks conceptually take a ciphertext as input and output whether or not the decryption key belongs to some known subset of keys. Partitioning oracle attacks allow an adversary to query multiple keys simultaneously, leading to practical attacks against low entropy keys (e. g., those derived from passwords). As users often choose low quality passwords, this presents a practical avenue of attack. We consider the practical setting where proxy servers are used to bypass censorship and where an adversary is actively engaged in targeting users. Our work adds to the understanding of the adversarial model and helps to highlight the criteria for designing cryptographic primitives and protocols.

1.3 Associated Publications

This thesis incorporates the following work.

- The basis for Chapters 3 and 5 is provided by:

  > Marcel Armour and Bertram Poettering. "Algorithm Substitution Attacks against Receivers". In: *International Journal of Information Security*, June 2022.

  This is a journal paper that gives a unified framework bringing together two prior publications (targeting MAC schemes and AEAD schemes) and extending them by showing that public-key encryption may also be targeted. The two publications are:

1.3 Associated Publications

1. Marcel Armour and Bertram Poettering. "Subverting Decryption in AEAD". in: *17th IMA International Conference on Cryptography and Coding*. Ed. by Martin Albrecht. Vol. 11929. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2019
2. Marcel Armour and Bertram Poettering. "Substitution Attacks against Message Authentication". In: *IACR Transactions on Symmetric Cryptology* 2019.3 (2019). ISSN: 2519-173X

- Chapter 4 comprises an extended version of :

  Marcel Armour and Elizabeth A. Quaglia. "Subverting Deniability". In: *Provable and Practical Security*. Ed. by Chunpeng Ge and Fuchun Guo. Cham: Springer Nature Switzerland, 2022.

- Chapter 6 is comprised of:

  Marcel Armour and Carlos Cid. "Partition Oracles from Weak Key Forgeries". In: *Cryptology and Network Security*. Ed. by Mauro Conti, Marc Stevens, and Stephan Krenn. Cham: Springer International Publishing, 2021.

Chapter 2

Preliminaries

Contents

This chapter provides an overview of the notation used within the thesis. We also recall standard cryptographic definitions and give a brief description of the provable security framework that we operate within.

2.1 Notation

Strings. We refer to an element $x \in \{0,1\}^*$ as a string, and denote its length by $|x|$; ε denotes the empty string. The set of strings of length ℓ is denoted $\{0,1\}^\ell$. In addition, we denote by $\perp \notin \{0,1\}^*$ a reserved special symbol. For $x \in \{0,1\}^*$, we let $x[i]$ denote the i-th bit of x, with the convention that we count from 0, i. e., we have $x = x[0] \dots x[|x|-1]$. For two strings x, x' we denote by $x \parallel x'$ their concatenation.

Numbers. Arbitrary finite fields are denoted by $\mathbb{F}$, or when we specify its characteristic by $\mathbb{F}_{p^r}$, with p prime.

Algorithms. We use code-based notation for probability and security experiments. We write $\leftarrow$ for the assignment operator (that assigns a right-hand-side value to a left-hand-side variable). If S, S' are sets, we write $S \xleftarrow{\cup} S'$ shorthand for $S \leftarrow S \cup S'$. If S is a finite set, then $s \leftarrow_\$ S$ denotes choosing s uniformly at random from S. We denote a γ-biased Bernoulli trial by $B(\gamma)$, i.e., a random experiment with possible outcomes 0 or 1 such that $\Pr[b \leftarrow B(\gamma) : b = 1] = \gamma$. The assignments $b \leftarrow_\$ \{0,1\}$ and $b \leftarrow B(1/2)$ are thus equivalent. In security games we use superscript notation $\mathcal{A}^{\mathcal{O}_1,\dots,\mathcal{O}_c} \Rightarrow 1$ to denote the event that the adversary outputs 1 after being given access to the c oracles. For a randomised algorithm A we write $y \leftarrow_\$ \mathsf{A}(x_1, x_2, \dots)$ to denote the operation of running A with inputs $x_1, x_2, \dots$ and assigning the output to variable y. An experiment terminates with a "stop with x" instruction, where value x is understood as the outcome of the experiment. We write "win" ("lose") as shorthand for "stop with 1" ("stop with 0"). We write "require C", for a Boolean condition C, shorthand for "if not C: lose". (We use require clauses typically to abort a game when the adversary performs some disallowed action, e.g.one that would lead to a trivial win.) We use Iverson brackets $[\cdot]$ to derive bit values from

Boolean conditions: For a condition C we have $[C] = 1$ if C holds; otherwise we have $[C] = 0$. The ":=" operator creates a symbolic definition; for instance, the code line "$A := E$" does *not* assign the value of expression E to variable A but instead introduces symbol A as a new (in most cases abbreviating) name for E.

2.2 Provable Security

2.2.1 Provable Security Framework

In this section, we give an overview of the provable security paradigm; much of the discussion is drawn from Katz and Lindell [83]. Provable security is an approach that, as the name suggests, provides a rigorous (mathematical logical) framework to provide arguments for a cryptographic scheme's security. The paradigm shift to provable security is credited to Goldwasser and Micali's 1982 paper [71], although the origins can be traced to the work of Shannon [110] who was the first to pursue a rigorous approach based on precise definitions and mathematical proofs [83]. Prior to the shift, cryptographic schemes were essentially deemed secure if no attacks had been found against the scheme; if attacks were found, the scheme would be patched to restore security. This state of affairs meant that there was no rigorous way to constructively argue for the security of a particular scheme.

Provable security provides a systematic way to argue for the security of a cryptographic scheme, and relies on three principles: a precise description of the protocol and security definitions to be achieved, precise assumptions and the security proof itself. Requiring rigorous definitions and assumptions allows security to be demonstrated by a valid proof that is correct. In theory at least, logical soundness means that the conclusion (scheme X is secure) holds as long as the assumptions are true. This has two effects: firstly, security reduces to an underlying computational hard-

ness assumption; secondly, verifying that a scheme is secure involves checking a mathematical proof – something that could potentially be automated by a formal verification tool. This would seem to suggest that a cryptographic scheme that has been proven secure in the provable security framework is "secure" in the intuitive sense – however, this is not necessarily the case as Koblitz and Menezes argue in a series of papers taking "Another Look at Provable Security" (see [85] for an overview). As they point out, things can go badly wrong in many different ways [85]:

1. The protocol description might implicitly assume that something is of the proper form, but attacks become possible in reality if that is not the case.
2. The definitions might not adequately model real-world adversaries.
3. The evidence for the assumptions might be weak, and we might need to assume that certain components of the protocol behave in an ideal fashion.
4. The proof might have a gap or fallacy.

The criticism presented by Koblitz and Menezes by no means invalidates the provable security framework; by highlighting the inherent limitations, it redirects attention to the possible locations where security may fail. An alternative way to formulate this is that provable security guarantees security against any attack in a given class [116]. In this work, we look at a set of definitions that, we argue, do not adequately model real-world adversaries (Line 2 of the list above). In particular, we consider powerful adversaries that have the means to undermine the integrity of design processes and supply chains to insert unreliability into the implementation of cryptographic primitives and schemes.

2.2.2 Computational Security

Computational security recognises that cryptographic schemes should be considered secure if an adversary with bounded computational power is able to break security with some small probability. Considering an encryption scheme for example, a scheme that allows an eavesdropper to learn an underlying plaintext with probability 2^{-60} after expending 200 years of computational effort with the fastest available supercomputer should reasonably be considered secure. Security definitions for computational security are given in the form of games (also known as experiments) in which an adversary, which we denote $\mathcal{A}$, interacts with a challenger. The challenger generates secret values, and the adversary is given oracle access to algorithms and seeks to "win". Deviating slightly from common usage, we use informal notions to describe the capabilities of an adversary. Our informal notions ("realistic" and "practical") are easily reformulated in terms of probabilistic polynomial-time (PPT) algorithms for readers who prefer a treatment in the asymptotic framework. Given that asymptotic notions don't reflect practice particularly well, we prefer to use the informal terms.

Definition 2.1. *A function $f\colon \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that*

$$f(n) < \frac{1}{p(n)} \ .$$

2.3 Standard Primitives

2.3.1 Pseudo-Random Functions and Permutations

We recall standard notions of pseudo-random functions and permutations.

Intuitively, a pseudo-random function is a "random-looking" function. This means that an adversary interacting with a PRF should be unable to predict outputs of the function. We model this by tasking the adversary with distinguishing between a given PRF and a function chosen at random from the space of all functions with the same domain and range. It makes little sense to talk about a particular function being pseudo-random, and to overcome this we consider a family of PRFs that are parameterised by a secret key.

A particular case is given by functions where the domain and range are the same. These are pseudo-random permutations (PRPs).

2.3.1.1 PRFs

A keyed pseudo-random function (PRF) for range R is an efficiently computable function $F\colon \{0,1\}^\ell \times \{0,1\}^* \to R$ taking a key $L \in \{0,1\}^\ell$ and input $s \in \{0,1\}^*$ to return an output $F(L,s) \in R$. Consider game $\mathrm{PRF}(\mathcal{D})$ in Figure 2.1 (left) associated to function F and distinguisher $\mathcal{D}$. For any adversary $\mathcal{D}$ we define the advantage

$$\mathrm{Adv}_F^{\mathrm{prf}}(\mathcal{D}) := \left|\Pr\left[\mathrm{PRF}^0(\mathcal{D})\right] - \Pr\left[\mathrm{PRF}^1(\mathcal{D})\right]\right|$$

and say that function F is pseudorandom if $\mathrm{Adv}_F^{\mathrm{prf}}(\mathcal{D})$ is negligibly small for all realistic $\mathcal{D}$.

2.3.1.2 PRPs

A keyed length-preserving pseudo-random permutation (PRP) is an efficiently computable function E where $E\colon \{0,1\}^\ell \times \{0,1\}^* \to \{0,1\}^*$ takes a key $L \in \{0,1\}^\ell$ and input $s \in \{0,1\}^*$ to return an output $E(L,s) \in \{0,1\}^{|s|}$. We require that any keyed

instance of E is a permutation on $\{0,1\}^n$ for all $n \in \mathbb{N}$ and also that its inverse E^{-1} is efficiently computable. Consider game $\mathrm{PRP}(\mathcal{D})$ in Figure 2.1 (right) associated to function E and distinguisher $\mathcal{D}$. For any adversary $\mathcal{D}$ we define the advantage

$$\mathrm{Adv}_F^{\mathrm{prp}}(\mathcal{D}) := \left|\Pr\left[\mathrm{PRP}^0(\mathcal{D})\right] - \Pr\left[\mathrm{PRP}^1(\mathcal{D})\right]\right|$$

and say that function E is pseudorandom if $\mathrm{Adv}_E^{\mathrm{prf}}(\mathcal{D})$ is negligibly small for all realistic $\mathcal{D}$.

Game $\mathrm{PRF}^b(\mathcal{D})$	**Game** $\mathrm{PRP}^b(\mathcal{D})$
00 $L \leftarrow_{\$} \{0,1\}^{\ell}, S \leftarrow \emptyset$	00 $L \leftarrow_{\$} \{0,1\}^{\ell}, S \leftarrow \emptyset$
01 $b' \leftarrow \mathcal{D}^{\mathrm{Func}}$	01 $b' \leftarrow \mathcal{D}^{\mathrm{Perm}}$
02 stop with b'	02 stop with b'
Oracle $\mathrm{Func}(s)$	**Oracle** $\mathrm{Perm}(s)$
03 if $(b = 1)$ then $y_s \leftarrow F(L, s)$	03 if $(b = 1)$ then $y_s \leftarrow E(L, s)$
04 else	04 else
05 if $s \notin S$ then $y_s \leftarrow_{\$} R$	05 if $s \notin S$ then $y_s \leftarrow_{\$} \{0,1\}^{\lvert s\rvert}$
06 $S \leftarrow S \cup \{s\}$	06 $S \leftarrow S \cup \{s\}$
07 return y_s	07 return y_s

Figure 2.1: Games to define prf and prp advantage of $\mathcal{D}$ with respect to F, E.

2.3.1.3 Block Ciphers

A block cipher enc is a family of permutations on $\{0,1\}^n$, with each permutation indexed by a key $k \in \mathcal{K}$, where the key space $\mathcal{K} = \{0,1\}^{\ell}$ for some fixed key length ℓ. The application of a block cipher to input $x \in \{0,1\}^n$ using key k will be denoted by $E_k(x)$. In practice, block ciphers are designed to be secure instantiations of pseudorandom permutations with some fixed key length and block length.

2.3.2 Hash Function

A secure cryptographic hash function with output length ℓ is a deterministic function $h\colon \{0,1\}^* \to \{0,1\}^\ell$ that takes an arbitrary-length input x and outputs a string $h(x) = y$ of length ℓ such that:

1. Given y, no realistic adversary can find x (pre-image resistance).
2. Given x, y, no realistic adversary can find an $x' \neq x$ such that $h(x') = y$ (second pre-image resistance).
3. No realistic adversary can find any x, x' such that $h(x) = h(x')$ (collision resistance).

2.3.3 Symmetric Encryption

We require a generic symmetric encryption algorithm for Section 4.2, where we discuss subverting symmetric encryption as an illustrative case study to introduce the concepts of deniability and subversion.

Symmetric encryption allows a sender and receiver who share a secret key to hide the contents of their communications from eavesdroppers. This is modelled using the notion of indistinguishability under chosen-ciphertext attack. The adversary is tasked distinguishing between the encryptions of two different messages; if they are unable to do so, the scheme is considered to be secure (provide confidentiality). We allow the adversary to choose the messages to distinguish between, and we also let the adversary interact with oracles to encrypt and decrypt. Of course, the adversary would trivially be able to distinguish if they query encryptions of the messages they are distinguishing between. As we don't consider this to have broken security in any meaningful sense, this trivial case is ruled out by keeping track of ciphertexts

that the adversary has encrypted and ensuring that these are not forwarded to the decryption oracle.

Our syntax for symmetric encryption surfaces the randomness. Formally, an encryption scheme SE consists of algorithms $\mathsf{SE.gen}, \mathsf{SE.enc}, \mathsf{SE.dec}$. Furthermore, the scheme has associated spaces $\mathcal{K}, \mathcal{R}, \mathcal{M}, \mathcal{C}$. The key generation algorithm $\mathsf{SE.gen}$ outputs a key $k \in \mathcal{K}$. The encryption algorithm $\mathsf{SE.enc}$ takes key $k \in \mathcal{K}$, randomness $r \in \mathcal{R}$ and message $m \in \mathcal{M}$, to produce ciphertext $c \in \mathcal{C}$. We write $c \leftarrow \mathsf{SE.enc}(k, m; r)$; dropping the last input is equivalent to $r \leftarrow_{\$} \mathcal{R}$. The decryption algorithm $\mathsf{SE.dec}$ takes key k and ciphertext $c \in \mathcal{C}$ to output either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection.

We formalise indistinguishability under chosen-ciphertext attack for a symmetric encryption scheme via the game IND-CCA in Figure 2.2 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{SE}}(\mathcal{A}) := \left|\Pr\left[\text{IND-CCA}^0(\mathcal{A})\right] - \Pr\left[\text{IND-CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme SE is indistinguishable against chosen-ciphertext attacks if $\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{SE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

2.3.4 Message Authentication Schemes

Message authentication schemes are a cryptographic primitive designed to provide authentication guarantees (that a message has not been forged or tampered with). MACs can be generically combined with encryption schemes to provide authenticated encryption – however, dedicated schemes providing authenticated encryption (AEAD) offer better performance. We recall definitions for AEAD in Section 2.3.5.

Game IND-CCA$^b(\mathcal{A})$	**Game** IND-CCA$^b(\mathcal{A})$	**Game** IND-CCA$^b(\mathcal{A})$
00 $C \leftarrow \emptyset$	00 $C \leftarrow \emptyset, N \leftarrow \emptyset$	00 $C \leftarrow \emptyset$
01 $k \leftarrow_{\$} \mathsf{SE.gen}$	01 $k \leftarrow_{\$} \mathsf{AEAD.gen}$	01 $(pk, sk) \leftarrow \mathsf{PKE.gen}$
02 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$	02 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$	02 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}(pk)$
03 stop with b'	03 stop with b'	03 stop with b'
Oracle Enc(m^0, m^1)	**Oracle** Enc(n, d, m^0, m^1)	**Oracle** Enc(m^0, m^1)
04 $c \leftarrow \mathsf{SE.enc}(k, m^b)$	04 require $n \notin N$	04 $c \leftarrow \mathsf{PKE.enc}(pk, m^b)$
05 $C \xleftarrow{\cup} \{c\}$	05 $N \xleftarrow{\cup} \{n\}$	05 $C \xleftarrow{\cup} \{c\}$
06 return c	06 $c \leftarrow \mathsf{AEAD.enc}(k, n, d, m^b)$	06 return c
	07 $C \xleftarrow{\cup} \{(n, d, c)\}$	
	08 return c	
Oracle Dec(c)	**Oracle** Dec(n, d, c)	**Oracle** Dec(c)
07 require $c \notin C$	09 require $(n, d, c) \notin C$	07 require $c \notin C$
08 $m \leftarrow \mathsf{SE.dec}(k, c)$	10 $m \leftarrow \mathsf{AEAD.dec}(k, n, d, c)$	08 $m \leftarrow \mathsf{PKE.dec}(sk, c)$
09 return m	11 return m	09 return m

Figure 2.2: Games modelling indistinguishability under chosen-ciphertext attacks (IND-CCA). **Left:** For a symmetric encryption scheme SE, as described in Section 2.3.3. **Centre:** For an authenticated encryption scheme with associated data AEAD, as described in Section 2.3.5.1. **Right:** For a public-key encryption scheme PKE, as described in Section 2.3.7.

Given a key k and a message m, a tag t is deterministically derived as per $t \leftarrow \mathsf{tag}(k, m)$. The (textbook) method to verify the authenticity of m given t is to recompute $t' \leftarrow \mathsf{tag}(k, m)$ and to consider m authentic if and only if $t' = t$. If this final tag comparison is implemented carelessly, a security issue might emerge: A natural yet naive way to perform the comparison is to check the tag bytes individually in left-to-right order until either a mismatch is spotted or the right-most bytes have successfully been found to match. Note that, if tags are not matching, such an implementation might easily give away, as timing side-channel information, the length of the matching prefix, allowing for practical forgery attacks via step-wise guessing.

This issue is understood by the authors of major cryptographic libraries, which thus contain carefully designed constant-time string comparison code. A consequence is that services for tag generation and verification are routinely split into two separate functions tag and vfy.[1] Our notion of a message authentication scheme follows this

[1] See https://nacl.cr.yp.to/auth.html for an example.

Game AUTH($\mathcal{A}$)	**Game** UF($\mathcal{A}$)
00 $k \leftarrow_{\$} \mathsf{AEAD.gen}$	00 $k \leftarrow_{\$} \mathsf{MAC.gen}$
01 $C \leftarrow \emptyset$, $N \leftarrow \emptyset$	01 $C \leftarrow \emptyset$
02 $\mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}$	02 $\mathcal{A}^{\mathrm{Tag},\mathrm{Vfy}}$
03 lose	03 lose
Oracle Enc(n, d, m)	**Oracle** Tag(m)
04 require $n \notin N$	04 $t \leftarrow \mathsf{MAC.tag}(k, m)$
05 $N \xleftarrow{\cup} \{n\}$	05 $C \xleftarrow{\cup} \{(m, t)\}$
06 $c \leftarrow \mathsf{AEAD.enc}(k, n, d, m)$	06 return (m, t)
07 $C \xleftarrow{\cup} \{(n, d, c)\}$	
08 return c	
Oracle Dec(n, d, c)	**Oracle** Vfy(m, t)
09 $m \leftarrow \mathsf{AEAD.dec}(k, n, d, c)$	07 $m \leftarrow \mathsf{MAC.vfy}(k, m, t)$
10 if $m \neq \perp \wedge (n, d, c) \notin C$:	08 if $[m \neq \perp] \wedge [(m, t) \notin C]$:
11 win	09 win
12 return m	10 return m

Figure 2.3: **Left:** Game modelling authenticity (AUTH) of an authenticated encryption scheme with associated data AEAD, as described in Section 2.3.5.2. **Right:** Game modelling unforgeability (UF) of a message authentication scheme MAC, as described in Section 2.3.4.

approach.

Formally, a scheme MAC providing message authentication consists of algorithms MAC.gen, MAC.tag, MAC.vfy and associated spaces $\mathcal{K}, \mathcal{M}, \mathcal{T}$. The key generation algorithm MAC.gen outputs a key $k \in \mathcal{K}$. The tagging algorithm MAC.tag takes a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and returns a message, tag pair $(m, t) \in \mathcal{M} \times \mathcal{T}$. The verification algorithm MAC.vfy takes a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$, and returns either the message m (indicating that the tag is accepted) or the special symbol $\perp$ to indicate rejection.[2] A shortcut notation for this syntax is

$$\mathsf{MAC.gen} \to \mathcal{K} \quad \text{and} \quad \mathcal{K} \times \mathcal{M} \to \mathsf{MAC.tag} \to \mathcal{M} \times \mathcal{T}$$
$$\text{and} \quad \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \mathsf{MAC.vfy} \to \mathcal{M} \cup \{\perp\}\,.$$

[2] It is more common to consider the output of a MAC verification algorithm to be a bit representing acceptance or rejection; this can be obtained from our syntax by evaluating $[\mathsf{MAC.vfy}(k, m, t) = m]$.

We formalise the (strong) unforgeability of a message authentication scheme via the game UF in Figure 2.3 (right). The adversary is tasked with producing a forgery. They are allowed to interact with oracles to tag messages and verify tags. For any adversary $\mathcal{A}$ we define the advantage $\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{MAC}}(\mathcal{A}) := \Pr[\mathrm{UF}(\mathcal{A})]$ and say that the scheme MAC is (strongly) unforgeable if $\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{MAC}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

2.3.5 AEAD Schemes

Authenticated Encryption (with associated data) is a symmetric encryption primitive that in addition to providing confidentiality, provides the guarantee that messages were not tampered with. The notions of confidentiality and authenticity are modelled using the same games as for symmetric encryption (indistinguishability under chosen-ciphertext attacks) and message authentication codes (authenticity).

We recall standard notions of (deterministic) nonce-based AEAD, as per [102]. Formally, a scheme AEAD providing authenticated encryption with associated data consists of algorithms $\mathsf{AEAD.gen}, \mathsf{AEAD.enc}, \mathsf{AEAD.dec}$. The scheme has associated spaces $\mathcal{K}, \mathcal{N}, \mathcal{D}, \mathcal{M}, \mathcal{C}$. The key generation algorithm $\mathsf{AEAD.gen}$ outputs a key $k \in \mathcal{K}$. The encryption algorithm $\mathsf{AEAD.enc}$ takes key $k \in \mathcal{K}$, nonce $n \in \mathcal{N}$, associated data $d \in \mathcal{D}$ and message $m \in \mathcal{M}$, to produce ciphertext $c \in \mathcal{C}$. The decryption algorithm $\mathsf{AEAD.dec}$ takes key k, nonce $n \in \mathcal{N}$, associated data $d \in \mathcal{D}$ and ciphertext $c \in \mathcal{C}$ to output either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection. A shortcut notation for this syntax is

$$\mathsf{AEAD.gen} \to \mathcal{K}, \quad \mathcal{K} \times \mathcal{N} \times \mathcal{D} \times \mathcal{M} \to \mathsf{AEAD.enc} \to \mathcal{C}$$
$$\text{and} \quad \mathcal{K} \times \mathcal{N} \times \mathcal{D} \times \mathcal{C} \to \mathsf{AEAD.dec} \to \mathcal{M} \cup \{\perp\}\,.$$

Scheme AEAD is said to be δ-correct if for $k \leftarrow_{\$} \mathsf{AEAD.gen}$ and $c \leftarrow \mathsf{AEAD.enc}(k, n, d, m)$

for some (n, d, m) and $m' \leftarrow \mathsf{AEAD.dec}(k, n, d, c)$ the probability that $m' \neq m$ is upper-bounded by δ, where the probability is over all coins involved.

2.3.5.1 IND-CCA

We formalise indistinguishability under chosen-ciphertext attack for an AEAD scheme via the game IND-CCA in Figure 2.2 (centre). For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{AEAD}}(\mathcal{A}) := \left|\Pr\left[\text{IND-CCA}^0(\mathcal{A})\right] - \Pr\left[\text{IND-CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme AEAD is indistinguishable against chosen-ciphertext attacks if $\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{AEAD}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

2.3.5.2 Authenticity

We formalise the authenticity of an AEAD scheme via the game AUTH in Figure 2.3 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\text{auth}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\text{AUTH}(\mathcal{A})]$$

and say that AEAD provides authenticity if $\mathrm{Adv}^{\text{auth}}_{\mathsf{AEAD}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

2.3.6 Public-Key Encryption Schemes

A public key (or asymmetric) encryption scheme allows secure communication between parties that have not shared a secret key with one another. PKE works by having two keys: the public key is used to encrypt messages and the private key is used to decrypt. The security of a PKE scheme is determined by the difficulty of determining any information about underlying messages for a given ciphertext. This is modelled using indistinguishability under chosen-ciphertext attacks, analogously to symmetric encryption.

Due to the high overheads associated with PKE, symmetric encryption is better suited to bulk communication. In most practical settings, PKE is used to establish a shared secret between the sender and receiver, so that the shared secret may be used as a key for communicating via symmetric encryption. This notion of sending keys for symmetric encryption via public key methods is formalised as a Key Encapsulation Mechanism (KEM). KEMs are typically used together with a Data Encapsulation Mechanism (DEM) in a so-called hybrid encryption scheme to PKE-encrypt messages. For completeness, a treatment of KEMs and DEMs is given in Appendices A.2.1 and A.2.2.

A PKE scheme $\mathsf{PKE} = (\mathsf{PKE.gen}, \mathsf{PKE.enc}, \mathsf{PKE.dec})$ consists of a triple of algorithms together with key spaces $\mathcal{K}_\mathsf{S}, \mathcal{K}_\mathsf{R}$, a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$. The key-generation algorithm $\mathsf{PKE.gen}$ returns a pair $(pk, sk) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$ consisting of a public key and a private key. The encryption algorithm $\mathsf{PKE.enc}$ takes a public key pk and a message $m \in \mathcal{M}$ to produce a ciphertext $c \in \mathcal{C}$. Finally, the decryption algorithm $\mathsf{PKE.dec}$ takes a private key sk and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection. The correctness requirement is that for $(pk, sk) \leftarrow_\$ \mathsf{gen}$ and $m \in \mathcal{M}$ and $c \leftarrow \mathsf{PKE.enc}(pk, m)$ and $m' \leftarrow \mathsf{PKE.dec}(sk, c)$ the probability that $m' \neq m$ is upper-bounded by δ, where the

probability is over all coins involved.

2.3.7 IND-CCA

We formalise the indistinguishability under chosen-ciphertext attack of a PKE scheme via the game IND-CCA in Figure 2.2 (right). For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{PKE}}(\mathcal{A}) := \left|\Pr\left[\text{IND-CCA}^0(\mathcal{A})\right] - \Pr\left[\text{IND-CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme PKE is indistinguishable against chosen-ciphertext attacks if $\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{PKE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

Chapter 3

Algorithm Substitution Attacks

Contents

This chapter introduces ASAs. We provide formal definitions for subversion attacks against generic cryptographic primitives consisting of a sender and receiver – including notions of undetectability and adversarial goals. Our syntax allows for both the sending and receiving party to be subverted, in contrast to previous work in this area which considered only subversion of the sender. As such, our definitions present a

subtle extension of prior work; in particular, they allow for a class of attack that was previously missed. We discuss a generic subversion method (rejection sampling) and show that our syntax applies to the primitives considered in this work: symmetric encryption, MACs and PKE schemes. We also give a brief overview of the history of the concept and situate our work in relation to other literature.

3.1 Introduction

Consider two parties communicating over an untrusted channel (in the presence of an adversary). Desired security properties for this scenario include confidentiality and integrity. Confidentiality means that the adversary is unable to learn anything about the messages sent between the parties. Integrity means that the parties can be sure that the messages have not been tampered with in transit. Both confidentiality and integrity are well-studied problems and there are many reliable and provably secure cryptographic solutions. These solutions rely on the assumption that the software or hardware in which they are implemented behaves as expected. However, we know that in the real world this assumption does not necessarily hold. As we discussed in Section 1.1, powerful adversaries have the means to insert unreliability into cryptography via external ("real-world") infrastructure: whether by influencing standards bodies to adopt "backdoored" parameters, inserting exploitable errors into software implementations, or compromising supply chains to interfere with hardware. The Snowden revelations showed that this is indeed the case, and that large and powerful adversaries (interested in mass surveillance) have sought to circumvent cryptography.

In the non-ASA setting, security is considered broken if the adversary has some non-negligible advantage in the security game. Translating this directly to the ASA setting, giving the adversary the additional power to replace an algorithm with a

subverted implementation, we can say that the adversary's aim is again to break a scheme's security guarantees. To make the discussion more concrete, let us consider the example of a symmetric encryption algorithm being used by a sender and receiver who share a secret key. In the non-ASA setting, let us assume that the encryption algorithm has been proven secure under chosen-ciphertext attacks. In the ASA setting, the adversary is allowed to replace the users' algorithms with subverted implementations. At first glance, it seems that this gives the adversary unlimited power to undermine confidentiality. As an example, what is stopping the adversary from simply not encrypting messages at all? So then, when a user encrypts their plaintext using the subverted encryption algorithm, the output "ciphertext" is simply the plaintext. When the user sends the "ciphertext" to the receiver, the adversary can simply read the plaintext in transmission and the communicating parties have no confidentiality at all. From the point of view of an adversary, this scenario (whilst a "successful subversion") is unsatisfactory for two reasons: firstly, it is trivially obvious that the subverted implementation is not working "as it should". That is, the subversion is easily detected as the implementation is quite clearly deviating from the algorithm's specification. Secondly, the adversary may want to preserve the users' confidentiality with respect to other eavesdroppers – recalling the case of Crypto AG (Section 1.1) for example. We conclude that although an adversary's aim is to "be able to undermine security", they would like to do this in a way that is hidden from users (or more generally, security auditors). We use the term "undetectability" to describe this notion. There is clearly a tension between successful subversion and detectability – intuitively, the less detectable a subversion is, the less scope for deviating from the implementation and thus successfully compromising users' communications. From the point of view of the adversary, the aim is to be as successful as possible whilst remaining as undetectable as possible.

Lastly, it is worth unpacking the notion of adversarial success. As we noted, in the non-ASA setting, security guarantees are very strict – continuing our example of

symmetric encryption, an adversary is considered to have broken security if they are able to distinguish between the encryptions of two different plaintexts. For the ASA adversaries that we are considering, that is a very weak goal and hardly worth committing the resources required to carry out the subversion attack. Considering that the aim is (mass) surveillance, the adversary ideally wants to know the underlying messages that the users are sending. The most efficient way of doing this is to learn the secret key that is used to encrypt messages. If the adversary learns the secret key, they can decrypt all the communication and obtain the full plaintext correspondence. We discuss adversarial goals in more detail in Section 5.2.

3.1.1 Structure of the Chapter

We give an abstract description of an ASA targeting generic cryptographic schemes consisting of a sender and receiver in Section 3.2, together with notions of undetectability (Section 3.2.1) and adversarial goals (Section 3.2.2); we also briefly describe rejection sampling, a generic subversion technique, in Section 3.2.3. We show that our generic syntax and ASA notions apply to AEAD schemes (Section 3.3.1.2), MAC schemes (Section 3.3.1.3) and PKE schemes (Section 3.3.1.4).

We finish by describing related work in Section 3.4, focussing on ASAs that target symmetric encryption, PKE and MACs. We also discuss defending against ASAs (Section 3.4.4) and consider some practical aspects of subversion through the discussion of "cryptographic versus non-cryptographic subversion" (Section 3.2.4).

The basis for this chapter is provided by:

> Marcel Armour and Bertram Poettering. "Algorithm Substitution Attacks against Receivers". In: *International Journal of Information Security*, June 2022.

3.2 Notions of Subversion Attacks

In this section, we provide formal definitions of success and undetectability for an ASA, as discussed in Section 3.1. We first need to introduce our syntax. We consider subversions of cryptographic schemes implementing encrypted communication between two parties: a sender and receiver. Abstractly, we consider a cryptographic scheme $\Pi = (\Pi.\mathsf{gen}, \{\Pi.\mathsf{S}^{(i)}\}_{0\leq i<n}, \Pi.\mathsf{R})$ consisting of three components: a key generation algorithm, together with a collection of $n \in \mathbb{N}_{>0}$ algorithms on the sender side and an algorithm on the receiver side. This abstraction is designed to be general enough to encompass the primitives that we are interested in: symmetric encryption (including AEAD), MACs and PKE, where there is only one sender algorithm; and deniable PKE, which has multiple sender algorithms.[1] Our abstract framework will be particularly useful in Chapter 5, where we describe a specific ASA that applies to any cryptographic scheme meeting our syntax and where the receiver holds a secret key: this means that our ASA applies automatically to AEAD, MACs and PKE.

In light of the fact that we mainly consider encryption schemes, we let $\Pi.\mathsf{S}^{(0)}$ represent encryption and write $\Pi.\mathsf{S} := \Pi.\mathsf{S}^{(0)}$; we will use the same convention for MAC schemes, where $\Pi.\mathsf{S} := \Pi.\mathsf{S}^{(0)}$ represents the tagging algorithm. We note that our generic syntax allows for the inclusion of randomness generators as well as applying to schemes such as Deniable PKE and FHE which require additional sender algorithms. The receiver algorithm $\Pi.\mathsf{R}$ represents decryption (or verification in the case of MACs).

We give a generic syntax to the scheme Π as follows: Key generation $\Pi.\mathsf{gen}$ outputs a key pair $(k_\mathsf{S}, k_\mathsf{R}) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$. Each sender algorithm $\Pi.\mathsf{S}^{(i)}$, for $0 \leq i < n$, has associated input and output spaces $\mathcal{X}^{(i)}, \mathcal{Y}^{(i)}$ (respectively) and takes as input a sender key $k_\mathsf{S} \in \mathcal{K}_\mathsf{S}$ $x \in \mathcal{X}^{(i)}$, outputting $y \in \mathcal{Y}^{(i)}$; we write $\mathcal{X} := \mathcal{X}^{(0)}, \mathcal{Y} := \mathcal{Y}^{(0)}$.

[1] We note that Fully Homomorphic Encryption (FHE) also fits into this rubric, with multiple sender algorithms, although we do not consider FHE in this work.

The receiver algorithm has associated input and output spaces $\mathcal{Y} := \mathcal{Y}^{(0)}, \mathcal{X}' := \mathcal{X}'^{(0)}$ (respectively). In the event that $n > 0$, we specify the syntax of the additional sender algorithms $\{\Pi.\mathsf{S}_i\}_{1 \leq i \leq n}$ as needed. We note that $\mathcal{X} \subsetneq \mathcal{X}'$; in particular, $\perp \in \mathcal{X}' \setminus \mathcal{X}$. This reflects the fact that $\perp$ is a special symbol that denotes rejection, and is not part of the message space. The receiver algorithm takes as input a receiver key $k_{\mathsf{R}} \in \mathcal{K}_{\mathsf{R}}$ and $y \in \mathcal{Y}$, outputting $x \in \mathcal{X}'$; the special symbol $\perp$ is used to indicate failure. A shortcut notation for this syntax is

$$\Pi.\mathsf{gen} \to \mathcal{K}_{\mathsf{S}} \times \mathcal{K}_{\mathsf{R}}, \quad \mathcal{K}_{\mathsf{S}} \times \mathcal{X}^{(i)} \to \Pi.\mathsf{S}^{(i)} \to \mathcal{Y}^{(i)}, \quad \text{and} \quad \mathcal{K}_{\mathsf{R}} \times \mathcal{Y} \to \Pi.\mathsf{R} \to \mathcal{X}'.$$

Lastly, we foreground the randomness used during encryption in our notation by writing $y \leftarrow \Pi.\mathsf{S}(k_{\mathsf{S}}, x; r)$ for some randomness space $\mathcal{R}$ where we split the input space accordingly as $\mathcal{X} \cong \tilde{\mathcal{X}} \times \mathcal{R}$; dropping the last input is equivalent to $r \leftarrow_{\$} \mathcal{R}$. This allows us to discuss particular values of r that arise during encryption, which will be particularly useful in the discussion of generic rejection sampling techniques (Section 3.2.3) and deniable PKE schemes (Chapter 4).

A scheme Π is said to be δ-correct if for all $(k_{\mathsf{S}}, k_{\mathsf{R}}) \leftarrow \Pi.\mathsf{gen}$ and $(x, r) \in \mathcal{X} \cong \tilde{\mathcal{X}} \times \mathcal{R}$, $y \leftarrow \Pi.\mathsf{S}(k_{\mathsf{S}}, x; r)$ and $x' \leftarrow \Pi.\mathsf{R}(k_{\mathsf{R}}, y)$ we have $\Pr[x' \neq x] \leq \delta$, where the probability is over all random coins involved. In the case that $\delta = 0$, the scheme is said to be perfectly correct. We note that this generic syntax applies to public-key encryption (Section 2.3.6) as well as symmetric encryption (Section 2.3.3), in which case we require $k_{\mathsf{S}} = k_{\mathsf{R}}$.

Having defined the syntax of the cryptographic schemes we consider, we now go on to give formal definitions for subversion of key generation, sender and receiver algorithms, together with the notion of *undetectability* (UD). In a nutshell, a subversion is formalised as undetectable if distinguishers with black-box access to either the original scheme or to its subverted variant cannot tell the two apart. Of course, the

adversary should be able to distinguish between a subverted implementation and an instantiation of the original scheme – in fact, their aim is to do substantially more than this, for example learn information about underlying plaintexts for the subverted version. The apparent contradiction – that a subversion should exhibit a dedicated functionality for the subverting party, but simultaneously be undetectable for all others – is resolved by parameterising the subverted algorithm with a secret subversion key, knowledge of which enables the extra functionality. In what follows we denote the corresponding subversion key spaces with $\mathcal{I}_{\mathsf{gen}}, \mathcal{I}_{\mathsf{S}}$ and $\mathcal{I}_{\mathsf{R}}$.

In this section we also specify, by introducing notions of subliminal information exfiltration Section 3.2.2, how we measure the quality of a subversion from the point of view of the subverting adversary (who is assumed to know the subversion keys). We measure success from the point of view of the adversary by the extent to which they can exfiltrate arbitrary strings – also referred to as a subliminal channel. A special case is obtained when the information exfiltrated is the secret key, discussed in Section 5.2.1, or a string that allows recovery of the secret key (discussed in Section 5.2.1.1).

3.2.1 Undetectable Subversion

We first define undetectability notions for subverted key generation, sender and receiver algorithms separately. We then offer a joint definition. Our definitions are inherited from prior work [23, 20, 51]. Whereas previous work assumed that only the sender algorithm might be subverted, we have generalised the definitions to reflect the possibility that any component (one or multiple) of the scheme could be subverted. Our undetectability games work by giving a detector $\mathcal{A}$ oracle access to either the specification algorithm or a subverted implementation (keyed with an index that is unknown to the detector). The detector interacts with its oracle and

outputs whether it believes the algorithm is real or subverted. In prior work [23, 51], undetectability is defined with respect to uniform keys. As code auditors and other security researchers looking for subversion attacks can specify keys during black-box testing according to their preferred distribution, we consider uniform-key constraints a rather severe limitation of undetectability notions. In particular, subversions that rely on "hiding" the subverted behaviour in a particular subset of the keyspace – that is, if the subverted algorithm behaves correctly when keyed with a non-triggering key, and deviates when keyed with a triggering key in the subset – would be detected more effectively with our notion. As a trivial example, consider a class of subversion that simply outputs the key rather than encrypting a message when the key used is in some subset of the keyspace.[2]

Subverted Key Generation. A subversion of the key generation algorithm $\Pi.\mathsf{gen}$ of a cryptographic scheme consists of a finite index space $\mathcal{I}_{\mathsf{gen}}$ and a family of algorithms $\mathcal{G}en = \{\Pi.\mathsf{gen}_i\}_{i \in \mathcal{I}_{\mathsf{gen}}}$ with

$$\Pi.\mathsf{gen}_i \rightarrow \mathcal{K}_{\mathsf{S}} \times \mathcal{K}_{\mathsf{R}}.$$

That is, for all $i \in \mathcal{I}_{\mathsf{gen}}$ the algorithm $\Pi.\mathsf{gen}_i$ can syntactically replace the algorithm $\Pi.\mathsf{gen}$.

As a security property we require that also the observable behaviour of $\Pi.\mathsf{gen}$ and $\Pi.\mathsf{gen}_i$ be effectively identical (for uniformly chosen $i \in \mathcal{I}_{\mathsf{gen}}$). This is formalised via the games $\mathrm{UDG}^0, \mathrm{UDG}^1$ in Figure 3.1 (left). For any adversary $\mathcal{A}$ we

[2] We note as an aside, that this class of subversion is quite plausible. There are known cases of weak key attacks against cryptographic schemes (see Section 6.2.5). An adversary could plausibly hide artificially introduced weak key behaviour when designing a subversion attack – although, this is outside of the black-box model that we consider here. As an attack vector, leveraging known cryptographic weaknesses to deliberately introduce subversion that retains "plausible deniability" would appear to be a sensible strategy for an ASA adversary.

Game $\mathrm{UDG}^b(\mathcal{A})$	**Game** $\mathrm{UDS}^b(\mathcal{A})$	**Game** $\mathrm{UDR}^b(\mathcal{A})$
00 $i \leftarrow_\$ \mathcal{I}_{\mathsf{gen}}$	00 $i \leftarrow_\$ \mathcal{I}_{\mathsf{S}}$	00 $i \leftarrow_\$ \mathcal{I}_{\mathsf{R}}$
01 $\mathsf{gen}^0 := \Pi.\mathsf{gen}_i$	01 $\mathsf{S}^0 := \Pi.\mathsf{S}_i$	01 $\mathsf{R}^0 := \Pi.\mathsf{R}_i$
02 $\mathsf{gen}^1 := \Pi.\mathsf{gen}$	02 $\mathsf{S}^1 := \Pi.\mathsf{S}$	02 $\mathsf{R}^1 := \Pi.\mathsf{R}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Gen,Send,Recv}}$	03 $b' \leftarrow \mathcal{A}^{\mathrm{Gen,Send,Recv}}$	03 $b' \leftarrow \mathcal{A}^{\mathrm{Gen,Send,Recv}}$
04 stop with b'	04 stop with b'	04 stop with b'
Oracle Gen	**Oracle** Gen	**Oracle** Gen
05 $(k_\mathsf{S}, k_\mathsf{R}) \leftarrow_\$ \mathsf{gen}^b$	05 $(k_\mathsf{S}, k_\mathsf{R}) \leftarrow_\$ \Pi.\mathsf{gen}$	05 $(k_\mathsf{S}, k_\mathsf{R}) \leftarrow_\$ \Pi.\mathsf{gen}$
06 return $(k_\mathsf{S}, k_\mathsf{R})$	06 return $(k_\mathsf{S}, k_\mathsf{R})$	06 return $(k_\mathsf{S}, k_\mathsf{R})$
Oracle $\mathrm{Send}(k_\mathsf{S}, x)$	**Oracle** $\mathrm{Send}(k_\mathsf{S}, x)$	**Oracle** $\mathrm{Send}(k_\mathsf{S}, x)$
07 $y \leftarrow \Pi.\mathsf{S}(k_\mathsf{S}, x)$	07 $y \leftarrow \mathsf{S}^b(k_\mathsf{S}, x)$	07 $y \leftarrow \Pi.\mathsf{S}(k_\mathsf{S}, x)$
08 return y	08 return y	08 return y
Oracle $\mathrm{Recv}(k_\mathsf{R}, y)$	**Oracle** $\mathrm{Recv}(k_\mathsf{R}, y)$	**Oracle** $\mathrm{Recv}(k_\mathsf{R}, y)$
09 $x \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y)$	09 $x \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y)$	09 $x \leftarrow \mathsf{R}^b(k_\mathsf{R}, y)$
10 return x	10 return x	10 return x

Figure 3.1: Games UDG, UDS and UDR modelling undetectability for the subversion of (respectively) key generation, sender and receiver algorithms for a cryptographic scheme Π. See Section 2.1 for the meaning of ":=". Note that in each game, the two unsubverted oracles are actually redundant.

define the advantage

$$\mathrm{Adv}^{\mathrm{udg}}_{\Pi}(\mathcal{A}) := \Big|\Pr[\mathrm{UDG}^1(\mathcal{A})] - \Pr[\mathrm{UDG}^0(\mathcal{A})]\Big|$$

and say that family $\mathcal{G}en$ undetectably subverts algorithm $\Pi.\mathsf{gen}$ if $\mathrm{Adv}^{\mathrm{udg}}_{\Pi}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

Subverted Sender. A subversion of the sender algorithm $\Pi.\mathsf{S}$ of a cryptographic scheme consists of a finite index space $\mathcal{I}_\mathsf{S}$ and a family $\mathcal{S} = \{\mathsf{S}_i\}_{i \in \mathcal{I}_\mathsf{S}}$ of algorithms

$$\mathcal{K}_\mathsf{S} \times \mathcal{X} \to \Pi.\mathsf{S}_i \to \mathcal{Y}.$$

That is, for all $i \in \mathcal{I}_\mathsf{S}$ the algorithm $\Pi.\mathsf{S}_i$ can syntactically replace the algorithm $\Pi.\mathsf{S}$.

As a security property we also require that the observable behaviour of $\Pi.\mathsf{S}$ and $\Pi.\mathsf{S}_i$

be effectively identical (for uniformly chosen $i \in \mathcal{I}_{\mathsf{S}}$). This is formalised via the games $\mathrm{UDS}^0, \mathrm{UDS}^1$ in Figure 3.1 (centre). Note that, in contrast to prior work like [23, 51], our distinguishers are given free choice over the keys to be used.

For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\mathrm{uds}}_{\Pi}(\mathcal{A}) := \left|\Pr[\mathrm{UDS}^1(\mathcal{A})] - \Pr[\mathrm{UDS}^0(\mathcal{A})]\right|$$

and say that family $\mathcal{S}$ undetectably subverts algorithm $\Pi.\mathsf{S}$ if $\mathrm{Adv}^{\mathrm{uds}}_{\mathsf{PKE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

Subverted Receiver. A subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a cryptographic scheme consists of a finite index space $\mathcal{I}_{\mathsf{R}}$ and a family $\mathcal{R} = \{\Pi.\mathsf{R}_i\}_{i \in \mathcal{I}_{\mathsf{R}}}$ of algorithms

$$\mathcal{K}_{\mathsf{R}} \times \mathcal{Y} \to \Pi.\mathsf{R}_i \to \mathcal{X}'.$$

That is, for all $i \in \mathcal{I}_{\mathsf{R}}$ the algorithm $\Pi.\mathsf{R}_i$ can syntactically replace the algorithm $\Pi.\mathsf{R}$.

As a security property we also require that the observable behaviour of $\Pi.\mathsf{R}$ and $\Pi.\mathsf{R}_i$ be effectively identical (for uniformly chosen $i \in \mathcal{I}_{\mathsf{R}}$). This is formalised via the games $\mathrm{UDR}^0, \mathrm{UDR}^1$ in Figure 3.1 (right). For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\mathrm{udr}}_{\Pi}(\mathcal{A}) := \left|\Pr[\mathrm{UDR}^1(\mathcal{A})] - \Pr[\mathrm{UDR}^0(\mathcal{A})]\right|$$

and say that family $\mathcal{R}$ undetectably subverts algorithm $\Pi.\mathsf{R}$ if $\mathrm{Adv}^{\mathrm{udr}}_{\mathsf{PKE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

The above undetectability notions demand that subversions do not change the observable behaviour of the key generation, sender and receiver algorithms. A consequence of this is that none of the correctness or security properties of the scheme are noticeably harmed by subversion. For each of the security properties associated

with the schemes we consider in this thesis (described in Section 2.3) there is a corresponding subverted security property, which is obtained by allowing the attacker to play the security game against the subverted implementation. The subverted security games are given in Section 3.3.

3.2.1.1 Hybrid Subversion of Key Generation, Sender and Receiver Algorithms

We give a joint definition of undetectability, in the case where the key generation, sender and receiver algorithms are subverted. This is the most general definition; in particular contexts it may not be appropriate to consider subversion of a particular algorithm – we discuss this below in Sections 3.3.2.1 to 3.3.2.3 and 5.2.2.

Game UD^b in Figure 3.2 (left) combines games UDG^b, UDS^b and UDR^b into one. We define

$$\mathrm{Adv}^{\mathrm{ud}}_{\Pi}(\mathcal{A}) := \left|\Pr[\mathrm{UD}^1(\mathcal{A})] - \Pr[\mathrm{UD}^0(\mathcal{A})]\right|.$$

By a hybrid argument, for all adversaries $\mathcal{A}$ there exist adversaries $\mathcal{A}', \mathcal{A}'', \mathcal{A}'''$ such that

$$\mathrm{Adv}^{\mathrm{ud}}_{\Pi}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{udg}}_{\Pi}(\mathcal{A}') + \mathrm{Adv}^{\mathrm{uds}}_{\Pi}(\mathcal{A}'') + \mathrm{Adv}^{\mathrm{udr}}_{\Pi}(\mathcal{A}''').$$

The hybrid argument proceeds by considering a series of games subverting each component (key generation, sender algorithm, receiver algorithm) in turn. Applying the triangle inequality results in the relationship given above.

3.2.2 Subversion Leading to Subliminal Information Exfiltration

We observed above that if the sender component $\Pi.\mathsf{S}$ of a cryptographic scheme Π is undetectably subverted, with uniformly chosen index i_{S} that remains unknown to

Game $\mathrm{UD}^b(\mathcal{A})$
00 $i_{\mathsf{gen}} \leftarrow_\$ \mathcal{I}_{\mathsf{gen}}; i_{\mathsf{S}} \leftarrow_\$ \mathcal{I}_{\mathsf{S}}; i_{\mathsf{R}} \leftarrow_\$ \mathcal{I}_{\mathsf{R}}$
01 $(\mathsf{gen}^0, \mathsf{S}^0, \mathsf{R}^0) := (\Pi.\mathsf{gen}_{i_{\mathsf{gen}}}, \Pi.\mathsf{S}_{i_{\mathsf{S}}}, \Pi.\mathsf{R}_{i_{\mathsf{R}}})$
02 $(\mathsf{gen}^1, \mathsf{S}^1, \mathsf{R}^1) := (\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R})$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Gen,Send,Recv}}$
04 stop with b'

Oracle Gen
05 $(k_{\mathsf{S}}, k_{\mathsf{R}}) \leftarrow_\$ \mathsf{gen}^b$
06 return $(k_{\mathsf{S}}, k_{\mathsf{R}})$

Oracle $\mathrm{Send}(k_{\mathsf{S}}, x)$
07 $y \leftarrow \mathsf{S}^b(k_{\mathsf{S}}, x)$
08 return y

Oracle $\mathrm{Recv}(k_{\mathsf{R}}, y)$
09 $x \leftarrow \mathsf{R}^b(k_{\mathsf{R}}, y)$
10 return x

Figure 3.2: Game UD modelling hybrid subversion undetectability for a cryptographic scheme Π. Note that not all of the algorithms need necessarily be subverted, although the syntax allows for this. See the discussion at Sections 3.3.2.1 to 3.3.2.3.

the participants, then all security guarantees are preserved from the original scheme. This may be different if i_{S} is known to an attacking party, and indeed we assume that mass-surveillance attackers leverage such knowledge to conduct attacks.

Abstractly, the aim of an adversary is to exfiltrate some subliminal information. In the context of prior work considering symmetric encryption, this information typically represents the secret key. We formalise this goal as the MR_μ game in Figure 3.3 (left), which assumes a passive attack in which the adversary eavesdrops on communication, observing the transmitted ciphertexts. We allow the adversary some influence over sender inputs, with the aim of closely modelling real-world settings. This influence on the sender inputs x is restricted by assuming a stateful "message sampler" algorithm MS (reflecting the fact that, in the contexts we consider, inputs to $\Pi.\mathsf{S}$ typically represent messages) that produces the inputs to $\Pi.\mathsf{S}$ used throughout the game. The syntax of this message sampler is

$$\Sigma \times A \to \mathrm{MS} \to \Sigma \times \mathcal{X} \times B, \quad (\sigma, \alpha) \mapsto \mathrm{MS}(\sigma, \alpha) = (\sigma', x, \beta),$$

Game $\mathrm{MR}_\mu(\mathcal{A})$
00 $i \leftarrow_\$ \mathcal{I}_{\mathsf{S}}$
01 $(k_{\mathsf{S}}, k_{\mathsf{R}}) \leftarrow_\$ \Pi.\mathsf{gen}$; $\sigma \leftarrow \diamond$
02 $\mu' \leftarrow \mathcal{A}^{\mathrm{Send}}(i)$
03 stop with $[\mu' = \mu]$

Oracle $\mathrm{Send}(\alpha)$
04 $(\sigma, x, \beta) \leftarrow \mathrm{MS}(\sigma, \alpha)$
05 $y \leftarrow \Pi.\mathsf{S}_i(k_{\mathsf{S}}, x)$
06 return (y, β)

Figure 3.3: Game MR_μ modelling subliminal message recoverability for passive adversaries. As we discuss, in the settings we consider the subliminal message will consist of the user's secret key, knowledge of which allows the adversary to completely break the security of the subverted scheme.

where $\sigma, \sigma' \in \Sigma$ are old and updated state, input $\alpha \in A$ models the influence that the adversary may have on message generation, and output $\beta \in B$ models side-channel outputs. In Figure 3.3 we write $\diamond$ for the initial state. Note that while we formalise the inputs α and the outputs β for generality (so that our models cover most real-world applications), our subversion attacks are independent of them. For any message sampler MS and adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\mathrm{mr}}_{\Pi,\mathrm{MS},\mu}(\mathcal{A}) := \Pr[\mathrm{MR}_\mu(\mathcal{A})].$$

We say that subversion family $\mathcal{S}$ is key recovering for passive attackers if for all practical MS there exists a realistic adversary $\mathcal{A}$ such that $\mathrm{Adv}^{\mathrm{mr}}_{\Pi,\mathrm{MS},\mu}(\mathcal{A})$ reaches a considerable value (e.g., 0.1).[3]

3.2.3 Generic Method: Rejection Sampling

We describe a generic method to embed a subliminal message μ with $|\mu| = \ell_\mu$ into ciphertexts of an encryption scheme $\Pi.\mathsf{S}$. Essentially, when computing a ciphertext,

[3] As discussed in Section 2.2.2, we prefer to use informal notions ("realistic" and "practical") which are easily reformulated in terms of probabilistic polynomial-time (PPT) algorithms for readers who prefer a treatment in the asymptotic framework.

Proc $\Pi.\mathsf{S}_i(k_\mathsf{S}, x, \mu)$	**Proc** $\mathcal{A}(i)$
00 while $[t \neq \mu]$:	00 pick any α
01 $\quad r \leftarrow_\$ \mathcal{R}$	01 $(y, \beta) \leftarrow \text{Send}(\alpha)$
02 $\quad y \leftarrow \Pi.\mathsf{S}_i(k_\mathsf{S}, x; r)$	02 $\mu \leftarrow F_i(y)$
03 $\quad t \leftarrow F_i(y)$	03 return μ
04 return y	

Figure 3.4: Rejection sampling subversion $\Pi.\mathsf{S}_i$ of encryption algorithm $\Pi.\mathsf{S}$. **Left:** Subverted encryption algorithm as in Section 3.2.1. **Right:** Message recovering adversary for game MR as in Section 3.2.2. The adversary need not have any influence over messages (modelled by α; see the discussion at Section 3.2.2).

the subverted algorithm uses rejection sampling to choose randomness that results in a ciphertext that encodes the subliminal message. We define a subversion of the encryption algorithm $\Pi.\mathsf{S}$ of a scheme Π in Figure 3.4 (left). It is parameterised by a large index space $\mathcal{I}$, a constant ℓ_μ and a PRF F_i. For the PRF we require that it be a family of functions $F_i \colon \mathcal{Y} \to \{0,1\}^{\ell_\mu}$ (that is: a pseudo-random mapping from the ciphertext space to the set strings of length ℓ_μ). We write $\Pi.\mathsf{S}_i$ for the subverted algorithm. We give a corresponding message recovery adversary in Figure 3.4 (right).

We note that the subverted encryption algorithm $\Pi.\mathsf{S}_i$ will resample randomness 2^{ℓ_μ} times on average. This means that longer messages result in exponentially slower running times of the algorithm; in practice, this means that the attack is limited to short messages (a few bits at most). We note that this technique embeds a message of length ℓ_μ in each ciphertext; in later sections we use this idea to exfiltrate a message that is derived from the plaintext being encrypted. More generally, each subliminal message μ could be the fragment of a larger message μ' (e.g. representing the secret key, as is the approach in prior work targeting symmetric encryption). It is straightforward to see how this would work for a stateful algorithm (simply send the bits in order); for a stateless algorithm, Bellare et al. [20] show that if individual ciphertexts embed messages of length ℓ_μ then it is possible to exfiltrate a string μ' of length 2^{ℓ_μ} by letting each individual μ encode the ${\ell_\mu}^{\text{th}}$ bit of μ'.

3.2.4 Cryptographic vs. Non-Cryptographic Subversion

In the literature on cryptography, the notion of an ASA assumes the malicious replacement of one or more algorithms of a scheme by a backdoored version, with the goal to leak key material, or at least to weaken some crucial security property. Different types of substitution attack appear in other areas of computing and communication. We discuss some examples in the following.

Program code in the domain of computer malware routinely modifies system functions to achieve its goals, where the latter comprises delivering some damaging payload, ensuring non-detection and thus survival of the malware on the host system, and in some cases even self-reproduction. Numerous techniques towards suitably modifying a host system have been developed and reported on by academic researchers and hackers. Standard examples include redirecting interrupt handlers, changing the program entry point of an executable file, and interfering with the OS kernel by overwriting its data structures [72].

Malicious modifications of implemented functionality are also a recognised threat in the hardware world. It is widely understood that circuit designers who do not possess the technical means to produce their own chips but instead out-source the production process to external foundries, risk that the chips produced might actually implement a maliciously modified version of what is expected. A vast number of independent options are known for when (within the production cycle) and how (functionally) subversions could be conducted. For instance, the survey provided in [33] reports that circuit design software (CAD) could be maliciously altered, that foundries could modify circuits before production, and that after production commercial suppliers could replace legitimate chips by modified ones. Further, [33] suggests that appealing types of functionality modification include deviating from specification when particular input trigger events are recognised, and/or to leak values of vital internal

registers via explicitly implemented side channels. Any such technique (or combination thereof) has an individual profile regarding the associated costs and attack detectability. Which of the many options is most preferable depends on the specific attack scenario and target.

We refer to the software and hardware based subversion techniques discussed above as "technology driven". This is in contrast to the techniques considered in this thesis which we refer to as "semantics driven". We consider the two approaches orthogonal: Our (semantics driven) proposed subversion *can* be implemented using techniques from e.g. [72, 33] (but likewise also through standard methods), and technology driven subversion proposals *can* be applied against cryptographic implementations (but likewise also against any other interesting target functionality). Our semantics driven approach in fact aims to maximise technology independence. As a consequence, the line of attacks proposed in this thesis can be implemented easily in software (e.g. in libraries or drop-in code), in hardware (e.g. in ASICs and FPGAs), and in mixed forms (e.g. firmware-programmed microcontrollers). The strategy to achieve this independence is to base the attacks and corresponding notions of (in)security on nothing but the abstract functionalities of the attacked scheme as they are determined by their definitions of syntax and correctness.

As the technology driven and semantics driven approaches are independent, they can in particular be combined. This promises particularly powerful subversions. For instance, consider that virtually all laptops and desktop PCs produced in the past decade are required to have an embedded trusted platform module (TPM) chip that supports software components (typically boot loaders and operating systems) with *trusted* cryptographic services. In detail, software can interact with a TPM chip through standardised API function calls and have cryptographic operations applied to provided inputs, with all key material being generated and held exclusively in the TPM. As TPMs are manufactured in hardware, it seems that the (technology

driven) subversion options proposed in [33] would be particularly suitable. However, as most of the attacks from [33] require physical presence of the adversary (e.g., to provide input triggers via specific supply voltage jitters or for extracting side channel information by operating physical probes in proximity of the attacked chip), only those options seem feasible where all attack conditions and events can be controlled and measured via the software interface provided by the API. This is precisely what our semantics driven attacks provide. We thus conclude by observing that dedicated cryptographic hardware like TPMs can only be trusted if *extreme* care is taken during design and production.

3.3 Subverting Primitives

In this section we note that our abstract syntax accommodates AEAD, MACs and PKE. In Chapter 5 we discuss a class of subversion attacks that generically targets any scheme meeting our syntax which thus applies to AEAD, PKE and MAC schemes.

3.3.1 Applying our Syntax

We note that the generic syntax introduced above in Section 3.2 is satisfied by symmetric encryption schemes, AEAD schemes, MAC schemes and PKE schemes. We may thus apply the generic notions of subversion and undetectability introduced in Section 3.2.1. For each primitive, we specify the subverted security games corresponding to the security properties given in Section 2.3. Each subverted security game models the adversary's ability to compromise the expected security properties of a scheme Π when that scheme has been subverted.

3.3.1.1 Symmetric Encryption

We note that symmetric encryption satisfies the generic syntax, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{SE.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{SE.enc}$ and receiver algorithm $\Pi.\mathsf{R} = \mathsf{SE.dec}$. We specify the game subIND-CCA in Figure 3.5 (left).

3.3.1.2 AEAD

We note that AEAD satisfies the generic syntax, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{AEAD.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{AEAD.enc}$ and receiver algorithm $\Pi.\mathsf{R} = \mathsf{AEAD.dec}$. We specify the games subIND-CCA and subAUTH in Figure 3.5 (right) and Figure 3.6 (left), respectively.

3.3.1.3 MACs

We note that MACs satisfy the generic syntax, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{MAC.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{MAC.tag}$, receiver algorithm $\Pi.\mathsf{R} = \mathsf{MAC.vfy}$. We specify the game subUF in Figure 3.6 (right).

3.3.1.4 PKE Schemes

We note that PKE schemes satisfy the generic syntax, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{PKE.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{PKE.enc}$, and receiver algorithm $\Pi.\mathsf{R} = \mathsf{PKE.dec}$. See Figure 3.7 for the game subIND-CCA.

3.3.2 Discussion

3.3.2.1 Symmetric Encryption and AEAD

We note that for symmetric primitives, key generation is unlikely to be subverted in practice as symmetric keys are typically generated by some external means not connected with or influenced by the scheme itself — e.g. through key agreement protocols, or by a trusted platform module. Nevertheless, we retain a syntax that allows for the more general case.

3.3.2.2 MACs

We note that for symmetric primitives, key generation is unlikely to be subverted, leaving us with the possibility that either the tagging or the verification algorithm (or both) could be subverted. However, as tagging and verification are typically performed by distinct, remote parties, successfully conducting such attacks would require replacing implementations of *two* participants, which we think is considerably more demanding for an adversary than replacing only one implementation.

3.3.2.3 PKE schemes

Berndt and Liśkiewicz [27] show that a generic ASA against an encryption scheme can only embed a limited number of bits per ciphertext. More concretely, they show that no universal and consistent[4] ASA is able to embed more than $\log(\kappa)$ bits of information into a single ciphertext in the random oracle model [27, Theorem 1.4],

[4]Here universal means that the ASA applies generically to any encryption scheme, and consistent essentially means that the ASA outputs genuine ciphertexts. We note that the rejection sampling ASA (Section 3.2.3) is universal and consistent, whereas IV replacement attacks (e.g. as discussed in Section 4.2) are not, failing to be universal.

where κ is the key length of the encryption scheme. In the setting of symmetric key encryption, this is sufficient to successfully leak the secret key over multiple ciphertexts ([23, 20]). However, for asymmetric primitives, subverting ciphertexts to leak the encryption key makes little sense as it is public; leaking plaintext messages is not possible due to the limited bandwidth. Thus for generic ASAs against PKE, the best possible adversarial goal is to exfiltrate sufficient information to compromise confidentiality – knowledge of one or two bits of the underlying plaintext message is sufficient to allow an adversary to break confidentiality in the sense of IND-CPA or IND$.[5] But as Bellare et al. [20] argue, this is not an attractive goal for a mass surveillance adversary, who would rather break confidentiality completely and recover plaintext messages. Thus for PKE schemes, subverting the receiver algorithm to leak secret keys is the only possible option for an effective ASA.

For PKE schemes, in contrast to symmetric encryption, subverting the key generation algorithm is a meaningful option, and we explain in Section 5.2.2 how subversion attacks can be amplified when applied together with a subverted key generation algorithm. In Chapter 4 we furthermore consider DPKE, a special type of PKE scheme that allows users to plausibly deny the message that they sent. As DPKE is not a standard notion, we leave the definition to Section 4.3, where we also introduce the notion in some depth.

3.4 Related Work

In this section, we give a brief overview of the history of ASAs, focussing on symmetric encryption and PKE, and situate our work in relation to other literature.

[5] Chen et al.[45] overcome these limitations by using non-generic techniques against KEM-DEM constructions to leak underlying plaintext messages representing (session) keys.

Game subIND-CCA$^b(\mathcal{A})$
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $C \leftarrow \emptyset$
02 $k \leftarrow \mathsf{SE.gen}_{i_{\mathsf{gen}}}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
04 stop with b'

Oracle Enc(m^0, m^1)
05 $c \leftarrow \mathsf{SE.enc}_{i_{\mathsf{S}}}(k, m^b)$
06 $C \xleftarrow{\cup} \{c\}$
07 return c

Oracle Dec(c)
08 require $c \notin C$
09 $m \leftarrow \mathsf{SE.dec}_{i_{\mathsf{R}}}(k, c)$
10 return m

Game subIND-CCA$^b(\mathcal{A})$
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $C \leftarrow \emptyset, N \leftarrow \emptyset$
02 $k \leftarrow \mathsf{AEAD.gen}_{i_{\mathsf{gen}}}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
04 stop with b'

Oracle Enc(n, d, m^0, m^1)
05 require $n \notin N$
06 $N \xleftarrow{\cup} \{n\}$
07 $c \leftarrow \mathsf{AEAD.enc}_{i_{\mathsf{S}}}(k, n, d, m^b)$
08 $C \xleftarrow{\cup} \{(n, d, c)\}$
09 return c

Oracle Dec(n, d, c)
10 require $(n, d, c) \notin C$
11 $m \leftarrow \mathsf{AEAD.dec}_{i_{\mathsf{R}}}(k, n, d, c)$
12 return m

Figure 3.5: Games modelling subverted indistinguishability under chosen-ciphertext attacks (subIND-CCA). **Left:** For a subverted symmetric encryption scheme SE, as described in Section 3.3.1.1. **Right:** For a subverted authenticated encryption scheme with associated data AEAD, as described in Section 3.3.1.2.

3.4.1 Symmetric Encryption

The idea that an adversary may embed a backdoor or otherwise tamper with the implementation or specification of a cryptographic scheme or primitive predates the Snowden revelations, and was initiated in a line of work by Young and Yung that they named *kleptography* [120, 119]. This area of study can be traced back to Simmons' work on *subliminal channels*, e.g. [111], undertaken in the context of nuclear non-proliferation during the Cold War. In the original conception [120], kleptography considered a saboteur who designs a cryptographic algorithm whose outputs are computationally indistinguishable from the outputs of an unmodified trusted algorithm. The saboteur's algorithm should leak private key data through the output of the system, which was achieved using the same principles as Simmons' earlier subliminal channels. Post-Snowden, work in this area was reignited by Bellare, Paterson and Rogaway (BPR) [23], who formalised the study of so-called Algorithm

Game subAUTH($\mathcal{A}$)
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $k \leftarrow_{\$} \mathsf{AEAD.gen}_{i_{\mathsf{gen}}}$
02 $C \leftarrow \emptyset$, $N \leftarrow \emptyset$
03 $\mathcal{A}^{\mathrm{Enc,Dec}}$
04 lose

Oracle Enc(n, d, m)
05 require $n \notin N$
06 $N \xleftarrow{\cup} \{n\}$
07 $c \leftarrow \mathsf{AEAD.enc}_{i_{\mathsf{S}}}(k, n, d, m)$
08 $C \xleftarrow{\cup} \{(n, d, c)\}$
09 return c

Oracle Dec(n, d, c)
10 $m \leftarrow \mathsf{AEAD.dec}_{i_{\mathsf{R}}}(k, n, d, c)$
11 if $m \neq \perp \wedge (n, d, c) \notin C$:
12 win
13 return m

Game subUF($\mathcal{A}$)
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $k \leftarrow_{\$} \mathsf{MAC.gen}_{i_{\mathsf{gen}}}$
02 $C \leftarrow \emptyset$
03 $\mathcal{A}^{\mathrm{Tag,Vfy}}$
04 lose

Oracle Tag(m)
05 $t \leftarrow \mathsf{MAC.tag}_{i_{\mathsf{S}}}(k, m)$
06 $C \xleftarrow{\cup} \{(m, t)\}$
07 return (m, t)

Oracle Vfy(m, t)
08 $m \leftarrow \mathsf{MAC.vfy}_{i_{\mathsf{R}}}(k, m, t)$
09 if $[m \neq \perp] \wedge [(m, t) \notin C]$:
10 win
11 return m

Figure 3.6: **Left:** Game modelling authenticity (subAUTH) of a subverted authenticated encryption scheme with associated data AEAD, as described in Section 3.3.1.2. **Right:** Game modelling unforgeability (subUF) of a subverted message authentication scheme MAC, as described in Section 3.3.1.3.

Game subIND-CCA$^b(\mathcal{A})$
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $C \leftarrow \emptyset$
02 $(pk, sk) \leftarrow \mathsf{PKE.gen}_{i_{\mathsf{gen}}}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}(pk)$
04 stop with b'

Oracle Enc(m^0, m^1)
05 $c \leftarrow \mathsf{PKE.enc}_{i_{\mathsf{S}}}(pk, m^b)$
06 $C \xleftarrow{\cup} \{c\}$
07 return c

Oracle Dec(c)
08 require $c \notin C$
09 $m \leftarrow \mathsf{PKE.dec}_{i_{\mathsf{R}}}(sk, c)$
10 return m

Figure 3.7: Game modelling indistinguishability under chosen-ciphertext attacks (IND-CCA) for a subverted public-key encryption scheme PKE.

Substitution Attacks (ASAs) through the example of symmetric encryption schemes.

BPR [23] demonstrate an attack against certain randomised encryption schemes that relies on influencing the randomness consumed in the course of encryption. Their attack, which they call the "biased-ciphertext attack", is a generic method that relies on rejection sampling. Randomness is resampled until ciphertexts satisfy a particular format (for example, implanting information in the least significant bits), resulting in a subliminal channel.

There is a tension for "Big Brother" between mounting a successful attack and being detected; clearly an attack that simply replaces the encryption algorithm with one that outputs the messages in plaintext would be devastating yet trivially detectable. BPR stipulate that ciphertexts generated with a subverted encryption algorithm should at the very least decrypt correctly with the unmodified decryption routine, in order to have some measure of resistance to detection. Furthermore, BPR define the success probability of a mass surveillance adversary in carrying out a successful attack, as well as the advantage of a user in detecting that a surveillance attack is taking place. The attack of BPR was later generalised by Bellare, Jaeger and Kane (BJK) [20] whose attack applies to all randomised schemes. Furthermore, whereas the attack of BPR is stateful and so vulnerable to detection through state reset, the BJK attack is stateless. BJK [20] later formalised the goal of key recovery as the desired outcome of an ASA from the point of view of a mass surveillance adversary. Lastly, BPR also establish a positive result that shows that under certain assumptions, it is possible for authenticated encryption schemes to provide resistance against subversion attacks.

Degabriele, Farshim and Poettering (DFP) [51] critiqued the definitions and underlying assumptions of BPR. Their main insight is that the perfect decryptability —a condition mandated by BPR— is a very strong requirement and artificially limits the adversary's set of available strategies. In practice, a subversion with negligible

detection probability, say 2^{-128}, should be considered undetectable.[6] As DFP note, decryption failures may happen for reasons other than a subverted encryption algorithm, and if they occur sporadically may easily go unnoticed. Thus a subverted encryption scheme that exhibits decryption failure with a very low probability is a good candidate for a practical ASA that is hard to detect. DFP demonstrate how this can be achieved with an input-triggered subversion, where the trigger is some message input that is difficult to guess, making detection practically impossible. Our work complements the trigger message approach of DFP by limiting ciphertext integrity and establishing a covert channel through decryption error events.

3.4.2 Public-Key Encryption

Yung and Young (YY) in [120] examine subverting asymmetric protocols in so-called "SETUP" attacks. Their core idea is to encode some information within the public key that allows the private key to be reconstructed. As a simple example, let the public key encode the encryption of the user's private key under the adversary's key. Subverted keys should be indistinguishable from real keys and only the adversary should be able to recover a user's private key from the subverted public key. As well as showing how to subvert RSA keys, YY also give examples of attacks against ElGamal, DSA and Kerberos. Later, Crépeau and Slakmon [49] gave an improved subversion attack against RSA which works by hiding half of the bits of p in the representation of the RSA modulus $N = pq$. Using Coppersmith's partial information attack [46], it is then possible to recover p and q.

For the prior work on symmetric encryption discussed above, the techniques can be translated naturally into a PKE setting. Attacks against the encryption algorithm of a PKE scheme however do not present an attractive attack to a mass surveillance

[6] This is analogous to the fundamental notion in cryptography that a symmetric encryption scheme be considered secure even in the presence of adversaries with negligible advantage.

adversary, as there is limited scope to undermine confidentiality. The covert channel usually has a bandwidth of a small number of bits per (subverted) ciphertext: not enough to leak the underlying messages. Leaking the private key would allow confidentiality to be broken completely, but the encryption algorithm does not have access to the private key. Chen, Huang and Yung [45] overcome these limitations by considering hybrid PKE constructions consisting of a KEM to send encapsulated session keys which are used for symmetric encryption with a DEM. Their non-generic attack applies to a particular class of practical KEM constructions and leaks session keys, that in turn break the security of the DEM. In contrast, for a PKE primitive not consisting of a hybrid KEM/DEM construction, targeting the decryption algorithm remains the only way to subvert the encryption/ decryption facility of a PKE scheme.

3.4.3 Further Work

Other works, briefly described here, consider subversion on different primitives and in different contexts. Berndt and Liśkiewicz [27] reunite the fields of cryptography and steganography. Goh, Boneh, Pinkas and Golle [70] show how to add key recovery to the SSH and SSL/TLS protocols. Ateniese, Magri and Venturi [12] study ASAs on signature schemes. Berndt et al. consider ASAs against protocols such as TLS, WireGuard and Signal [28]. Dodis, Ganesh, Golovnev, Juels and Ristenpart [55] provide a formal treatment of backdooring PRGs, another form of subversion. This work was extended by Degabriele, Paterson, Schuldt and Woodage [52] to look at robust PRNGs with input. Camenisch, Drijvers and Lehmann [38] consider Direct Anonymous Attestation in the presence of a subverted Trusted Platform Module.

3.4.4 Defending Against Subversion Attacks

Achieving security against adversaries mounting ASAs is difficult, and essentially reduces to assuming trust in particular components or architectures. The three main theoretical approaches to preventing or mitigating against ASAs in the literature are reverse firewalls, self-guarding protocols and watchdogs. Other approaches include: deterministic PKE schemes that defend against the subversion of random number generators, as discussed by Bellare and Hoang [19]; large keys that make exfiltration infeasible, as explored by Bellare, Kane and Rogaway [21]; the use of state reset to detect ASAs, as studied by Hodges and Stebila [76].

Cryptographic reverse firewalls [93, 54, 90, 114, 37] represent an architecture to counter ASAs against asymmetric cryptography via trusted code in network perimeter filters. At a high level, the approach is for a trusted third party to re-randomise ciphertexts before transmission over a public network to destroy any subliminal messages. Fischlin and Mazaheri show how to construct 'self-guarding' ASA-resistant (asymmetric) encryption and signature algorithms given initial access to a trusted base scheme [67]. Their approach uses trusted samples to essentially perform re-randomisation of ciphertexts.

In a series of works, Russell, Tang, Yung and Zhou [103, 105, 106, 104] study ASAs on one-way functions, trapdoor one-way functions and key generation as well as defending randomised algorithms against ASAs using so-called watchdogs. The watchdog model considers splitting a primitive into constituent algorithms that are run as subroutines by a trusted "amalgamation" layer. This allows the constituent algorithms to be individually checked and sanitised, in a variety of different assumptions (e.g. on- or offline, black- or whitebox access). Combiners are often used to provide subversion resilience, particularly in the watchdog model. A combiner [69, 98] essentially combines the output from different algorithms (or runs of the same algorithm)

in such a way as to produce secure (in this case, unsubverted) combined output as long as any one of the underlying outputs is secure. Aviram et al. [13] consider combining (potentially maliciously chosen) keys for post-quantum protocols such as TLS. Bemman, Chen and Jager [26] show how to construct a subversion-resilient KEM, using a variant of a combiner and a subversion resilient randomness generator. Their construction considers Russell et al.'s watchdog from a practical perspective, meaning an offline watchdog that runs in linear time. Another line of work, [65, 16, 57], examined backdoored hash functions, showing how to immunise hash functions against subversion.

Chapter 4

Subverting Deniability

Contents

This chapter provides an introduction to subversion, in preparation for the next chapter where we give details of a specific ASA against receiver algorithms. We consider DPKE as an interesting case study, with the ulterior motive of explaining the intuition behind ASAs. Intuitively, DPKE is a cryptographic primitive that allows the sender of an encrypted plaintext message to later claim that a different faked plaintext was sent. DPKE schemes find their simplest application in the setting of users communicating in the presence of repressive authorities that monitor communications, for example to suppress dissent or otherwise restrict the rights of their citizens. We show that subversion attacks against DPKE schemes present an attractive opportunity for such an adversary. We note that whilst deniable public key

encryption is a widely accepted notion, there are as yet no practical DPKE schemes; we demonstrate the feasibility of ASAs against deniable encryption using a representative scheme as a proof of concept. We also provide a formal model and discuss how to mitigate against ASAs targeting DPKE schemes.

4.1 Introduction

Deniable public-key encryption (DPKE) is a primitive that allows a sender to successfully lie about which plaintext message was originally encrypted. DPKE schemes find their simplest application in the setting of users communicating in the presence of repressive authorities that monitor communications, for example to suppress dissent or otherwise restrict the rights of their citizens.

In particular, suppose that Alice encrypts a plaintext m under some public key, using randomness r, to give ciphertext c which she sends to Bob. At some point in the future – perhaps Bob falls under suspicion of being a dissident, or harbouring subversive beliefs – Alice is coerced to reveal the message she encrypted, together with the randomness she used. DPKE allows Alice to claim that she sent m^*, by providing r^* such that $\mathsf{enc}(m^*, r^*) = \mathsf{enc}(m, r)$. Beyond its immediate use case, deniable encryption finds applications in electronic voting, where deniability allows voters to cast their ballots without coercion and prevents vote-buying, as well as in secure multiparty computation [39].

The adversarial model for deniable encryption assumes strong capabilities for the adversary, in which they are able to coerce individuals to reveal the messages they encrypted; it is therefore reasonable to consider other advanced adversarial capabilities, such as the ability to subvert algorithms. Powerful adversaries, such as state security services run by repressive regimes, have the means to insert unreliability into cryptography via external infrastructure, as we discussed in Chapter 3. Deniable encryption thus provides an interesting case study to build some intuition on how ASAs work.

As we discuss in Section 3.3.2.3, we usually consider the adversarial aim to be exfil-

trating the user's secret key. Indeed, our generic attacks against receivers (presented in Chapter 5) apply to PKE schemes and thus DPKE as a special case and allows an adversary to exfiltrate the receiver's (private) key. This is one avenue to break the deniability of user's messages; an adversary who has learnt the private key effectively undermines the confidentiality of a DPKE scheme and hence the deniability guarantees. Simply put, if an adversary is able to read Alice's messages as they are sent, then Alice is unable to lie about those messages in the future.

In this chapter however, we are concerned with a different approach: we consider subverting the sender with the specific aim of undermining deniability. In particular, our subversion attacks targeting deniability leave the confidentiality of messages intact. We argue in this chapter that subversion attacks against DPKE schemes present an attractive opportunity for an adversary, in particular since it seems that deniable encryption requires large structured ciphertexts, which allow for exfiltration of more than a few bits. An adversary who is able to monitor communications and record ciphertexts may later coerce the participants to reveal their underlying plaintexts; we show that a subverted deniable encryption scheme can embed information into ciphertexts that undermines the sender's ability to lie about which message was sent.

4.1.1 Structure of the Chapter

Having previously covered the standard definitions needed (Chapter 2) and introduced the concept of an ASA (Chapter 3), in this chapter we show that subverting deniability is a well-defined concept. In doing so, we hope to give the reader a greater intuition of ASAs. We first consider, in Section 4.2, subverting deniability in the context of symmetric encryption as an illustrative case study. Section 4.3 discusses deniable public-key encryption schemes, giving a definition and notions of security (Section 4.3.1), as well as a brief survey of the literature and a description of the "Parity Scheme" [42] of Canetti et al.(Section 4.3.2). Section 4.4 introduces notions of subverted deniability, including adversarial goals (Section 4.4.1). As a proof of concept, we show that the Parity Scheme is easily subverted (Section 4.4.2). We indicate approaches to mitigate against subversion of deniable schemes in Section 4.4.3 and conclude in Section 4.4.4.

This chapter consists of an extended version of:

> Marcel Armour and Elizabeth A. Quaglia. "Subverting Deniability". In: *Provable and Practical Security*. Ed. by Chunpeng Ge and Fuchun Guo. Cham: Springer Nature Switzerland, 2022.

4.2 Case Study: Subverting Deniable Symmetric Encryption

In this section we describe an illustrative case study that serves to introduce the concepts of deniability and subversion; in order to highlight the intuition behind our ideas, our discussion proceeds rather informally. We show how a subversion attack against symmetric encryption schemes can undermine the deniability that the scheme provides. Later, in Section 4.3, we develop the idea further and describe subversion attacks against DPKE, where the notions of deniability are more nuanced and require a formal treatment.

Symmetric encryption schemes are intuitively deniable, in the following sense: If Alice and Bob share a secret key, then any ciphertext could have been created by either party. If we consider messages in the direction from Alice to Bob, this means that Bob is unable to present an adversary with a convincing proof that Alice sent a particular message (by revealing a key, message and ciphertext that he claims were sent by Alice). As a result, it is ineffective for an adversary to coerce Bob to reveal Alice's messages. Further, it also means that Bob is unable to convincingly "frame" Alice for messages she didn't send. The inherent deniability provided by symmetric encryption is usually considered in the context of non-repudiation, where it is regarded as a weakness. Non-repudiation can be achieved via digital signatures, an asymmetric primitive that allows a signer to create signatures for messages such that only the signer could have created the signature.

Consider a scenario where a symmetric encryption scheme is used for its deniability property. We show that a subversion adversary who can subvert the scheme's encryption algorithms is able to undermine deniability by introducing a subliminal channel. The subliminal channel can be used to provide a commitment to the mes-

sages, in the form of a digital signature or a Message Authentication Code (MAC). This means that if Bob reveals Alice's messages, her ability to deny that she sent the messages is undermined, as the adversary can check whether the message Alice claims to have sent matches the commitment. Furthermore, this subliminal channel committing ciphertexts to underlying messages is undetectable according to the undetectability notions of Section 3.2.1 – that is, any detector with black-box access to the subverted scheme will be unable to determine whether the scheme is subverted.

We first describe two standard methods to implant a subliminal channel into ciphertexts generated using symmetric encryption: rejection sampling and IV replacement. We then go on to informally describe how to subvert deniability of symmetric encryption schemes using a subliminal channel, which serves as a useful case study for our results in Section 4.3.

Rejection Sampling. Rejection sampling, as discussed in Section 3.2.3, allows a subliminal channel to be implanted in ciphertexts. The idea is that randomness is resampled until the ciphertext encodes the subliminal channel (e.g. in the first bit, or when a hash function is applied to the ciphertext). As noted before, in practice the subliminal channel's bandwidth needs to be small (one or two bits) in order to ensure that the subverted algorithm is not prohibitively slow.

Initialisation Vector (IV) Replacement. IV replacement, following [23], utilises the "randomness surfacing" property of some encryption schemes to directly implant a subliminal message. Randomness surfacing schemes are such that the randomness (usually referred to as an IV in this context) used in encryption can be directly recovered from ciphertexts; a subversion adversary can simply replace the honestly generated IV with their subliminal message. In practice, the adversary will want to hide their message by encrypting so that the replacement IV still looks random, to ensure that the subversion is undetectable. Consider a randomised stateless symmetric encryption scheme $\mathsf{SE} = (\mathsf{SE.gen}, \mathsf{SE.enc}, \mathsf{SE.dec})$, following the notation introduced in Section 2.3.3. We write $c \leftarrow \mathsf{SE.enc}(k, m; IV)$ to highlight the fact that we surface the randomness input IV (for initialisation vector) to the encryption algorithm. Such a scheme is said to surface its IV if there is an efficient algorithm χ such that $\chi(\mathsf{SE.enc}(k, m; IV)) = IV$ for all k, m, IV. The condition says that χ can recover the IV from the ciphertext. A simple example of a scheme that surfaces

its IV is CBC$, namely CBC mode with random IV. Another example is CTR$, counter mode with random starting point.

4.2.1 Subverting Deniability of Symmetric Encryption

We assume that the adversary subverts the encryption algorithm $\mathsf{SE.enc}$ of a symmetric encryption scheme so that the ciphertexts contain a subliminal channel, either using IV replacement or rejection sampling. Rather than using the channel to exfiltrate the user's key, as is the approach in other settings,[1] we let the adversary transmit a commitment to the underlying message in the form of a MAC tag t.

Alice generates ciphertext $c \leftarrow \mathsf{SE.enc}_i(k, m)$ which encrypts a message m under key k, using the subverted encryption algorithm $\mathsf{SE.enc}_i$ so that c encodes the tag t. At a later point in time, the adversary can coerce Bob to reveal ciphertext, key, message c^*, k^*, m^* and can then compare the message m^* to the tag encoded in the ciphertext c^*. We note that for an adversary, knowing whether or not the message that Bob claims was sent is the real message is sufficient to identify when Bob is lying, implying that Alice and Bob exchanged illicit messages.

In more detail, assume that the adversary has subverted the encryption algorithm $\mathsf{SE.enc}$ so that ciphertexts encode ℓ-bits of subliminal information. As per the discussion in Section 3.2, we assume that the subverted algorithm has an embedded subversion key $i \in \mathcal{I}_{\mathsf{S}}$ known only to the adversary. On input a message m, the subverted encryption algorithm $\mathsf{SE.enc}_i$ first calculates an ℓ-bit MAC tag $t_\ell \leftarrow \mathsf{MAC.tag}_\ell(i, m)$.[2] Then, using the subliminal channel, the tag t_ℓ is encoded into a valid ciphertext c. An adversary who is given the ciphertext c and knows the subversion key i can recover the tag t_ℓ and check (by recomputing the tag) that it verifies against the message m^* that Bob claims was encrypted.

[1] While leaking the secret key would allow an adversary to compromise users, we are looking ahead to DPKE where the user's key is public and thus pointless to leak.

[2] We write tag_ℓ to denote the tagging algorithm of a MAC scheme that outputs ℓ-bit tags. Such a scheme can straightforwardly be instantiated from MAC scheme MAC with associated tag space $\mathcal{T}$ by applying a cryptographically secure hash function $H\colon \mathcal{T} \to \{0,1\}^\ell$ to tags $t \in \mathcal{T}$. In order to verify a message, tag pair (m, t_ℓ), test whether $H(\mathsf{MAC.tag}(i, m)) = t_\ell$. If so, return m and otherwise return $\perp$.

4.2.1.1 Success of the Subversion

We first note that the distribution of subverted ciphertexts is indistinguishable from the distribution of unsubverted ciphertexts, assuming that the MAC scheme outputs tags whose distribution is (computationally) indistinguishable from random.[3] In both cases (real or subverted), a distinguisher playing the subversion detection game UDS observes ciphertexts that are indistinguishable from random. This means that a detector with black box access to the subverted encryption algorithm is unable to distinguish $\mathsf{SE.enc}$ from $\mathsf{SE.enc}_i$ with any meaningful probability – that is, the attack is undetectable according to the notion in Section 3.2.1.

Furthermore, the attack potentially allows the adversary to tell whether a particular message corresponds to a ciphertext or not with some (non-negligible) probability. Applying Kerckhoffs' principle, we assume that the communicating parties (Alice and Bob) know that the encryption is subverted, but do not have access to the secret signing key i. In effect, we assume that Alice (and Bob) have black-box access to the subverted encryption algorithm.[4] This means that Alice and Bob have access to an oracle that on input a message m returns a tag $\mathsf{MAC.tag}_\ell(i, m)$. When Bob is coerced by the adversary, in order to be convincing he will need to produce c^*, k^*, m^* such that c^* encodes t_ℓ with $\mathsf{MAC.tag}_\ell(i, m^*, t_\ell) \neq \perp$. So now Bob plays the role of adversary in an unforgeability game UF, as discussed in Section 2.3.4. Informally, if the MAC scheme is secure then Bob's advantage in this task is negligible. In the scenario we have just discussed, we assume that Bob tries to construct a fake message *after* being coerced. An alternative is for Alice to prepare a pair of messages m, m^* in advance so that the $\mathsf{tag}_\ell(m) = \mathsf{tag}_\ell(m^*)$; however, this too should be possible only with negligible probability for a secure MAC scheme.

We thus conclude that the subverted encryption scheme is no longer inherently deniable, and in fact the deniability of the subverted scheme reduces to the security of the MAC scheme that the subverted algorithm runs as a subroutine. This security

[3] This requirement is equivalent to stating that the MAC scheme is a PRF, following the definition at Figure 2.1. We note that while a MAC scheme is not necessarily a PRF, most MAC schemes satisfy this property in practice. For example, Bellare [17] showed that HMAC is a PRF if the underlying hash function is a PRF.

[4] This assumption is a common approach in work on ASAs, e.g.[23, 20]. The embedded subversion key may be obfuscated in code or stored in a trusted execution environment that a user is unable to tamper with. Using techniques from malware [72], this is a plausible outcome for an adversary.

is a function of the length of tags, expressed above as ℓ.

4.2.2 Discussion

IV replacement allows for $|IV|$ bits of information – commonly 128, if AES is the block cipher used – to be encoded into ciphertexts, which would make it unrealistic for Bob to deny messages (successfully evade the subverted deniability by forging a 128 bit tag). This case is less practically relevant, as IV surfacing schemes are not widely used, but allows us to conclude that subverting deniability is a meaningful concept. We note that the rejection sampling method allows only a few bits to be implanted into the subliminal channel, which means that tags are not long enough to be unforgeable by Bob. In particular, short tags increase the probability of collisions which provide a generic method for Bob to construct a forgery. This means that the subversion is unsuccessful from the point of view of an adversary. Of course, this assumes that Alice and Bob are aware of the subversion and actively craft convincing c^*, k^*, m^* such that the encoded tag verifies over m^*. We note that in practice, it may be the case that the subversion goes unnoticed by Alice and Bob – and for an unscrupulous adversary this may be sufficient to undermine deniability in practice.

To conclude, in this section we discussed how the deniability of symmetric encryption schemes can be undermined if the algorithms are subverted. The subversion techniques and deniability notions translate loosely onto deniable public-key encryption, which we discuss in the next section. For DPKE, proposed schemes are commonly IV surfacing, or else allow for a covert channel with a large bandwidth, so that subverting deniability becomes more feasible – and is also highly relevant, because we are then considering a primitive that is designed to provide deniability.

4.3 Deniable Public-Key Encryption

Deniable (public key) encryption allows a sender to lie about the messages that were encrypted.[5] In particular, suppose that a user encrypts message m to obtain c which is sent to the recipient. Later, the user is required to reveal the randomness and message that were used to derive the ciphertext c. Deniable encryption allows the sender to choose a different message m^* and reveal fake randomness r^* which explains c as the encryption of m^*. Notice that this necessarily implies that the scheme can't be perfectly correct as $\mathsf{dec}(\mathsf{enc}(m^*, r^*)) = m$. This counter-intuitive observation is resolved by noticing that for a given message m, there are "sparse trigger" values r_i such that encrypting m with an r_i results in an incorrect ciphertext. Deniable public-key encryption schemes rely on the fact that finding such r_i should be easy with some trapdoor knowledge, and hard otherwise.

In this chapter we focus on non-interactive sender deniable public-key encryption, as introduced by Canetti et al. (CDNO) [42], who showed that a sender-deniable scheme can be used to construct receiver-deniable (and thus bi-deniable) schemes. Other notions of deniability include weak (or "multi-distributional") deniability in which a sender uses an alternative ("fake") encryption algorithm to encrypt deniable messages – when coerced, they claim to have run the regular algorithm. Canetti et al.describe such a scheme in [42]; later O'Neill et al.[95] proposed a non-interactive encryption scheme with negligible deniability simulatable encryption. Another line of work uses Indistinguishability Obfuscation (iO) to achieve deniable encryption: Sahai and Water's sender deniable scheme [108] and Canetti Park and Poburinnaya's bi-deniable interactive scheme [40]. However, the current state of iO means that these result serve more as a theoretical feasibility result. De Caro, Iovino and O'Neill [50] studied the notion of receiver deniable functional encryption, but instantiating their constructions required fully fledged functional encryption, which in turn is known to imply iO.

To date, no practical deniable schemes has been proposed. Either deniability is not practically achievable, as in the case of the CDNO Parity Scheme whose ciphertexts

[5] We are considering *sender-deniable* encryption here, in which the sender is able to deny messages that they encrypted. One can similarly consider *receiver-deniable* encryption, where a receiver is able to deny a message that they received. Bi-deniable encryption combines both directions.

grow inversely proportional to the deniability probability, or else the construction requires strong assumptions such as iO or functional encryption. Recent work by Agrawal et al. [2] is promising in this regard, as their construction for deniable fully homomorphic encryption (FHE) provides compact ciphertexts and is based on the security of Learning with Errors. Nevertheless, their construction requires a running time that is inversely proportional to detection probability. In the absence of practical schemes, we demonstrate the feasibility of our ASA targeting deniable encryption schemes (Section 4.4) by focussing on the illustrative case study of the CDNO "Parity Scheme". Our technique applies generically to any deniable PKE scheme.

The remainder of this section sets the scene for our attack in Section 4.4; we first recap the formal definition of a deniable PKE scheme in Section 4.3.1 before describing the CDNO Parity Scheme in Section 4.3.2.

4.3.1 Definition of Deniable PKE Schemes

A deniable PKE scheme $\mathsf{DE} = (\mathsf{DE.gen}, \mathsf{DE.enc}, \mathsf{DE.dec}, \mathsf{DE.Fake})$ consists of a tuple of algorithms together with key spaces $\mathcal{K}_\mathsf{S}, \mathcal{K}_\mathsf{R}$, randomness space $\mathcal{R}$, a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$.

- The key-generation algorithm $\mathsf{DE.gen}$ returns a pair $(pk, sk) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$ consisting of a public key and a private key.
- The encryption algorithm $\mathsf{DE.enc}$ takes a public key pk, randomness $r \in \mathcal{R}$ and a message $m \in \mathcal{M}$ to produce a ciphertext $c \in \mathcal{C}$.
- The decryption algorithm $\mathsf{DE.dec}$ takes a private key sk and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection.
- Finally, the faking algorithm $\mathsf{DE.Fake}$ takes a public key pk, a pair of messages and randomness m, r as well as a fake message m^*, and outputs faking randomness $r^* \in \mathcal{R}$.

Game $\text{INDEXP}^b(\mathcal{A})$
00 $(dpk, dsk) \leftarrow \mathsf{DE.gen}$
01 $b' \leftarrow \mathcal{A}^{\text{Exp}}(dpk)$
02 stop with b'

Oracle $\text{Exp}(m, m^*)$
03 $r \leftarrow_{\$} \mathcal{R}$
04 $r^* \leftarrow \mathsf{DE.Fake}(dpk, m, r, m^*)$
05 if $b = 0$:
06 return $(m^*, r, \mathsf{DE.enc}(dpk, m^*; r))$
07 else:
08 return $(m^*, r^*, \mathsf{DE.enc}(dpk, m; r))$

Game $\text{subINDEXP}^b(\mathcal{A})$
00 $(dpk, dsk) \leftarrow \mathsf{DE.gen}$; $i \in \mathcal{I}_{\mathsf{S}}$
01 $b' \leftarrow \mathcal{A}^{\text{Exp}}(dpk)$
02 stop with b'

Oracle $\text{Exp}(m, m^*)$
03 $r \leftarrow_{\$} \mathcal{R}$
04 $r^* \leftarrow \mathsf{DE.Fake}(dpk, m, r, m^*)$
05 if $b = 0$:
06 return $(m^*, r, \mathsf{DE.enc}_i(dpk, m^*; r))$
07 else:
08 return $(m^*, r^*, \mathsf{DE.enc}_i(dpk, m; r))$

Figure 4.1: Games modelling the deniability (indistinguishability of explanation) of a deniable PKE scheme (left) and a subverted deniable PKE scheme (right).

A scheme DE is correct and secure if the key generation, encryption and decryption algorithms considered as a PKE scheme $(\mathsf{DE.gen}, \mathsf{DE.enc}, \mathsf{DE.dec})$ satisfy the standard notions of correctness and IND-CPA security properties of public-key encryption, as in Section 2.3.6. We formalise the deniability of the scheme via the game INDEXP in Figure 4.1, using the standard definition from the literature [42]. Essentially, the INDEXP game is an indistinguishability game in which a distinguisher must choose between two cases: INDEXP^0 represents the adversary's view of an honest encryption of m^*; INDEXP^1 represents the adversary's view when the sender lies about the underlying plaintext. The corresponding advantage is, for any distinguisher $\mathcal{A}$, given by

$$\text{Adv}^{\text{indexp}}_{\mathsf{DE}}(\mathcal{A}) := \left|\Pr[\text{INDEXP}^0(\mathcal{A})] - \Pr[\text{INDEXP}^1(\mathcal{A})]\right|$$

and say that scheme DE is deniable if $\text{Adv}^{\text{indexp}}_{\mathsf{DE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

Note that a scheme cannot simultaneously satisfy perfect correctness and deniability, so negligible decryption error in correctness is inherent.

4.3.2 Parity Scheme of Canetti, Dwork, Naor, Ostrovsky

Here we describe the sender deniable "Parity Scheme" of Canetti et al.[42]. Informally, ciphertexts consist of a tuple of elements where each element is either chosen randomly from a set $T = \{0,1\}^\tau$ or a so-called "translucent set" S_τ, where S satisfies the following properties:

- $S_\tau \subset T$ and $|S_\tau| \leq 2^{\tau - k}$, for sufficiently large k.
- It is easy to generate random elements $x \in S_\tau$.
- Given $x \in T$ and trapdoor information d_τ, it is easy to check whether $x \in S_\tau$.
- Without d_τ it is computationally infeasible to decide whether $x \in S_\tau$.

For specificity, we consider the construction of translucent sets given in [42] based on a trapdoor permutation $f\colon \{0,1\}^s \to \{0,1\}^s$ and its hard-core predicate $B\colon \{0,1\}^s \to \{0,1\}$ (see Appendix A.1 for the definition of trapdoor permutations and hard-core predicates). Let $\tau = s + k$. Represent each $x \in T$ as $x = x_0 \parallel b_1 \parallel b_2 \parallel \ldots \parallel b_k$, where $x_0 \in \{0,1\}^s$ is followed by k bits. Then the translucent set is defined as:

$$S = \left\{ x = x_0 \parallel b_1 \parallel b_2 \parallel \ldots \parallel b_k \in \{0,1\}^{s+k} | (\forall i \leq k) B(f^{-i}(x_0)) = b_i \right\}.$$

The trapdoor information d_τ plays the role of a private key.

We give a description in pseudo-code of the encryption algorithm PS.enc in Figure 4.2. On input a bit value b, the encryption algorithm first chooses a random number $0 < \ell \leq n$ with parity b in Line 00. Next, ℓ elements in S are generated in Lines 02 to 06. Lastly, before outputting the ciphertext in Line 09, $n - \ell$ elements in T are generated in Lines 07 and 08. We refer the reader to [42, 41] for full details of the scheme, including decryption and faking algorithms as well as proofs of the security and deniability of the scheme. In particular, it is shown that the Parity Scheme is a $4/n$-sender deniable encryption scheme, which means that the probability of a successful attack of a coercer vanishes linearly in the security parameter n.

Proc $\mathsf{PS.enc}(pk, m)$

00 while $\ell \mod 2 \neq b$:
01 $\quad \ell \leftarrow_\$ [0\,..\,n+1]$
02 for $i \in [0\,..\,\ell]$:
03 $\quad x_0^{(i)} \leftarrow_\$ \{0,1\}^s$
04 $\quad$ for $j \in [0\,..\,k]$:
05 $\quad\quad b_j^{(i)} \leftarrow B(f^{-j}(x_0^{(i)}))$
06 $\quad x^{(i)} \leftarrow x_0^{(i)} \parallel b_0^{(i)} \parallel \ldots \parallel b_k^{(i)}$
07 for $i \in [\ell\,..\,n+1]$:
08 $\quad x^{(i)} \leftarrow_\$ \{0,1\}^t$
09 return $c = (x^{(0)}, x^{(1)}, \ldots, x^{(n)})$

Proc $\mathsf{PS.enc}_i(pk, m)$

00 $t \leftarrow \mathsf{MAC.tag}_{(n+1)s}(sk_i, m)$
01 while $\ell \mod 2 \neq b$:
02 $\quad \ell \leftarrow_\$ [0\,..\,n+1]$
03 for $i \in [0\,..\,\ell]$:
04 $\quad x_0^{(i)} \leftarrow_\$ t[is : (i+1)s]$
05 $\quad$ for $j \in [0\,..\,k]$:
06 $\quad\quad b_j^{(i)} \leftarrow B(f^{-j}(x_0^{(i)}))$.
07 $\quad x^{(i)} \leftarrow x_0^{(i)} \parallel b_0^{(i)} \parallel \ldots \parallel b_k^{(i)}$
08 for $i \in [\ell\,..\,n+1]$:
09 $\quad x_0^{(i)} \leftarrow_\$ t[is : (i+1)s]$
10 $\quad x_1^{(i)} \leftarrow_\$ \{0,1\}^k$
11 $\quad x^{(i)} \leftarrow x_0^{(i)} \parallel x_1^{(i)}$
12 return $c = (x^{(0)}, x^{(1)}, \ldots, x^{(n)})$

Figure 4.2: **Left:** CDNO Parity Scheme encryption algorithm $\mathsf{PS.enc}$. **Right:** Subverted encryption algorithm $\mathsf{PS.enc}_i$.

4.4 Subverting deniable PKE

A first approach to subvert a deniable PKE scheme is to target the faking algorithm. A subverted faking algorithm $\mathsf{DE.Fake}_i(pk, m, r, m^*)$ could output subverted r^* which alerts the adversary to the fact that m^*, r^* are fake; for example, if r^* commits to the real message m. However, this fake randomness r^* still needs to be convincing from the point of view of the deniability of the scheme – the scheme's security properties should be maintained by the subversion, otherwise a detector playing the UDS game, in which they are tasked with differentiating between a subverted algorithm and a reference version (see Section 3.2.1), will be able to tell that the algorithm is subverted. In particular, r^* should satisfy $\mathsf{DE.enc}(pk, m^*, r^*) = c$. However, for a deniable PKE scheme there is no reason why this should hold for an arbitrary value of r^*. This approach does not seem to be workable without adding considerable structure to the subverted scheme that means it would be easily detected.[6]

A second approach turns out to be a feasible attack route: subverting the scheme

[6] As an interesting aside, the approach for iO deniability schemes is to hide an encoding of the faked ciphertext within randomness; the encryption algorithm first checks whether the randomness encodes a ciphertext c and if so outputs c; if not, it proceeds to encrypt the message. The security follows from the fact that iO obfuscates the inner working of the algorithm so that it appears as a black box. This results in large, structured randomness inputs which would seem to facilitate subversion against iO based deniable schemes.

to let randomness commit to the message. This way, when Alice is coerced by the adversary to reveal her message and randomness, the adversary is able to test whether this is the case. If Alice is trying to lie about her message and randomness, she will be unable to provide the adversary with randomness that commits to the message. This means that an adversary playing the INDEXP game (Figure 4.1) is able to distinguish between real and fake cases and win the game with non-negligible probability. We demonstrate this approach in the case of the CDNO Parity Scheme (Section 4.3.2) as an illustrative example. As the scheme surfaces its randomness, the ciphertexts reveal the underlying randomness making it easy to embed a subliminal channel. For a scheme that is not randomness surfacing, Alice has to reveal the randomness (either real or fake) when coerced; thus the issue remains that if randomness is subverted, then deniability can be undermined.

4.4.1 Subverting Deniable PKE

We note that a deniable PKE scheme satisfies the generic syntax introduced above in Section 3.2, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{DE.gen}$, sender algorithms $(\Pi.\mathsf{S}_0, \Pi.\mathsf{S}_1) = (\mathsf{DE.enc}, \mathsf{DE.Fake})$ and receiver algorithm $\Pi.\mathsf{R} = \mathsf{DE.dec}$. We may thus apply the generic notions of subversion and undetectability introduced in Sections 3.2.1 and 3.2.2. We obtain the game subINDEXP given in Figure 4.1 (right), modelling the adversary's ability to compromise the deniability property of a subverted scheme.

4.4.2 Subverting CDNO Parity Scheme

In Figure 4.2 (right), we give a subversion attack against the CDNO Parity Scheme, described in Section 4.3.2. At a high level, we subvert randomness so that it encodes a commitment to the original message in the form of a MAC tag. Our technique is analogous to the IV replacement attack described in Section 4.2.

On input a public key and bit message m, the subverted algorithm first generates a tag of length $(n+1)s$ bits over the plaintext m using its secret subversion key i

(Line 00). This tag will be embedded into the "random" elements that constitute the ciphertext in chunks. When generating elements from S in Lines 03 to 07, the subverted encryption algorithm uses chunks of the tag (Line 04) rather than sampling a random element. Lines 05 to 07 ensures that we are generating elements in S. When generating elements from T in Lines 08 to 11, the subverted encryption algorithm sets the first s bits to be a chunk of the tag, then samples the remaining bits at random. Finally, the subverted algorithm outputs the ciphertext in Line 09.

We note that ciphertexts are well-formed and the subverted ciphertext c encrypts the message m as intended. To see this, the recipient with knowledge of the trapdoor d_τ will be able to identify which elements are in S.

4.4.2.1 Success of the Subversion

We first note that the distribution of subverted ciphertexts is indistinguishable from the distribution of unsubverted ciphertexts, assuming that the message authentication scheme outputs tags whose distribution is (computationally) indistinguishable from random.[7] In both cases (real or subverted), a distinguisher without knowledge of the trapdoor d_τ playing the subversion detection game UDS observes ciphertexts that are indistinguishable from random. This means that a detector with black box access to the subverted encryption algorithm is unable to distinguish $\mathsf{PS.enc}$ from $\mathsf{PS.enc}_i$ with any meaningful probability – that is, the attack is undetectable according to the notion in Section 3.2.1.

An adversary who is given the ciphertext c (or indeed the randomness r) and knows the subversion key i can recover the tag t and check (using the subversion key i) that it verifies against the message m^* that Bob claims was encrypted. More formally, a subversion adversary playing the subINDEXP game from Figure 4.1 (right) is able to distinguish between $(m^*, r, \mathsf{DE.enc}_i(dpk, m^*; r))$ and $(m^*, r^*, \mathsf{DE.enc}_i(dpk, m; r))$ by recovering t from the randomness and testing whether $\mathsf{MAC.vfy}(i, m^*, t)$ verifies. We note that as the randomness encodes the tag, recovering the tag is independent of the ciphertext and this method applies generically even to encryption schemes that do not surface their randomness.

[7] See the discussion at Footnote 4.

Lastly, we note that even if Alice is aware that her encryption algorithm is subverted, as long as she does not have access to the secret subversion key i she is unable to forge a tag which would allow her to claim she sent a fake message. We thus conclude that the subverted scheme is no longer deniable, and in fact the deniability of the subverted scheme reduces to the security of the MAC scheme that the subverted algorithm runs as a subroutine. Thus security is a function of the length of the tag – in the example of the Parity Scheme, this is sufficiently large to give a meaningful success probability to the adversary.

4.4.3 Subversion Resilient Deniable PKE Schemes

Following the discussion in Section 3.4.4, we may apply any of the standard approaches (reverse firewalls, self-guarding protocols or watchdogs) to sanitise the scheme and prevent subliminal channels in ciphertexts. One way to achieve this generically is to simply compose the deniable PKE scheme DE with a subversion resilient PKE scheme PKE_{SR} so that the output of $\mathsf{DE.enc}$ is encrypted under PKE_{SR} before being sent to the receiver. Particular deniable PKE constructions may allow a more efficient approach; for example, reverse firewalls apply directly to deniable FHE.

However, subliminal channels is not sufficient to protect against subversion as the adversary is still able to coerce the sender to reveal randomness at some point in the future. To mitigate against this, deniable PKE constructions should explicitly separate randomness generation from encryption so that $\mathsf{DE.enc}$ is deterministic, following the approach of [19]. This does not necessarily result in ciphertexts that are free of a subliminal channel, however – whilst it protects against the rejection sampling method, other non-generic methods, such as [118], could potentially result in subverted ciphertexts. Thus, a combination of mitigating measures are appropriate.

4.4.4 Conclusion

In this chapter, we have explored the notion of ASAs and we have seen that the security provided by a scheme (such as deniability) no longer holds if the adversary is given control over the implementation of the algorithms used. Our treatment of subversion attacks in this chapter is quite informal, with the view of building intuition. In the next chapter, we consider a more concrete ASA.

In this chapter, we have also taken a look at deniable encryption, one particular primitive providing deniability whose definition is widely agreed upon in the literature and for which the applications are clear (including in e-voting, multi-party computation and to protect against coercion). The threat model for deniable encryption usually considers an adversary who is willing to coerce users; in this chapter we extend the model to consider adversaries who also undermine deniability by using subversion attacks. This seems a reasonable additional assumption to make of an adversary who is willing to coerce users. We hope that showcasing how to subvert deniability can help to enable a better understanding of what deniable communication should provide.

Deniable communication more generally is a subtle concept and it is unclear what it should mean "in the real world". Intuitively, the notion is clear: deniability should allow Alice to plausibly claim that she is not a participant in a particular communication [66]. However, the adversarial model and evaluation of real world protocols claiming deniability is not agreed upon; Celi and Symeonidis [43] give an overview of the current state of play and a discussion of open problems.

Chapter 5

Concrete Subversion Attacks via Acceptance vs. Rejection

Contents

This chapter discusses our subversion attacks targeting the receiver. Concretely, we alter the behaviour of the receiver's algorithm to leak information through (artificially

induced) decryption error events – the subverted algorithm either rejects (particular) valid ciphertexts or accepts (particular) bogus ciphertexts. These particular ciphertexts can be thought of as "trigger" values. An adversary observing the receiver who is able to determine whether a ciphertext has been accepted or rejected learns some information; this subliminal channel can be used to exfiltrate the user's key.

5.1 Introduction

In this chapter, we describe how an ASA against the receiver can be used to exfiltrate the (receiver's) key, which represents the most effective attack from the point of view of an attacker (and the most devastating from the point of view of the users). The framework for ASAs against receivers was introduced in Chapter 3, where we showed that the syntax is met by symmetric encryption, PKE and MACs.

Concretely, we alter the behaviour of the receiver's algorithm to leak information through (artificially induced) decryption error events – the subverted algorithm either rejects (particular) valid ciphertexts or accepts (particular) bogus ciphertexts. These particular ciphertexts can be thought of as "trigger" values. An adversary observing the receiver who is able to determine whether a ciphertext has been accepted or rejected learns some information; this subliminal channel can be used to exfiltrate the user's key. The assumption that a surveillance adversary is able to observe whether a receiver's algorithm implementation accepts or rejects a ciphertext is a mild one in many practical scenarios; for example, a decryption error may result in a network packet being dropped and automatically retransmitted. A subverted algorithm could, furthermore, go beyond this by e.g.influencing timing information in future messages sent to the network. We conclude that this attack represents an attractive and easy to implement opportunity for a mass surveillance adversary.

5.1.1 Structure of the Chapter

We first discuss adversarial goals. In Section 3.2.2 we gave the most general adversarial goal, namely to leak some arbitrary information. For this chapter, we consider the aim of an adversary to undermine confidentiality of encryption communication (or authenticity in the case of MACs). As such, the most efficient way to achieve this is usually to leak the secret key; in some cases, the scheme to be subverted exhibits some additional structure that allows security to be broken more efficiently than leaking the secret key. We discuss two such scenarios in Sections 5.2.2 and 5.2.3.

We go on to describe our concrete subversion attacks, in two variants: a passive attack (Section 5.3.2) and an active attack (Section 5.3.2) quantifying the success probability of an adversary and the detectability of the subversion.

Lastly, in Section 5.4 we describe the results of an experiment where we ran a proof-of-concept implementation of our attack against AES-GCM. Our experimental results conform to the theoretical results of Section 5.3.

We end the chapter by discussing mitigation of subversion attacks in Section 5.5 before making some concluding remarks in Section 5.6.

The basis for this chapter is provided by:

> Marcel Armour and Bertram Poettering. "Algorithm Substitution Attacks against Receivers". In: *International Journal of Information Security*, June 2022.

Additionally, the proof-of-concept implementation (Section 5.4) is taken from:

> Marcel Armour and Bertram Poettering. "Subverting Decryption in

AEAD". in: *17th IMA International Conference on Cryptography and Coding*. Ed. by Martin Albrecht. Vol. 11929. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2019

5.2 Adversarial Goals

In Section 3.2.2 we gave the most general adversarial goal, namely to leak some arbitrary information. This arbitrary information might for example be the user's encryption key for another application, or the internal state of its random number generator. But as Bellare et al. [20] argue, this is not an attractive goal for a mass surveillance adversary, who would rather break confidentiality completely and recover plaintext messages. For this chapter, we thus consider the special case where the information leaked leads to a complete break of security (confidentiality for encryption schemes and authenticity for MAC schemes).

5.2.1 Subversion Leading to Key Recovery

We observed above that if any of the components $\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R}$ of a cryptographic scheme Π is undetectably subverted, with uniformly chosen indices $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}}$ that remain unknown to the participants, then all security guarantees are preserved from the original scheme. This may be different if (any of) $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}}$ are known to an attacking party, and indeed we assume that mass-surveillance attackers leverage such knowledge to conduct attacks. For any cryptographic scheme, the most devastating attack goal for an attacker is key recovery (KR): Users generate keys using their key generation algorithm $(k_{\mathsf{S}}, k_{\mathsf{R}}) \leftarrow_{\$} \Pi.\mathsf{gen}_{i_{\mathsf{gen}}}$.[1] Generated secret keys are kept hidden,

[1] To preserve generality, our syntax suggests that key generation is subverted, however this need not be the case. Simply set $\Pi.\mathsf{gen}_{i_{\mathsf{gen}}} := \Pi.\mathsf{gen}$ for all $i_{\mathsf{gen}} \in \mathcal{I}_{\mathsf{gen}}$. This applies similarly to $\Pi.\mathsf{S}$ and $\Pi.\mathsf{R}$.

and the adversary aims at recovering these keys through the subversion. Note that in the symmetric case, $k_{\mathsf{S}} = k_{\mathsf{R}}$, whereas in the asymmetric case the receiver's key k_{R} represents the private key. In either case, the value k_{R} is the target of a KR adversary.

We formalise this attack goal in two versions. The KRP game in Figure 5.1 (centre) assumes a passive attack in which the adversary cannot manipulate inputs or outputs (typically representing messages or ciphertexts) to the sender or receiver, and the KRA game in Figure 5.1 (right) assumes an active attack in which the adversary can inject and test arbitrary receiver inputs (which potentially correspond to sender outputs). In both cases, with the aim of closely modelling real-world settings, we restrict the adversary's influence on the sender inputs x by assuming a stateful "message sampler" algorithm MS (reflecting the fact that, in the contexts we consider, inputs to $\Pi.\mathsf{S}$ typically represent messages) that produces the inputs to $\Pi.\mathsf{S}$ used throughout the game. The syntax of this message sampler is

$$\Sigma \times A \to \mathrm{MS} \to \Sigma \times \mathcal{X} \times B, \quad (\sigma, \alpha) \mapsto \mathrm{MS}(\sigma, \alpha) = (\sigma', x, \beta),$$

where $\sigma, \sigma' \in \Sigma$ are old and updated state, input $\alpha \in A$ models the influence that the adversary may have on message generation, and output $\beta \in B$ models side-channel outputs. In Figure 5.1 we write $\diamond$ for the initial state. Note that while we formalise the inputs α and the outputs β for generality (so that our models cover most real-world applications), our subversion attacks are independent of them.[2] For any message sampler MS and adversary $\mathcal{A}$ we define the advantages

$$\mathrm{Adv}^{\mathrm{krp}}_{\Pi,\mathrm{MS}}(\mathcal{A}) := \Pr[\mathrm{KRP}(\mathcal{A})] \text{ and } \mathrm{Adv}^{\mathrm{kra}}_{\Pi,\mathrm{MS}}(\mathcal{A}) := \Pr[\mathrm{KRA}(\mathcal{A})].$$

We say that subversion family $\mathcal{G}en, \mathcal{S}, \mathcal{R}$ is key recovering for passive attackers if for all practical MS there exists a realistic adversary $\mathcal{A}$ such that $\mathrm{Adv}^{\mathrm{krp}}_{\mathrm{MS}(\mathcal{A})}$ reaches

[2] ... meaning that the reader may safely choose to ignore them.

a considerable value (e. g., 0.1).[3] The key recovery notion for active attackers is analogous.

5.2.1.1 Discussion

We note that the adversary need not necessarily exfiltrate each individual bit of the user's key,[4] in order to successfully recover it – this is implicit in our definitions of key recovery. To formalise this, we let the "leakage key" $k_\ell \in \{0,1\}^\lambda$ be a string such that knowledge of k_ℓ is sufficient for an adversary to break the security of the primitive. At worst, from the perspective of the adversary, the leakage key may simply be the bit representation of the user's key. We note that in practice a leakage key consisting of *most* of the user's key is sufficient for an adversary to recover the full key using brute force; the exact number of bits to be brute forced would depend on the context and would involve a trade-off for the adversary. Nevertheless, the notion is intuitively clear. Note that in the key recovery games in Figure 5.1 the "leakage key" is implicit: the game ends with the adversary returning (their approximation to) the user's secret key. This may well have been recovered by reconstructing the key from the sufficient information encoded in a "leakage key" during the experiment.

Furthermore, in some contexts there may be some redundancy or structure that allows for a shorter leakage key. As an example, one may consider DES keys as being 64-bit strings with 8 bits of redundancy, so that an effective leakage key would be of size 56 bits. As another example, the private key in RSA encryption is knowledge of the factorisation of the public modulus $N = pq$. Supposing that the modulus N can be represented using $n = \lfloor \log N \rfloor$-bits, one may consider RSA private keys as being $n/2 = \lfloor \log p \rfloor$-bit strings. However, knowledge of around half the bits of p is

[3] Our informal notions ("realistic" and "practical") are easily reformulated in terms of probabilistic polynomial-time (PPT) algorithms for readers who prefer a treatment in the asymptotic framework. Given that asymptotic notions don't reflect practice particularly well, we prefer to use the informal terms.

[4] or a bit representation thereof, if it is not a bit string

sufficient to be able to factorise N using Coppersmith's partial information attack [46], so that an effective leakage key might have length $\lambda = \lfloor \log p \rfloor / 2$.[5]

A different approach might be to leak, for example, the seed of a pseudo-random number generator. We discuss breaking security without extracting the full key further in Section 5.2.3.

Game KRP($\mathcal{A}$)	**Game** KRA($\mathcal{A}$)
00 $C \leftarrow \emptyset$	00 ~~$C \leftarrow \emptyset$~~
01 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$	01 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
02 $(k_{\mathsf{S}}, k_{\mathsf{R}}) \leftarrow_{\$} \Pi.\mathsf{gen}_{i_{\mathsf{gen}}}$; $\sigma \leftarrow \diamond$	02 $(k_{\mathsf{S}}, k_{\mathsf{R}}) \leftarrow_{\$} \Pi.\mathsf{gen}_{i_{\mathsf{gen}}}$; $\sigma \leftarrow \diamond$
03 $k' \leftarrow \mathcal{A}^{\mathrm{Send},\mathrm{Recv}}(i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}})$	03 $k' \leftarrow \mathcal{A}^{\mathrm{Send},\mathrm{Recv}}(i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}})$
04 stop with $[k' = k_{\mathsf{R}}]$	04 stop with $[k' = k_{\mathsf{R}}]$
Oracle Send(α)	**Oracle** Send(α)
05 $(\sigma, x, \beta) \leftarrow \mathrm{MS}(\sigma, \alpha)$	05 $(\sigma, x, \beta) \leftarrow \mathrm{MS}(\sigma, \alpha)$
06 $y \leftarrow \Pi.\mathsf{S}_{i_{\mathsf{S}}}(k_{\mathsf{S}}, x)$	06 $y \leftarrow \Pi.\mathsf{S}_{i_{\mathsf{S}}}(k_{\mathsf{S}}, x)$
07 $C \xleftarrow{\cup} \{y\}$	07 ~~$C \xleftarrow{\cup} \{y\}$~~
08 return (y, β)	08 return (y, β)
Oracle Recv(y)	**Oracle** Recv(y)
09 require $y \in C$	09 ~~require $y \in C$~~
10 $x \leftarrow \Pi.\mathsf{R}_{i_{\mathsf{R}}}(k_{\mathsf{R}}, y)$	10 $x \leftarrow \Pi.\mathsf{R}_{i_{\mathsf{R}}}(k_{\mathsf{R}}, y)$
11 return x	11 return x

Figure 5.1: Games KRP and KRA modelling key recoverability for passive and active attackers, respectively. Note that the adversary's aim is to recover the receiver's key k_{R}, as in both symmetric and asymmetric settings this value is secret.

5.2.2 Hybrid Subversion

Previous work on subversion has looked at either subverted key generation[6] or subverted encryption/ decryption, but not considered the case where these are subverted in tandem. For key generation, this has meant that the subverted algorithm needs to leak the whole key in a single operation. This setting was studied by Young and Yung [120] under the name "kleptography", and they showed how it is possible to

[5] We note that in practice the security for an RSA modulus of size n is far less than $n/2$ bits; for example, an RSA modulus of size 1024 is believed to have security at most 80 bits [15], corresponding to the computational effort required to factorise an RSA modulus.

[6] and potentially, the associated public parameters if those form part of the formalisation used.

subvert key generation such that the adversary is able to recover the private key sk from the public key pk (together with any public parameters and knowledge of secret trapdoor information). They show how such attacks against key generation could look in the case of RSA and ElGamal cryptosystems. Such subversion imposes a large cost on the subverter: requiring that all key bits are leakable in one operation means that the subverted keys are given some structure (e. g., the public key is the encryption of the secret key under the attacker's key). This overhead would likely lead to detection in a real world setting (using either timing information, code review or hardware inspection). Considering the subversion of key generation and sender/receiver algorithms in tandem, it is possible to reduce this overhead.

Generically, this tandem subversion can be achieved by subverting key generation to produce weaker keys and combining this with a subverted sender and/or receiver that provides a subliminal channel. Consider a subverted key generation algorithm $\Pi.\mathsf{gen}_i$ that outputs (receiver keys) in some reduced key set $\widetilde{\mathcal{K}}_{\mathsf{R}} \subset \mathcal{K}_{\mathsf{R}}$. The smaller this subverted key set $\widetilde{\mathcal{K}}_{\mathsf{R}}$, the less information needs to be leaked via the subliminal channel. One method to implement such a subverted key generation algorithm is to use the rejection sampling method described in Section 3.4.1, so that $\Pi.\mathsf{gen}_i$ runs the unsubverted algorithm $\Pi.\mathsf{gen}$ as a subroutine and resamples until keys are in $\widetilde{\mathcal{K}}_{\mathsf{R}}$. There are certainly more targeted attacks that take into account the specific structure of keys being generated – and that may leverage more specific attacks than the generic weakening of keys.

5.2.3 Breaking Security without Extracting the Full Key

The KRA and KRP notions introduced in Section 5.2.1 assume that key recovery is the ultimate goal in subversion. This suggests that longer keys make a scheme more resilient, an approach explored in big key cryptography [21]. In practice, it may be

more efficient to exploit non-generic features of a particular scheme to minimise the information to be leaked. In this section, we will consider AEAD schemes as an illustrative example.

As we detail, many current AEAD schemes have inner building blocks that maintain their own secret values, and scaling up key sizes does not automatically also increase the sizes of these internal values. We note that proposed ASAs against AEAD schemes (including our attacks presented in Section 5.3) can easily be adapted to leak this internal information instead of the key. As the recovery of such values might not always directly lead to full message recovery, the assessment of whether the resulting overall attack is more or less effective than our generic attacks has to be made on a per scheme basis. We exemplify this on the basis of two of the currently best-performing AES-based AEAD schemes: GCM [60] and OCB3 [88]. In both cases, the size of the crucial internal value and the block size of the cipher have to coincide and the latter value is fixed to 128 bits for AES (independently of key size).

AES-GCM. We consider the following abstraction of GCM. The AEAD key k is used directly to create an instance E_k of the AES blockcipher. To encrypt a message m with respect to associated data d and nonce n, E_k is operated in counter mode, giving a pad $E_k(n+1) \parallel E_k(n+2) \parallel \ldots$, where a specific nonce encoding ensures there are no collisions between counter values of different encryption operations. The first part c_1 of the ciphertext $c = c_1c_2$ is obtained by XORing the pad into the message, and finally the authentication tag c_2 is derived by computing $c_2 \leftarrow E_k(n) + H_{hk}(d, c_1)$. Here H_{hk} is an instance of a universal hash function H indexed (that is, keyed) with the 128-bit value $hk = E_k(0^{128})$. Concretely, $H_{hk}(d, c_1) = \sum_{i=1}^{l} v_i h^{l-i+1}$, where coefficients $v_1, \ldots, v_l$ are such that a prefix $v_1 \ldots v_j$ is a length-padded copy of the associated data d, the middle part $v_{j+1} \ldots v_{l-1}$ is a length-padded copy of ciphertext component c_1, and the last item v_l

is an encoding of the lengths of d and c_1. The addition and multiplication operations deployed in this computation are those of a specific representation of the Galois field $\mathrm{GF}(2^{128})$.

In executing a practical ASA against AES-GCM, it might suffice to leak the value hk (which has length 128-bits independently of the AES key length, and furthermore stays invariant across encryption operations). The insight is that if the key of a universal hash function is known, then it becomes trivial to compute collisions. Concretely, assume the adversary is provided with the AES-GCM encryption $c = c_1c_2 = \mathsf{enc}(k, n, d, m)$ for unknown k, m but chosen d, n. Then by the above we have $c_2 = R + \sum_{i=1}^{j} v_i h^{l-i+1}$ where the coefficients $v_1 \dots v_j$ are an encoding of d and R is some residue. If, having been successfully leaked by the ASA, the internal value h is known, by solving a linear equation it is easy to find an associated data string $d' \neq d$, $|d'| = |d|$, such that for its encoding $v'_1 \dots v'_j$ we have $\sum_{i=1}^{j} v'_i h^{l-i+1} = \sum_{i=1}^{j} v_i h^{l-i+1}$. Overall this means that we have found $d' \neq d$ such that $\mathsf{enc}(k, n, d', m) = c = \mathsf{enc}(k, n, d, m)$. In a CCA attack the adversary can thus query for the decryption of c with associated data d' and nonce n, and thus fully recover the target message m. We finally note that this attack can be directly generalised to one where also the c_1 and c_2 components are modified, resulting in the decryption of a message $m' \neq m$ for which the XOR difference between m and m' is controlled by the adversary.

OCB3. Multiple quite different versions of the OCB encryption scheme exist [78], but a common property is that the associated data input is incorporated via "ciphertext translation" [102]. To encrypt a message m under key k with associated data d and nonce n, in a first step the message m is encrypted with a pure AE scheme[7]) to an intermediate ciphertext $c^* \leftarrow \mathsf{enc}^*(k, n, m)$. Then to obtain the final ciphertext c, a pseudo-random function value $F_k(d)$ of the associated data string is XORed

[7] Assuming the above notation for AEAD schemes, we give a similar syntax to AE schemes: an AE scheme encrypts a message m under key k with nonce n to produce a ciphertext denoted $\mathsf{enc}^*(k, n, m)$.

into the trailing bits of c^*. Concretely, in OCB3 we have $F_k(d) = \sum_{i=1}^{l} E(v_i + C_i)$ where all addition operations are XOR combinations of 128 bit values, $E_k(\cdot)$ stands for AES enciphering with key k, values $v_1, \ldots, v_l$ represent a length-padded copy of associated data d, and coefficients $C_1, \ldots, C_l$ are (secret) constants deterministically derived from the value $L = E(0^{128})$.

In the context of an ASA we argue that it is sufficient to leak the 128 bit value L. The attack procedure is, roughly, as in the AES-GCM case. Assume the adversary is provided with the OCB3 encryption $c = \mathsf{enc}(k, n, d, m)$ for unknown k, m but chosen d, n, and assume the adversary knows L and thus $C_1, \ldots, C_l$. Now let $1 \leq s < t \leq l$ be any two indices, let $\Delta = C_s + C_t$ and let $d' \neq d$, $|d'| = |d|$, be the associated data string with encoding $v'_1, \ldots, v'_l$ such that we have $v'_s = v_t + \Delta$ and $v'_t = v_s + \Delta$ and $v'_i = v_i$ for all $i \neq s, t$. Then we have $E_k(v'_s + C_s) = E_k(v_t + \Delta + C_s) = E_k(v_t + C_t)$ and $E_k(v'_t + C_t) = E_k(v_s + \Delta + C_t) = E_k(v_s + C_s)$, which leads to $F_k(d) = F_k(d')$ and ultimately $\mathsf{enc}(k, n, d', m) = \mathsf{enc}(k, n, d, m)$. In a CCA attack environment, this can immediately be leveraged to the full recovery of m. As in the AES-GCM case, we note that many variants of our attack exist (against all versions of OCB), including some that manipulate message bits in a controlled way.

5.3 Concrete Subversion Attacks via Acceptance vs. Rejection

We assume that the objective of a subverted receiver algorithm is to leak a bit string $k_\ell \in \{0,1\}^\lambda$ representing either some leakage that will enable recovery of the secret (private) key k_{R}, following the discussion at Sections 5.2.1.1 and 5.2.2, or else a string that is sufficient to break security in the sense of Section 5.2.3. We refer to k_ℓ as the leakage key in what follows. At worst, from the subverter's perspective, the leakage key will simply be a bit string representation of k_{R}.

We propose two key-recovering subversion attacks against a scheme $\Pi = (\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R})$ satisfying the syntax given in Section 3.2. While both attacks subvert the receiver algorithm only, they differ in that our first attack is passive (can be mounted by a mass surveillance adversary who eavesdrops) and our second attack is active (requires intercepting and modifying sender outputs in transmission – i. e., ciphertexts, in the case of AEAD or PKE, or message-tag pairs in the MAC case.). The driving principles behind the two attacks are closely related: In both cases the receiver algorithm of the attacked scheme is manipulated such that it marginally deviates from the regular accept/reject behaviour; by making these deviations depend on the leakage key, the bits of the latter are leaked one by one.

Our passive attack rejects a sparse subset of the receiver inputs that the unmodified algorithm would accept. Our active attack does the opposite by accepting certain receiver inputs that the unmodified algorithm would reject. A property of the former (passive) attack is that the scheme's probability of incorrect decryption is increased by a small amount (rendering it detectable with the same probability); we believe however that in settings where rejected messages are automatically retransmitted by the sender (for example, in low-level network encryption like IPSec), this attack is still practical and impactful. Our active attack does not influence correctness. However, as key bits are leaked only when the receiver algorithm is exposed to bogus inputs, successful adversaries are necessarily active. The active attack furthermore has the following attractive property: The underlying receiver outputs (i. e., messages) corresponding to the injected inauthentic receiver inputs are not arbitrary (and thus unexpected to the processing application), but identical with sender inputs previously sent by the sender algorithm. This allows attacks to be kept "under the radar": the receiver will not realise that an attack has been mounted, as all accepted messages it receives will be those sent by the sender.

We note that both of our subversions are stateless, which not only allows for much

easier backdoor implementation from a technical perspective but also should decrease the likelihood that an implemented attack is detected through code review or observing memory usage. That said, our passive attack also has a stateful variant with an interesting additional practicality feature. We discuss this further below. We note that our subversion approach, for leaking at most one bit per operation, remains on the conservative side. Depending on the circumstances, in practice, more aggressive methods that leak more than one bit per operation, are expected to be easily derived from our subversion proposals.

5.3.1 Combinatorics: Coupon Collection

The passive and active attacks both exfiltrate secret key material one bit at a time. The following lemma recalls a standard coupon collector statement that will help analysing the efficiency of this approach, in particular how long it takes until all bits are extracted. For a proof of the lemma, see e.g. [80, §8.4].

Lemma 5.1. *Fix a finite set S (of "coupons") and a probability $0 < \eta \leq 1$. Experiment $\mathrm{CC}(S, \eta)$ in Figure 5.2 measures the number of iterations it takes to visit all elements of S ("collect all coupons") when picked uniformly at random and considered with probability η. The expected number of iterations is given by $O(n \log n)$, where $n = |S|$. More precisely we have*

$$\mathbb{E}[\mathrm{CC}(S, \eta)] = \frac{|S|}{\eta}\left(\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{|S|}\right) = O(n \log n).$$

Note that parameter η is fully absorbed by the $O(\cdot)$ notation.

Exp $\mathrm{CC}(S, \eta)$
00 $S' \leftarrow \emptyset$; $l \leftarrow 0$
01 while $S' \subsetneq S$:
02 $\quad s \leftarrow_{\$} S$
03 $\quad$ if $B(\eta)$:
04 $\quad\quad S' \xleftarrow{\cup} \{s\}$
05 $\quad l \leftarrow l + 1$
06 stop with l

Figure 5.2: Coupon collector experiment (see Lemma 5.1). Recall that $B(\eta)$ denotes a Bernoulli trial with success probability η (see Section 2.1).

5.3.2 Passive Attack

We first give an intuition of our passive attack. Our attack subverts the receiver algorithm so that an adversary who observes decryption error events in a "normal" run of communication between sender and receiver is able to learn bits of the leakage key. The subverted receiver monitors incoming ciphertexts. It applies a hash function to each of them to obtain a pointer to a bit of the leakage key. It then, with a configurable probability, artificially rejects the ciphertext if the indicated bit of the leakage key does not match some hard-coded reference value. The adversary is able to apply the same hash function to the ciphertext and thus learns whether the bit position deviates from the reference value. By bit-wise learning the difference between the leakage key and the reference value, eventually the adversary can put the complete leakage key together.

In the remaining part of this section, we describe the specification of our subversion and KRP adversary in detail, and analyse their effectiveness.

5.3.2.1 Description of our Passive Attack

We define our passive subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a scheme Π in Figure 5.3 (left). It is parameterised by a probability $0 \leq \gamma \leq 1$, a large index space $\mathcal{I}_{\mathsf{R}}$, a PRF $(F_i)_{i \in \mathcal{I}_{\mathsf{R}}}$, and a family $(G_i)_{i \in \mathcal{I}_{\mathsf{R}}}$ of random constants. For the PRF we require that it be a family of functions $F_i \colon \mathcal{Y} \to [0 \,..\, \lambda - 1]$ (that is: a pseudo-random mapping from the ciphertext space to the set of bit positions of a leakage key k_ℓ), and for the constants we require that $G_i \in \{0,1\}^\lambda$ (that is: a random element of the set of leakage keys $\{0,1\}^\lambda$). (That we use the same index space $\mathcal{I}_{\mathsf{R}}$ for two separate primitives is purely for notational convenience; our analyses will actually assume that (F_i) and (G_i) are independent.[8])

Proc $\Pi.\mathsf{R}_i(k_{\mathsf{R}}, y)$	**Proc** $\mathcal{A}(i)$
00 $k_\ell' \leftarrow G_i$	09 $k_\ell' \leftarrow G_i$
01 $x \leftarrow \Pi.\mathsf{R}(k_{\mathsf{R}}, y)$	10 while k_ℓ' incorrect:
02 if $x = \perp$: return x	11 pick any $\alpha \in A$
03 if $B(\gamma)$:	12 $(y, \beta) \leftarrow \mathrm{Send}(\alpha)$
04 $\iota \leftarrow F_i(y)$	13 $x' \leftarrow \mathrm{Recv}(y)$
05 if $k_\ell'[\iota] \neq k_\ell[\iota]$:	14 if $x' = \perp$:
06 $x \leftarrow \perp$	15 $\iota \leftarrow F_i(y)$
07 // $k_\ell'[\iota] \leftarrow \,!\, G_i[\iota]$	16 $k_\ell'[\iota] \leftarrow \,!\, G_i[\iota]$
08 return x	17 return k_ℓ'

Figure 5.3: Passive subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a scheme Π. As in Figure 5.2, $B(\cdot)$ denotes a Bernoulli trial. We let $!\, b := 1 - b$ denote the inversion of a bit value $b \in \{0,1\}$. **Left:** Decryption subversion as in Section 3.2.1. Line 07 is redundant if the attack is stateless; in a stateful attack this line is meaningful – see the discussion below. **Right:** Key recovering adversary for game KRP as in Section 5.2.1.

We provide details on our attack. The idea is that k_ℓ' (Line 00), which is shared by the subverted algorithm and the key recovering adversary through knowledge of G_i, represents an initial reference key[9]; the key recovery adversary, throughout

[8] At the expense of introducing more symbols we could also have formally separated the index spaces of (F) and (G). We believe that our concise notation adds significantly to readability.

[9] For generality, we consider G a keyed family of constants. The simplest case would have all constants fixed to the same hardcoded string (say, the string of all zeroes), which would aid in

the attack, learns the bits that differ between k_ℓ and k_ℓ'. This means that the subversion only needs to leak (on average) half as many bits compared to leaking the whole of k_ℓ. The Bernoulli trial (Line 03) controls the rate with which such differing bits are exfiltrated, and the PRF (Line 04) controls which bit position ι is affected in each iteration. By PRF security, these bit positions can be assumed uniformly distributed (though knowledge of the subversion index i allows tracing which ciphertext is mapped to which position). Any bit difference is communicated to the adversary by artificially rejecting (Line 06) the ciphertext, although it is actually valid.

We specify a corresponding KRP adversary in Figure 5.3 (right). It starts with the same random string G_i as the subversion and traces the bit updates of $\Pi.\mathsf{R}_i$ until eventually the full key k_ℓ is reconstructed. We assume that $\mathcal{A}$ can tell whether the full leakage key k_ℓ has been recovered (Line 10), e. g., by recovering the secret key k_R from k_ℓ and verifying one or more recorded authentic outputs with it.

Note that our adversary $\mathcal{A}$ does not need to know the sender inputs $x \in \mathcal{X}$, which typically represent plaintext messages in the settings we consider, emerging throughout the experiment: The core of the attack, in Line 13 to Line 16, is independent of the value of x. This considerably adds to the practicality of our attack: While messages are not always secret information, *in practice* they might be hard to obtain. Conducting mass-surveillance attacks is certainly easier if the attacks depend exclusively on the knowledge of ciphertexts (like in our case, Line 15).

While we present our subversion as stateless (i. e., the reference key is kept static between invocations), it also works if the $\Pi.\mathsf{R}_i$ algorithm maintains state between any two invocations and remembers which differing bit positions have already been communicated. Activate Line 07, and execute Line 00 only during the first invocation, to obtain the stateful attack. With respect to the detectability and key

reducing the size of the implementation code.

recovery notions from Section 3.2, the attack's performance is the same whether the subversion is stateful or not. The stateful version offers better correctness *after* exfiltration, in the sense that the algorithm will only behave unexpectedly at most $|k_\ell| = \lambda$ occasions; once the leakage key k_ℓ has been exfiltrated, the subverted scheme $\Pi.\mathsf{R}_i$ behaves identically to the honest scheme $\Pi.\mathsf{R}$. (This case is practically less relevant and not covered by our formal models.)

We establish the following statements about the key recoverability and undetectability of our passive subversion attack.

Theorem 5.2. *For a δ-correct scheme Π, let $\Pi.\mathsf{R}_i$ be defined as in Figure 5.3 (left) and $\mathcal{A}$ as defined in Figure 5.3 (right). If F_i behaves like a random function and constants G_i are uniformly distributed, then for any message sampler* MS*, the key recovery advantage $\mathrm{Adv}^{\mathrm{krp}}_{\mathrm{MS}}(\mathcal{A})$ is expected to reach value 1 once the receive algorithm was invoked on $O(\lambda \log \lambda)$ different inputs.*

Proof. We model algorithm $\mathcal{A}(i)$ by experiment $\mathrm{CC}(S, \eta)$ from Figure 5.2, with $S = [0\,..\,\lambda - 1]$ and $\eta = 1-(\delta-1)(\gamma/2-1)$. The (pseudo-)randomness of F_i ensures that elements of $s \in S$, here representing the possible values of the index ι (Line 04), are picked uniformly at random. The probability $\eta = 1-(\delta-1)(\gamma/2-1) = \delta+(1-\delta)(\gamma/2)$ arises through success of the CC experiment being equivalent to the $\Pi.\mathsf{R}_i$ outputting $x = \perp$. This occurs either:

- Through an early exit with $\perp = \Pi.\mathsf{R}(k_\mathsf{R}, y)$ at Line 02, which has probability δ.
- Else, continuing to Line 03 with $\perp \neq \Pi.\mathsf{R}(k_\mathsf{R}, y)$ and triggering both Line 03 (with probability γ) and Line 05 (with probability 1/2, as $\Pr\left[k_\ell'[\iota] \neq k_\ell[\iota]\right]$ for $\iota \leftarrow F_i(t)$ is 1/2).

Applying Lemma 5.1 gives the expected number of messages to be sent as $O(\lambda \log \lambda)$. □

Theorem 5.3. *Let $\mathcal{A}$ be an adversary playing the* UDR *game (as in Figure 3.1, right), such that $\mathcal{A}$ makes at most q queries to the receiver oracle* Recv. *The undetectability advantage of the subversion $\Pi.\mathsf{R}_i$, as defined in Figure 5.3 (left), is bounded by*

$$\mathrm{Adv}^{\mathrm{UDR}}_{\Pi}(\mathcal{A}) \leq 1 - (1-\gamma)^q.$$

Proof. Any adversary playing the UDR game against the subverted $\Pi.\mathsf{R}_i$ must, in order to win, trigger $x = \perp$ with a valid sender output (receiver input) y. More precisely, the adversary $\mathcal{A}$ must find y such that $\Pi.\mathsf{R}(k_{\mathsf{R}}, y) \neq \perp$ but $\Pi.\mathsf{R}_i(k_{\mathsf{R}}, y) = \perp$. Figure 5.4 (left) shows the (obviously) best adversarial strategy. Even if the adversary can submit y such that $\iota \leftarrow F_i(y)$ would be assigned in Line 04 (Figure 5.3), this is contingent on $B(\gamma)$ succeeding in Line 03; thus $\Pr[x = \perp] \leq \gamma$ in Line 05 (Figure 5.4). Clearly, detection adversary $\mathcal{A}$ (Figure 5.4, left) always returns 1 when interacting with the unsubverted receiver algorithm, as always $x \neq \perp$. Thus,

$$\mathrm{Adv}^{\mathrm{UDR}}_{\Pi}(\mathcal{A}) = |\Pr[\mathrm{UDR}^1(\mathcal{A})] - \Pr[\mathrm{UDR}^0(\mathcal{A})]| \leq 1 - (1-\gamma)^q.$$

□

5.3.3 Active Attack

In this section we describe our second subversion attack. In contrast to the previous attack, key recovery requires an active adversary, i. e., one who injects crafted ciphertexts into the regular transmission stream. Our ASA has the desirable property (from the point of view of the subverter) that correctness is maintained.

We give an overview of our attack for the case of AEAD. (The generalisation to MAC and PKE is immediate; for the generic version following our abstract syntax see Fig-

Proc $\mathcal{A}$

```
00 (k_S, k_R) ←$ Π.gen
01 repeat q times:
02     pick any x ∈ 𝒳
03     y ← Π.S(k_S, x)
04     x ← Recv(k_S, y)
05     if x = ⊥:
06         return 0
07 return 1
```

Proc $\mathcal{A}$

```
00 (k_S, k_R) ←$ Π.gen
01 S ← {y}
02 ct = 0
03 while ct < q:
04     pick any x ∈ 𝒳
05     y ← Π.S(k_S, x)
06     y' ←$ 𝒴 \ S
07     if Π.R(y') = ⊥:
08     ct ← ct + 1
09         x ← Recv(k_S, y')
10         S ←∪ {y'}
11         if x ≠ ⊥:
12             return 0
13 return 1
```

Figure 5.4: Detection adversaries for Game UDR as in Figure 3.1. **Left:** For the passive attack from Figure 5.3. **Right:** For the active attack from Figure 5.5.

ure 5.5.) A prerequisite of the attack is a keyed random permutation P_i of the AEAD ciphertext space. The key is known exclusively to the subversion adversary. The AEAD encryption algorithm S remains unmodified. For honestly generated ciphertexts, the (subverted) algorithm R_i implements the unmodified AEAD decryption routine R. This ensures correctness.

To start a key recovery attack, the subversion adversary waits for an honest ciphertext c and replaces it with $P_i(c)$. That is, the adversary suppresses the delivery of c and instead injects a "randomised same-length version" of the ciphertext. By the authenticity property of the AEAD scheme, the unmodified R algorithm would reject this ciphertext. This is where the (subverted) R_i deviates from R: If any incoming ciphertext is deemed invalid upon decryption with R, with the expectation that this could be the case due to a KRA attack being in operation, R_i applies P_i^{-1} to c and tries to decrypt the result (with R). If R rejects, R_i rejects also (interpretation: c was simply a random ciphertext, not injected by the KRA adversary). If however R accepts, then R_i concludes that a KRA attack is in operation, and that it (R_i) is

supposed to leak key material.[10] Observing that R_i just recovered the originally encrypted message m, we let R_i either deliver that message, or we let it return $\perp$, i. e., indicate decryption failure. That way, if a message is delivered by R_i, it is always correct. As in Section 5.3.2, we modulate key bits into the decision of delivering vs rejecting.

The above should make clear how the attack works. Details, and the generic version, are in Section 5.3.3.2. We note that a technical prerequisite of the attack is that for valid ciphertexts c, $P_i(c)$ should not also be a valid ciphertext. As P is a random permutation, the standard AUTH and UF properties of AEAD and MAC ensure this. However, the situation is different for PKE where it is easy to define schemes that accept every ciphertext input, e. g., by outputting a valid but dummy message. We resolve this technicality by requiring the mild assumption of ciphertext sparseness: We say that a PKE scheme has a sparse ciphertext space if the decryption algorithm makes an internal decision about the validity of an incoming ciphertext, with the property that uniformly picked ciphertexts are deemed invalid with overwhelming probability.

We studied a range of practically relevant PKE schemes and observe that all of them satisfy the ciphertext sparseness demand. For reference we provide corresponding details for OAEP and Cramer-Shoup encryption in Appendix A.3. We further observe that the general classes of plaintext-aware schemes [22] (see also Appendix A.4) and of schemes with publicly verifiable ciphertexts [94] have this property as well. We confirm also for all four of the NIST post-quantum cryptography round 3 finalists[11], that are specifically designed to mask decryption failures by accepting every ciphertext and outputting a random value rather than the rejection symbol, that

[10] Multiple options for this exist, for instance could the full k_{R} be embedded into the returned m; this would be powerful, but also has practical disadvantages that would hinder effectiveness. We hence pursue a different, milder approach.

[11] Classic McEliece, CRYSTALS-KYBER, NTRU and SABER. Detailed information on the submissions, in particular their specifications, are available at the NIST PQC website https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

they provide ciphertext sparseness: The mechanics of the decryption/decapsulation algorithms are such that first an internal yet explicit ciphertext validity decision is made and then either the correct or an independent, randomised value is output. Our subversions can easily adapt to such specifications and be directly based on the outcome of the validity check.

Lastly, we note that for the active attack there is no advantage to the subverted receiver keeping state. This is because the subverted receiver reveals key bits only when explicitly queried by the adversary – thus, the adversary is able to maintain all necessary state. Note this is in contrast to our passive attack where the adversary observes the receiver but does not interact with it, and ultimately thus the attack could benefit mildly from the subversion R_i keeping state.

In the remaining part of this section, we define ciphertext sparseness, describe the specification of our subversion and KRA adversary, and analyse their effectiveness.

5.3.3.1 Ciphertext Sparseness

We define ciphertext sparseness for a scheme Π as follows: We say that $\Pi = (\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R})$ is ciphertext ε-sparse if $\Pr[\mathsf{R}(y) \neq \perp] \leq \varepsilon$ for $y \leftarrow_{\$} \mathcal{Y}$. If ε is negligibly small, we refer to the scheme as being ciphertext sparse. For AEAD and MAC schemes, ciphertext sparseness is a corollary of the unforgeability (authenticity) properties. In particular: an AEAD scheme that has $\mathrm{Adv}^{\mathrm{auth}}_{\mathsf{AEAD}}(\mathcal{A}) \leq \varepsilon$ is ciphertext ε-sparse, as if $\Pr[\mathsf{R}(y) \neq \perp] > \varepsilon$ for $y \leftarrow_{\$} \mathcal{Y}$, then an adversary who simply chooses an element of $\mathcal{Y}$ uniformly at random would win the AUTH game with probability greater than ε. Similarly, a MAC scheme with $\mathrm{Adv}^{\mathrm{uf}}_{\mathsf{MAC}}(\mathcal{A}) \leq \varepsilon$ is ciphertext ε-sparse. As discussed above, ciphertext sparseness is a reasonable assumption also for many practical PKE schemes.

Proc $\Pi.\mathsf{R}_i(k_\mathsf{R}, y)$
00 $k_\ell' \leftarrow G_i$
01 $x \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y)$
02 if $x \neq \bot$:
03 return x
04 $y' \leftarrow P_i^{-1}(y)$
05 $x' \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y')$
06 if $x' = \bot$:
07 return $\bot$
08 $\iota \leftarrow F_i(y')$
09 if $k_\ell[\iota] \neq k_\ell'[\iota]$:
10 return x'
11 return $\bot$

Proc $\mathcal{A}(i)$
12 $k_\ell' \leftarrow G_i$
13 while k_ℓ' incorrect:
14 pick any α
15 $(y, \beta) \leftarrow \mathrm{Send}(\alpha)$
16 $y' \leftarrow P_i(y)$
17 if $y' = y$, jump to line 14
18 $x \leftarrow \mathrm{Recv}(y')$
19 if $x \neq \bot$:
20 $\iota \leftarrow F_i(y')$
21 $k_\ell'[\iota] \leftarrow \,!\, G_i[\iota]$
22 return k_ℓ'

Figure 5.5: Active subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a ciphertext ε-sparse scheme Π (see the discussion at Section 5.3.3). **Left:** Decryption subversion as in Section 3.2.1. **Right:** Key recovering adversary for game KRA as in Section 5.2.1. The adversary needs to have no influence over messages (modelled by α; see the discussion at Section 5.2.1). As before we let ! denote the inversion of a bit value.

5.3.3.2 Description of our Active Attack

We define our active subversion of the receiver algorithm of ciphertext-sparse scheme Π in Figure 5.5 (left). It is parameterised by a large index space $\mathcal{I}_\mathsf{R}$, a PRF $(F_i)_{i\in\mathcal{I}_\mathsf{R}}$, a PRP $(P_i)_{i\in\mathcal{I}_\mathsf{R}}$, and a family $(G_i)_{i\in\mathcal{I}_\mathsf{R}}$ of random constants. (As in Section 5.3.2, our analyses will assume that (F_i) and (P_i) and (G_i) are independent.) For the PRF we require that it be a family of functions $F_i\colon \mathcal{Y} \to [0\,..\,\lambda-1]$ (that is: a pseudo-random mapping from the output space to the set of bit positions of a leakage key $k_\ell \in \{0,1\}^\lambda$), for the PRP we require that it be a family of length-preserving permutations $P_i\colon \mathcal{Y} \to \mathcal{Y}$ (that is: a pseudo-random bijection on the sender output space), and for the constants we require that $G_i \in \{0,1\}^\lambda$ (that is: a random element of the set of leakage keys).

The idea of our attack is as follows. Lines Line 01 to Line 03 of $\Pi.\mathsf{R}_i$ ensure that authentic receiver inputs are always accepted (no limitation on correctness). If however a receiver input (sender output) y is identified as not valid, i. e., is unauthentic, then

a secret further check is performed: The original value y is mapped to an unrelated value y' using the random permutation (Line 04), and the result y' is checked for validity (Line 05). For standard (invalid) sender outputs y this second validity check should also fail, and in this case algorithm $\Pi.\mathsf{R}_i$ rejects as expected (lines Line 06 and Line 07). The normally not attainable case that the second validity check succeeds is used to leak key bits. The mechanism for this (Line 08 to Line 11) is as in our passive attack from Section 5.3.2, namely by communicating via accept/reject decisions the positions where the bits of a hard-coded random reference value k_ℓ' and the to-be-leaked key k_ℓ differ.

The corresponding key recovery adversary crafts these required bogus receiver inputs by obtaining a valid sender output[12] y (Line 15) and modifies it in Line 16. The information thus leaked by the validity checking routine is used to reconstruct target leakage key k_ℓ in the obvious way (Line 19 to Line 21).

We establish the following statements about the key recoverability and undetectability of our active subversion attack.

Theorem 5.4. *For an ε-sparse scheme Π, let $\Pi.\mathsf{R}_i$ be defined as in Figure 5.5 (left) and $\mathcal{A}$ as in Figure 5.5 (right). If F_i and P_i behave like random functions, and constants G_i are uniformly distributed, then for any message sampler* MS*, the key recovery advantage $\mathrm{Adv}^{\mathrm{kra}}_{\mathrm{MS}}(\mathcal{A})$ is expected to reach value 1 once the receive algorithm was invoked on $O(\lambda \log \lambda)$ different inputs.*

Proof. By the ciphertext ε-sparseness of the scheme Π, each invocation of algorithm $\Pi.\mathsf{R}_i$ in an execution of attack $\mathcal{A}(i)$ has $x \neq \perp$ in Line 02 (Figure 5.5) with probability ε and thus Line 04 is reached with probability $1 - \varepsilon$. We model algo-

[12] Note that in the symmetric case, such authentic outputs are obtained by intercepting valid communications between the sender and receiver; in the public key case, an adversary can easily craft their own authentic outputs using the public key. We consider adversaries that have no influence on message choices for the most powerful attack (hence the arbitrary value of α in Line 14); adversaries who are able to utilise α (and β) may be even more effective.

rithm $\mathcal{A}(i)$ by the experiment $\mathrm{CC}(S, \eta)$ from Figure 5.2, with $S = [0\,..\,\lambda - 1]$ and $\eta = (1 - \varepsilon)/2$. The (pseudo-)randomness of F_i ensures that elements of $s \in S$, here representing the possible values of the index ι (Line 20), are picked uniformly at random. The probability $1/2$ arises through success of the CC experiment being equivalent to the condition $x \neq \bot$ in Line 19. This occurs precisely when $\Pi.\mathsf{R}_i$ returns $x' \neq \bot$ in Line 10, which is conditional on $\Pi.\mathsf{R}_i$ reaching past Line 07. The probability that $x \neq \bot$ in Line 19 is $1/2$ as this is the probability that for any sender output y' and $\iota \leftarrow F_i(y')$, $k_\ell[\iota] \neq G_i[\iota]$ (Line 09). We now apply Lemma 5.1, which gives us that the expected number of messages to be sent is $O(\lambda \log \lambda)$. □

Theorem 5.5. *Let $\mathcal{A}$ be an adversary playing the* UDR *game (as in Figure 3.1, right), such that $\mathcal{A}$ makes at most q queries to the verification oracle* Recv*. If P_i behaves like a random function, and the scheme Π is ciphertext ε-sparse, then the undetectability advantage of the subversion $\Pi.\mathsf{R}_i$, as defined in Figure 5.5 (left), is given by $\mathrm{Adv}^{\mathrm{udr}}_{\Pi}(\mathcal{A}) \leq 1 - (1 - \varepsilon)^q$.*

Proof. Any detection adversary $\mathcal{A}$ playing the UDR game against the subverted $\Pi.\mathsf{R}_i$ must, in order to win, trigger $\Pi.\mathsf{R}_i(y) \neq \bot$ with a bogus y. That is, a sender output y with $\Pi.\mathsf{R}(k_\mathsf{R}, y) = \bot$ but $\Pi.\mathsf{R}_i(k_\mathsf{R}, y) \neq \bot$. This will occur if $y = P_i(y')$, where $\Pi.\mathsf{R}(k_\mathsf{R}, y') \neq \bot$. As i is chosen uniformly randomly from $\mathcal{I}_\mathsf{R}$ and P is a (pseudo-)random function, the optimal strategy is to sample values of y' and test whether $\mathrm{Recv}(k_\mathsf{R}, y') \neq \bot$. Algorithm $\mathcal{A}$ in Figure 5.4 (right) shows this strategy. When $\mathcal{A}$ interacts with the unsubverted receiver algorithm, we have that $\Pr[\mathrm{UDR}^1(\mathcal{A})] = 1$ by construction. When interacting with the subverted receiver algorithm, $\mathcal{A}$ returns $x \neq \bot$ by either triggering Line 02 or Line 10 of $\Pi.\mathsf{R}_i$. By the ciphertext sparseness of the scheme, Line 02 is triggered with probability $1 - \varepsilon$. Triggering Line 10 happens with probability $\leq \varepsilon$. Thus we have

$$\mathrm{Adv}_{\Pi}^{\mathrm{UDR}}(\mathcal{A}) = |\Pr[\mathrm{UDR}^1(\mathcal{A})] - \Pr[\mathrm{UDR}^0(\mathcal{A})]| \leq 1 - (1-\varepsilon)^q \,.$$

□

5.4 Implementation

We implemented our attacks in proof-of-concept Python code (for the specific case of an AEAD scheme, as a representative case) to verify their functionality and effectiveness. The particular AEAD scheme we attack is AES-GCM [60], using black-box access to the implementation provided by [121]. We simulated both active and passive attacks 10,000 times, and recorded the number of queries for successful extraction of a 128-bit key (thus, $\ell = 128$). Messages, nonces and associated data were generated using the `random.getrandbits` method from the `Crypto.Random` library. The plots in Figures 5.6 and 5.7 are of histograms showing the distribution (in blue) of the recorded number of queries q, and (in red) the cumulative success probability as a function of q. Note that the variable being measured (number of queries) is a discrete value that is measured exactly.

Our results confirm the theoretical estimates from Theorems 5.2 and 5.4; in particular, the exponential success rate. While the attacks have different application and success profiles, both reliably recover keys.

Passive. The expected number of calls to the transcript oracle for successful exfiltration is given by $\frac{2\ell}{\eta}\sum_{i=1}^{\ell}\frac{1}{i}$ (see proof of Thm. 5.2). We set $\delta = 0$ and $\gamma = 0.2$ for illustration. This gives us an expected value of $q = 13910$ compared to the recorded mean of 13920.59. Alternatively, the result from Thm. 5.2 gives a key recovery advantage of $\approx 1/2$ with $q = 14000$, compared to the recorded median of 13380. The

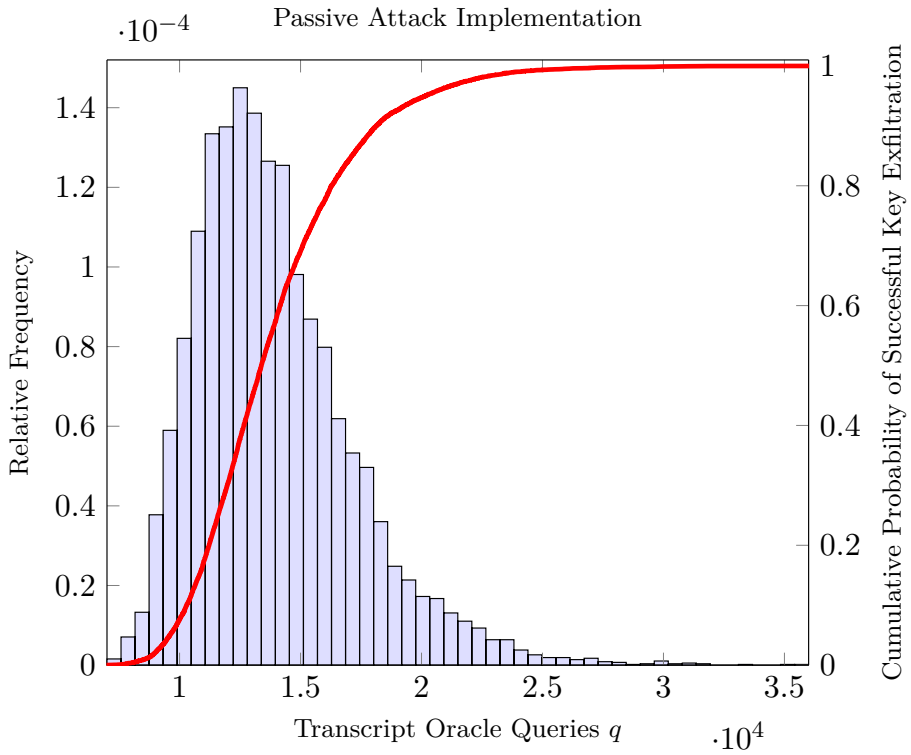

Figure 5.6: Results of running an implementation of the passive attack 10,000 times. Key length $\ell = 128$, and parameter $\delta = 0.1$. **Left axis:** The blue histogram shows the distribution of the number of queries required for successful key exfiltration. The data has been sorted into 50 bins. **Right axis:** The red curve shows the cumulative probability of successful key exfiltration against q.

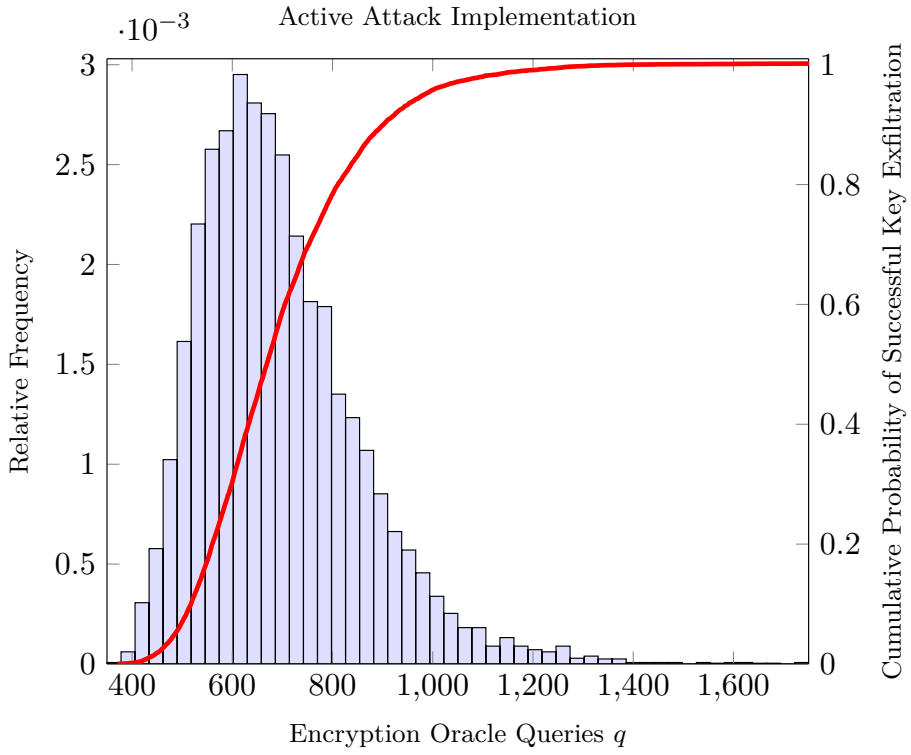

Figure 5.7: Results of running an implementation of the active attack 10,000 times with key length $\ell = 128$. **Left axis:** The blue histogram shows the distribution of the number of queries required for successful key exfiltration. The data has been sorted into 50 bins. **Right axis:** The red curve shows the cumulative probability of successful key exfiltration against q.

discrepancy is due to the exponential approximation in the proof.

Active. For the purposes of illustration, we set $\varepsilon = 0$. We assume that for AES-GCM, $\mathrm{Adv}^{\mathrm{auth}}_{\mathsf{AEAD}}(\mathcal{A}) \approx 0$ for any realistic adversary $\mathcal{A}$. The expected number of encryption calls for successful exfiltration is then $\ell \sum_{i=1}^{\ell} \frac{1}{i}$ (see proof of Theorem 5.4). This gives an expected value of $q = 696$ compared to the recorded mean of 695.05. Alternatively, the result from Thm. 5.4 gives a key recovery advantage of $\approx 1/2$ with $q = 710$ compared to the recorded median of 670. Again, the difference is due to exponential approximation.

5.5 Mitigating Subversion

As the discussion of cryptographic ("semantics driven") vs.non-cryptographic ("technology driven") subversion in Section 3.2.4 shows, achieving security against adversaries mounting ASAs is difficult, and essentially reduces to assuming trust in particular components or architectures. The three main theoretical approaches to preventing or mitigating against ASAs in the literature, discussed above in Section 3.4.4, are reverse firewalls, self-guarding protocols and watchdogs. We note that these approaches apply in the main to asymmetric primitives, and so (appropriately adapted to target receiver algorithms) would be suitable to defend against our attack against asymmetric schemes in Section 2.3.6.

Defending against our attacks on AEAD and MACs is more difficult. We note that the watchdog model applies in theory, while reverse firewalls and self-guarding approaches are ineffective against symmetric primitives. The watchdog model considers splitting a primitive into constituent algorithms that are run as subroutines by a trusted "amalgamation" layer. This allows the constituent algorithms to be individually checked and sanitised. Considering the verification algorithm of a MAC scheme as an example, the canonical approach of recalculating and checking the tag is modelled by letting the verification algorithm be a trusted amalgamation of the tagging algorithm with an identity test. The tagging algorithm typically runs a hash function as a subroutine, and so applying results from [65, 16, 57] would allow for the claim that the verification algorithm can be made subversion-resilient in the watchdog model. The assumption of a trusted amalgamation is precisely what makes our attack infeasible, but this assumption is questionable in real world settings. In particular, as we discussed above, the presence of non-cryptographic vectors makes this assumption unlikely to hold in practice.

Lastly, we note that none of the theoretical approaches are fully satisfying, requiring

strong or impractical assumptions. Indeed, it is telling that there are no implementations of subversion-resilient primitives to date, although some recent work seems promising in this regard [37, 26]. The best defense seems to be the unglamorous task of minimising risk by implementing a variety of control mechanisms across the whole infrastructure, in a process of security management. In particular: software implementations could be protected by measures including regular integrity tests and secure boot, hardware implementations could be protected by technical controls such as threshold implementations or testing amplification [62], and both cases can be strengthened by relying on open source implementations and verified supply chains. Whilst such measures can go some way towards minimising risk, we emphasise that there are no security guarantees.

5.6 Conclusion

In this chapter we described a class of attack targeting the receiving party, a class of ASA that was missed by previous work. Our class of attack applies to any scheme meeting the syntax introduced in Chapter 3 – in particular, AEAD, MACs and PKE. We conclude this chapter with some remarks regarding our ASA when applied to each scheme in turn. In Chapter 7 we discuss defending against our proposed subversion attacks.

5.6.1 AEAD

Our results stand in opposition to previous work [23, 51, 20] which proposed subversion resilience of a large class of AEAD schemes to which many if not all real-world constructions such as GCM, CCM and OCB belong, as long as their nonces are generated deterministically via a shared state maintained by both encryptor and

decryptor. The crucial observation to resolve this apparent contradiction is that previous work has assumed, besides explicitly spelled out requirements like uniqueness of ciphertexts and perfect decryptability, implicit notions such as integrity of ciphertexts. In the ASA setting for AEAD where undermining the confidentiality of a scheme is the primary goal of an adversary, it seems just as natural to assume that the adversary is also willing to compromise the integrity guarantees as well.

The internal details of our attacks (described in Sections 5.3.2.1 and 5.3.3.2) are such that we require a PRF $(F_i)_{i \in \mathcal{I}_R}$ to uniformly hash ciphertexts to bit positions. In the AEAD setting, this requirement can be dropped where the AEAD scheme meets the widespread design goal of IND$ security [102], i. e., ciphertexts indistinguishable from random bits. Combined with the fact that symmetric keys are typically 256 bits, the first 8 bits of the (uniformly distributed) ciphertexts are sufficient to point to the bit position. This allows for a reduced footprint (and thus significantly adds to the practicability of the attack for an adversary).

5.6.2 MACs

Applying our attack to a MAC leaks the secret key to an adversary, allowing them to forge any tag. This is an attractive goal for an adversary in real world settings, as once integrity has been compromised this can often be leveraged to perform any number of other attacks, for example: enabling attacks against ("encrypt-then-MAC") confidentiality; getting users to accept compromised (authenticated) software updates; injecting malicious packets into (secured) communication streams to de-anonymise users.

5.6.3 PKE

Our ASA attacks on PKE require a fairly large number of ciphertexts to be sent and observed to reject erroneously in order for the private key to be exfiltrated (e. g., 14000 to achieve a success probability of approximately 0.5 for the parameters considered in Section 5.4). In practice, this condition will be met: consider a server that hosts traffic for a large number of clients. The server will have a private/public key pair which is held static over long periods of time. Observing the server receive ciphertexts from many clients will allow an adversary to witness a large enough amount of traffic to recover the server's private key, rendering all communications between clients and server compromised.

Due to the high overheads associated with PKE, symmetric encryption is better suited to bulk communication. In most practical settings, PKE is used to establish a shared secret between the sender and receiver, so that the shared secret may be used as a key for communicating via symmetric encryption. This notion of sending keys for symmetric encryption via public key methods is formalised as a KEM. We show how our notions of subversion apply also to KEMs in Appendix A.2.1. KEMs are typically used together with a data encapsulation mechanism (DEM) in a so-called hybrid encryption scheme to PKE-encrypt messages. We give the definition of a DEM in Appendix A.2.2 for completeness.

5.6.3.1 DPKE

We note that as DPKE (as discussed in Chapter 4) represents a special case of PKE, our results apply there too. Exfiltrating the receiver's key would allow the adversary to break the confidentiality of encrypted messages, which in turn undermines the deniability of the scheme. If the adversary knows which message was originally sent,

then the sender is unable to later claim that they sent an alternative, fake, message. The absence of any practical DPKE schemes makes this a hypothetical scenario.

Chapter 6

Partitioning Oracles

Contents

In this chapter, we continue the theme of powerful adversaries undermining privacy of users. Whereas earlier chapters focussed on the powerful adversaries who are

motivated to carry out mass surveillance, this chapter considers the related aim of targeting users attempting to bypass internet censorship. We consider the specific scenario of users accessing the wider, free, internet via a proxy server. We note that the techniques and results we develop are of wider interest than the particular scenario considered. Our focus is a particular class of attack that allows an adversary to test whether a ciphertext was encrypted using any of a particular set of encryption keys using only one decryption query. Such an attack is known as a partitioning oracle attack, and was introduced by Len et al. (Usenix'21) as a new class of decryption error oracle which, conceptually, takes a ciphertext as input and outputs whether or not the decryption key belongs to some known subset of keys. Partitioning oracle attacks allow for a more efficient brute force search of the key space than querying one key per ciphertext, leading to practical attacks against low entropy keys (e. g., those derived from passwords). We discuss the applications to proxy servers and other practical scenarios.

Our main contribution is to show that weak key forgeries against polynomial hash-based AE schemes can be leveraged to launch partitioning oracle attacks. Weak key forgeries were given a systematic treatment in the work of Procter and Cid (FSE'13), who showed how to construct MAC forgeries that effectively test whether the decryption key is in some (arbitrary) set of target keys. Consequently, it would appear that weak key forgeries naturally lend themselves to constructing partition oracles; we show that this is indeed the case.

Lastly, our attacks in Chapter 5 relied on the assumption that an adversary has access to a decryption oracle – that is, can observe whether a receiver's algorithm implementation accepts or rejects a ciphertext. In this chapter, we give an illustration of how decryption oracles occur in practice, supporting the practicality of our attacks in Chapter 5.

6.1 Introduction

Authenticated Encryption (AE) schemes are designed to provide the core properties of confidentiality and message integrity against chosen-ciphertext attacks (CCA). A particularly important practical class of AE schemes offer Authenticated Encryption with Associated Data (AEAD); AEAD schemes are widely standardised and implemented due to their efficiency and security. As a result of their widespread adoption, AEAD schemes have in some cases been used in contexts that require additional properties beyond standard CCA security. One particular property that has attracted recent attention is key-commitment [74, 56, 3], also known as robustness [63], which (informally) states that a ciphertext will only decrypt under the key that was used to encrypt it.

A lack of key-commitment in particular AEAD schemes was exploited by Len et al. who introduced a new class of attack they call "partitioning oracle attacks" [89]. Conceptually, a partitioning oracle takes as input a ciphertext and outputs whether the decryption key belongs to some known subset of keys. Len et al. first construct so-called "splitting ciphertexts" for AES-GCM and ChaCha20Poly1305 that decrypt under every key in a set of target keys. This splitting ciphertext is submitted to a decryption oracle; on observing whether the ciphertext is accepted or rejected, the adversary learns whether or not the decryption key is in the set of target keys. As a result, the adversary is able to query multiple keys simultaneously, speeding up a brute force attack. Combining this with low entropy keys, such as those derived from passwords, results in practical attacks. Len et al. give a number of examples including against Shadowsocks [109], a censorship evasion tool, where the attack results in key recovery.

The concept of weak keys shares some similarities with that of partitioning oracles. While there is no precise definition in the literature, the concept is intuitively clear;

6.1 Introduction

Handschuh and Preneel [75] describe a weak key as a key that results in an algorithm behaving in an unexpected way (that can easily be detected) – the idea is that a weak key can be tested for with less effort than brute force. Procter and Cid [100] give a framework that neatly captures weak key forgeries (forgeries that are valid if the key is "weak"), which generalised previous attacks against polynomial hash-based message authentication codes (MACs) by Handschuh and Preneel [75] and Saarinen [107]. Procter and Cid's results showed that for these cases the term is a misnomer: in fact, for a polynomial hash-based MAC, any set of keys can be considered weak using their forgery techniques.

Abstractly, weak key forgeries and splitting ciphertexts share the same structure: ciphertexts whose successful decryption is contingent on the user's key being in a set of target keys. This suggests that weak key forgeries are a good candidate to carry out partitioning oracle attacks; we show that this is indeed the case. As in previous chapters, we are interested in powerful adversaries, modelling 'state level actors' who present the most sophisticated capabilities and who are able to access resources at a level of magnitude greater than non-state actors.[1] However, where earlier chapters focussed on the powerful adversaries who are motivated to carry out mass surveillance, this chapter considers the related aim of targeting users attempting to bypass internet censorship. We consider the specific scenario of users accessing the wider, free, internet via a proxy server. With that aim in mind, we first generalise the attack formalisation of Len et al. to allow the adversary to act as a machine-in-the-middle, in a more realistic reflection of an attacker's capabilities. Our model of the adversary's capabilities and strategy is informed by observed adversarial strategies against censorship evasion [31, 115].

As a result we obtain a more abstract definition that encompasses weak key forgeries and splitting ciphertexts. We show how to carry out a partitioning oracle attack

[1] See the discussion at Section 1.1.

using weak key forgeries, and discuss some practical applications of the attack. An advantage of our attack is the control that an adversary obtains over underlying plaintexts, allowing for partitioning oracle attacks in settings that are resistant to the attack of [89], in particular where there are format requirements on underlying plaintexts – including format requirements that are designed to render schemes key-committing. Our results reinforce the conclusions of [89], especially on the danger of deriving encryption keys from user-generated passwords. Furthermore, our results suggest that resistance to weak key forgeries should be considered a related design goal to key-commitment, particularly in settings that are vulnerable to partitioning oracle attacks. Concretely, our results demonstrate – in contrast to the suggestions of prior work – that adding structure to underlying plaintexts (e.g. packet headers that prefix every plaintext message, or an appended block of all zeros) is not a sufficient mitigation against partitioning oracle attacks.

6.1.1 Structure of the Chapter

We begin by providing the relevant background material on polynomial hash-based schemes in Section 6.2. Partitioning Oracle Attacks are introduced in Section 6.3, and our extension based on Weak Key Forgeries in Section 6.4. Section 6.5 describes our experiments with Shadowsocks, as well as other protocols. The chapter ends with some concluding remarks in Section 6.6.

This chapter comprises:

> Marcel Armour and Carlos Cid. "Partition Oracles from Weak Key Forgeries". In: *Cryptology and Network Security*. Ed. by Mauro Conti, Marc Stevens, and Stephan Krenn. Cham: Springer International Publishing, 2021.

6.2 Background: Polynomial Hashing

MACs (see Section 2.3.4) are a symmetric cryptographic primitive that allows two parties sharing a secret key to communicate with the assurance that their messages have not been tampered with. Many popular MAC schemes are constructed from universal hash functions that are realised by polynomial evaluation; such MACs based on polynomial hashing are discussed in Section 6.2.1. They are often used to provide the authentication component for AEAD schemes, which are discussed in Section 6.2.2, where we give an overview of the two most widely used polynomial hash based AEAD constructions, McGrew and Viega's Galois/Counter Mode (GCM) [91] and Bernstein's ChaCha20-Poly1305 [30].

6.2.1 MACs from Polynomial Hashing

A polynomial hash-based authentication scheme is built on a family of universal hash functions that are based on polynomial evaluation. It takes as input an authentication key hk and message m (consisting of plaintext or ciphertext blocks depending on context). Let $m = m_1 \parallel \cdots \parallel m_{|p|} \parallel m_{|p|+1}$ with $m_{|p|+1} = \text{len}(m)$ and all m_i considered as elements of a field $\mathbb{F}$ (typically $\mathbb{F}_{2^n}$), and $G_m(x)$ be the polynomial in $\mathbb{F}[x]$ defined as $G_m(x) = \sum_{i=1}^{|p|+1} m_i x^{|p|+2-i}$. If we also consider $hk \in \mathbb{F}$, the polynomial hash $H_{hk}(m)$ of m is calculated by evaluating $G_m(x)$ at hk, i. e.,

$$H_{hk}(m) := G_m(hk) = \sum_{i=1}^{|p|+1} m_i hk^{|p|+2-i} \in \mathbb{F}\,.$$

The hash value is usually encrypted with a pseudo-random one-time pad, to provide the output authentication tag.

The underlying properties of polynomials are inherited by the hash function and

thus the authentication scheme; in particular, the fact that adding a zero valued polynomial will not change the value of the hash (which gives rise to "weak key" forgeries, discussed in Section 6.2.5) and the fact that it is possible to construct a polynomial that passes through a set of given points (giving rise to multi-key collisions, discussed in Section 6.2.4).

6.2.2 AEAD

In this section we consider two specific instantiations of AEAD schemes. See the definitions at Section 2.3.5 for our syntax. A common paradigm for constructing AEAD schemes is to use an Encrypt-then-MAC (EtM) construction with a stream cipher for encryption and an authentication component from a polynomial based universal hash function. Recall that in our notation, the encryption algorithm of an AEAD scheme returns a single ciphertext c. For an AEAD scheme derived from an EtM construction, output ciphertexts take the form $c = (C, \tau)$ consisting of a constituent encryped component C and tag component τ. We now give a brief overview of the most widely adopted and standardised schemes: McGrew and Viega's AES Galois/Counter Mode (AES-GCM) [91] and Bernstein's ChaCha20-Poly1305 [30, 99].

6.2.3 AES-GCM

We briefly discussed AES-GCM in Section 5.2.3; for this chapter we require a more detailed exposition in order to clearly describe weak key forgeries. AES-GCM encryption takes as input: an AES key k, a nonce n, plaintext $p = p_1 \parallel \cdots \parallel p_{|p|}$ and associated data $d = d_1 \parallel \cdots \parallel d_{|d|}$. The key is 128, 192 or 256 bits long, the nonce n should preferably be 96 bits long although any length is supported. For each i, $|p_i| = |d_i| = 128$ except for perhaps a partial final block. With this input,

AES-GCM returns a ciphertext $C = C_1 \parallel \cdots \parallel C_{|p|}$ (the same length as the plaintext) and an authentication tag τ. From here on, we will omit associated data for simplicity. The plaintext is encrypted using an instance of the AES in counter mode, under key k with counter value starting at CTR_1. If the nonce is 96 bits long the initial counter value (CTR_0) is $n \parallel 0^{31}1$, otherwise it is a polynomial evaluation-based hash of n after zero padding (using the hash key described below). For each i, $\mathrm{CTR}_i = \mathrm{inc}(\mathrm{CTR}_{i-1})$, where $\mathrm{inc}(\cdot)$ increments the last 32 bits of its argument (modulo 2^{32}).

The authentication tag is computed from GHASH, a polynomial evaluation hash (in $\mathbb{F}_{2^{128}}$), described below. First, the ciphertext C is parsed as 128-bit blocks (with partial final blocks zero padded) and each block is interpreted as an element of $\mathbb{F}_{2^{128}}$. We denote by ℓ an encoding of the length of the (unpadded) ciphertext and additional data. The hash key hk is derived from the AES block cipher key: $hk = E_k(0^{128})$. The hash function GHASH is then computed as:

$$H_{hk}(C) = \ell \cdot hk \oplus C^*_{|p|} \cdot hk^2 \oplus C_{|p|-1} \cdot hk^3 \oplus \cdots \oplus C_2 \cdot hk^{|p|} \oplus C_1 \cdot hk^{|p|+1}, \quad (6.1)$$

where all operations are in $\mathbb{F}_{2^{128}}$, and $C^*_{|p|}$ denotes the zero-padded last block. The authentication tag is given by: $\tau = E_k(\mathrm{CTR}_0) \oplus H_{hk}(C)$.

ChaCha20-Poly1305. Poly1305 is similar to GHASH, and to form AEAD schemes it is most commonly combined with the ChaCha20 stream cipher [29], although Poly1305-AES is also an option [30]. For concreteness, we will give a description of ChaCha20-Poly1305 and note that the differences are trivial.

ChaCha20-Poly1305 encryption takes as input: a 32-byte ChaCha20 key k, a 12-byte nonce n, plaintext p and additional data d. With this input, ChaCha20-Poly1305 returns a ciphertext C (the same length as the plaintext) and an authentication tag τ of length 16 bytes. From here on, we will omit the associated data for simplicity.

First, the plaintext is divided into 64 byte blocks, except perhaps for a partial final block, and encrypted using the ChaCha20 stream cipher, under key k.

The authentication tag is next computed from a polynomial evaluation hash in the finite field $\mathbb{F}_{2^{130}-5}$. The ciphertext to be hashed is divided into 16-byte blocks with any partial final block zero-padded to 16 bytes. We denote by ℓ an encoding of the (unpadded) ciphertext and additional data. Each block is encoded as an integer modulo $2^{130}-5$ by first appending `0x01` to each block, and interpreting the resulting block as a little-endian integer X_i.

The authentication tag is computed from a polynomial evaluation hash (in $\mathbb{F}_{2^{130}-5}$). First we derive the hashing key r and the pseudo-random one time pad s: the first 32 bytes of $hk = E_k(n_0 \parallel n)$ is divided into two 16-byte strings $\tilde{r}$ and s. Here n_0 represents 0 encoded as a 4-btye little-endian integer.

The hashing key r is obtained from the string $\tilde{r}$ by setting some of the bits to zero in a process referred to as "clamping"; we gloss over the specific details. The hash function is then computed as

$$H_r(C) = \ell \cdot r \oplus C^*_{|p|} \cdot r^2 \oplus C_{|p|-1} \cdot r^3 \oplus \cdots \oplus C_2 \cdot r^{|p|} \oplus C_1 \cdot r^{|p|+1},$$

where all operations are in $\mathbb{F}_{2^{130}-5}$, and $C^*_{|p|}$ denotes the last zero-padded block. The authentication tag is given by:

$$\tau = (s \oplus H_r(C)) \mod 2^{128},$$

where s and $H_r(C)$ are interpreted as elements of $\mathbb{F}_{2^{128}}$, and the result as an integer modulo 2^{128}.

6.2.4 Key Commitment

A committing AE scheme is one which satisfies the property of *key commitment*, which (informally) states that a ciphertext will only decrypt under the key that was used to encrypt it. Equivalently, for a committing AE scheme, it should be infeasible to find a ciphertext that will decrypt under two different keys. Security goals for committing AE were first formalised by Farshim et al. [63] under the name "robustness". Although key commitment is not part of the design goal of AE schemes, there are natural scenarios where a lack of key commitment results in security issues. Dodis et al. [56] and Grubbs et al. [74] show how to exploit non-committing AE schemes in the context of abuse reporting in Facebook Messenger. Albertini et al. [3] give some further practical examples where a lack of key commitment leads to practical attacks, e. g., in the setting of paywalled subscription material where a malicious publisher might prepare a ciphertext that decrypts to different content for different users.

Generic AE solutions, the so-called *generic composition* constructions such as Encrypt-then-MAC, can provide key-commitment, as shown by Farshim et al. [63] who suggested using a keyed hash function such as HMAC [18] for authentication. However, if a key-committing scheme is required for security in some particular setting, then performance considerations may mean that switching to e. g., encrypt-then-HMAC is not practical. This is illustrated by the choice of Facebook Messenger to use AES-GCM to encrypt message attachments despite work showing that this was insecure. Albertini et al. [3] propose two generic fixes that minimise the changes needed to add key-commitment to widely deployed, highly efficient schemes such as AES-GCM:

1. **Padding Fix**. Prepend a constant string to messages before encrypting; check for the presence of the constant string after decrypting. This fix is also given in an early draft of an OPAQUE protocol RFC [87], and discussed in [89]. This

solution – essentially adding redundancy to the message – is not generically secure and must be analysed per scheme. Albertini et al. [3] perform this analysis for AES-GCM and ChaCha20-Poly1305, showing that in both cases the resulting scheme is key-committing.

2. **Generic Fix**. From a given key k, derive an encryption key $k_{\text{enc}} = F_{\text{enc}}(k)$ and a commitment to the key $k_{\text{com}} = F_{\text{com}}(k)$. Here F_{enc} and F_{com} are collision resistant hash functions. Ciphertexts for the resulting key-committing scheme consist of a regular ciphertext (for the underlying AEAD scheme) together with the commitment to the key. Albertini et al. [3] show that this construction provides key-commitment, if the functions F_{enc} and F_{com} used to derive the encryption key and commitment are collision resistant pseudorandom functions.

6.2.5 Weak Key Forgeries

In symmetric cryptography, a class of keys is called a *weak key class* if the algorithm behaves in an unexpected way when operating under members of that class, and this behaviour is easy to detect. In addition, identifying that a key belongs to such a weak key class should require trying fewer than N keys by exhaustive search (or verification queries), where N is the size of the class [75]. In the context of polynomial hash-based authentication schemes, e.g. the GCM mode, Handschuh and Preneel [75] and Saarinen [107] identified several weak key classes. In [100], Procter and Cid proposed a generic framework to mount forgery attacks against polynomial-based MAC schemes based on weak keys. Their framework encompasses the previous forgery attacks from [75] and [107], as well as the earlier Joux's Forbidden Attack [82], and is based on a malleability property present in polynomial-based MAC schemes.

6.2.5.1 Procter and Cid's Weak Key Forgery Framework

If H_{hk} is a polynomial hash under key hk and m is a message input, let $H_{hk}(m) = G_m(hk)$, where $G_m(x) = \sum_{i=1}^{|p|+1} m_i x^{|p|+2-i} \in \mathbb{F}[x]$ and $hk \in \mathbb{F}$ (as in Section 6.2.1). Now let $Q(x) = \sum_{i=1}^{|p|+1} q_i x^{|p|+2-i} \in \mathbb{F}[x]$ be a polynomial with constant term zero, such that $Q(H) = 0$. Then

$$H_{hk}(m) = G_m(hk) = G_m(hk) + Q(hk) = G_{m+q}(hk) = H_{hk}(m+q),$$

where $q = q_1 \parallel q_2 \parallel \ldots \parallel q_\ell$ and the addition $m+q$ is done block-wise[2]. It follows that given a polynomial $Q(x)$ satisfying these properties, it is straightforward to construct collisions for the hash function. In fact, we have that $Q(x)$ is in the ideal $\langle x^2 - hkx \rangle$, and any polynomial in this ideal can be used to produce collisions. On the other hand, collisions in the hash function correspond to MAC forgeries, by substituting the original message for the one that yields a collision in the polynomial hash. Thus this method allows an adversary to create forgeries when they have seen a tuple of (nonce, message, tag), by simply modifying the message, as above. Saarinen's cycling attacks [107] are a special case of this attack. Forgeries for GCM and variants are presented in [100]. Later, an efficient method for constructing forgery polynomials which have disjoint sets of roots (i.e. keys) was proposed in [1].

6.3 Partitioning Oracle Attacks

Partitioning oracles, introduced by Len et al. [89] are a class of decryption error oracles which, conceptually, take a ciphertext and return whether the decryption key belongs to some known subset of keys. This allows an adversary to speed up an exhaustive search by querying multiple keys at once; in effect, partitioning

[2] The shorter message is zero-padded if required.

the key space. The approach of [89] relies on two conditions: (1) the non-key committing property of polynomial hash-based AE schemes is exploited to craft targeted "splitting" ciphertexts that will decrypt under multiple keys; and (2) a decryption oracle that reveals whether decryption (with the user's key) of such a splitting ciphertext succeeds or not.

The partitioning oracle attack of Len et al. proceeds as follows. They construct a ciphertext $\hat{c}$ that decrypts under every key in a set of target keys $K^* = \{k_1, \cdots, k_\ell\}$ by constructing a linear equation whose variables are the blocks of ciphertext; $\hat{c}$ is the solution to the equation. We describe the technique using AES-GCM for concreteness.

Given K^* and nonce n, first derive the associated GHASH key $hk_i = E_{k_i}(0^n)$ for each $k_i \in K^*$. Then construct the linear equation

$$\tau = C_1 \cdot hk_i^{|p|-1} \oplus \cdots \oplus C_{|p|-1} \cdot hk_i^2 \oplus \ell \cdot hk_i \oplus E_{k_i}(n \parallel 0^{31}1)\,,$$

which is arrived at by assigning hk_i to hk in Equation (6.1) and substituting the result into the expression for the tag $T = H_{hk}(C) \oplus E_{k_i}(\mathrm{CTR}_0)$. The result is a system of ℓ equations in ℓ unknowns which can be solved; this can be done more efficiently using a clever trick (fixing τ and adding one block of ciphertext as a new variable, giving a Vandermonde matrix). We refer the reader to [89] for further detail.

Abstractly, a partitioning oracle will (in the optimal case) allow a binary search of the key space, giving a logarithmic improvement over naïve exhaustive search. This requires being able to query half the keys in the key space. In practice however, there is a limit to the number of keys that can be queried at once – e. g., for AES-GCM, messages are required to be less than approx. 64GB ($2^{39} - 256$ bits [59]), and applications may impose further restrictions depending on context. Nevertheless, as

shown in [89], it is still possible to launch practical attacks by combining partitioning oracles with knowledge of non-uniform key distributions, which arise in particular when human memorable passwords are used to derive keys, and can be estimated from password breaches [96].

We note that the conditions for a partitioning oracle attack can be satisfied with weak key forgeries, following the work of Procter and Cid [100] (see Section 6.2.5). Weak key forgeries require a valid ciphertext to construct the forgery; a crucial difference to [89], which considers adversaries that only have access to a decryption oracle. In practice this is a limitation of the adversary that does not tally with observed adversarial strategies against censorship evasion [31, 115]. We thus extend the model by allowing an adversary to obtain valid ciphertexts from chosen plaintexts, a standard adversarial model for AE. In fact, this assumption is stronger than required; as we later show, adversaries with only "machine-in-the-middle" capabilities can carry out effective partitioning oracle attacks using weak key forgeries. Known and chosen plaintext capabilities lead to more powerful attacks, as we briefly describe in Section 6.5.1.

Example: Generic Encryption. Consider a client and server communicating with end-to-end encryption, using an AEAD scheme and a shared key k derived from password pw. The client encrypts message p (together with any associated data d), using key k and nonce n to obtain a ciphertext tag pair $(C, \tau) \leftarrow \mathsf{AEAD.enc}(k, n, d, p)$. The conditions for a partitioning oracle attack are met if the server reveals whether or not decryption succeeds; it might for example output an observable error message, or reveal the information via a side-channel.

Example: Password-authenticated Key Exchange. A Password Authenticated Key Exchange (PAKE) is a cryptographic key exchange protocol in which a client authenticates to a server using a password pw that the server has stored (as

the equivalent of a hash). Len et al. show how to launch a partitioning oracle attack against OPAQUE, a modern PAKE protocol currently undergoing standardisation. OPAQUE uses an AEAD scheme as a component, and Len et al. show the necessity of the AEAD scheme being key-committing by considering deviations from the specification in some early prototype implementations. OPAQUE works by composing an oblivious PRF with an authenticated key exchange; Len et. al.'s attack relies on the fact that the server sends a ciphertext C encrypted using the password during an execution of the protocol.

6.3.0.1 Related Work

Bellovin and Merritt introduced partition attacks against encrypted key exchange: trial decryption of intercepted traffic allowed multiple keys to be eliminated at once [25]. Other oracle attacks include padding oracles [35, 113] or other format oracles [5, 4, 68]; these attacks are similar to but distinct from partitioning oracles as they recover information regarding plaintexts rather than secret keys.

6.3.1 Formal Definition of a Partitioning Oracle

Following [89], we consider settings in which an attacker targets AE and seeks to recover a user's key $k \in \mathcal{K}$, where the key is deterministically derived from secret password $pw \in \mathcal{D}$. We write $\mathcal{K}(\mathcal{D}) \subseteq \mathcal{K}$ for the set of keys derived from passwords and $k(pw) \in \mathcal{K}(\mathcal{D})$ to denote a key derived from password pw. The attacker is given access to an interface that takes as input ciphertext c, and outputs whether or not the ciphertext decrypts correctly (passing any format checks) under the user's key $k(pw)$. The attacker is further given access to an interface that will encrypt plaintexts of the attacker's choosing and return the ciphertext. This set-up represents a "partitioning oracle" if it is computationally tractable for the adversary,

given any set $K \subseteq \mathcal{K}(\mathcal{D})$, to compute a value $(n, d, \hat{c})$ that partitions K into two sets K^* and $K \setminus K^*$, with $|K^*| \leq |K \setminus K^*|$, such that $\mathsf{AEAD.dec}(k, n, d, \hat{c}) \neq \perp$ for all $k \in K^*$ and $\mathsf{AEAD.dec}(k, n, d, \hat{c}) = \perp$ for all $k \in K \setminus K^*$. We call such a $\hat{c}$ a *splitting ciphertext* and refer to $|K^*|$ as the degree of $\hat{c}$. We distinguish between targeted splitting ciphertexts, where the adversary can select the secrets in K^*, and untargeted attacks.

In general, the definition can be applied to arbitrary cryptographic functionalities by considering a Boolean function f that takes as input a string and a key, returning 1 if some cryptographic operation succeeds and 0 otherwise. The attacker has access to an interface that takes as input a bit string V, and uses it plus k to output the result of some Boolean function $f_k : \{0,1\}^* \to \{0,1\}$. Here f_k is an abstraction of some cryptographic operations that may succeed or fail depending on k and V; set $f_k(V) = 1$ for success and $f_k(V) = 0$ for failure. We note that partitioning oracles may output more than two possible outputs, for example if there are multiple distinguishable error messages, following [36].

6.3.2 Multi-Key Contingent Forgeries

Central to launching a partitioning oracle attack is the ability to craft splitting ciphertexts. This is formalised in the notion of "Targeted Multi-Key Contingent Forgeries", which quantifies an adversary's advantage in crafting splitting ciphertexts against a particular AEAD scheme, with oracle access to encryption. Our definition is a slight generalisation of the "Targeted Multi-Key Collision" notion from [89]; their notion can be obtained from ours by removing the adversary's encryption oracle.[3]

Targeted multi-key contingent forgery resistance (TMKCR) security is defined by

[3]We hope the reader forgives our abuse of nomenclature; although we refer to both notions as TMKCR, ours is a (slight) generalisation of Len et al.'s, and we use the term "key contingent forgery" to encompass both.

the game given in Figure 6.2. It is parameterised by a scheme AEAD and a target key set $K^* \subseteq \mathcal{K}$. A possibly randomised adversary $\mathcal{A}$ is given input a target set K^* and must produce nonce n^*, associated data d^* and ciphertext C^* such that $\mathsf{AEAD.dec}(n^*, d^*, C^*) \neq \perp$ for all $k \in K^*$. We define the advantage via

$$\mathsf{Adv}^{\text{tmk-cr}}_{\mathsf{AEAD},K^*}(\mathcal{A}) = \Pr\left[\text{TMKCR}^{\mathcal{A}}_{\mathsf{AEAD},K^*} \Rightarrow 1\right] \tag{6.2}$$

where "$\text{TMKCR}^{\mathcal{A}}_{\mathsf{AEAD},K^*} \Rightarrow 1$" denotes the event that $\mathcal{A}$ succeeds in finding n^*, d^*, C^* that decrypt under all keys in K^*. The event is defined over the coins used by $\mathcal{A}$.

We can define a similar untargeted multi-key contingent forgery resistance goal, called $\text{MKCR}^{\mathcal{A}}_{\mathsf{AEAD},\kappa}$. The associated security game, given in Figure 6.2, is the same except that the adversary gets to output a set K^* of its choosing in addition to the nonce n^*, associated data d^*, and ciphertext C^*. The adversary wins if $|K^*| \geq \kappa$ for some parameter $\kappa > 1$ and decryption of n^*, d^*, C^* succeeds for all $k \in K^*$. We define the advantage via

$$\mathsf{Adv}^{\text{mk-cr}}_{\mathsf{AEAD},\kappa}(\mathcal{A}) = \Pr\left[\text{MKCR}^{\mathcal{A}}_{\mathsf{AEAD},\kappa} \Rightarrow 1\right] \tag{6.3}$$

where "$\text{MKCR}^{\mathcal{A}}_{\mathsf{AEAD},\kappa} \Rightarrow 1$" denotes the event that $\mathcal{A}$ succeeds in finding K^* and n^*, d^*, C^* that decrypt under all keys in K^*. The event is defined over the coins used by $\mathcal{A}$.

6.4 Partitioning Oracle Attacks from Weak Key Forgeries

At a high level, our attack works as follows: Construct key-contingent forgeries from captured ciphertexts using weak-key forgery techniques and submit these to a decryption oracle; that is, an oracle that reveals whether a ciphertext is accepted or rejected. The weak key forgery ensures that the ciphertext will only be accepted if

Game $\mathrm{TMKCR}^{\mathcal{A}}_{\mathsf{AEAD},K^*}$
00 require $K^* \subset \mathcal{K}$
01 $k \leftarrow_{\$} K^*, N \leftarrow \emptyset$
02 $(n^*, d^*, C^*) \leftarrow \mathcal{A}^{\mathrm{Enc}}(K^*)$
03 stop with $[\mathsf{AEAD.dec}([k]n^*, d^*, C^* \neq \perp]$

Oracle $\mathrm{Enc}(n, d, p)$
04 require $n \notin N$
05 $N \xleftarrow{\cup} n$
06 return $\mathsf{AEAD.enc}(k, n, d, p$

Figure 6.1: Targeted Multi-Key Collision Resistance game modelling targeted multi-key contingent forgery resistance for an AEAD scheme. Note that an adversary who can produce a ciphertext C^* that decrypts under every key in K^* will win the game with probability 1.

Game $\mathrm{MKCR}^{\mathcal{A}}_{\mathsf{AEAD},\kappa}$
00 $K^* \leftarrow_{\$} \mathcal{A}(\kappa)$; require $K^* \subset \mathcal{K}$ and $|K^*| \geq \kappa$
01 $k \leftarrow_{\$} K^*, N \leftarrow \emptyset$
02 $(n^*, d^*, C^*) \leftarrow \mathcal{A}^{\mathrm{Enc}}(K^*)$
03 stop with $[\mathsf{AEAD.dec}(k)(n^*, d^*, C^*) \neq \perp]$

Oracle $\mathrm{Enc}(n, d, p)$
04 require $n \notin N$
05 $N \xleftarrow{\cup} n$
06 return $\mathsf{AEAD.enc}(k, n, d, p)$

Figure 6.2: Multi-Key Contingent Forgery Resistance game modelling multi-key contingent forgery resistance for an AEAD scheme. Note that Multi-Key Contingent Forgery Resistance is a weaker notion than Targeted Multi-Key Collision Resistance (fig. 6.1) which lets the adversary choose the set of target keys K^*.

the user's key is in the set of weak keys.

More specifically: (1) In an offline phase, the adversary pre-computes a set of ciphertext masks. Each mask corresponds to a set of passwords to be tested. (2) In an online phase, the adversary intercepts a ciphertext and, using a ciphertext mask, constructs a key-contingent forgery which it forwards to the partitioning oracle. Observing whether or not the key-contingent forgery is accepted reveals whether or not the user's key is in the set of target keys corresponding to the ciphertext mask. Our attack relies on the ability of the adversary to act as a "machine-in-the-middle" between sender and receiver. We first give an abstract description of a key contingent forgery consisting of ℓ ciphertext blocks which encompasses two special cases: a targeted key-contingent forgery testing ℓ keys, (Section 6.4.1); and a targeted forgery passing format requirements on underlying plaintexts, (Section 6.4.2).

1. **Offline phase.** The attack takes a set of target keys $K^* = \{k_1, \ldots, k_{\ell-1}\}$ as input and outputs a ciphertext mask. We note that one key is lost per ciphertext block that is not a free variable.

   (a) First derive the associated authentication (GHASH) keys by setting $K_H^* = \{E_k(0^{128}) | k \in K^*\}$.

   (b) Set $Q(x) = \sum_{i=1}^{\ell} q_i \cdot x^{\ell+1-i} = x \cdot \prod_{hk \in K_H^*} (x \oplus hk)$.

2. **Online phase.** The online phase takes as further input a valid nonce, ciphertext, tag tuple (n, C, τ) and outputs a key-contingent forgery consisting of tuple $(n, \hat{c}, \tau)$. The key-contingent forgery is forwarded to the partitioning oracle. In what follows, we assume that $\ell - 1 \geq p = \lceil \text{len}(C/128) \rceil$

   (a) First parse the captured ciphertext as $C = C_1 \parallel \cdots \parallel C_{|p|}^*$, i. e., as blocks of the appropriate length. Let $\alpha = \text{len}(C) \oplus \text{len}(\hat{c})$ and β be constants.

Now set $Q'(x) = \sum_{i=1}^{\ell+1} q'_i \cdot x^{\ell+2-i} = (a \oplus bx) \cdot Q(x)$, with

$$a = \alpha \cdot q_\ell^{-1} \text{ and } b = \beta \cdot q_2^{-1} \oplus \alpha \cdot q_1 \cdot q_2^{-1} \cdot q_\ell^{-1}. \tag{6.4}$$

Set $q' = q'_1 \parallel \cdots \parallel q'_\ell$. Note that $q'_{\ell+1} = q_\ell \cdot a = \alpha$ and $q'_1 = a \cdot q_1 \oplus b \cdot q_2 = \beta$. This step can take place offline if $\text{len}(C)$ is known in advance.

(b) Let $\hat{c} = C^* \oplus q'$, where $C^* = 0^{128} \parallel \cdots \parallel 0^{128} \parallel C_1 \parallel \cdots \parallel C^*_{|p|}$ denotes the ciphertext C padded (pre-pended) with blocks of zeros to match the length of q'. As $\ell \geq p + 1$, at least one block of padding is pre-pended.

Note that if the user key $k \in K^* \cup \{K_\ell\} \cup \{0\}$, where $K_\ell = a \cdot b^{-1}$, then for $hk = E_k(0^{128})$,

$$\begin{aligned} H_{hk}(\hat{c}) &= \text{len}(\hat{c}) \cdot hk \oplus \hat{c}^*_\ell \cdot hk^2 \oplus \hat{c}_{\ell-1} \cdot hk^3 \oplus \cdots \oplus \hat{c}_1 \cdot hk^{\ell+1} \\ &= (\alpha \oplus \text{len}(C)) \cdot hk \oplus \left(C^*_{|p|} \oplus q'_\ell\right) \cdot hk^2 \oplus \cdots \oplus (0^{128} \oplus q'_1) \cdot hk^{\ell+1} \\ &= Q'(hk) \oplus H_{hk}(C). \end{aligned}$$

Consequently, the tag is a valid forgery and $\mathsf{AEAD.enc}(k, n, \varepsilon, \hat{c}) \neq \perp$.

6.4.1 Targeted Key Contingent Forgery Testing ℓ keys

We first consider key contingent ciphertext forgeries that test ℓ keys with no restrictions on the format of the underlying plaintext. Setting $\beta = a \cdot q_2 \cdot hk_\ell^{-1} \oplus a \cdot q_1$ in Equation (6.4) for $hk_\ell = E_{K_\ell}(0^{128})$ gives $a = b \cdot hk_\ell$. Thus,

$$Q'(x) = b \cdot (x + hk_\ell) \cdot x \cdot \prod_{hk \in K^*_H} (x \oplus hk) = b \cdot x \cdot \prod_{hk \in K^*_H \cup \{hk_\ell\}} (x \oplus hk)\,.$$

The ciphertext forgery $\hat{c}$ is a valid forgery if $k \in K^* \cup K_\ell \cup \{0\}$. Thus, we are in effect able to test target key sets of size $|K^*| + 1 = \ell$.

Performance. The attack description above is for a fixed set of target keys K^*; in practice, an attacker would prepare a collection of ciphertext masks corresponding to disjoint target key sets $\{K_i^*\}_{i \in I}$, such that $p_{i+1} \geq p_i$ for all i, where p_i denotes the aggregate success probability of target key set K_i^*. Given $\lambda = |K^*|$ hashing keys, the coefficients of the polynomial $Q(x)$ can be computed using $\mathcal{O}(\lambda^2)$ time and $\mathcal{O}(\lambda)$ space. We note that the offline phase need only occur once, allowing the adversary to amortise the upfront cost of pre-computation over multiple targets. This is especially useful in cases where generating target keys from passwords is particularly slow.

In the online phase, splitting ciphertexts are then submitted in order until a query is successful; we note that a negative result is returned immediately. For a successful query, we know that the key $k \in K_i^*$ for some particular i. As our result relies on pre-computation to be practical, in order to perform a binary search on K_i^* appropriate forgery masks would have to be pre-computed – this would require $\mathcal{O}(\lambda \log \lambda)$ space. In most cases it is probably more efficient, once an adversary knows that $k \in K_i^*$, to perform the first few iterations of a binary search (having precomputed the necessary values) before switching to trial decryption of C with each key in K_i^*. We assume that the cost of querying a ciphertext is low and that either (1) there is a steady supply of ciphertexts to intercept or (2) it is possible to reuse the same nonce – the server may or may not enforce unique nonces depending on context. Regarding point (1), we note that a common adversarial model introduced by the BEAST attack [58] gives an attacker the ability to inject arbitrary plaintexts via client-side JavaScript in some window in the user's browser (see e. g., [6, 5, 32]).

Our attack is limited to scenarios where keys are deterministically derived from passwords; that is, if passwords are salted (using randomly generated salts) then pre-computation is no longer feasible. This highlights the fact that whilst salts are not secret values, they should be unpredictable when used to derive encryption keys

from passwords, in a direct analogue to password storage. Better security in any case is obtained by using password authenticated key exchange protocols such as [81], rather than deriving session keys statically from passwords.

6.4.2 Targeted Key Contingent Forgery Passing Format Checks

The targeted multi-key contingent forgery attack from the previous section results in ciphertexts that decrypt under the user's key to plaintexts that are "garbage". This is a problem in cases where plaintexts are required to meet some format check. The most common form of format check will be a header field containing (for example) protocol data, sender and receiver addresses, serial numbers or integrity check values. The weak key forgery method of [100] allows full control over the underlying plaintext, with the caveat that the ciphertext forgery represents an (untargeted) multi-key contingent forgery – for every block of underlying plaintext that is part of the format check, the number of *targeted* keys being tested will decrease by one, with one extra untargeted key gained. In practice this will not make much difference: usually, the prefix is designed to be as short as possible, which means one or at most two blocks. We would typically expect splitting ciphertexts of degree $\approx$500 so that losing one or two blocks represents only a small fraction of the total.

Let us assume that the captured nonce, ciphertext, tag tuple (n, C, τ) corresponds to some underlying plaintext matching the (known) required format. For concreteness, assume that the first block of plaintext (respectively, ciphertext) corresponds to the format to be checked. This means that we need to leave the first block of plaintext unchanged. We thus set $\beta = C_1 \oplus 0$ in Equation (6.4) and note that the method may easily be adapted to "flip bits" in the underlying plaintext by using a suitable value of $\beta = C_1 \oplus \delta$; furthermore, it is straightforward to extend the method to deal with multiple blocks. By construction, $\hat{c}_1 = q'_1 = \beta$, which gives $\hat{c} = C_1 \parallel \hat{c}_2 \parallel \cdots \parallel \hat{c}_\ell$, i. e.,

a ciphertext forgery $\hat{c}$ with the same first block of ciphertext (and thus underlying plaintext) as the original intercepted ciphertext C. Note that we gain $k_\ell = a \cdot b^{-1}$ as an untargeted key.

Len et al. [89] show how to craft (untargeted) multi-key collisions to pass format checks with fixed prefixes, however their method is impractical for prefixes longer than a couple of bytes; in contrast, our method can easily be applied to arbitary prefixes and is targeted. Lastly, we observe that this method circumvents the key committing "padding fix" discussed in Section 6.2.4, i. e., to prepend a constant string to messages before encrypting. The ability to control underlying plaintexts in this way allows an attacker to apply partitioning oracle attacks using weak key forgeries where attacks based on exploiting non-committment are infeasible.

6.5 Partitioning Oracle Attacks against Shadowsocks

Originally written by a pseudonymous developer, Shadowsocks [109] is an encrypted proxy for TCP and UDP traffic, based on SOCKS5. Shadowsocks was first built to help evade censorship in China, and it underlies other tools such as Jigsaw's Outline VPN. To use Shadowsocks, a user first deploys the Shadowsocks proxy server on a remote machine, provisions it with a static password and chooses an encryption scheme to use for all connections. The most up-to-date implementations only support AEAD schemes for encryption, with the options consisting of AES-GCM (128-bit or 256-bit) or ChaCha20/Poly1305. Next the user configures the Shadowsocks client on their local machine, and can then forward TCP or UDP traffic from their machine to the Shadowsocks proxy server.

Len et al. [89] showed how to build a practical partitioning oracle attack against Shadowsocks proxy servers. At a high level, their attack exploited the non-key com-

mitting property of the AEAD schemes used, making it possible to craft ciphertexts which decrypt correctly under a set of target keys. Furthermore, the attack exploits the fact that the proxy server opens an ephemeral UDP port in response to a valid request (and otherwise does not) which reveals whether a ciphertext has been accepted or rejected. The attack depends on a particular configuration: password derived keys and UDP traffic. As a response to [89], users are advised to mitigate against the attack by generating good quality passwords and disabling UDP mode [7]. In this section, we first describe the Shadowsocks protocol and the partitioning oracle attack of Len et al. before going on to describe how weak key forgeries can be used to launch a partitioning oracle attack. We note that whilst our attack is rendered impractical by the per-message salt used in the Shadowsocks protocol, a description of a hypothetical attack still offers a useful case study, which we describe below.

The Shadowsocks Protocol. The client starts by hashing the user password pw to obtain a key $k = H(pw)$. The client then samples a random sixteen-byte salt s and computes a session key $k_s \leftarrow \text{HKDF}(k, s, \mathit{info})$ using HKDF [86], where info is the string `ss-subkey`. A new salt and session key are generated for every message. The client encrypts its plaintext payload p by computing $C \leftarrow \mathsf{AEAD.enc}(k_s, Z, \varepsilon, \mathit{flag} \parallel \mathit{ip} \parallel \mathit{port} \parallel \mathit{payload})$ where Z denotes a nonce set to a string of zero bytes (12 for AES-GCM); the value ε empty associated data; and flag is a one-byte header indicating the format of ip with the following convention: $\mathit{flag} = 01$ indicates that ip is a 4-byte IPv4 address, $\mathit{flag} = 03$ indicates that ip consists of a one byte length and then hostname, and $\mathit{flag} = 04$ indicates that ip is a 16-byte IPv6 address. The port field port is two bytes long. The client sends (s, C) to the server via UDP. If the client is using TCP, the process is the same except that the ciphertext is prefixed with a two-byte encrypted length (and authentication tag) before being sent to the server via TCP.

When the Shadowsocks server receives (s, C), it extracts the salt and uses it together with pw to re-derive the session key k_s. It decrypts the remainder of the ciphertext with k_s. If decryption fails, no error message is sent back to the client. If decryption succeeds, the plaintext's format is checked by verifying that its first byte is equal to a valid *flag* value. If that check passes, the next bytes are interpreted as an appropriately encoded address *ip*, and two-byte port number *port*. Finally, the rest of the payload is sent to the remote server identified by *ip* and *port*. The proxy then listens on an ephemeral source UDP port assigned by the kernel networking stack for a reply from the remote server. When Shadowsocks receives a reply on the ephemeral port, the server generates a random salt and uses it with pw to generate a new session key. It then encrypts the response, and sends the resulting salt and ciphertext back to the client. The same encryption algorithm is used in both directions.

The Attack of Len et al. The proxy server opens an ephemeral UDP port in repsonse to a valid request (and otherwise not). One can view this as a remotely observable logical side-channel that reveals whether decryption succeeds. The attacker starts with knowledge of a password dictionary $\mathcal{D}$ and an estimate $\hat{p}$ of the probability distribution over keys in the dictionary. The attack has two steps, a pre-computation phase and an active querying phase.

In the pre-computation phase, the attacker chooses an arbitrary salt s and derives a set of session keys $K = K(\mathcal{D})$ by $k_s^i \leftarrow \text{HKDF}(H(pw_i), s, \texttt{ss-subkey})$ for all $pw_i \in \mathcal{D}$; the nonce is set as a string of all zeroes. The adversary then outputs a ciphertext $\hat{c}$ of length 4093 (to meet the length restriction imposed by Shadowsocks servers) and a set K^* of 4091 keys such that $\hat{c}$ decrypts under every key in S to give a plaintext with first byte 01. We gloss over the details of how $\hat{c}$ is constructed and refer the reader to [89]; we note that the construction is not a targeted multi-key collision.

In the querying phase, the attacker then submits (s^*, C^*) to the proxy server. Should the user's key be in the set of target keys, $k(pw) \in K^*$, the server will interpret the decrypted plaintext as a 01 byte followed by a random IPv4 address, destination port, and payload. The IPv4 and destination port will be accepted by the server's network protocol stack with high probability, and so the server will send the payload as a UDP packet and open a UDP source port to listen for a response, which the attacker can observe by port scanning.

We note that this assumes an attacker will be able to conduct arbitrary port scanning reliably, which in practice is unlikely to be the case. Strict firewall rules, or other defences such as software solutions, can disrupt or prevent port scanning. Our attack, outlined in Section 6.5.1, does not require port scanning but relies on the trivial assumption that IP addresses for receivers may be derived from packet captures.

6.5.1 Our Attack: Partitioning Oracles from Weak Key Forgeries

We now describe how to launch a partitioning oracle attack using weak key forgeries against Shadowsocks (in the same configuration as the attack of Len et al. described above). As noted above, our attack is impractical as session keys are salted on a per-message basis in the Shadowsocks protocol, making pre-computation of forgery masks infeasible. Nevertheless, a weak key forgery partitioning oracle attack against Shadowsocks is an instructive case study, demonstrating the feasibility of the approach and allowing us to point out some interesting features; in particular, we are able to construct targeted multi-key contingent forgeries that meet arbitrary format requirements as we explain below.

Basic Version. We separate the attack into two steps, a computation phase and

an active querying phase. The attacker starts with knowledge of a password dictionary $\mathcal{D}$ and an estimate $\hat{p}$ of the probability distribution over keys in the dictionary and then intercepts a salt, ciphertext tuple (s, C).

In the computation phase, the attacker first chooses a set of passwords $\mathcal{D}^*$ with $|\mathcal{D}^*| = 4092$, such that the set has the maximum aggregate probability according to $\hat{p}$. The attacker then derives a set of session keys K^* from the salt s and set of passwords $\mathcal{D}^*$ by $k_s^i \leftarrow \text{HKDF}(H(pw_i), s, \texttt{ss-subkey})$; the nonce is set as a string of all zeroes. Using the weak key forgery method described in Section 6.4.2, the attacker outputs a ciphertext $\hat{c}$ of length 4093 (to meet the length restriction imposed by Shadowsocks servers) such that $\hat{c}$ decrypts under the users key k if $k \in K^*$. Furthermore, the underlying plaintext $p \leftarrow \mathsf{dec}(k, \hat{c})$ passes the format check.

In the querying phase, the attacker then submits (s, C^*) to the proxy server. Should the user's key be in the set of target keys, the server will interpret the decrypted plaintext as *flag* $\|$ *ip* $\|$ *port* $\|$ *payload*; that is, an IP address, destination port and payload. Note that these are unchanged from the original plaintext that was sent by the user, so will be accepted by the server's network protocol stack. The server will send the payload as a UDP packet and open a UDP source port to listen for a response, which the attacker can observe by port scanning.

Extension 1: Redirection (Known Plaintext Attack). If the attacker knows the first 7 bytes of an underlying plaintext, which we write as *prefix*, then they can use the weak key forgery technique to redirect the user's payload to arbitrary destinations. In particular, the first 7 bytes can be modified to give 01 $\|$ *ip*$'$ $\|$ *port*$'$, with *ip*$'$ a four-byte IPv4 address, and *port*$'$ a two-byte destination port. This is the idea behind Peng's "redirect attack" [64, 97], discovered in February 2020, which exploited the use of stream ciphers without integrity protection in the Shadowsocks protocol. Obtaining plaintexts with known *prefix* is relatively easy in the server to

client direction, as many common server protocols start with the same bytes (e. g., `HTTP/1.` for HTTP). In the client to server direction, underlying plaintexts will be in the format [destination][payload], so that the adversary needs to know the target address (and its encoding), perhaps through injecting plaintexts via client-side JavaScript in some window in the user's browser [6, 5, 32, 58]. Note that if an adversary is able to launch chosen plaintext attacks, they could target the TCP configuration of Shadowsocks (the recommended option) by crafting plaintexts with the maximum length to overcome the fact that for TCP the length is sent encrypted together with the encrypted payload.

The adversary intercepts a ciphertext C from server to client, and using weak key forgery techniques modifies C to give a splitting ciphertext $\hat{c}$ whose underlying plaintext begins with $\mathit{prefix}' = 01 \parallel \mathit{ip}' \parallel \mathit{port}'$, i. e., an address that the adversary controls. The splitting ciphertext is then sent to the Shadowsocks server: if the splitting ciphertext is accepted, the payload is sent to the adversary, revealing that the user's key is in the set of target keys associated to $\hat{c}$. To produce $\hat{c}$, we modify the basic attack above as follows: when it comes to constructing the weak key forgery mask, following the technique outlined in Section 6.4.2, we use a non-zero value of β in Equation (6.4); specifically, $\beta = \mathit{prefix} \oplus (01 \parallel \mathit{ip}' \parallel \mathit{port}')$, interpreted as an element of $\mathbb{F}_{2^{128}}$. The effect is to flip some bits in the 7-byte prefix *prefix*, so that we obtain the attacker's address prefix'.

We note that this attack allows the adversary to efficiently and reliably determine whether the ciphertext has been accepted; it is no longer necessary to scan the server for open ports, which is time consuming and not necessarily completely reliable. Furthermore, if the splitting ciphertext is accepted, the adversary receives the payload *payload* which means that it can efficiently test target keys against the ciphertext by encrypting one block of plaintext and checking whether it matches. Without this, the adversary would need to calculate the authentication tag of the

captured ciphertext for each target key.

Extension 2: Bypassing the Padding Fix. As discussed in Section 6.2.4, prior work on non-key committing AEAD schemes showed that applying a "padding fix", that is prepending a fixed constant string to underlying plaintexts, transforms the scheme to be key-committing. Applying a padding fix is recommended by Len et al. as a way to mitigate against partitioning oracle attacks; however, a partitioning oracle attack using weak key forgeries will still be successful despite that mitigation. To see this, we simply modify the description of the "basic attack version" in the previous subsection to leave one further block unaltered, at the cost of testing one less key per ciphertext $\hat{c}$. We note that the reason our attack is impractical is due to the salting of passwords to derive per-message ephemeral keys, rather than because of the non-key committing property of the AEAD scheme used.

6.5.2 Other Proxy Servers (VPNs)

Virtual Private Networks (VPNs) are often used to achieve similar objectives to Shadowsocks (allowing a user to access the internet via a proxy server), although Shadowsocks was designed specifically to circumvent internet censorship, which is not part of the threat model for VPNs. VPNs allow users to interact with what appears to be a private network, despite the interaction taking place over a public network (typically, the internet). This is achieved by encrypting packets in transit so that the contents are hidden from the public network. VPNs have a number of applications, including enabling users to remotely access local resources, or allowing individuals to improve their anonymity and privacy online (by masking their IP and hiding their traffic). Users connect to a proxy server via an encrypted tunnel, and the proxy server acts as an intermediary for the client and the internet (or a portion thereof). The most widely used protocols for VPNs are TLS and the IPsec protocol

At a high level, IPsec works as follows: the user first composes a TLS packet that will be sent to the end destination. This is encapsulated in an IPsec Encapsulating Security Payload (ESP) packet in tunnelling mode, which essentially adds a header and encrypts the whole packet to give a ciphertext C. This encrypted packet C is sent to the proxy server, where it is decrypted to recover the underlying TLS packet. The proxy server now forwards the TLS packet to its intended destination. There are many configuration options for how the user and proxy server authenticate and/or encrypt the ESP packets, including to provision the user and proxy server with static keys [84]. This is known as "manual management", and is suited to small static environments. However, the standard does not allow AES-GCM (or ChaCha20-Poly1305) with manual keys, although they are available in other configurations, due to concerns over the brittleness when a nonce/key combination is reused. AES with HMAC is preferred, which happens to be both key-committing and not vulnerable to weak key forgeries. Similarly, OpenVPN disallows AEAD cipher mode with static keys to avoid the insecurity of potential nonce/key reuse. We have thus not been able to find any vulnerable applications "in the wild", but note that partitioning oracle attacks are theoretically possible against implementations incorrectly deviating from the specification. Following on from the discussion of ASAs in Chapter 3, an attractive avenue for a malicious adversary would be to craft an implementation that facilitates partition oracle attacks. This task is eased by the fact that users typically download an implementation from code sharing sites, where there are multiple implementations (in different programming languages, targeted at different operating systems) to choose from.

6.6 Conclusions

In this chapter we considered an adversary that is willing to engage in active attacks against users to block access to the wider internet and potentially de-anonymise

users. Censoring the internet can be considered a complementary approach to conducting mass surveillance (the type of adversary considered in Chapter 5); de-anonymising users is a related notion to undermining deniability of communication (following the adversarial goals in Chapter 4). This chapter thus serves to broaden the model of the motivations and approach of powerful adversaries, and continues our investigation of cryptographic attacks available to such an adversary.

The main technical contribution of this chapter was to demonstrate a new class of attack that extends prior work and to extend the definition of partitioning oracle attacks. Prior work demonstrated that key commitment is an important security property of AEAD schemes. Our results suggest that resistance to weak key forgeries should be considered a related design goal to key-commitment, particularly in settings that are vulnerable to partitioning oracle attacks. Concretely, our results demonstrate – in contrast to the suggestions of prior work – that structured underlying plaintexts (e.g. packet headers that prefix every plaintext message, or an appended block of all zeros) is not a sufficient mitigation against partitioning oracle attacks.

Lastly, our discussion of partitioning oracles demonstrated that decryption oracles are a practical concern in the sense that adversaries are able to determine whether or not a ciphertext is accepted or not by observing some phenomenon. Our attacks in Chapter 5 rely on the assumption that an adversary has access to a decryption oracle. This chapter supports our assumption that an adversary has access to decryption oracles.

Chapter 7

Conclusion

In this thesis, our motivation was to examine some aspects of security in the face of adversaries who are powerful and determined. The archetype for a powerful adversary is a nation state actor that has access to resources (in terms of funding, computation, expertise and political capital) of a magnitude greater than even well-organised, well-funded and technically proficient "professional" adversaries. The attacks we considered in this work arise from a consideration of motives particular to such powerful adversaries (modelling the interests and capabilities of nation state agencies): firstly, to conduct mass surveillance against populations; and secondly, to restrict access to the wider internet. Chapter 3 expanded the definitions modelling mass surveillance adversaries, and Chapter 5 introduced a new class of practical attack enabled by the expanded definition. Chapter 4 considered a novel setting of interest to powerful adversaries: deniable encryption. Lastly, Chapter 6 focussed on a different but related setting, censoring users' internet access.

In the rest of this chapter, we will give a more detailed summary of the contributions in Section 7.1, and finish with some open questions in Section 7.2.

7.1 Contributions

Chapter 3 introduced Algorithm Substitution Attacks. After discussing related literature, we provided formal definitions for subversion attacks against generic cryptographic primitives consisting of a sender and receiver – including notions of undetectability and adversarial goals. Our syntax allows for both the sending and receiving party to be subverted, in contrast to previous work in this area which con-

sidered only subversion of the sender. As such, our definitions are a subtle extension of prior work which allow for a class of attack that was previously missed. We discuss a generic subversion method (rejection sampling) and show that our syntax applies to the primitives considered in this work: symmetric encryption, MACs and PKE schemes. By extending the definitions to capture a broader class of attack, we add to the understanding of the adversarial model.

Chapter 4 served as an introduction demonstrating what is possible with ASAs and illustrating the techniques and framework. We considered subversion of DPKE schemes, a primitive that allows the sender of an encrypted plaintext message to later claim that a different plaintext was sent. This chapter, beyond providing an introduction to the power of ASAs, demonstrates that subversion attacks should be considered part of the adversarial model for deniability and should be considered in the design of deniable schemes. Our subversion attacks against deniable encryption are generic, and beyond providing an introductory case study to the later attacks in Chapter 5, are designed to serve as a proof of concept demonstrating the feasibility of such an attack. The current absence of practical deniable encryption schemes means that our attacks are theoretical at present. An interesting open problem is to consider non-generic techniques to subvert deniable encryption.

Chapter 5 introduced our subversion attacks targeting the receiver. Our subversion attack allows an adversary to learn the user's secret key by observing their communication; once the adversary has learnt the key, the user's cryptography is completely undermined. We consider a passive adversary, following prior work which considers a mass surveillance adversary to be engaged in eavesdropping on a huge scale, but we also consider an active variant. The active variant allows an adversary to target users far more effectively, which makes the attack attractive from the point of view of an adversary. Our attacks work by altering the behaviour of the receiver's algorithm to leak information through (artificially induced) decryption error events – the subverted algorithm either rejects (particular, "trigger") valid ciphertexts or accepts (particular, "trigger") bogus ciphertexts. An adversary observing the receiver who

is able to determine whether a ciphertext has been accepted or rejected learns some information; this subliminal channel can be used to exfiltrate the user's key. Our attacks are highly practical and, as we discuss in Section 5.5 in particular, can be mounted using a combination of logical and physical techniques.

Chapter 6 continued the theme of powerful adversaries undermining privacy of users. Whereas earlier chapters focussed on powerful adversaries who are motivated to carry out mass surveillance, this chapter considers the related aim of targeting users attempting to bypass internet censorship. We consider the specific scenario of users accessing the wider, free, internet via a proxy server. Our focus is a particular class of attack that allows an adversary to test whether a ciphertext was encrypted using any of a particular set of encryption keys using only one decryption query. Such an attack, known as a partitioning oracle attack, allows for a more efficient brute force search of the key space than querying one key per ciphertext, leading to practical attacks against low entropy keys (e. g., those derived from passwords). Our main contribution is to show that weak key forgeries against polynomial hash-based AE schemes can be leveraged to launch partitioning oracle attacks. We discuss the applications to proxy servers and other practical scenarios. Lastly, our attacks in Chapter 5 relied on the assumption that an adversary has access to a decryption oracle – that is, can observe whether a receiver's algorithm implementation accepts or rejects a ciphertext. In this chapter, we give an illustration of how decryption oracles occur in practice, supporting the practicality of our attacks in Chapter 5.

7.2 Further Work

It is an open question to consider other primitives that may be vulnerable to receiver subversion, following the notions introduced in Chapter 3 and Chapter 5.

An interesting open problem, following on from Chapter 4, is to consider subversion attacks against deniable protocols, for example the Signal protocol [112] or deniable key agreement. At present, subverting protocols has not received a great deal of

attention other than recent work by Berndt et al. [28], who considered subverting protocols including Signal with the aim of leaking a secret key. Another line of enquiry is to consider subverting protocols in general: Is it sufficient to subvert the implementation of constituent primitives, or is it necessary to subvert the implementation of the protocol as well? An example suggested by Chapter 6 is to consider subverting an implementation to make a vulnerability exploitable.

In the discussion at Section 5.2.3, we give an ad hoc description of ways in which our subversion attacks may be made more efficient in practice by leaking information that allows an adversary to reconstruct secret keys. In general, this is an interesting open question: in practice, what is the minimum amount of information required to "break" security? And could this be minimised further using subversion?

Bibliography

[1] Mohamed Ahmed Abdelraheem et al. “Twisted Polynomials and Forgery Attacks on GCM”. In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2015, pp. 762–786. DOI: 10.1007/978-3-662-46800-5_29 (cit. on p. 125).

[2] Shweta Agrawal, Shafi Goldwasser, and Saleet Mossel. “Deniable Fully Homomorphic Encryption from Learning with Errors”. In: *Advances in Cryptology – CRYPTO 2021, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 641–670. DOI: 10.1007/978-3-030-84245-1_22 (cit. on p. 74).

[3] Ange Albertini et al. *How to Abuse and Fix Authenticated Encryption Without Key Commitment*. Cryptology ePrint Archive, Report 2020/1456. https://eprint.iacr.org/2020/1456. 2020 (cit. on pp. 116, 123, 124).

[4] Martin R. Albrecht and Kenneth G. Paterson. “Lucky Microseconds: A Timing Attack on Amazon’s s2n Implementation of TLS”. In: *Advances in Cryptology – EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. Lecture Notes in Computer Science. Springer, Heidelberg, May 2016, pp. 622–643. DOI: 10.1007/978-3-662-49890-3_24 (cit. on p. 128).

[5] Nadhem J. AlFardan and Kenneth G. Paterson. “Lucky Thirteen: Breaking the TLS and DTLS Record Protocols”. In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 526–540. DOI: 10.1109/SP.2013.42 (cit. on pp. 128, 134, 141).

[6] Nadhem J. AlFardan et al. “On the Security of RC4 in TLS”. In: *USENIX Security 2013: 22nd USENIX Security Symposium*. Ed. by Samuel T. King. USENIX Association, Aug. 2013, pp. 305–320 (cit. on pp. 134, 141).

[7] Anonymous et al. *A Practical Guide to Defend Against the GFW’s Latest Active Probing*. https://gfw.report/blog/ss_advise/en/. Retrieved May 2021. 2021 (cit. on p. 137).

[8] Marcel Armour and Carlos Cid. “Partition Oracles from Weak Key Forgeries”. In: *Cryptology and Network Security*. Ed. by Mauro Conti, Marc Stevens, and Stephan Krenn. Cham: Springer International Publishing, 2021, pp. 42–62. ISBN: 978-3-030-92548-2 (cit. on pp. 22, 118).

[9] Marcel Armour and Bertram Poettering. "Substitution Attacks against Message Authentication". In: *IACR Transactions on Symmetric Cryptology* 2019.3 (2019), pp. 152–168. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i3.152-168 (cit. on p. 22).

[10] Marcel Armour and Bertram Poettering. "Subverting Decryption in AEAD". In: *17th IMA International Conference on Cryptography and Coding*. Ed. by Martin Albrecht. Vol. 11929. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2019, pp. 22–41. DOI: 10.1007/978-3-030-35199-1_2 (cit. on pp. 22, 84).

[11] Marcel Armour and Elizabeth A. Quaglia. "Subverting Deniability". In: *Provable and Practical Security*. Ed. by Chunpeng Ge and Fuchun Guo. Cham: Springer Nature Switzerland, 2022, pp. 52–59. ISBN: 978-3-031-20917-8 (cit. on pp. 22, 68).

[12] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. "Subversion-Resilient Signature Schemes". In: *ACM CCS 2015: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 364–375. DOI: 10.1145/2810103.2813635 (cit. on p. 62).

[13] Nimrod Aviram et al. *Practical (Post-Quantum) Key Combiners from One-Wayness and Applications to TLS*. Cryptology ePrint Archive, Report 2022/065. https://eprint.iacr.org/2022/065. 2022 (cit. on p. 64).

[14] James Ball, Julian Borger, and Glenn Greenwald. "Revealed: How US and UK Spy Agencies Defeat Internet Privacy and Security". In: *The Guardian* (2013). URL: https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security (visited on 09/05/2013) (cit. on p. 17).

[15] Elaine Barker. "Nist special publication 800-57 part 1, revision 5". In: *Recommendation for Key Management* (2020). DOI: 10.6028/NIST.SP.800-57pt1r5 (cit. on p. 88).

[16] Balthazar Bauer, Pooya Farshim, and Sogol Mazaheri. "Combiners for Backdoored Random Oracles". In: *Advances in Cryptology – CRYPTO 2018, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2018, pp. 272–302. DOI: 10.1007/978-3-319-96881-0_10 (cit. on pp. 64, 109).

[17] Mihir Bellare. "New Proofs for NMAC and HMAC: Security without Collision-Resistance". In: *Advances in Cryptology – CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2006, pp. 602–619. DOI: 10.1007/11818175_36 (cit. on p. 71).

[18] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Keying Hash Functions for Message Authentication". In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1996, pp. 1–15. DOI: 10.1007/3-540-68697-5_1 (cit. on p. 123).

[19] Mihir Bellare and Viet Tung Hoang. "Resisting Randomness Subversion: Fast Deterministic and Hedged Public-Key Encryption in the Standard Model". In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2015, pp. 627–656. DOI: 10.1007/978-3-662-46803-6_21 (cit. on pp. 63, 80).

[20] Mihir Bellare, Joseph Jaeger, and Daniel Kane. "Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks". In: *ACM CCS 2015: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 1431–1440. DOI: 10.1145/2810103.2813681 (cit. on pp. 44, 51, 57, 60, 71, 85, 110).

[21] Mihir Bellare, Daniel Kane, and Phillip Rogaway. "Big-Key Symmetric Encryption: Resisting Key Exfiltration". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 373–402. DOI: 10.1007/978-3-662-53018-4_14 (cit. on pp. 63, 89).

[22] Mihir Bellare and Adriana Palacio. "Towards Plaintext-Aware Public-Key Encryption without Random Oracles". In: *Advances in Cryptology – ASIACRYPT 2004*. Ed. by Pil Joong Lee. Vol. 3329. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2004, pp. 48–62. DOI: 10.1007/978-3-540-30539-2_4 (cit. on pp. 101, 169).

[23] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. "Security of Symmetric Encryption against Mass Surveillance". In: *Advances in Cryptology – CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2014, pp. 1–19. DOI: 10.1007/978-3-662-44371-2_1 (cit. on pp. 44, 45, 47, 57, 58, 60, 69, 71, 110).

[24] Mihir Bellare and Phillip Rogaway. "Optimal Asymmetric Encryption". In: *Advances in Cryptology – EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Springer, Heidelberg, May 1995, pp. 92–111. DOI: 10.1007/BFb0053428 (cit. on pp. 168, 169).

[25] Steven M. Bellovin and Michael Merritt. "Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks". In: *1992 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1992, pp. 72–84. DOI: 10.1109/RISP.1992.213269 (cit. on p. 128).

[26] Pascal Bemmann, Rongmao Chen, and Tibor Jager. "Subversion-Resilient Public Key Encryption with Practical Watchdogs". In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Juan Garay. Vol. 12710. Lecture Notes in Computer Science. Springer, Heidelberg, May 2021, pp. 627–658. DOI: 10.1007/978-3-030-75245-3_23 (cit. on pp. 64, 110).

[27] Sebastian Berndt and Maciej Liskiewicz. "Algorithm Substitution Attacks from a Steganographic Perspective". In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham et al. ACM Press, Oct. 2017, pp. 1649–1660. DOI: 10.1145/3133956.3133981 (cit. on pp. 56, 62).

[28] Sebastian Berndt et al. *ASAP: Algorithm Substitution Attacks on Cryptographic Protocols*. Cryptology ePrint Archive, Report 2020/1452. https://eprint.iacr.org/2020/1452. 2020 (cit. on pp. 62, 148).

[29] Daniel J Bernstein. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 2008, pp. 3–5 (cit. on p. 121).

[30] Daniel J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *Fast Software Encryption – FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2005, pp. 32–49. DOI: 10.1007/11502760_3 (cit. on pp. 119–121).

[31] Jan Beznazwy and Amir Houmansadr. "How China Detects and Blocks Shadowsocks". In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 111–124 (cit. on pp. 117, 127).

[32] Karthikeyan Bhargavan and Gaëtan Leurent. "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN". In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl et al. ACM Press, Oct. 2016, pp. 456–467. DOI: 10.1145/2976749.2978423 (cit. on pp. 134, 141).

[33] Swarup Bhunia et al. "Hardware Trojan Attacks: Threat Analysis and Countermeasures". In: *Proceedings of the IEEE* 102.8 (2014), pp. 1229–1247 (cit. on pp. 52–54).

[34] James Birkett and Alexander W. Dent. "Relations Among Notions of Plaintext Awareness". In: *PKC 2008: 11th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Ronald Cramer. Vol. 4939. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2008, pp. 47–64. DOI: 10.1007/978-3-540-78440-1_4 (cit. on p. 169).

[35] Daniel Bleichenbacher. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1". In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1998, pp. 1–12. DOI: 10.1007/BFb0055716 (cit. on p. 128).

[36] Alexandra Boldyreva et al. "On Symmetric Encryption with Distinguishable Decryption Failures". In: *Fast Software Encryption – FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2014, pp. 367–390. DOI: 10.1007/978-3-662-43933-3_19 (cit. on p. 129).

[37] Angèle Bossuat et al. “Designing Reverse Firewalls for the Real World”. In: *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part I*. Ed. by Liqun Chen et al. Vol. 12308. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2020, pp. 193–213. DOI: 10.1007/978-3-030-58951-6_10 (cit. on pp. 63, 110).

[38] Jan Camenisch, Manu Drijvers, and Anja Lehmann. “Anonymous Attestation with Subverted TPMs”. In: *Advances in Cryptology – CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2017, pp. 427–461. DOI: 10.1007/978-3-319-63697-9_15 (cit. on p. 62).

[39] Ran Canetti and Rosario Gennaro. “Incoercible Multiparty Computation (extended abstract)”. In: *37th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Oct. 1996, pp. 504–513. DOI: 10.1109/SFCS.1996.548509 (cit. on p. 66).

[40] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. “Fully Deniable Interactive Encryption”. In: *Advances in Cryptology – CRYPTO 2020, Part I*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12170. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2020, pp. 807–835. DOI: 10.1007/978-3-030-56784-2_27 (cit. on p. 73).

[41] Ran Canetti et al. *Deniable Encryption*. Cryptology ePrint Archive, Report 1996/002. https://eprint.iacr.org/1996/002. 1996 (cit. on p. 76).

[42] Ran Canetti et al. “Deniable Encryption”. In: *Advances in Cryptology – CRYPTO’97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1997, pp. 90–104. DOI: 10.1007/BFb0052229 (cit. on pp. 67, 73, 75, 76).

[43] Sofía Celi and Iraklis Symeonidis. “The Current State of Denial”. In: *PETS. HotPETS*. 2020 (cit. on p. 81).

[44] Stephen Checkoway et al. “On the Practical Exploitability of Dual EC in TLS Implementations”. In: *USENIX Security 2014: 23rd USENIX Security Symposium*. Ed. by Kevin Fu and Jaeyeon Jung. USENIX Association, Aug. 2014, pp. 319–335 (cit. on p. 18).

[45] Rongmao Chen, Xinyi Huang, and Moti Yung. “Subvert KEM to Break DEM: Practical Algorithm-Substitution Attacks on Public-Key Encryption”. In: *Advances in Cryptology – ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2020, pp. 98–128. DOI: 10.1007/978-3-030-64834-3_4 (cit. on pp. 57, 62).

[46] Don Coppersmith. “Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known”. In: *Advances in Cryptology – EUROCRYPT’96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Springer, Heidelberg, May 1996, pp. 178–189. DOI: 10.1007/3-540-68339-9_16 (cit. on pp. 61, 88).

[47] Ronald Cramer and Victor Shoup. "A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack". In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1998, pp. 13–25. DOI: 10.1007/BFb0055717 (cit. on p. 169).

[48] Ronald Cramer and Victor Shoup. "Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack". In: *SIAM Journal on Computing* 33.1 (2003), pp. 167–226 (cit. on p. 168).

[49] Claude Crépeau and Alain Slakmon. "Simple Backdoors for RSA Key Generation". In: *Topics in Cryptology – CT-RSA 2003*. Ed. by Marc Joye. Vol. 2612. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2003, pp. 403–416. DOI: 10.1007/3-540-36563-X_28 (cit. on p. 61).

[50] Angelo De Caro, Vincenzo Iovino, and Adam O'Neill. "Deniable Functional Encryption". In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Chen-Mou Cheng et al. Vol. 9614. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2016, pp. 196–222. DOI: 10.1007/978-3-662-49384-7_8 (cit. on p. 73).

[51] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. "A More Cautious Approach to Security Against Mass Surveillance". In: *Fast Software Encryption – FSE 2015*. Ed. by Gregor Leander. Vol. 9054. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 579–598. DOI: 10.1007/978-3-662-48116-5_28 (cit. on pp. 44, 45, 47, 60, 110).

[52] Jean Paul Degabriele et al. "Backdoors in Pseudorandom Number Generators: Possibility and Impossibility Results". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 403–432. DOI: 10.1007/978-3-662-53018-4_15 (cit. on p. 62).

[53] Alexander W. Dent. "The Cramer-Shoup Encryption Scheme Is Plaintext Aware in the Standard Model". In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, Heidelberg, May 2006, pp. 289–307. DOI: 10.1007/11761679_18 (cit. on p. 169).

[54] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. "Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 341–372. DOI: 10.1007/978-3-662-53018-4_13 (cit. on p. 63).

[55] Yevgeniy Dodis et al. "A Formal Treatment of Backdoored Pseudorandom Generators". In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2015, pp. 101–126. DOI: 10.1007/978-3-662-46800-5_5 (cit. on p. 62).

[56] Yevgeniy Dodis et al. "Fast Message Franking: From Invisible Salamanders to Encryptment". In: *Advances in Cryptology – CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2018, pp. 155–186. DOI: 10.1007/978-3-319-96884-1_6 (cit. on pp. 116, 123).

[57] Yevgeniy Dodis et al. "Towards Defeating Backdoored Random Oracles: Indifferentiability with Bounded Adaptivity". In: *TCC 2020: 18th Theory of Cryptography Conference, Part III*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12552. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 2020, pp. 241–273. DOI: 10.1007/978-3-030-64381-2_9 (cit. on pp. 64, 109).

[58] Thai Duong and Juliano Rizzo. *Here Come the ⊕ Ninjas*. Unpublished manuscript. Retrieved May 2021. URL: https://tlseminar.github.io/docs/beast.pdf (cit. on pp. 134, 141).

[59] Morris J Dworkin. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Tech. rep. 2007 (cit. on p. 126).

[60] Morris J. Dworkin. *SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. US National Institute of Standards and Technology. Gaithersburg, MD, United States, 2007. DOI: 10.6028/NIST.SP.800-38D (cit. on pp. 90, 106).

[61] Jason Dymydiuk. "RUBICON and Revelation: the Curious Robustness of the 'Secret' CIA-BND Operation with Crypto AG". In: *Intelligence and National Security* 35.5 (2020), pp. 641–658. DOI: 10.1080/02684527.2020.1774853. eprint: https://doi.org/10.1080/02684527.2020.1774853. URL: https://doi.org/10.1080/02684527.2020.1774853 (cit. on p. 18).

[62] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. "Private Circuits III: Hardware Trojan-Resilience via Testing Amplification". In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl et al. ACM Press, Oct. 2016, pp. 142–153. DOI: 10.1145/2976749.2978419 (cit. on p. 110).

[63] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. "Security of Symmetric Primitives under Incorrect Usage of Keys". In: *IACR Transactions on Symmetric Cryptology* 2017.1 (2017), pp. 449–473. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.449-473 (cit. on pp. 116, 123).

[64] David Fifield. *Decryption Vulnerability in Shadowsocks Stream Ciphers*. https://github.com/net4people/bbs/issues/24. Retrieved May 2021. (Visited on 02/25/2020) (cit. on p. 140).

[65] Marc Fischlin, Christian Janson, and Sogol Mazaheri. "Backdoored Hash Functions: Immunizing HMAC and HKDF". In: *CSF 2018: IEEE 31st Computer Security Foundations Symposium*. Ed. by Steve Chong and Stephanie Delaune. IEEE Computer Society Press, 2018, pp. 105–118. DOI: 10.1109/CSF.2018.00015 (cit. on pp. 64, 109).

[66] Marc Fischlin and Sogol Mazaheri. "Notions of Deniable Message Authentication". In: *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. WPES '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, 55?64. ISBN: 9781450338202. DOI: 10.1145/2808138.2808143. URL: https://doi.org/10.1145/2808138.2808143 (cit. on p. 81).

[67] Marc Fischlin and Sogol Mazaheri. "Self-Guarding Cryptographic Protocols against Algorithm Substitution Attacks". In: *CSF 2018: IEEE 31st Computer Security Foundations Symposium*. Ed. by Steve Chong and Stephanie Delaune. IEEE Computer Society Press, 2018, pp. 76–90. DOI: 10.1109/CSF.2018.00013 (cit. on p. 63).

[68] Christina Garman et al. "Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage". In: *USENIX Security 2016: 25th USENIX Security Symposium*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, Aug. 2016, pp. 655–672 (cit. on p. 128).

[69] Federico Giacon, Felix Heuer, and Bertram Poettering. "KEM Combiners". In: *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10769. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2018, pp. 190–218. DOI: 10.1007/978-3-319-76578-5_7 (cit. on p. 63).

[70] Eu-Jin Goh et al. "The Design and Implementation of Protocol-Based Hidden Key Recovery". In: *ISC 2003: 6th International Conference on Information Security*. Ed. by Colin Boyd and Wenbo Mao. Vol. 2851. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2003, pp. 165–179 (cit. on p. 62).

[71] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *14th Annual ACM Symposium on Theory of Computing*. ACM Press, May 1982, pp. 365–377. DOI: 10.1145/800070.802212 (cit. on p. 25).

[72] Dieter Gollmann. *Computer Security (3. ed.)* Wiley, 2011. ISBN: 978-0-470-74115-3. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118801326.html (cit. on pp. 52, 53, 71).

[73] Glen Greenwald et al. "Microsoft Handed the NSA Access to Encrypted Messages". In: *The Guardian* (2013). URL: https://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data (visited on 07/11/2013) (cit. on p. 17).

[74] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. "Message Franking via Committing Authenticated Encryption". In: *Advances in Cryptology – CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2017, pp. 66–97. DOI: 10.1007/978-3-319-63697-9_3 (cit. on pp. 116, 123).

[75] Helena Handschuh and Bart Preneel. "Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms". In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2008, pp. 144–161. DOI: 10.1007/978-3-540-85174-5_9 (cit. on pp. 117, 124).

[76] Philip Hodges and Douglas Stebila. "Algorithm Substitution Attacks: State Reset Detection and Asymmetric Modifications". In: *IACR Transactions on Symmetric Cryptology* 2021.2 (2021), pp. 389–422. ISSN: 2519-173X. DOI: 10.46586/tosc.v2021.i2.389-422 (cit. on p. 63).

[77] Nick Hopkins. "UK Gathering Secret Intelligence via Covert NSA Operation". In: *The Guardian* (2020). URL: https://www.theguardian.com/technology/2013/jun/07/uk-gathering-secret-intelligence-nsa-prism (visited on 06/07/2013) (cit. on p. 17).

[78] Akiko Inoue et al. "Cryptanalysis of OCB2: Attacks on Authenticity and Confidentiality". In: *Advances in Cryptology – CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2019, pp. 3–31. DOI: 10.1007/978-3-030-26948-7_1 (cit. on p. 91).

[79] Amnesty International. "Forensic Methodology Report: How to Catch NSO Group's Pegasus". In: (July 18, 2021). URL: https://www.amnesty.org/en/documents/doc10/4487/2021/en/ (cit. on p. 17).

[80] Richard Isaac. *The Pleasures of Probability*. Springer Science & Business Media, 2013 (cit. on p. 94).

[81] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks". In: *Advances in Cryptology – EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2018, pp. 456–486. DOI: 10.1007/978-3-319-78372-7_15 (cit. on p. 135).

[82] Antoine Joux. *Authentication Failures in NIST Version of GCM*. Tech. rep. 2006, p. 3 (cit. on p. 124).

[83] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC press, 2020 (cit. on p. 25).

[84] Stephen Kent and Karen Seo. *Security Architecture for the Internet Protocol*. RFC 4301. https://tools.ietf.org/html/rfc4301. RFC Editor, Dec. 2005, pp. 1–101 (cit. on p. 143).

[85] Neal Koblitz and Alfred Menezes. *Critical Perspectives on Provable Security: Fifteen Years of "Another Look" Papers*. Cryptology ePrint Archive, Report 2019/1336. https://eprint.iacr.org/2019/1336. 2019 (cit. on p. 26).

[86] Hugo Krawczyk. "Cryptographic Extraction and Key Derivation: The HKDF Scheme". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2010, pp. 631–648. DOI: 10.1007/978-3-642-14623-7_34 (cit. on p. 137).

[87] Hugo Krawczyk. *The Opaque Asymmetric PAKE Protocol (Draft)*. Tech. rep. https://datatracker.ietf.org/doc/html/draft-krawczyk-cfrg-opaque-02. 2018 (cit. on p. 123).

[88] Ted Krovetz and Phillip Rogaway. *The OCB Authenticated-Encryption Algorithm*. https://tools.ietf.org/html/rfc7253. 2014 (cit. on p. 90).

[89] Julia Len, Paul Grubbs, and Thomas Ristenpart. "Partitioning Oracle Attacks". In: *USENIX Security 2021: 30th USENIX Security Symposium*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, Aug. 2021, pp. 195–212 (cit. on pp. 4, 116, 118, 123, 125–129, 136–138).

[90] Hui Ma et al. "Concessive Online/Offline Attribute Based Encryption with Cryptographic Reverse Firewalls - Secure and Efficient Fine-Grained Access Control on Corrupted Machines". In: *ESORICS 2018: 23rd European Symposium on Research in Computer Security, Part II*. Ed. by Javier López, Jianying Zhou, and Miguel Soriano. Vol. 11099. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2018, pp. 507–526. DOI: 10.1007/978-3-319-98989-1_25 (cit. on p. 63).

[91] David McGrew and John Viega. *The Galois/Counter Mode of Operation (GCM)*. Tech. rep. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf. 2004, pp. 0278–0070 (cit. on pp. 119, 120).

[92] Greg Miller. "The Intelligence Coup of the Century". In: *Washington Post* 11 (2020). URL: https://www.washingtonpost.com/graphics/2020/world/national-security/cia-crypto-encryption-machines-espionage/ (visited on 02/11/2020) (cit. on p. 18).

[93] Ilya Mironov and Noah Stephens-Davidowitz. "Cryptographic Reverse Firewalls". In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2015, pp. 657–686. DOI: 10.1007/978-3-662-46803-6_22 (cit. on p. 63).

[94] Juan Manuel González Nieto et al. "Publicly Verifiable Ciphertexts". In: *J. Comput. Secur.* 21.5 (2013), pp. 749–778. DOI: 10.3233/JCS-130473. URL: https://doi.org/10.3233/JCS-130473 (cit. on p. 101).

[95] Adam O'Neill, Chris Peikert, and Brent Waters. "Bi-Deniable Public-Key Encryption". In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2011, pp. 525–542. DOI: 10.1007/978-3-642-22792-9_30 (cit. on p. 73).

[96] Bijeeta Pal et al. "Beyond Credential Stuffing: Password Similarity Models Using Neural Networks". In: *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2019, pp. 417–434. DOI: 10.1109/SP.2019.00056 (cit. on p. 127).

[97] Zhiniang Peng. *Redirect Attack on Shadowsocks Stream Ciphers*. https://github.com/edwardz246003/shadowsocks. Retrieved May 2020. (Visited on 02/12/2020) (cit. on p. 140).

[98] Bertram Poettering and Paul Rösler. "Combiners for AEAD". In: *IACR Transactions on Symmetric Cryptology* 2020.1 (2020), pp. 121–143. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i1.121-143 (cit. on p. 63).

[99] Gordon Procter. *A Security Analysis of the Composition of ChaCha20 and Poly1305*. Cryptology ePrint Archive, Report 2014/613. https://eprint.iacr.org/2014/613. 2014 (cit. on p. 120).

[100] Gordon Procter and Carlos Cid. "On Weak Keys and Forgery Attacks Against Polynomial-Based MAC Schemes". In: *Fast Software Encryption – FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2014, pp. 287–304. DOI: 10.1007/978-3-662-43933-3_15 (cit. on pp. 117, 124, 125, 127, 135).

[101] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on p. 169).

[102] Phillip Rogaway. "Authenticated-Encryption With Associated-Data". In: *ACM CCS 2002: 9th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 98–107. DOI: 10.1145/586110.586125 (cit. on pp. 34, 91, 111).

[103] Alexander Russell et al. "Cliptography: Clipping the Power of Kleptographic Attacks". In: *Advances in Cryptology – ASIACRYPT 2016, Part II*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10032. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2016, pp. 34–64. DOI: 10.1007/978-3-662-53890-6_2 (cit. on p. 63).

[104] Alexander Russell et al. "Correcting Subverted Random Oracles". In: *Advances in Cryptology – CRYPTO 2018, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2018, pp. 241–271. DOI: 10.1007/978-3-319-96881-0_9 (cit. on p. 63).

[105] Alexander Russell et al. *Destroying Steganography via Amalgamation: Kleptographically CPA Secure Public Key Encryption*. Cryptology ePrint Archive, Report 2016/530. https://eprint.iacr.org/2016/530. 2016 (cit. on p. 63).

[106] Alexander Russell et al. "Generic Semantic Security against a Kleptographic Adversary". In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham et al. ACM Press, Oct. 2017, pp. 907–922. DOI: 10.1145/3133956.3133993 (cit. on p. 63).

[107] Markku-Juhani Olavi Saarinen. "Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes". In: *Fast Software Encryption – FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2012, pp. 216–225. DOI: 10.1007/978-3-642-34047-5_13 (cit. on pp. 117, 124, 125).

[108] Amit Sahai and Brent Waters. "How to use indistinguishability obfuscation: deniable encryption, and more". In: *46th Annual ACM Symposium on Theory of Computing*. Ed. by David B. Shmoys. ACM Press, May 2014, pp. 475–484. DOI: 10.1145/2591796.2591825 (cit. on p. 73).

[109] *Shadowsocks - A Fast Tunnel Proxy That Helps You Bypass Firewalls*. https://shadowsocks.org. Retrieved May 2021. (cit. on pp. 116, 136).

[110] Claude E. Shannon. "Communication theory of secrecy systems". In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715 (cit. on p. 25).

[111] Gustavus J. Simmons. "The Prisoners' Problem and the Subliminal Channel". In: *Advances in Cryptology – CRYPTO'83*. Ed. by David Chaum. Plenum Press, New York, USA, 1983, pp. 51–67 (cit. on p. 58).

[112] Nihal Vatandas et al. "On the Cryptographic Deniability of the Signal Protocol". In: *ACNS 20: 18th International Conference on Applied Cryptography and Network Security, Part II*. Ed. by Mauro Conti et al. Vol. 12147. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2020, pp. 188–209. DOI: 10.1007/978-3-030-57878-7_10 (cit. on p. 147).

[113] Serge Vaudenay. "Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS..." In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2002, pp. 534–546. DOI: 10.1007/3-540-46035-7_35 (cit. on p. 128).

[114] Yi Wang et al. "Secure Anonymous Communication on Corrupted Machines with Reverse Firewalls". In: *IEEE Transactions on Dependable and Secure Computing* (2021), pp. 1–1. DOI: 10.1109/TDSC.2021.3107463 (cit. on p. 63).

[115] Philipp Winter and Stefan Lindskog. "How the Great Firewall of China is Blocking Tor". In: *2nd USENIX Workshop on Free and Open Communications on the Internet, FOCI '12, Bellevue, WA, USA, August 6, 2012*. Ed. by Roger Dingledine and Joss Wright. USENIX Association, 2012. URL: https://www.usenix.org/conference/foci12/workshop-program/presentation/winter (cit. on pp. 117, 127).

[116] Joanne Woodage. "Provable Security in the Real World: New Attacks and Analyses". English. PhD thesis. Royal Holloway, University of London, 2019 (cit. on p. 26).

[117] Joanne Woodage and Dan Shumow. "An Analysis of NIST SP 800-90A". In: *Advances in Cryptology – EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. Lecture Notes in Computer Science. Springer, Heidelberg, May 2019, pp. 151–180. DOI: 10.1007/978-3-030-17656-3_6 (cit. on p. 18).

[118] Zhichao Yang et al. "On the Security of LWE Cryptosystem against Subversion Attacks". In: *The Computer Journal* 63.4 (Sept. 2019), pp. 495–507. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxz084. eprint: https://academic.oup.com/comjnl/article-pdf/63/4/495/33153492/bxz084.pdf. URL: https://doi.org/10.1093/comjnl/bxz084 (cit. on p. 80).

[119] Adam Young and Moti Yung. "Kleptography: Using Cryptography Against Cryptography". In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, Heidelberg, May 1997, pp. 62–74. DOI: 10.1007/3-540-69053-0_6 (cit. on pp. 17, 58).

[120] Adam Young and Moti Yung. "The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone?" In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 1996, pp. 89–103. DOI: 10.1007/3-540-68697-5_8 (cit. on pp. 17, 58, 61, 88).

[121] Bo Zhu. *AES-GCM-Python*. https://github.com/bozhu/AES-GCM-Python/blob/master/aes_gcm.py. 2013 (cit. on p. 106).

Appendix

A.1 Trapdoor Permutations

A.1.1 One-Way Permutation

A triple $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ of probabilistic polynomial-time algorithms is a family of permutations if the following hold:

1. The parameter-generation algorithm Gen, on input 1^n, outputs parameters I with $|I| \geq n$. Each value of I defines a set D_I that constitutes the domain and range of a permutation (i.e., bijection) $f_I \colon D_I \to D_I$.

2. The sampling algorithm Samp, on input I, outputs a uniformly distributed element of D_I.

3. The deterministic evaluation algorithm f, on input I and $x \in D_I$, outputs an element $y \in D_I$. We write this as $y := f_I(x)$.

Given a family of functions Π, consider the following experiment for any algorithm $\mathcal{A}$ and parameter n: The inverting experiment $\mathrm{INVERT}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ is run to obtain I, and then $\mathsf{Samp}(I)$ is run to choose a uniform $x \in D_I$. Finally, $y := f_I(x)$ is computed.

2. $\mathcal{A}$ is given I and y as input, and outputs x'.

3. The output of the experiment is 1 if and only if $f_I(x') = y$.

The family of permutations $\Pi = (\mathsf{Gen}, \mathsf{Samp}, f)$ is one-way if for all probabilistic

polynomial-time algorithms $\mathcal{A}$ there exists a negligible function $negl$ such that

$$\Pr\left[\mathrm{INVERT}_{\mathcal{A},\Pi}(n) = 1\right] \leq negl(n).$$

A.1.2 Trapdoor Permutation

A tuple of polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Samp}, f, Inv)$ is a family of trapdoor permutations (or a trapdoor permutation) if:

- The probabilistic parameter-generation algorithm Gen, on input 1^n, outputs (I, td) with $|I| \geq n$. Each value of I defines a set D_I that constitutes the domain and range of a permutation (i.e., bijection) $f_I \colon D_I \to D_I$.
- Let Gen_1 denote the algorithm that results by running Gen and outputting only I. Then $(\mathsf{Gen}_1, \mathsf{Samp}, f)$ is a family of one-way permutations.
- Let (I, td) be an output of $\mathsf{Gen}(1^n)$. The deterministic inverting algorithm Inv, on input td and $y \in D_I$, outputs $x \in D_I$. We denote this by $x := \mathsf{inv}_{td}(y)$. It is required that with all but negligible probability over (I, td) output by $\mathsf{Gen}(1^n)$ and uniform choice of $x \in D_I$, we have

$$\mathsf{inv}_{td}(f_I(x)) = x.$$

A.1.3 Hardcore Predicate

Let $\Pi = (\mathsf{Gen}, f, \mathsf{Inv})$ be a family of trapdoor permutations, and let hc be a deterministic polynomial-time algorithm that, on input I and $x \in D_I$, outputs a single bit $\mathsf{hc}_I(x)$. We say that hc is a hard-core predicate of Π if for every probabilistic polynomial-time algorithm $\mathcal{A}$ there is a negligible function $negl$ such that

$$\Pr\left[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)\right] \leq 1 + negl(n)$$

Game IND-CCA$^b(\mathcal{A})$	**Game** subIND-CCA$^b(\mathcal{A})$
00 $C \leftarrow \emptyset$	00 $C \leftarrow \emptyset$
01 $(pk, sk) \leftarrow \mathsf{KEM.gen}$	01 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
02 $b' \leftarrow \mathcal{A}^{\mathrm{Encap,Decap}}(pk)$	02 $(pk, sk) \leftarrow \mathsf{KEM.gen}_{i_{\mathsf{gen}}}$
03 stop with b'	03 $b' \leftarrow \mathcal{A}^{\mathrm{Encap,Decap}}(pk)$
	04 stop with b'
Oracle Encap	**Oracle** Encap
04 $(k^0, c) \leftarrow \mathsf{KEM.enc}(pk)$	05 $(k^0, c) \leftarrow \mathsf{KEM.enc}_{i_{\mathsf{S}}}(pk)$
05 $k^1 \leftarrow_{\$} \mathcal{K}$	06 $k^1 \leftarrow_{\$} \mathcal{K}$
06 $C \xleftarrow{\cup} \{c\}$	07 $C \xleftarrow{\cup} \{c\}$
07 return (k^b, c)	08 return (k^b, c)
Oracle Decap(c)	**Oracle** Decap(c)
08 require $c \notin C$	09 require $c \notin C$
09 $k \leftarrow \mathsf{KEM.dec}(sk, c)$	10 $k \leftarrow \mathsf{KEM.dec}_{i_{\mathsf{R}}}(sk, c)$
10 return k	11 return k

Figure A.1: Games modelling indistinguishability under chosen-ciphertext attacks (IND-CCA), and subverted indistinguishability under chosen-ciphertext attacks (subIND-CCA) for a key encapsulation mechanism KEM.

where the probability is taken over the experiment in which $\mathsf{Gen}(1^n)$ is run to generate (I, td) and then x is chosen uniformly from D_I. The asymmetry provided by trapdoor permutations implies that anyone who knows the trapdoor td associated with I can recover x from $f_I(x)$ and thus compute $\mathsf{hc}_I(x)$ from $f_I(x)$. But given only I, it is infeasible to compute $\mathsf{hc}_I(x)$ from $f_I(x)$ for a uniform x.

A.2 Key and Data Encapsulation Mechanisms

A.2.1 Key Encapsulation Mechanisms

For completeness, we give the corresponding definitions of subversion attacks against key encapsulation mechanisms, together with notions of undetectability and key recovery.

A.2.1.1 KEM Definition

A KEM scheme $\mathsf{KEM} = (\mathsf{KEM.gen}, \mathsf{KEM.enc}, \mathsf{KEM.dec})$ for a finite session key space $\mathcal{K}$ is a triple of algorithms together with a key space $\mathcal{K}_{\mathsf{S}} \times \mathcal{K}_{\mathsf{R}}$ and ciphertext

space $\mathcal{C}$. The key generation algorithm $\mathsf{KEM.gen}$ returns a pair $(pk, sk) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$ consisting of a public key and a secret key. The encapsulation algorithm $\mathsf{KEM.enc}$ takes a public key pk to produce a session key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$. Finally, the decapsulation algorithm $\mathsf{KEM.dec}$ takes a secret key sk and a ciphertext $c \in \mathcal{C}$, and outputs either a session key $K \in \mathcal{K}$ or the special symbol $\perp \notin \mathcal{K}$ to indicate rejection. The correctness requirement is that for all $(pk, sk) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$ we have $\Pr[\mathsf{KEM.dec}(sk, c) \neq k] \leq \delta$ for $(k, c) \leftarrow \mathsf{KEM.enc}(pk)$.

A.2.1.2 IND-CCA

For a key encapsulation mechanism, we formalise the indistinguishability under chosen-ciphertext attack via the game IND-CCA in Figure A.1 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\text{IND-CCA}^0(\mathcal{A})\right] - \Pr\left[\text{IND-CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme KEM is indistinguishable against chosen-ciphertext attacks if $\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{KEM}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

A.2.1.3 Subverting KEM

We note that KEM schemes satisfy the generic syntax introduced above in Section 3.2, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{KEM.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{KEM.enc}$, receiver algorithm $\Pi.\mathsf{R} = \mathsf{KEM.dec}$. We may thus apply the generic notions of subversion introduced in Section 3.2.1, and observe that the passive attack in Section 5.3.2 applies. If the KEM scheme is in addition ciphertext sparse, according to the notion in Section 5.3.3.1, then the attacks in Section 5.3.3 will also apply. Figure A.1 (right) shows the game modelling subverted indistinguishability under chosen-ciphertext attacks.

A.2.2 Data Encapsulation

A DEM scheme $\mathsf{DEM} = (\mathsf{DEM.gen}, \mathsf{DEM.enc}, \mathsf{DEM.dec})$ is a triple of algorithms together with associated key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. The key generation algorithm $\mathsf{DEM.gen}$ returns key $k \in \mathcal{K}$. The encapsulation algorithm $\mathsf{DEM.enc}$ takes key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$. The decapsulation algorithm $\mathsf{DEM.dec}$ takes a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection. The correctness requirement is that for all keys $k \in \mathcal{K}, m \in \mathcal{M}$ it holds that $\Pr[\mathsf{DEM.dec}(k, c) \neq m] \leq \delta$ for $c \leftarrow \mathsf{DEM.enc}(k, m)$.

A.2.2.1 IND-CCA

We formalise the indistinguishability under *one-time* chosen-ciphertext attack of a data encapsulation mechanism via the game IND-CCA in Figure A.2 (left). Note how Line 04 and Line 08 ensure that the adversary's first query is an encryption query, and that all further queries are decryption queries. (This precisely matches the typical situation as it emerges in a KEM/DEM hybrid.) For any adversary $\mathcal{A}$ we define the advantage

$$\mathrm{Adv}^{\text{ind-cca}}_{\mathsf{DEM}}(\mathcal{A}) := \left|\Pr\left[\text{IND-CCA}^0(\mathcal{A})\right] - \Pr\left[\text{IND-CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme DEM is indistinguishable against chosen-ciphertext attacks if $\mathrm{Adv}^{\text{ind-cca}}_{\mathcal{A}}$ is negligibly small for all realistic $\mathcal{A}$.

A.2.2.2 Subversion of DEM

We note that Data Encapsulation Mechanism schemes satisfy the generic syntax introduced above in Section 3.2, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{DEM.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{DEM.enc}$, receiver algorithm $\Pi.\mathsf{R} = \mathsf{DEM.dec}$. We may thus apply the generic notions of subversion introduced in Section 3.2.1, and observe that the passive attack in Section 5.3.2 applies. If the DEM scheme is in addition

Game IND-CCA$^b(\mathcal{A})$	**Game** subIND-CCA$^b(\mathcal{A})$
00 $C \leftarrow \emptyset$	00 $C \leftarrow \emptyset$
01 $k \leftarrow_\$ \mathsf{DEM.gen}$	01 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_\$ \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
02 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$	02 $k \leftarrow_\$ \mathsf{DEM.gen}_{i_{\mathsf{gen}}}$
03 stop with b'	03 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
	04 stop with b'
Oracle Enc(m^0, m^1)	**Oracle** Enc(m^0, m^1)
04 require $C = \emptyset$	05 require $C = \emptyset$
05 $c \leftarrow \mathsf{DEM.enc}(k, m^b)$	06 $c \leftarrow \mathsf{DEM.enc}_{i_{\mathsf{S}}}(k, m^b)$
06 $C \xleftarrow{\cup} \{c\}$	07 $C \xleftarrow{\cup} \{c\}$
07 return c	08 return c
Oracle Dec(c)	**Oracle** Dec(c)
08 require $C \neq \emptyset$	09 require $C \neq \emptyset$
09 require $c \notin C$	10 require $c \notin C$
10 $m \leftarrow \mathsf{DEM.dec}(k, c)$	11 $m \leftarrow \mathsf{DEM.dec}_{i_{\mathsf{R}}}(k, c)$
11 return m	12 return m

Figure A.2: Games modelling indistinguishability under one-time chosen-ciphertext attacks (IND-CCA), and subverted indistinguishability under one-time chosen-ciphertext attacks (subIND-CCA) for a data encapsulation mechanism DEM.

ciphertext sparse, according to the notion in Section 5.3.3.1, then the attacks in Section 5.3.3 will also apply. Figure A.2 (right) shows the game modelling subverted indistinguishability under chosen-ciphertext attacks.

Discussion. Typically, a DEM is used together with a KEM in a so-called hybrid encryption scheme that uses the KEM to share (symmetric) session keys with which plaintext messages are encrypted under the DEM. In such a setting, subverting the KEM is sufficient to undermine the security of messages sent via the hybrid scheme. Following the discussion at Section 5.2.2, it is conceivable to subvert a KEM and DEM in tandem so that the KEM's secret key is leaked by the both together. This could allow the subversion to effectively be distributed between the two primitives, aiding undetectability in practice.

A.3 Example Ciphertext Sparse PKE Schemes

We describe two widespread PKE schemes that satisfy the notion of ciphertext sparseness described in Section 5.3.3.1.

OAEP. Optimal Asymmetric Encryption Padding (OAEP) was introduced by Bellare and Rogaway [24] and is a widely deployed and standardised PKE scheme. The encryption algorithm of OAEP works on message space $\mathcal{M} = \{0,1\}^\ell$ with fixed message length ℓ. Let k_0 and k_1 be integers, and $G\colon \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$ and $H\colon \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$ be two hash functions. Messages are padded before being encrypted using the trapdoor permutation (typically RSA): To pad a message $m \in \mathcal{M}$, set $m' \leftarrow m \parallel 0^{k_1}$ and choose $r \leftarrow_\$ \{0,1\}^{k_0}$. Then set $s \leftarrow m' \oplus G(r)$, $t \leftarrow r \oplus H(s)$ and $\hat{m} \leftarrow s \parallel t$. To decrypt a ciphertext, first decrypt (i.e.apply the trapdoor inverse) before unpadding the resulting padded message $\hat{m}$: Parse $\hat{m}$ as $s \parallel t$ with $s \in \{0,1\}^{\ell+k_1}$ and $t \in \{0,1\}^{k_0}$. Now compute $r \leftarrow t \oplus H(s)$ and $m' \leftarrow s \oplus G(r)$. If $m' \neq m \parallel 0^{k_1}$ for some m then reject, otherwise return m.
For a randomly chosen element c in the ciphertext space $\mathcal{C}$, the redundancy introduced by padding will ensure that decrypting c results in a valid message with probability 2^{-k_1}. This is because choosing $c \leftarrow_\$ \mathcal{C}$ is equivalent to choosing a random $m' \leftarrow_\$ \{0,1\}^{\ell+k_1}$, assuming that the trapdoor permutation and hash functions all behave like random functions. Equivalently, the scheme is ciphertext 2^{-k_1}-sparse, according to the definition in Section 5.3.3.1.

Cramer-Shoup. The Cramer-Shoup PKE scheme was introduced in [48]. The encryption scheme $\mathsf{CS} = (\mathsf{CS.gen}, \mathsf{CS.enc}, \mathsf{CS.dec})$ is defined in relation to a set of public parameters consisting of finite group $\mathbb{G}$ with $|\mathbb{G}| = q$ and a pair of generators $g, \hat{g}$ for $\mathbb{G}$, together with a hash key hk for a family of keyed collision resistant universal hash functions $H_{hk} : \mathbb{G}^3 \to \mathbb{Z}_q$. The family of keyed hash functions is such that given a randomly chosen tuple of group elements and randomly chosen hash function key, it is computationally infeasible to find a different tuple of group elements that hashes to the same value using the given hash key. We give details of

Proc $\mathsf{CS.gen}(\mathbb{G}, g, \hat{g}, hk)$
00 $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow_\$ \mathbb{Z}_q$
01 $a \leftarrow g^{x_1}\hat{g}^{x_2}, b \leftarrow g^{y_1}\hat{g}^{y_2}, d \leftarrow g^{z_1}\hat{g}^{z_2}$
02 $pk \leftarrow (a, b, d)$
03 $sk \leftarrow (x_1, x_2, y_1, y_2, z_1, z_2)$
04 output (sk, pk)

Proc $\mathsf{CS.enc}(pk, m)$
05 $u \leftarrow_\$ \mathbb{Z}_q, w \leftarrow g^u, \hat{w} \leftarrow \hat{g}^u$
06 $e \leftarrow d^u \cdot m$
07 $\rho \leftarrow H_{hk}(a, \hat{a}, e)$
08 $v \leftarrow a^u b^{u\rho}$
09 output $c = (a, \hat{a}, e, v)$

Proc $\mathsf{CS.dec}(sk, c)$
10 parse c as $(a, \hat{a}, e, v)$
11 $\rho \leftarrow H_{hk}(a, \hat{a}, e)$
12 if $v \neq a^{x_1+y_1\rho} \cdot \hat{a}^{x_2+y_2\rho}$
13 return $\perp$
14 else:
15 $m \leftarrow c \cdot (a^{z_1}\hat{a}^{z_2})^{-1}$
16 return m

Figure A.3: Cramer-Shoup PKE scheme $\mathsf{CS} = (\mathsf{CS.gen}, \mathsf{CS.enc}, \mathsf{CS.dec})$.

Cramer-Shoup in Figure A.3.

For a randomly chosen element $c = (a, \hat{a}, e, v)$ in the ciphertext space $\mathbb{G}^4$, the redundancy introduced by the hash function will ensure that decrypting c results in a valid message with probability q^{-1}. To see this, consider fixed $a, \hat{a}, e$: this gives a fixed value of $a^{x_1+y_1\rho} \cdot \hat{a}^{x_2+y_2\rho} \in \mathbb{G}$ and thus $\Pr\left[v = a^{x_1+y_1\rho} \cdot \hat{a}^{x_2+y_2\rho}\right] = q^{-1}$.

A.4 Plaintext Awareness

We here give the definition of plaintext awareness, a property of PKE schemes that implies ciphertext sparseness (see Section 5.3.3.1). Plaintext awareness essentially means that an adversary is unable to create ciphertexts without knowing the underlying plaintext message. This means that ciphertexts which have not been generated from underlying plaintext messages should be rejected, implying that ciphertexts chosen uniformly at random from the ciphertext space are unlikely to be valid. Both the Cramer-Shoup cryptosystem [47, 53] and RSA+OAEP [101, 24], outlined in Section 5.3.3.1, satisfy plaintext awareness.

The formal definitions of plaintext awareness in the standard model were proposed by Bellare and Palacio [22], and were slightly extended by Dent and Birkett [53, 34]. A scheme is plaintext aware if for all ciphertext creators (attackers) $\mathcal{A}$, there exists a

plaintext extractor K which takes as input the random coins of A and can answer the decryption queries of A in a manner that A cannot distinguish from a real decryption oracle. In order to model the attacker's ability to obtain ciphertexts for which it does not know the underlying plaintext, the ciphertext creator is equipped with an oracle that will return the encryption of a randomly chosen message $m \leftarrow_{\$} \mathrm{MS}$, where MS is an arbitrary (stateful) message sampling algorithm that takes as input α allowing an adversary (ciphertext creator) to specify a distribution on messages. In Figure A.4, we write $\diamond$ for the initial state of MS. After interacting with either the real decryption algorithm or the knowledge extractor simulating decryption, the ciphertext creator outputs a ciphertext c. A distinguisher $\mathcal{D}$ is now tasked with guessing which case we are in. Note that the knowledge extractor K does not have access to the distinguisher's randomness.

We formalise plaintext awareness of a public-key encryption scheme via the game PA in Figure A.4. For any distinguisher $\mathcal{D}$ we define the advantage

$$\mathrm{Adv}^{\mathrm{pa}}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D}) := \left|\Pr\left[\mathrm{PA}^{0}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})\right] - \Pr\left[\mathrm{PA}^{1}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})\right]\right|.$$

We say that a scheme is plaintext aware if for all realistic ciphertext creators $\mathcal{A}$, there exists a knowledge extractor K such that for all message samplers MS and distinguishers $\mathcal{D}$, the advantage $\mathrm{Adv}^{\mathrm{pa}}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})$ is negligibly small.

We note that a PKE scheme that satisfies plaintext awareness and indistinguishability against chosen-ciphertext attacks (Section 2.3.6) is necessarily ciphertext sparse. To see this, suppose that the PKE scheme is not ciphertext sparse. For a randomly chosen ciphertext $c \leftarrow_{\$} \mathcal{C}$, the real game $\mathrm{PA}^{0}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})$ will output a valid message m. However, in the random game $\mathrm{PA}^{1}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})$ the knowledge extractor will not be able to output m without contradicting the plaintext awareness and CCA security of the scheme. We thus conclude that the scheme is ciphertext sparse.

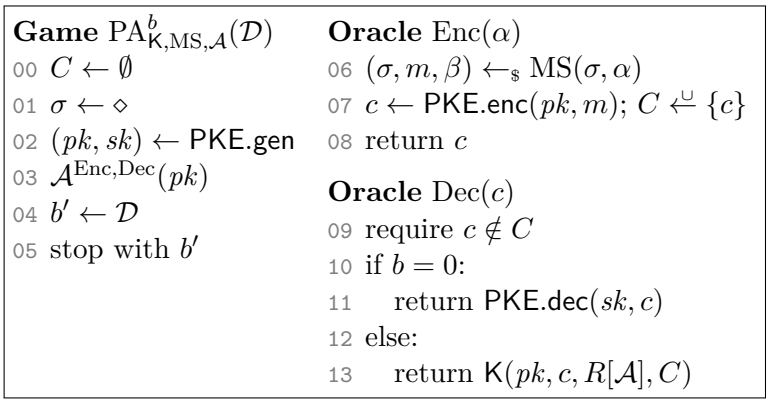

Figure A.4: Game modelling plaintext awareness (PA), for a public-key encryption scheme PKE. Note that we retain β (modelling side-channel information) in the syntax of message sampler MS for consistency, but the adversary is not given output β.