On the Verification of Computation and Data Retrievability

Christian Janson

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

Information Security Group
School of Mathematics and Information Security
Royal Holloway, University of London
2016

Declaration

These doctoral studies were conducted under the supervision of Professor 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.

Christian Janson
July, 2016

Acknowledgements

First and foremost, I am very grateful to my PhD supervisor Carlos Cid for his supervision and advice guiding me through the process of completing this thesis. Thank you for letting me work on exciting research areas even though they were not completely in your area of expertise, and for your support of my attendance of many exciting and useful events.

I also wish to express my gratitude to my adviser Kenny Paterson for helpful discussions about cryptography in general as well as assisting me with career related advice. I wish to thank the former director of the ISG, Keith Martin, for all the opportunities and financial support to attend conferences, as well as many exciting squash and badminton matches. I thank Martin R. Albrecht for many helpful discussions and suggesting the Information Security Group to start a PhD in the first place which was certainly a great choice. I also wish to thank Michael Hortmann for introducing me to the exciting world of cryptography during my math studies.

I am thankful to have collaborated with James Alderman, Frederik Armknecht, Carlos Cid, Jason Crampton, Sarah Louise Renwick and Christian A. Reuter.

I am indebted to all PhD students and members of staff of the ISG for providing me with such a pleasant environment as well as friends I have met outside the academic scope. In particular, I wish to thank Alex, Alexandre, Andrew, Bertfried, Christian, Dale, Dean, Ela, George, Greg, Gordon, Guillermo, Johanna, Jovares, Martin, Muzamil, Mwawi, Rob, Sam, Thalia, Thyla and Victor.

I deeply thank Eugenio, Konstantinos, Matteo and Pavlo for many memories that will always be part of my life. Especially the many matches of squash at Kingswood and badminton at the sports centre provided me with many hours of happiness and memorable stories.

A special thank you goes out to Team Llama a.k.a Dan, James, Naomi and Rachel. Many thanks for all the outings in and around London, the many escape rooms we successfully escaped, coffee, sometimes tea, teaching me proper (northern) English as well as discussing cryptographic problems. Especially, I thank James for being an incredible co-author.

To my friends at home I would like to thank Matthias, Mischa, Thilo and Thommy for the many adventures we have experienced and the ones ahead of us.

Many thanks to my family. Especially, I thank my aunt Silvia for her endless support, love, and help when it was needed the most. I also wish to thank my late parents, Elisabeth and Joachim, for their love and sacrifices made to provide me with a good education.
Last, but certainly not least, I thank Wiebke for all her love, patience and encouragement over this period of time. Thank you for supporting me going abroad and pursuing this path. All of this would have not been possible without you and the real adventure has just started.

Christian Janson

Abstract

The cloud model offers many useful services such as storage and computing solutions that enrich our daily lives. However, there is a restriction in using the cloud's optimal potential since the client relies on trusting the cloud completely. This blind faith can easily be exploited by the cloud by lying about computational results or deleting data; making verification of results a desirable property to obtain a level of assurance while relaxing the trust assumption.

Publicly verifiable computation (PVC) enables a computationally-limited client to outsource computations to an untrusted server and to verify correctness of the returned results. Servers providing such a service may be rewarded per computation, providing an incentive to cheat by returning malformed results rather than devoting time and resources to compute a valid result. In this thesis, we extend a previous approach using attribute-based encryption (ABE) to enable a broader system model for PVC such that servers may compute multiple functions and if found cheating, are revoked from the system. We show that different types of ABE accommodate different system models and ultimately show that dual-policy ABE unifies all ABE based PVC models into a hybrid model which can flexibly switch between the models at the cost of a single setup.

Proofs of retrievability (PoR) enable a client to outsource data to an untrusted server and allow the client to request a proof that the data stored can be retrieved, which the client can verify. We construct a somewhat practical scheme that enables the client to request proofs of retrievability of multiple different-sized files with a single request. This is achieved by using homomorphic properties to aggregate a proof into a small value. Furthermore, using combinatorial and statistical tools we derive strategies obtaining an assurance whether the server retains enough information to deliver the original data.

Contents

List of Figures

List of Tables

Publications

The work in this thesis originates from the five papers listed below.

1. James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. *Revocation in Publicly Verifiable Outsourced Computation.* In D. Lin, M. Yung and J. Zhou, editors, Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers, volume 8957 of Lecture Notes in Computer Science, pages 51–71. Springer, 2014.

2. James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. *Access Control in Publicly Verifiable Outsourced Computation.* In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2015, pages 657–662, New York, NY, USA, 2015. ACM.

3. James Alderman, Christian Janson, Keith M. Martin, and Sarah Louise Renwick. *Extended Functionality in Verifiable Searchable Encryption.* In E. Pasalic and L. R. Knudsen, editors, Cryptography and Information Security in the Balkans - Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers, volume 9540 of Lecture Notes in Computer Science, pages 187–205. Springer, 2015.

4. James Alderman, Christian Janson, Carlos Cid, and Jason Crampton. *Hybrid Publicly Verifiable Computation.* In K. Sako, editor, Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, volume 9610 of Lecture Notes in Computer Science, pages 147–163. Springer, 2016.

5. Christian Janson, Christian A. Reuter, Frederik Armknecht, and Carlos Cid. *Cloud Storage Proofs of Retrievability.* 2016. In Submission.

All authors contributed equally to the above publications.

Chapter 1

Introduction

Contents

This chapter provides an overview of this thesis. We provide motivation for the undertaken research and present the structure of this thesis.

1.1 Motivation

Cloud service providers are continuously gaining importance over the last few years. They offer various services in numerous application domains such as storage, computing services and key management services. The huge success of the cloud model is based on offering various benefits such as flexible scalability, accessibility and easy manageability to companies and individuals to employ cloud services in a cost effective manner.

The combination of Software-as-a-Service and the increasing use of mobile devices being employed as general computing devices give rise to a considerable difference in computational power between cloud service providers (or servers) and clients. Thus, there is a tremendous desire for clients to outsource the evaluation of complex functions to an external server. Servers providing such a service may be rewarded per computation, and as such have an incentive to cheat by returning malformed computational results rather than devoting their precious resources and time to compute a correct result. Thus, this enables servers to offer computational services to more costumers within the same time frame and potentially increase their rewards.

For example, nowadays it is natural that a company may operate a "bring your own device" policy, enabling employees to use personal smartphones and tablets for work. Due to resource limitations, it may not be possible for these devices to perform complex computations locally. Instead, a computation is outsourced over some network to a more powerful server (possibly outside the company, offering Software-as-a-Service, and hence untrusted) and the result of the computation is returned to the client device. In formal terms, this means given a function F to be computed by a server S, the client sends her personal input x to the server S that should return the computational result

$F(x)$ to the client. However, there may be an incentive for the server to cheat and thus return an invalid result y to the client that does not correspond to the actual computational result $F(x)$. The reason for such a behaviour may be that the server is simply too busy or may not wish to devote any resources on performing computations, and thus wishes to convince the client to accept a malformed result. Therefore, the client has a natural desire to obtain some assurance that the returned result is indeed correct.

This problem, known as *verifiable outsourced computation* (VC), has attracted a lot of attention in the research community recently. VC schemes provide a solution of the above problem and enable a client to outsource computations to a powerful server while providing her with the possibility of verifying the correctness of the computational result. The underlying efficiency requirement for VC schemes simply expects that outsourcing and verifying of the computation take less time than computing the function from scratch. Usually, VC schemes use an amortised notion of complexity for the client, i.e. the client may perform an expensive pre-processing phase, but after this stage, she is required to run very efficiently.

Further activities in this area lead to the notion of *publicly verifiable outsourced computation* (PVC). This notion follows the same principles as above but enables after the pre-processing any client to delegate and verify computations since these actions simply rely on the public information of the system. PVC has the same efficiency requirement as VC. PVC also uses an amortised notion of efficiency that requires a single client to invest in the expensive pre-processing while enabling many other clients to benefit from her initial effort, and thus amortises this expensive step over the number of clients within the system. Therefore, PVC can be seen as a more practical scheme accommodating multiple clients outsourcing function evaluations to a server but not necessarily on joint inputs.

Those VC and PVC solutions can be seen as a contribution towards ensuring that an untrusted server has returned a correct result or otherwise can be detected cheating. Yet another popular application domain within the cloud model is the storage domain. Here, cloud service providers offer a scalable and very cost effective storage solution compared to, from an organisational point of view, building an own data storage centre, or from an individual client's perspective, owning several hard drives. Thus it is very tempting to use this storage service. However, an enormous drawback is that cloud service providers do not provide provable storage guarantees about the outsourced data.

A solution towards achieving provable storage guarantees is the notion of *proofs of retrievability* (PoR). Here the guarantees are formulated in terms of a verifiable statement whether or not the client's outsourced data is *authentic* and *retrievable*. The first

property expresses that the client wants to be able to verify that the received data is correct whereas the latter property reflects the need to assure that no data loss has occurred on the cloud storage provider's side.

In this thesis, we propose solutions to enhance current PVC and PoR notions in terms of practicability and functionality.

1.2 Organisation of Thesis

Chapter 2. This chapter contains the preliminaries and will establish the notational conventions that will be used throughout this thesis. All the necessary primitives and notions of security can be found in this chapter.

Our contributions are then presented in the remaining chapters.

Chapter 3. In this chapter we consider the setting of *publicly verifiable outsourced computation* (PVC) for which it has been shown that key-policy attribute-based encryption can be used. We propose extensions and improvements to the current PVC proposal in order to accommodate a more practical framework and enable a server to compute multiple functions rather than only being certified to compute a single function. This is achieved by a simple encoding trick and further the chapter provides a method to revoke misbehaving servers from future evaluations within the system.

Chapter 4. This chapter considers the problem in which an untrusted server holds a data set in such a way that any client can ask the server to compute a function on any input portion of the data set. We provide a solution to this problem by introducing a scheme called *verifiable delegable computation* (VDC) and argue that this system model is a reversed system architecture compared to our proposal in Chapter 3. We give a provably secure construction based on ciphertext-policy attribute-based encryption and discuss relevant security models. Furthermore, we observe that this notion has some natural applications to verifiable queries on remote databases and verifiable parallel processing using the MapReduce framework.

Chapter 5. In this chapter we use the already introduced system models from the previous chapters and define an umbrella scheme called *hybrid PVC*. This scheme only requires a single setup stage in order to provide a flexible outsourced computation solution. Furthermore, we briefly introduce another mode of publicly verifiable computation that extends the proposal from Chapter 3 enabling us to enforce graph-based access control policies over the delegators, servers and verifiers. We give a provably secure construction based on dual-policy attribute-based encryption showing that all

introduced system models are captured within this general framework and discuss relevant security models for the different modes.

Chapter 6. This chapter investigates the application of PVC techniques developed in this thesis to the realm of verifiable searchable encryption (VSE). We introduce a VSE scheme based upon ciphertext-policy attribute-based encryption that permits a user to verify that search results are correct and complete. Our scheme also permits verifiable computational queries over keywords and specific data values, that go beyond the standard keyword matching queries to allow functions such as averaging or counting operations.

Chapter 7. In this chapter we turn our attention to the setting of providing provable data storage guarantees, i.e. verifying whether a server still retains the client's outsourced data. A possible concept achieving such guarantees is known as proofs of retrievability (PoR). We propose extensions and improvements to the current PoR proposals in order to accommodate a more practical framework checking whether multiple files are retained simultaneously. Thus, we overcome limitations of previous schemes which were only able to check a single file at a time. We discuss different strategies a client can use in order to obtain a verifiable statement about the retrievability of all her outsourced data from a server, and also evaluate the performance of our proposed scheme.

Chapter 2

Background Material

Contents

This chapter introduces the necessary notation and discusses in detail various cryptographic primitives and security notions that will be used throughout this thesis.

2.1 Preliminaries

2.1.1 Notation

Let us fix all necessary notations that are used in the remainder of this thesis. The set of integers is denoted by $\mathbb{Z}$, the set of all non-zero integers is denoted by $\mathbb{Z}^\star = \mathbb{Z} \setminus \{0\}$ and we denote the set of natural numbers in the natural way as being the set of non-negative integers $\mathbb{N} = \{n \in \mathbb{Z} | n > 0\}$. We denote the set of consecutive integers $\{i, \ldots, j\}$ by $[i, j]$, the set $\{1, \ldots, n\}$ will be written as $[n]$ for $n \geq 1$, and $\emptyset$ denotes the empty set. If X is a set, we denote the *power set* of X by 2^X which is the set of all subsets, and $|X|$ denotes its size. A *partially ordered set* (*poset*) is a set L equipped with a reflexive, anti-symmetric and transitive binary relation $\leqslant$. In other words, for all $x, y, z \in L$ it holds that $x \leqslant x$ (reflexivity); if $x \leqslant y$ and $y \leqslant x$ then $x = y$ (anti-symmetry); and if $x \leqslant y$ and $y \leqslant z$ then $x \leqslant z$ (transitivity). We may write $x < y$ if $x \leqslant y$ and $x \neq y$, and write $y \geqslant x$ if $x \leqslant y$. We say that x *covers* y, written $y \lessdot x$, if $y < x$ and no z exists in L such that $y < z < x$. The *Hasse Diagram* of a poset $(L, \leqslant)$ is the directed acyclic graph $H = (L, \lessdot)$ where vertices are labelled by the elements of L and an edge connects vertex v to w if and only if $w \lessdot v$.

If A is a deterministic algorithm, we use the notation $y \leftarrow A(x_1, \ldots, x_n)$ to denote the action of running A on the given inputs x_1 to x_n and assigning the result y. Similarly, if A is a randomised (probabilistic) algorithm, we simply write $y \xleftarrow{\$} A(\cdot)$ to denote A outputting the variable y. This notion can also be seen as sampling an output y from the range of possible outputs of A according to an underlying distribution defined by the input arguments. Equivalently, one may consider the algorithm A taking an additional input r of random coins which determine the outcome of randomised choices during the execution of the algorithm. Thus, more formally we would write $y \leftarrow A(\cdot; r)$ since the explicit choice of r renders the algorithm A deterministic. Finally, if B denotes a set, we write $y \xleftarrow{\$} B$ to express sampling a value y uniformly at random from the set B. The set of all finite binary strings is denoted by $\{0,1\}^*$. For any two binary strings a and b, we denote their concatenation by $a \| b$ and $a \oplus b$ denotes their bitwise XOR.

When considering cryptographic schemes in this thesis, we denote the message space by $\mathcal{M}$, the key space by $\mathcal{K}$, the security parameter by $\lambda \in \mathbb{N}$ and its respective unary representation as 1^λ. The key space consists of all bit strings of length λ and it can be increased to asymptotically increase the strength of the cryptographic primitive. We denote by ϵ the empty string or an empty list, $\perp \notin \mathcal{M}$ denotes a distinguished failure symbol outputted by an algorithm, and we abbreviate probabilistic polynomial-time by PPT. Let F be a function then we denote the domain of the function by $\mathsf{Dom}(F)$ and the range by $\mathsf{Ran}(F)$. A function f from the natural numbers to the non-negative real numbers is said to be *negligible* on its input if for every positive polynomial p there exists an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$. We denote an arbitrary negligible function by negl.

A *Boolean function* $F\colon \{0,1\}^n \to \{0,1\}$ takes a string of n bits as input and outputs a single bit result. We use the notation 1 and `true`, and similarly 0 and `false`, to denote the outcome of this function. We denote by $\vee$ the binary OR operator, and by $\wedge$ the binary AND operator. If a Boolean function F evaluates to `true` on a set of input strings I, we say that the set I is a *satisfying set*. Furthermore, it is possible to describe F in terms of satisfying sets. In case this representation is used, we refer to F as an *access structure*. A Boolean function is said to be *monotonic* if for all sets $S \subset T \subseteq [n]$, it holds that $F(S) \leq F(T)$. In other words, increasing the size of the set can only increase the value of F, or alternatively, replacing a 0 with a 1 in the input can only increase the value of F. Finally, we denote by $\mathcal{F}_n$ the family of n-ary monotone functions closed under complement. In more detail, if $F \in \mathcal{F}_n$ then the complement function $\overline{F} = F \oplus 1$ also belongs to $\mathcal{F}_n$. In case n is clear from the context, we abbreviate $\mathcal{F}_n$ as $\mathcal{F}$. A more formal treatment of Boolean functions and their properties can be found in [44, 59].

2.1.2 Provable Security

A central task in cryptography is to analyse whether a particular cryptographic scheme is "secure". However, this simple question is hard to answer. In the "dark ages" of cryptography, security of a cryptographic scheme was assumed until someone broke the scheme. At this point, one tried either to fix the scheme to make it "secure" again or discarded it if the security issue could not be fixed. In the age of *modern* cryptography, such an approach is not acceptable and thus we require rigorous techniques to analyse and assess the security of a cryptographic scheme and protocol. Such techniques fall into the realm of *provable security* which refers to a well-studied paradigm in modern cryptography. On a high-level, the goal is to formalise a security property and prove that the property holds – relative to definitions and assumptions – which corresponds to an assurance that the scheme is secure. The approach uses ideas and techniques from theoretical computer science as well as mathematics.

In order to formalise a proof of security, we require two necessary ingredients. The first ingredient is to formalise precise *security definitions*. These specify concrete security properties that cryptographic schemes are intended to achieve. A security definition is normally expressed as an *experiment* (or *game*) presented in pseudo-code and is designed to reflect potential security threats against the scheme as well as a realistic system execution. The experiment is run by a computationally unbounded entity known as the *challenger* $\mathcal{C}$ and played by a computationally bounded *adversary* $\mathcal{A}$ with respect to a cryptographic scheme. The adversary is usually modelled as a PPT algorithm which is called at relevant points during the execution of the experiment by the challenger. Furthermore, $\mathcal{A}$ may maintain a state throughout the experiment which reflects the adversary being called several times at different points during the execution of the system. Security of a cryptographic scheme is usually defined in terms of the adversary's maximal winning probability against the formalised experiment.[1] Thus, we define a scheme to be *secure* if a class of adversaries have at most a *negligible* winning probability (in terms of the security parameter) of winning against the formalised security definition. Otherwise, we say a scheme is *insecure*.

The second fundamental ingredient is the *reduction proof technique* which enables one to prove that a cryptographic scheme is computationally secure. As mentioned above, a security proof is conditional in the sense that we require to assume that some mathematical problem is hard. That is, security can rely on the assumption that the underlying primitive, on which the scheme is built, is secure. The reduction proof technique then enables one to *prove* that a cryptographic scheme is secure under this assumption. In

[1] Note that this sometimes varies depending on the underlying experiment. For example, IND-CPA security is defined in terms of the maximal winning probability minus the probability of a random guess.

more detail, the technique boils down to present an explicit *reduction* that transforms any efficient algorithm $\mathcal{A}$ that succeeds in winning against the security notion of the scheme ("breaking") into an efficient algorithm $\mathcal{B}$ that solves the underlying problem that was initially assumed to be hard. A very good and detailed explanation of the required steps for a proof by reduction can be found in [100, 130].

Throughout this thesis, we will construct concrete cryptographic schemes and provide different security definitions that we model the scheme to be secure against. To prove the security of the scheme we make intensive use of the above introduced reduction technique.

2.2 Encryption Schemes

Encryption is mainly a technique in order to preserve the confidentiality of messages exchanged between a sender and a receiver. The primitive we consider here is called an *encryption scheme*. Such a scheme specifies a *key-generation algorithm* which produces key material that the parties need to share. Furthermore, it specifies an *encryption algorithm*, which provides the sender with an instruction how to process the plaintext using the key to produce a ciphertext that is transmitted to the receiver. An encryption scheme also specifies a *decryption algorithm*, which provides the receiver with instructions how to retrieve the original message from the transmitted ciphertext using the key while possibly performing some verification as well. In this section, we begin to review the definitions of *symmetric encryption schemes* and *public-key encryption schemes*.

2.2.1 Symmetric Encryption Schemes

A *symmetric encryption scheme* relies on a secret key k held by both entities, namely the sender and receiver of a ciphertext. Note that we do not address here how both entities came into joint possession of this key k, and thus assume that the key has been shared in some way. More formally a symmetric encryption scheme is described as follows.

Definition 2.1. *A* symmetric encryption (SE) scheme *consists of the following three algorithms.*

- $k \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$: *this randomised algorithm takes as input the security parameter* λ *and randomly selects a symmetric key* k *from the key space* $\mathcal{K}$*;*
- $ct \xleftarrow{\$} \mathsf{Encrypt}(m, k)$: *this randomised algorithm takes as input the symmetric key* k *and a message* m *from the message space* $\mathcal{M}$*. It outputs a ciphertext* ct*;*
- $m \leftarrow \mathsf{Decrypt}(ct, k)$: *this deterministic algorithm takes as input the symmetric key* k *and the ciphertext* ct*. It outputs the underlying message* m *encrypted in the ciphertext* ct*.*

Note that we could extend the decryption algorithm to output a failure symbol $\perp$ in case it is presented with an invalid ciphertext.

Correctness of a symmetric encryption scheme requires that for all security parameters and all messages, the decryption of an honestly generated ciphertext under a correctly generated key will return the correct message. This is captured more formally as follows.

Definition 2.2. *A symmetric encryption scheme is* correct *if for every security parameter λ, every key k outputted by $\mathsf{KeyGen}(1^\lambda)$, and every message $m \in \mathcal{M}$, it holds that $\mathsf{Decrypt}(\mathsf{Encrypt}(m,k),k) = m$.*

There are many notions of security for symmetric encryption schemes and the relation amongst them is discussed in [26]. The choice of security property of a symmetric encryption scheme depends on the context in which the scheme is used as well as on what information an adversary may observe in practice. In the following we present two main notions of security for a symmetric encryption scheme, namely *indistinguishability against chosen-plaintext attacks* and *indistinguishability against chosen-ciphertext attacks*.

Indistinguishability against Chosen-plaintext Attacks

The most commonly discussed security notion, and the one that we use mainly throughout this thesis, is *indistinguishability against chosen-plaintext attacks* (IND-CPA). This notion basically models that an adversary that is not in possession of the secret key running in polynomial-time chooses two messages of the same length and has the ability to request any encryptions of other arbitrary messages via accessing an *encryption oracle*. Then one of the two messages is encrypted, and the ciphertext is returned to the adversary. The scheme is considered secure if the adversary is not able to distinguish which one of the two messages of its choice was encrypted. Informally, the encryption scheme should hide all information about the underlying plaintext such that a ciphertext does not reveal anything about which message was encrypted.

The IND-CPA notion for a symmetric encryption scheme is formally defined in Figure 2.1. The game begins with the challenger running the KeyGen algorithm to generate a challenge key k and sampling a bit b uniformly at random. Next the adversary is called with the security parameter as input, and it is given access to a "left-or-right" (LoR) encryption oracle $\mathcal{O}^{\mathsf{LoR}}$. The oracle inputs a pair of messages and first checks whether the messages are of equal length. In case this check is positive then it chooses one of the messages according to the sampled bit b and outputs the encryption of m_b under the challenge key k which is then given to the adversary. That is, if $b = 0$ then the adversary receives an encryption of the "left" plaintext, and if $b = 1$ it receives an

encryption of the "right" plaintext. Eventually, the adversary returns a bit b' and the game outputs 1 indicating that b' corresponds to the bit b chosen by the challenger, otherwise it returns 0. This also encompasses the adversary's access to a simple encryption oracle, since the adversary can simply query $\mathcal{O}^{\mathsf{LoR}}(m, m, k, b)$ to obtain $\mathsf{Encrypt}(m, k)$.

$\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{A}}\left[\mathcal{SE}, 1^\lambda\right]$

1 : $k \leftarrow\$ \mathsf{KeyGen}(1^\lambda)$
2 : $b \leftarrow\$ \{0,1\}$
3 : $b' \leftarrow\$ \mathcal{A}^{\mathcal{O}^{\mathsf{LoR}}(\cdot,\cdot,k,b)}(1^\lambda)$
4 : **if** $b' = b$ **then**
5 : **return** 1
6 : **else return** 0

$\mathcal{O}^{\mathsf{LoR}}(m_0, m_1, k, b)$

1 : **if** $(|m_0| \neq |m_1|)$ **then**
2 : **return** $\perp$
3 : **else**
4 : **return** $\mathsf{Encrypt}(m_b, k)$

Figure 2.1: The IND-CPA experiment $\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{A}}\left[\mathcal{SE}, 1^\lambda\right]$

Note that we can provide the adversary with a little more power by letting it choose a whole sequence of pairs of equal-length messages. This approach is similar and has the advantage of modelling attackers that can *adaptively* choose plaintexts to be encrypted even after observing previous ciphertexts.

We define the advantage of an adversary to be the difference between the probability of the adversary guessing the bit b correctly which indicates which message was encrypted and the probability of a random guess.

Definition 2.3. *The* advantage *of a PPT adversary in the* IND-CPA *game for a symmetric encryption scheme $\mathcal{SE}$ is defined as:*

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathcal{SE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{A}}\left[\mathcal{SE}, 1^\lambda\right] \rightarrow 1\right] - \frac{1}{2}.$$

We say that the symmetric encryption scheme $\mathcal{SE}$ is IND-CPA *secure if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathcal{SE}}(1^\lambda) \leq \text{negl}(\lambda).$$

Indistinguishability against Chosen-ciphertext Attacks

Similarly to the above notion, one can consider the security notion of *indistinguishability against chosen-ciphertext attacks* (IND-CCA) for a symmetric encryption scheme as formally defined in Figure 2.2. In the IND-CCA notion the adversary is not only able to obtain arbitrary encryptions of her choice, but also may request decryptions of

ciphertexts of her choice. The adversary's access to the decryption oracle is unlimited *except* for the adversary is not allowed to request a decryption of the challenge ciphertext as this would lead to a trivial win and thus is an *illegitimate* query.

In more detail, the IND-CCA game proceeds similar to the IND-CPA game presented in Figure 2.1 with the difference that the challenger additionally maintains a set S of ciphertexts that is initially empty. The "left-or-right" (LoR) encryption oracle $\mathcal{O}^{\mathsf{LoR}}$ is modified in such a way that it additionally adds generated ciphertexts to the set S if the input messages are of equal length and distinct. Furthermore, the adversary is given access to a decryption oracle $\mathcal{O}^{\mathsf{Decrypt}}$ to which it can submit ciphertexts to recover respective plaintexts as long as the submitted ciphertext does not correspond to the challenge ciphertext in order to avoid a trivial win. More precisely, in case the queried ciphertext corresponds to a previously generated output from the LoR encryption oracle $\mathcal{O}^{\mathsf{LoR}}$ then the decryption oracle $\mathcal{O}^{\mathsf{Decrypt}}$ outputs an error symbol $\perp$. Obviously, given the decryption of the ciphertext, it would be possible to determine the value of b based on which message is returned.

$\mathbf{Exp}^{\text{IND-CCA}}_{\mathcal{A}}\left[\mathcal{SE}, 1^\lambda\right]$

1 : $k \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$
2 : $b \leftarrow\!\!\$\ \{0,1\}$
3 : $S \leftarrow \emptyset$
4 : $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{LoR}}(\cdot,\cdot,k,b)\mathcal{O}^{\mathsf{Decrypt}}(\cdot,k)}(1^\lambda)$
5 : **if** $b' = b$ **then**
6 : **return** 1
7 : **else return** 0

$\mathcal{O}^{\mathsf{LoR}}(m_0, m_1, k, b)$

1 : **if** $(|m_0| \neq |m_1|)$ **then**
2 : **return** $\perp$
3 : **else**
4 : $ct \leftarrow\!\!\$\ \mathsf{Encrypt}(m_b, k)$
5 : $S \leftarrow S \cup ct$
6 : **return** ct

$\mathcal{O}^{\mathsf{Decrypt}}(ct, k)$

1 : **if** $(ct \in S)$ **then**
2 : **return** $\perp$
3 : **else**
4 : **return** $\mathsf{Decrypt}(ct, k)$

Figure 2.2: The IND-CCA experiment $\mathbf{Exp}^{\text{IND-CCA}}_{\mathcal{A}}\left[\mathcal{SE}, 1^\lambda\right]$

Definition 2.4. *The* advantage *of a PPT adversary in the* IND-CCA *game for a symmetric encryption scheme $\mathcal{SE}$ is defined as:*

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathcal{SE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{IND-CCA}}_{\mathcal{A}}\left[\mathcal{SE}, 1^\lambda\right] \rightarrow 1\right] - \frac{1}{2}.$$

We say that the symmetric encryption scheme $\mathcal{SE}$ is IND-CCA *secure if for all PPT*

adversaries $\mathcal{A}$, it holds that

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\mathcal{SE}}(1^\lambda) \leq \text{negl}(\lambda).$$

2.2.2 Public-key Encryption Schemes

A *public-key encryption scheme* (PKE), or *asymmetric encryption scheme*, eases the above problem of agreeing on a common secret key. Instead, each entity is now equipped with two keys, namely a *public key* and a *private key*. An encryptor may use an entity's A public key to encrypt a message and send it to A. The entity A is now able to use her private key to decrypt messages initially encrypted under A's public key. Note that the encryptor is not required to know the decryption key. In this setting, the public key may be transmitted or published in the clear such that there is no need to initialise secure channels before transmitting a message. The main difference between public-key encryption and symmetric encryption is that the latter assumes complete secrecy of all cryptographic keys, whereas the former requires secrecy only for the private key.

More formally a public-key encryption scheme is described as follows.

Definition 2.5. *A* public-key encryption scheme *consists of the following three algorithms.*

- $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$: *this randomised algorithm takes as input the security parameter λ and outputs a key pair (pk, sk). We refer to the first of these keys as the* public key *and to the second as the* private key *(or* secret key*);*
- $ct \xleftarrow{\$} \mathsf{Encrypt}(m, pk)$: *this randomised algorithm takes as input the public key pk and a message m from some message space $\mathcal{M}$. It outputs a ciphertext ct;*
- $m \leftarrow \mathsf{Decrypt}(ct, sk)$: *this deterministic algorithm takes as input the private key sk and the ciphertext ct. It outputs the message m encrypted in the ciphertext ct.*

As in the previous section, we could extend the decryption algorithm to output a failure symbol $\perp$ in case it is presented with an invalid ciphertext.

Correctness of a public-key encryption scheme requires that for all security parameters and all messages, the decryption of an honestly generated ciphertext under an honestly generated key pair will return the correct message. This is captured more formally as follows.

Definition 2.6. *A public-key encryption scheme is* correct *if there exists a negligible function* negl *such that for every security parameter λ, every key pair (pk, sk) outputted by $\mathsf{KeyGen}(1^\lambda)$, and every message $m \in \mathcal{M}$ it holds that* $\Pr[\mathsf{Decrypt}(\mathsf{Encrypt}(m, pk), sk) \neq m] = \text{negl}(\lambda)$.

The main disadvantage of public-key encryption schemes is that they are significantly slower than symmetric encryption schemes. However, it is not easy to detail an exact comparison between both schemes since the relative efficiency depends on the concrete scheme under consideration as well as their implementation details. In fact, to overcome this problem, a symmetric encryption scheme is used in the public key setting to enhance the efficiency for the public-key encryption of long messages. Such an encryption scheme is called *hybrid encryption* and basically a message is encrypted using the symmetric key (called *data encapsulation mechanism*) while the symmetric key itself is encrypted under the public-key encryption scheme (called *key encapsulation mechanism*). In other words, the public-key encryption scheme is only used in order to encrypt a short symmetric key, whilst the more efficient symmetric key is used to encrypt the larger message.

In the context of public-key encryption schemes it is also possible to define security in terms of *indistinguishability against chosen-plaintext attacks* (IND-CPA) and *indistinguishability against chosen-ciphertext attacks* (IND-CCA). It is straightforward to obtain those security notions by adapting the syntax from the public-key encryption scheme (cf. Definition 2.5) into Figures 2.1 and 2.2.

2.3 Attribute-based Encryption

Attribute-based Encryption (ABE) is a public-key, functional encryption primitive that allows decryption of a ciphertext if and only if some policy formula is satisfied. This means that the encrypted data and the decrypting entity satisfy certain properties formulated in terms of attribute sets. As such, ABE is also well-suited for the cryptographic enforcement of attribute-based access control policies [93]. In general, we define a universe $\mathcal{U}$ of *attributes* which represent labels that may describe entities or data. We then form a set of attributes $A \subseteq \mathcal{U}$ and a *policy* (or *access structure*) $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \{\emptyset\}$ and decryption is successful if and only if $A \in \mathbb{A}$.

There exist several variants of ABE which we introduce in the course of this section. These different notions mainly differ in the connection between attribute sets and access structures to ciphertexts and decryption keys.

2.3.1 Key-policy Attribute-based Encryption

In *key-policy attribute-based encryption* (KP-ABE)[89], the decryption key is associated with a policy and each ciphertext is associated with a set of attributes. In more detail, each secret decryption key is associated with an access structure, that is some family of satisfying attribute sets $\mathbb{A} = \{A_1, \ldots, A_n\}$, while each ciphertext is generated using the system-wide public parameters and is associated with a single subset of attributes

A. Decryption succeeds if the decryption key associated with the access structure $\mathbb{A}$ includes the attribute set under which the ciphertext was generated, i.e. if $A_i = A$ for some $i \in [n]$. In most schemes the access structure is considered to be *monotonic*. This means that $A' \in \mathbb{A}$ whenever there exists $A \subset A'$ such that $A \in \mathbb{A}$. A notable non-monotonic scheme was presented by Ostrovsky et al. [112].

Definition 2.7. *A* key-policy attribute-based encryption (KP-ABE) scheme *consists of the following four algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm takes as input the security parameter λ and attribute universe $\mathcal{U}$ and generates public parameters pp and a master secret key mk;*
- $ct_A \xleftarrow{\$} \mathsf{Encrypt}(m, A, pp)$: *this randomised algorithm takes as input a message m, an attribute set A and public parameters pp, and outputs a ciphertext ct_A;*
- $sk_{\mathbb{A}} \xleftarrow{\$} \mathsf{KeyGen}(\mathbb{A}, mk, pp)$: *this randomised algorithm takes as input an access structure $\mathbb{A}$, the master secret key mk and public parameters pp. It generates a secret decryption key $sk_{\mathbb{A}}$ for this access structure;*
- $pt \leftarrow \mathsf{Decrypt}(ct_A, sk_{\mathbb{A}}, pp)$: *this algorithm takes as input the ciphertext ct_A associated with an attribute set A, a secret decryption key $sk_{\mathbb{A}}$ generated under an access structure $\mathbb{A}$ and the public parameters pp. The algorithm outputs a plaintext pt which corresponds to m if and only if the attribute set satisfies the access structure, $A \in \mathbb{A}$, or else corresponds to $\perp$ indicating that decryption failed.*

Note that we refer to the access policies used in KP-ABE as *objective* policies.

Definition 2.8. *A KP-ABE scheme is* correct *if for all messages $m \in \mathcal{M}$, access structures $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \{\emptyset\}$ and attribute sets $A \subseteq \mathcal{U}$ where $A \in \mathbb{A}$, it holds that*

$$\begin{aligned}\Pr[&(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U}),\\ &ct_A \xleftarrow{\$} \mathsf{Encrypt}(m, A, pp),\\ &sk_{\mathbb{A}} \xleftarrow{\$} \mathsf{KeyGen}(\mathbb{A}, mk, pp),\\ &pt \leftarrow \mathsf{Decrypt}(ct_A, sk_{\mathbb{A}}, pp)]\\ &= 1 - \mathrm{negl}(\lambda).\end{aligned}$$

In general security for ABE schemes can be classified into the categories *full* (or adaptive) and *selective* security. The full security notion is the ideal notion we wish to achieve. However, the selective security notion has usually been easier to formalise and to achieve. The notions mainly differ in whether the adversary receives the public parameters before or after outputting its challenge input choice. The selective notion enables the client (or user[2]) to partition the system into queries that it must be able to

[2]We will use the terms client and user interchangeably throughout this thesis.

answer and to queries it will not answer as some queries may be restricted in order to avoid a trivial win for the adversary. Therefore, the challenger is able to embed secrets for a reductive proof.

However, we do not provide the standard security properties for KP-ABE here as we will be interested to use an extended version of KP-ABE supporting *revocation* which we recap in Section 2.3.2. Note that it is easy to obtain the standard security properties for KP-ABE by "swapping" the access policy and attribute set in the IND-CPA and sIND-CPA security notions for ciphertext-policy attribute-based encryption (CP-ABE) as defined in Figure 2.4 and Figure 2.5.

2.3.2 Revocable Key-policy Attribute-based Encryption

The mechanism of *revocation* plays a central role in cryptography. In the setting of attribute-based encryption, users may hold keys for the same functionality, i.e. being provided with keys which grant certain access to objects. The revocation mechanism disables the given functionality for a certain user or group of users. For example, if an employee leaves the company and thus the internal system, it is necessary to revoke the employee from the system in order to stop her from further accessing data. In the framework of attribute-based encryption, either target specific attributes (to disable certain policies within the system) or specific users (to accommodate changing user populations) can be revoked. Throughout this thesis (e.g. in Chapters 3 and 5), we focus on the latter case as we aim to prevent cheating servers from participating in the system model at all.
Revocable ABE schemes can support two different modes [17]:

- *Direct revocation* allows users to specify a revocation list at the point of encryption. This means that periodic re-keying is not required but the encryptors must have knowledge of, or be able to choose, the current revocation list.
- *Indirect revocation* requires ciphertexts to be associated with a time period (as an additional attribute) and for a key authority to issue key update material at each time period which enables non-revoked users to update their key to be functional during that time period. A revoked user will not be able to use the update material and thus their key will not succeed at decrypting ciphertexts associated with the current time period attribute. With indirect revocation, users only need to know the current time attribute during encryption, but increased communication costs are incurred due to the dissemination of the key update material.

In this thesis, we will make use of the *indirectly revocable KP-ABE scheme* given by Attrapadung and Imai [17], itself a more formal definition of that given by

Boldyreva et al. [35].

An indirectly revocable KP-ABE scheme defines the universe of attributes to be $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}}$. In more detail, $\mathcal{U}_{\text{attr}}$ is the normal attribute universe for describing ciphertexts and forming access policies, $\mathcal{U}_{\text{time}}$ comprises attributes for time periods, and $\mathcal{U}_{\text{ID}}$ contains attributes encoding entity identities. More formally, an indirectly revocable KP-ABE scheme is presented in the following definition.

Definition 2.9. *An* indirectly revocable key-policy attribute-based encryption scheme *consists of the following algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm takes as input the security parameter λ and the universe of attributes $\mathcal{U}$ and outputs public parameters pp and master secret key mk;*
- $ct_{A,t} \xleftarrow{\$} \mathsf{Encrypt}(m, A, t, pp)$: *this randomised encryption algorithm inputs a message m, an attribute set $A \subset \mathcal{U}_{\text{attr}}$, the current time period $t \in \mathcal{U}_{\text{time}}$ and the public parameters pp. It outputs a ciphertext ct that is valid for time t;*
- $sk_{\mathsf{id},\mathbb{A}} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{id}, \mathbb{A}, mk, pp)$: *this randomised key generation algorithm takes as input an identity $\mathsf{id} \in \mathcal{U}_{\text{ID}}$ for a user, an access structure $\mathbb{A}$ encoding a policy, as well as the master secret key mk and public parameters pp. It outputs a decryption key $sk_{\mathsf{id},\mathbb{A}}$ for the user id;*
- $uk_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, mk, pp)$: *this randomised algorithm takes a revocation list $R \subseteq \mathcal{U}_{\text{ID}}$ containing the identities of revoked entities, the current time period t, as well as the master secret key mk and public parameters pp. It outputs updated key material $uk_{R,t}$;*
- $pt \leftarrow \mathsf{Decrypt}(ct_{A,t}, sk_{\mathsf{id},\mathbb{A}}, uk_{R,t}, pp)$: *this decryption algorithm takes as input a ciphertext ct, a decryption key $sk_{\mathsf{id},\mathbb{A}}$, an update key $uk_{R,t}$ and the public parameters pp. The algorithm outputs a plaintext pt which corresponds to the correct message m if and only if the attributes associated with ct satisfy $\mathbb{A}$* and *the value of t in the update key matches that specified during the encryption of ct, or outputs $\perp$ if the decryption failed.*

Correctness of a revocable KP-ABE scheme is defined as follows:

Definition 2.10. *A revocable KP-ABE scheme is* correct *if for all $m \in \mathcal{M}$, all $\mathsf{id} \in \mathcal{U}_{\text{ID}}$, all $R \subseteq \mathcal{U}_{\text{ID}}$, all $\mathbb{A} \subseteq 2^{\mathcal{U}_{\text{attr}}} \setminus \{\emptyset\}$, all $A \subset \mathcal{U}_{\text{attr}}$ and all $t \in \mathcal{U}_{\text{time}}$, if $A \in \mathbb{A}$ and*

$\mathsf{id} \notin R$, *it holds that*

$$\begin{aligned}\Pr[&(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U}),\\ &ct_{A,t} \xleftarrow{\$} \mathsf{Encrypt}(m, A, t, pp),\\ &sk_{\mathsf{id},\mathbb{A}} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{id}, \mathbb{A}, mk, pp),\\ &uk_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, mk, pp),\\ &m \leftarrow \mathsf{Decrypt}(ct_{A,t}, sk_{\mathsf{id},\mathbb{A}}, uk_{R,t}, pp)]\\ &= 1 - \mathrm{negl}(\lambda).\end{aligned}$$

The schemes [17, 35] use the Complete-subtree method to arrange users as the leaves of a binary tree such that the required key-update material can be reduced from the naive method of $\mathcal{O}(n - r)$, where n is the number of users and r is the number of revoked users, to $\mathcal{O}(r \log(\frac{n}{2}))$. This approach works as follows for a revocation list R. For a leaf node $l \in \mathcal{U}_{\mathrm{ID}}$, let $Path(l)$ be the set of nodes on the path between the root node and l inclusively. Then, for each $l \in R$, mark all nodes in $Path(l)$. Define $Cover(R)$ to be the set of all unmarked children of marked nodes, and generate update keys for these nodes.

Note that the time parameter in the above algorithms could be a literal clock value where all entities have access to some synchronised clock or network clock. In this case, re-keying must occur at every time period regardless of whether a revocation has occurred in the prior period. Alternatively, the time parameter could simply be a counter that is updated when a revocation takes place and the KeyUpdate algorithm is performed.

Attrapadung and Imai [17] defined several security notions for revocable KP-ABE schemes. In this thesis, the security property we consider for a revocable KP-ABE scheme is *indistinguishability against selective-target with semi-static query attack* (IND-sHRSS) which is formally defined in Figure 2.3.
This is a *selective* notion of security where the adversary must declare at the beginning of the game the set of attributes $(t^\star, A^\star)$ to be challenged upon. The challenger runs Setup and provides the adversary with the resulting public parameters. The adversary must choose a target revocation set $\overline{R}$ which is the set of entities that should be revoked at time $t^\star$. The nature of the *semi-static* notion requires that this revocation list is chosen before the adversary is given access to the KeyGen and KeyUpdate oracles as specified in Figure 2.3.
In order to prevent trivial wins, for a key generation query, the adversary may not query for any key $sk_{\mathsf{id},\mathbb{A}}$ where the target attribute set $A^\star$ satisfies $\mathbb{A}$ *and* the identity is not revoked at time $t^\star$. If the adversary would be allowed to query for this, it would hold a secret decryption key and would receive key update material (as the identity

$\mathbf{Exp}^{\text{IND-sHRSS}}_{\mathcal{A}}\left[\mathcal{KP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right]$

$(t^\star, A^\star) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda, \mathcal{U})$
$(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{U})$
$\overline{R} \leftarrow\!\!\$\ \mathcal{A}(pp)$
$(m_0, m_1) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,mk,pp),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,mk,pp)}(\overline{R}, pp)$
if $(|m_0| \neq |m_1|)$ **then return** 0
$b \leftarrow\!\!\$\ \{0,1\}$
$ct^\star \leftarrow\!\!\$\ \mathsf{Encrypt}(m_b, t^\star, A^\star, pp)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,mk,pp),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,mk,pp)}(ct^\star, \overline{R}, pp)$
if $b' = b$ **then**
  return 1
else return 0

$\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{id}, \mathbb{A}, mk, pp)$

if $(A^\star \in \mathbb{A})$ **and** $(\mathsf{id} \notin \overline{R})$ **then**
  return $\perp$
else $\mathsf{KeyGen}(\mathsf{id}, \mathbb{A}, mk, pp)$
  return $sk_{\mathsf{id},\mathbb{A}}$

$\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, mk, pp)$

if $(t = t^\star)$ **and** $(\overline{R} \not\subseteq R)$ **then**
  return $\perp$
else $\mathsf{KeyUpdate}(R, t, mk, pp)$
  return $uk_{R,t}$

Figure 2.3: The IND-sHRSS experiment $\mathbf{Exp}^{\text{IND-sHRSS}}_{\mathcal{A}}\left[\mathcal{KP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right]$

is not revoked) for the challenge time period and thus could successfully decrypt the challenge ciphertext.
Similarly, for a key update request, the adversary is prevented from learning an update key $uk_{R,t^\star}$ for the challenge time period $t^\star$ for a less restrictive revocation list R than the challenge list $\overline{R}$. Otherwise, the adversary could obtain an update key which could be combined with a queried secret key to form a functional decryption key for a server that the adversary claimed would be revoked.

As in a standard IND-CPA notion, the adversary outputs two messages and the challenger chooses one of them at random to encrypt and passes the resulting ciphertext to the adversary. The adversary is again provided with access to the oracles and even-

tually guesses which message was encrypted. The advantage of the adversary is given in the following definition.

Definition 2.11. *The* advantage *of a PPT adversary in the* IND-sHRSS *game for a revocable KP-ABE construction $\mathcal{KP}$-$\mathcal{ABE}$ is defined as:*

$$\mathbf{Adv}^{\text{IND-sHRSS}}_{\mathcal{A},\mathcal{KP}\text{-}\mathcal{ABE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{IND-sHRSS}}_{\mathcal{A}}\left[\mathcal{KP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right] \to 1\right] - \frac{1}{2}.$$

We say that the indirectly revocable KP-ABE scheme is secure in the sense of indistinguishability against selective-target with semi-static query attack (IND-sHRSS) *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{IND-sHRSS}}_{\mathcal{A},\mathcal{KP}\text{-}\mathcal{ABE}}(1^\lambda) \leq \text{negl}(\lambda).$$

We note that it is also possible to define a stronger, *full* notion of security whereby the adversary may receive the public parameters and may query the oracles *before* selecting the set of challenge attributes under the restriction that no such query would lead to a trivial win. To the best of our knowledge, current known primitives supporting indirect revocation in the KP-ABE setting only achieve the selective security notion as described above.

2.3.3 Ciphertext-policy Attribute-based Encryption

Ciphertext-policy attribute-based encryption (CP-ABE) [31] "behaves" conversely to KP-ABE. Here a ciphertext is associated with a policy while the set of attributes is assigned to the secret decryption key. More concretely, each ciphertext is associated with an access structure, i.e. some family of attribute sets $\mathbb{A} = \{A_1, \ldots, A_n\}$. Each private key is computed using the system-wide public parameters and is associated with a single subset of attributes A. Decryption succeeds if the ciphertext includes the attribute set under which the decryption key was generated.

Definition 2.12. *A* ciphertext-policy attribute-based encryption (CP-ABE) scheme *consists of the following four algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm takes as input the security parameter λ and attribute universe $\mathcal{U}$ and generates public parameters pp and a master secret key mk;*
- $ct_{\mathbb{A}} \xleftarrow{\$} \mathsf{Encrypt}(m, \mathbb{A}, pp)$: *this randomised algorithm takes as input a message m, an access structure $\mathbb{A}$ and public parameters pp, and outputs a ciphertext $ct_{\mathbb{A}}$;*
- $sk_A \xleftarrow{\$} \mathsf{KeyGen}(A, mk, pp)$: *this randomised algorithm takes as input an attribute set A, the master secret key mk and public parameters pp. It generates a secret decryption key sk_A for this attribute set;*

- $pt \leftarrow \mathsf{Decrypt}(ct_{\mathbb{A}}, sk_A, pp)$: *this algorithm takes as input the ciphertext* $ct_{\mathbb{A}}$ *generated under an access structure* $\mathbb{A}$*, a secret decryption key* sk_A *generated under an attribute set* A *and the public parameters* pp*. The algorithm outputs a plaintext* pt *which corresponds to* m *if and only if the attribute set satisfies the access structure,* $A \in \mathbb{A}$*, or else corresponds to* $\perp$ *indicating that decryption failed.*

Note that we refer to the access policies used in CP-ABE as *subjective* policies as the access structure describes objects for which the key may be used to access.

Definition 2.13. *A CP-ABE scheme is* correct *if for all messages* $m \in \mathcal{M}$*, access structures* $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \{\emptyset\}$ *and attribute sets* $A \subseteq \mathcal{U}$ *where* $A \in \mathbb{A}$*, it holds that*

$$\begin{aligned}\Pr[&(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U}),\\ &ct_{\mathbb{A}} \xleftarrow{\$} \mathsf{Encrypt}(m, \mathbb{A}, pp),\\ &sk_A \xleftarrow{\$} \mathsf{KeyGen}(A, mk, pp),\\ &pt \leftarrow \mathsf{Decrypt}(ct_{\mathbb{A}}, sk_A, pp)]\\ &= 1 - \mathrm{negl}(\lambda).\end{aligned}$$

The security goals of a CP-ABE scheme are similar to those of symmetric and public-key encryption schemes, namely the adversary should not be able to distinguish which of the two messages was encrypted.

Note that different CP-ABE keys provide access to different classes of documents and ciphertexts are not necessarily generated for a particular user but rather a class of users. Therefore, it is important to consider in ABE schemes that users may not collude with each other in order to decrypt a ciphertext which no one of them could have decrypted alone. For example in the CP-ABE setting, consider two users assigned with the following attributes sets {`Professor, Psychology`} and {`Student, Computer Science`} respectively, and the ciphertext was encrypted under the policy (`Professor` $\wedge$ `Computer Science`). Both users now should not be able to combine their attributes `Professor` and `Computer Science` in order to decrypt the ciphertext as neither of them satisfy the policy. To model collusion between clients into the security notion we provide the adversary with an additional KeyGen oracle such that it can request multiple decryption keys for different attribute sets. We require that all queried attribute sets do not satisfy the adversary's challenge policy because then the adversary could trivially win against the security game as it would hold a valid key and could decrypt the challenge ciphertext itself.

Full IND-CPA Notion

The first security property we consider here is the full notion of *indistinguishability against chosen-plaintext attacks* (IND-CPA) for a CP-ABE scheme represented in

$\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CPA}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^{\lambda}, \mathcal{U}\right]$

1 : $\mathbb{A}^{\star} \leftarrow \{\emptyset\}$
2 : $Q \leftarrow \epsilon$
3 : $(pp, mk) \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, \mathcal{U})$
4 : $(m_0, m_1, \mathbb{A}^{\star}) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, mk, pp)}(pp)$
5 : **if** $(|m_0| \neq |m_1|)$ **then return** 0
6 : **for all** $A \in Q$ **do**
7 : **if** $A \in \mathbb{A}^{\star}$ **then return** 0
8 : $b \leftarrow_{\$} \{0, 1\}$
9 : $ct^{\star} \leftarrow_{\$} \mathsf{Encrypt}(m_b, \mathbb{A}^{\star}, pp)$
10 : $b' \leftarrow_{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, mk, pp)}(ct^{\star}, pp)$
11 : **if** $b' = b$ **then**
12 : **return** 1
13 : **else return** 0

$\mathcal{O}^{\mathsf{KeyGen}}(A, mk, pp)$

1 : **if** $A \notin \mathbb{A}^{\star}$ **then**
2 : $Q \leftarrow Q \cup A$
3 : **return** $\mathsf{KeyGen}(A, mk, pp)$
4 : **else**
5 : **return** $\perp$

Figure 2.4: The IND-CPA experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CPA}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^{\lambda}, \mathcal{U}\right]$

Figure 2.4. The game begins with the challenger initialising the challenge access structure $\mathbb{A}^{\star}$ and a list Q of queried attribute sets which is initially set to be empty. The challenger then runs Setup and provides the adversary with the generated public parameters. The adversary now is equipped with access to a KeyGen oracle which returns to the adversary a valid secret decryption key for its choice of attribute set A but only if the attribute set does not satisfy the access structure $\mathbb{A}$. Otherwise, if the attribute set satisfies the access structure then the oracle returns $\perp$ in order to avoid allowing the adversary a trivial win. After a polynomial number of queries the adversary chooses the challenge access structure $\mathbb{A}^{\star}$ and also returns two messages of equal length. Note that in case the adversary returns messages of unequal length the game is aborted immediately and $\mathcal{A}$ loses. The adversary also loses the game if the outputted access structure $\mathbb{A}^{\star}$ is satisfied by any attribute set previously queried to the KeyGen oracle $\mathcal{O}^{\mathsf{KeyGen}}$ and therefore appears in Q as the adversary has not found a valid attack target. The challenger chooses a bit b uniformly at random and uses this to determine which message will be encrypted under the challenge access structure $\mathbb{A}^{\star}$. The adversary is provided with the created ciphertext and is again allowed to access the KeyGen oracle $\mathcal{O}^{\mathsf{KeyGen}}$ as before. Eventually, the adversary outputs a guess b' of b to determine which message was encrypted. If the guess was correct the adversary wins and the game outputs 1, and 0 otherwise.

Definition 2.14. *The* advantage *of a PPT adversary in the* IND-CPA *game for a*

CP-ABE construction $\mathcal{CP}$-$\mathcal{ABE}$ is defined as:

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathcal{CP}\text{-}\mathcal{ABE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{IND-CPA}}_{\mathcal{A}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right] \to 1\right] - \frac{1}{2}.$$

We say that the scheme $\mathcal{CP}$-$\mathcal{ABE}$ is IND-CPA *secure if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathcal{A},\mathcal{CP}\text{-}\mathcal{ABE}}(1^\lambda) \leq \text{negl}(\lambda).$$

Selective IND-CPA Notion

$\mathbf{Exp}^{\text{sIND-CPA}}_{\mathcal{A}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right]$

1 : $\mathbb{A}^\star \leftarrow \mathcal{A}(1^\lambda, \mathcal{U})$
2 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mathcal{U})$
3 : $(m_0, m_1) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, mk, pp)}(pp)$
4 : **if** $(|m_0| \neq |m_1|)$ **then return** 0
5 : $b \leftarrow_\$ \{0, 1\}$
6 : $ct^\star \leftarrow_\$ \mathsf{Encrypt}(m_b, \mathbb{A}^\star, pp)$
7 : $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, mk, pp)}(ct^\star, pp)$
8 : **if** $b' = b$ **then**
9 : **return** 1
10 : **else return** 0

$\mathcal{O}^{\mathsf{KeyGen}}(A, mk, pp)$

1 : **if** $A \notin \mathbb{A}^\star$ **then**
2 : **return** $\mathsf{KeyGen}(A, mk, pp)$
3 : **else**
4 : **return** $\perp$

Figure 2.5: The selective IND-CPA experiment $\mathbf{Exp}^{\text{sIND-CPA}}_{\mathcal{A}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right]$

We also consider the *selective* IND-CPA notion (sIND-CPA) represented in Figure 2.5. The selective notion is similar to the full notion while the main difference is that the game starts with the adversary selecting an access structure $\mathbb{A}^\star$. Otherwise the game proceeds similarly to the full version.

Definition 2.15. *The* advantage *of a PPT adversary in the selective* IND-CPA *game for a CP-ABE construction $\mathcal{CP}$-$\mathcal{ABE}$ is defined as:*

$$\mathbf{Adv}^{\text{sIND-CPA}}_{\mathcal{A},\mathcal{CP}\text{-}\mathcal{ABE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{sIND-CPA}}_{\mathcal{A}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right] \to 1\right] - \frac{1}{2}.$$

We say that the scheme $\mathcal{CP}$-$\mathcal{ABE}$ is sIND-CPA *secure if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sIND-CPA}}_{\mathcal{A},\mathcal{CP}\text{-}\mathcal{ABE}}(1^\lambda) \leq \text{negl}(\lambda).$$

2.3.4 Dual-policy Attribute-based Encryption

Dual-policy attribute-based encryption was introduced by Attrapadung and Imai [19]. This scheme combines both approaches of KP-ABE (enforcing objective policies) and CP-ABE (enforcing subjective policies) such that both the ciphertext and the decryption key comprise an attribute set and an access structure. In more detail, the ciphertext is associated with a subjective policy specifying which entities may decrypt it and an objective attribute set describing the data, while the decryption key is associated with an objective policy and a subjective attribute set. Finally, decryption is successful if and only if *both* attribute sets satisfy their access policy, respectively.

Definition 2.16. *A* dual-policy attribute-based encryption (DP-ABE) scheme *consists of the following four algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm takes as input the security parameter λ and attribute universe $\mathcal{U}$ and generates public parameters pp and a master secret key mk which is kept private by the client;*
- $ct_{\omega,\mathbb{S}} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), pp)$: *this randomised algorithm takes as input a message m, an objective attribute set ω, a subjective access policy $\mathbb{S}$ and the public parameters pp. It outputs a ciphertext $ct_{\omega,\mathbb{S}}$;*
- $sk_{\mathbb{O},\psi} \xleftarrow{\$} \mathsf{KeyGen}((\mathbb{O}, \psi), mk, pp)$: *this randomised algorithm takes as input an objective access policy $\mathbb{O}$, a subjective attribute set ψ, the master secret key mk and the public parameters pp. It generates a secret decryption key $sk_{\mathbb{O},\psi}$;*
- $pt \leftarrow \mathsf{Decrypt}(ct_{\omega,\mathbb{S}}, sk_{\mathbb{O},\psi}, pp)$: *this algorithm takes as input the ciphertext $ct_{\omega,\mathbb{S}}$, the decryption key $sk_{\mathbb{O},\psi}$ and the public parameters pp. The algorithm outputs a plaintext pt which corresponds to the correct message m if and only if the set of objective attributes ω satisfies the objective access policy $\mathbb{O}$ and the set of subjective attributes ψ satisfies the subjective access policy $\mathbb{S}$, i.e. $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$. Otherwise, pt corresponds to $\perp$ indicating that decryption failed.*

Note we assume that the policies and attributes are implicit from the relevant keys and ciphertexts. Otherwise, these can be given as additional arguments to the decryption algorithm.

Definition 2.17. *A DP-ABE scheme is* correct *if for all messages $m \in \mathcal{M}$, for all access structures $\mathbb{O}, \mathbb{S} \subseteq 2^{\mathcal{U}} \setminus \{\emptyset\}$ and for all attribute sets $\omega, \psi \subseteq \mathcal{U}$ where $\omega \in \mathbb{O}$ and*

$\psi \in \mathbb{S}$, it holds that

$$\begin{aligned}\Pr[&(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U}),\\ &ct_{\omega,\mathbb{S}} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), pp),\\ &sk_{\mathbb{O},\psi} \xleftarrow{\$} \mathsf{KeyGen}((\mathbb{O}, \psi), mk, pp),\\ &pt \leftarrow \mathsf{Decrypt}(ct_{\omega,\mathbb{S}}, sk_{\mathbb{O},\psi}, pp)]\\ &= 1 - \mathrm{negl}(\lambda).\end{aligned}$$

Selective IND-CPA Notion

Security for DP-ABE is defined similarly to the security notion of CP-ABE. The selective security notion is defined in Figure 2.6. As before, it is easy to define an adaptive notion of security by providing the adversary with the public parameters before selecting the challenge input. Note that the respective notions of sIND-CPA for both KP-ABE and CP-ABE can be obtained by ignoring the relevant attribute sets and access structures in the formalisation below.

$\mathbf{Exp}^{\text{sIND-CPA}}_{\mathcal{A}}\left[\mathcal{DP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right]$

1 : $(\omega^\star, \mathbb{S}^\star) \leftarrow \mathcal{A}(1^\lambda, \mathcal{U})$
2 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mathcal{U})$
3 : $(m_0, m_1, \mathbb{A}^\star) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}((\cdot,\cdot), mk, pp)}(pp)$
4 : **if** $(|m_0| \neq |m_1|)$ **then return** 0
5 : $b \leftarrow_\$ \{0, 1\}$
6 : $ct^\star \leftarrow_\$ \mathsf{Encrypt}(m_b, (\omega^\star, \mathbb{S}^\star), pp)$
7 : $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}((\cdot,\cdot), mk, pp)}(ct^\star, pp)$
8 : **if** $b' = b$ **then**
9 : **return** 1
10 : **else return** 0

$\mathcal{O}^{\mathsf{KeyGen}}((\mathbb{O}, \psi), mk, pp)$

1 : **if** $(\omega^\star \notin \mathbb{O})$ **or** $(\psi \notin \mathbb{S}^\star)$ **then**
2 : **return** $\mathsf{KeyGen}((\mathbb{O}, \psi), mk, pp)$
3 : **else**
4 : **return** $\perp$

Figure 2.6: The selective IND-CPA experiment $\mathbf{Exp}^{\text{sIND-CPA}}_{\mathcal{A}}\left[\mathcal{DP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right]$

Definition 2.18. *The* advantage *of a PPT adversary in the selective* IND-CPA *game for a DP-ABE construction $\mathcal{DP}$-$\mathcal{ABE}$ is defined as:*

$$\mathbf{Adv}^{\text{sIND-CPA}}_{\mathcal{A},\mathcal{DP}\text{-}\mathcal{ABE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{sIND-CPA}}_{\mathcal{A}}\left[\mathcal{DP}\text{-}\mathcal{ABE}, 1^\lambda, \mathcal{U}\right] \rightarrow 1\right] - \frac{1}{2}.$$

We say that the scheme $\mathcal{DP}$-$\mathcal{ABE}$ is sIND-CPA *secure if for all PPT adversaries $\mathcal{A}$,*

it holds that

$$\mathbf{Adv}^{\text{sIND-CPA}}_{\mathcal{A},\mathcal{DP}\text{-}\mathcal{ABE}}(1^\lambda) \leq \text{negl}(\lambda).$$

2.3.5 Instantiation of Attribute-based Encryption Schemes

In the instantiation of many ABE schemes a secret value is chosen uniformly at random during either the key generation or the encryption and use a *linear secret sharing scheme* to divide the secret over a set of attributes or clauses in a policy. Those schemes then use *Lagrange Interpolation* to reconstruct the secret if and only if a satisfying set of attributes is provided to the decryption procedure. Additionally, many ABE schemes are built using bilinear maps which give rise to hardness assumptions based on the Diffie-Hellman problem in bilinear groups.

In this section, we provide tools used in instantiating an ABE scheme as we introduce a new ABE scheme in Section 5.4.

2.3.5.1 Linear Secret Sharing Schemes

Secret sharing is a basic and fundamental cryptographic tool that enables a secret s to be shared amongst a set of entities in such a way that all *authorised* sets of entities can combine their individual share in order to reconstruct the secret value s. For example, any k out of the n entities may form an authorised set and thus are able to reconstruct the secret value. Any set of entities that does not form an authorised set cannot learn more than their individual shares and thus cannot reconstruct the secret s. A secret sharing scheme is *linear* if the reconstruction operation is a linear function of the shares and note that almost all known secret sharing schemes are linear [24].

We provide a definition of an *access structure* which is a collection of satisfying sets of a Boolean formula. Beimel [24] provides an equivalent generic formulation as follows.

Definition 2.19. *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of parties (or attributes). A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if for all B, C we have that if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotonic access structure) is a collection (respectively, monotone collection) $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the* authorised sets *and the sets not in $\mathbb{A}$ are called* unauthorised sets.

A linear secret sharing scheme can be defined as follows [142].

Definition 2.20. *Let $\mathcal{P}$ be a set of parties. Let M be a matrix of size $l \times k$. Let $\pi \colon \{1, \ldots, l\} \to \mathcal{P}$ be a function that maps a row to a party for labelling. A secret sharing scheme Π for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$ is a linear secret-sharing scheme (LSSS) in $\mathbb{Z}_p$ and is represented by (M, π) if it consists of two polynomial-time algorithms:*

- $M\boldsymbol{v} \xleftarrow{\$} \mathsf{Share}(s, (M, \pi))$*: this randomised algorithm takes as input* $s \in \mathbb{Z}_p$ *which is to be shared and the LSSS* (M, π)*. It randomly chooses* $y_2, \ldots, y_k \in \mathbb{Z}_p$ *and sets* $\boldsymbol{v} = (s, y_2, \ldots, y_k)$*. It outputs* $M\boldsymbol{v}$ *as a vector of* l *shares. The share* $\lambda_{\pi(i)} := \boldsymbol{M_i} \cdot \boldsymbol{v}$ *belongs to party* $\pi(i)$*, where we denote* $\boldsymbol{M_i}$ *as the* i*th row in* M*.*

- $\{(i, \mu_i)\}_{i \in I} \leftarrow \mathsf{Recon}(S, \{\lambda_{\pi(i)}\}_{\pi(i) \in S}, (M, \pi))$*: this algorithm takes as input an authorised set* $S \in \mathbb{A}$*, the set of shares for this set* $\{\lambda_{\pi(i)}\}_{\pi(i) \in S}$ *and the LSSS* (M, π)*. Let* $I = \{i : \pi(i) \in S\}$*. It outputs reconstruction constants* $\{(i, \mu_i)\}_{i \in I}$ *such that the secret can be linearly reconstructed as* $s = \sum_{i \in I} \mu_i \cdot \lambda_{\pi(i)}$*.*

Note that the set $\{(\mu_i)\}_{i \in I}$ can be found in polynomial-time in the size of M [24, 142].

In Section 5.4, we will require the following important fact [142]:

Proposition 2.21. *Let* (M, π) *be a LSSS for access structure* $\mathbb{A}$ *over a set of parties* $\mathcal{P}$*, where* M *is a matrix of size* $l \times k$*. For any authorised set* $S \in \mathbb{A}$*, the target vector* $(1, 0, \ldots, 0)$ *is in the span of* $I = \{i : \pi(i) \in S\}$*. For all unauthorised sets* $S \notin \mathbb{A}$*, the target vector is not in the span of* I*, and there exists a polynomial time algorithm that outputs a vector* $\boldsymbol{w} = (w_1, \ldots, w_k) \in \mathbb{Z}_p^k$ *such that* $w_1 = -1$ *and for all* $i \in I$ *it holds that* $\boldsymbol{M_i} \cdot \boldsymbol{w} = 0$*.*

In Section 5.4, we make use of Lagrange interpolation as the reconstruction algorithm for LSSSs. The reconstruction procedure can be defined following Attrapadung and Imai [17] in the following way.

Definition 2.22. *For* $i \in \mathbb{Z}$ *and* $S \subseteq \mathbb{Z}$*, the Lagrange basis polynomial is defined as* $\Delta_{i,S}(z) = \prod_{j \in S, j \neq i} \frac{z-j}{i-j}$*. Let* $f(z) \in \mathbb{Z}[z]$ *be a* d*th degree polynomial. If* $|S| = d + 1$*, from a set of* $d + 1$ *points* $\{(i, f(i))\}_{i \in S}$*, one can reconstruct* $f(z)$ *as*

$$f(z) = \sum_{i \in S} f(i) \cdot \Delta_{i,S}(z).$$

In Section 5.4, we especially use the interpolation for a first degree polynomial. In particular, let $f(z)$ *be a first degree polynomial, one can obtain* $f(0)$ *from two points* $(i_1, f(i_1)), (i_2, f(i_2))$ *where* $i_1 \neq i_2$ *by computing*

$$f(0) = f(i_1)\frac{i_2}{i_2 - i_1} + f(i_2)\frac{i_1}{i_1 - i_2}.$$

2.3.5.2 Bilinear Maps and Hardness Assumptions

Most ABE schemes are instantiated over groups with efficiently computable bilinear maps. Thus, we review the notions of bilinear maps and the hardness assumption on which we base the security of our revocable DP-ABE scheme in Section 5.4. We follow the formalisation in [17, 18].

Definition 2.23. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative groups of order p, and let g be a generator of $\mathbb{G}$. A* bilinear map *is a map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that:*

1. *e is bilinear: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$ we have $e(u^a, v^b) = e(u, v)^{ab}$*
2. *e is non-degenerate: $e(g, g) \neq 1$*

We say that $\mathbb{G}$ is a bilinear group if the group action in $\mathbb{G}$ can be computed efficiently and there exists $\mathbb{G}_T$ for which $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is efficiently computable.

Definition 2.24. *Let $\mathbb{G}$ be a bilinear group of prime order p. The* decisional q-bilinear Diffie-Hellman exponent problem *(q-BDHE) in $\mathbb{G}$ is stated as follows. Given a vector*

$$\left(g, h, g^a, g^{(a^2)}, \ldots, g^{(a^q)}, g^{(a^{q+2})}, \ldots, g^{(a^{2q})}, Z\right) \in \mathbb{G}^{2q+1} \times \mathbb{G}_T$$

as input, determine whether $Z = e(g, h)^{a^{q+1}}$. We write g_i to denote $g^{a^i} \in \mathbb{G}$. Let $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$. An algorithm $\mathcal{A}$ that outputs $b \in \{0, 1\}$ has advantage ϵ in solving the decisional q-BDHE problem in $\mathbb{G}$ if

$$|\Pr[\mathcal{A}\left(g, h, \boldsymbol{y}_{g,a,q}, e(g_{q+1}, h)\right) \to 0] - \Pr[\mathcal{A}(g, h, \boldsymbol{y}_{g,a,q}, Z) \to 0]| \geq \epsilon,$$

where the probability is over the random choices of generators and groups $g, h \in \mathbb{G}$, $a \in \mathbb{Z}_p$, $Z \in \mathbb{G}_T$, and the randomness of $\mathcal{A}$. We refer to the distribution on the left as $\mathcal{P}_{BDHE}$ and the one on the right as $\mathcal{R}_{BDHE}$. The decisional q-BDHE assumption holds in $\mathbb{G}$ if no polynomial-time $\mathcal{A}$ has a non-negligible advantage in solving the problem.

2.3.5.3 Terminology for Binary Trees

A binary tree is a directed, rooted tree in which each node has at most two children such that there exists a unique path from the root to each node. Let $\mathcal{L} = \{1, \ldots, n\}$ be the set of leaves of a complete binary tree. Let $\mathcal{X}$ be the set of node names via some systematic naming order. For a leaf $i \in \mathcal{L}$, let $\mathsf{Path}(i) \subset \mathcal{X}$ be the set of nodes on the path from node i to the root (including i and the root). For $R \subseteq \mathcal{L}$, let $\mathsf{Cover}(R) \subset \mathcal{X}$ be defined as follows. First mark all the nodes in $\mathsf{Path}(i)$ if $i \in R$. Then $\mathsf{Cover}(R)$ is the set of all unmarked children of marked nodes. It can be shown to be the minimal set that contains no node in $\mathsf{Path}(i)$ if $i \in R$ but contains at least one node in $\mathsf{Path}(i)$ if $i \notin R$.

2.4 Searchable Encryption

Searchable encryption (SE) is a cryptographic primitive that enables a client to search over *encrypted* data that has been outsourced to an untrusted server. The untrusted server receives a query to perform a search over the encrypted data on behalf of the

client without learning information about the underlying plaintexts. The standard entity population considered in SE consists of a data owner and a remote server. Here the data owner is responsible to initialise the system and prepares her database to be stored at the remote server. Depending on the motivation, sometimes the scheme also accommodates a group of users (representing a multi-user SE scheme) that get authorised by the data owner in order to form search queries over the data too.

The seminal paper by Song et al. [132] introduced a cryptographic solution to the problem of searching on encrypted data. Goh [77] introduced a SE construction that associates an *index* to each document in a collection which enables a server to search each of the indexes for keywords. An index can be seen as a data structure that enables a querier with a trapdoor for a word w to test whether the index contains w. The index does not reveal any content about the documents without a valid trapdoor which can only be generated by the data owner holding a secret key.

We review the definition of a (single-client, index-based) searchable symmetric encryption scheme following Curtmola et al. [57]. The model includes a user that wishes to store an encrypted document collection $D = (D_1, \ldots, D_n)$ on a server, while preserving the ability to search through them. A document collection D is a subset of the set of all possible documents 2^{Δ} where Δ represents a dictionary of d words. More formally a SSE scheme can be defined as follows.

Definition 2.25. *A* searchable symmetric encryption (SSE) scheme *consists of the following four algorithms:*

- $k \xleftarrow{\$} \mathsf{KeyGen}(1^{\lambda})$: *this randomised algorithm is run by the user to initialise the system. It takes a security parameter λ as input, and returns a secret key k such that the length of k is polynomially bounded in λ;*
- $\mathcal{I} \leftarrow \mathsf{BuildIndex}(D, k)$: *this (possibly randomised) algorithm is run by the user to generate indexes. It takes a secret key k and a document collection D as inputs, and returns an index $\mathcal{I}$;*
- $T_w \leftarrow \mathsf{Trapdoor}(w, k)$: *this algorithm is run by the user to generate a trapdoor for a given word. It takes a secret key k and a word w as input, and returns a trapdoor T_w;*
- $D(w) \leftarrow \mathsf{Search}(T_w, \mathcal{I})$: *this algorithm is run by the server in order to search for the documents in D that contain the word w. It takes an index $\mathcal{I}$ for a collection D and a trapdoor T_w for word w as inputs, and returns the set of identifiers of documents containing w denoted by $D(w)$.*

Definition 2.26. *A searchable symmetric encryption (SSE) scheme is* correct *if for every security parameter λ, every key k outputted by $\mathsf{KeyGen}(1^{\lambda})$, every word $w \in D \subseteq 2^{\Delta}$,*

and every index $\mathcal{I}$ *outputted by* $\mathsf{BuildIndex}(D, k)$*, it holds that* $\mathsf{Search}(\mathsf{Trapdoor}(w, k), \mathcal{I}) = D(w)$.

Security for searchable encryption is typically characterised as the requirement that nothing is leaked beyond the outcome of the search. However, it is not straightforward to achieve this notion of security. Curtmola et al. [57] point out that the above notion can be achieved in its generality using the work of Goldreich and Ostrovsky on *oblivious RAM* [78]. Most SE schemes leak additionally to the search outcome also the search pattern as for example in [77, 132]. More accurately SE schemes try to formalise security in terms of nothing is leaked beyond the outcome of the search, the search pattern and the access pattern. The first notion of security in SE was introduced by Goh [77] who proposed the notion of *indistinguishability against chosen-keyword attacks* (IND1-CKA). This notion ensures that an adversary is not able to deduce any information about the document's content from its index. In other words this means for example that given two encrypted documents of equal size and an index, then an adversary is not able to decide which document is encoded in the index. Chang and Mitzenmacher [49] introduced a stronger simulation-based IND-CKA notion that is a stronger notion compared to IND1-CKA in the sense that this formalises that an adversary cannot even distinguish indexes from two documents of unequal sizes. Additionally to this, Goh introduced the IND2-CKA security notion that protects documents of unequal size like Chang and Mitzenmacher [49].

Curtmola et al. [57] revisited the existing security definitions and provided a discussion about previous security definitions not being adequate for SSE. They pointed out that the security of indexes and the security of trapdoors are inherently linked. They introduce two new adversarial models, a non-adaptive IND-CKA1 and an adaptive IND-CKA2 notion of security, which are widely used as the standard security definitions for SSE. Those security notions include security for the trapdoors and guarantee that the trapdoors do not leak any information about the keywords other than what can be inferred from the search and access pattern anyway.

Boneh et al. [36] discuss security in the public-key setting. In this setting, the security definition guarantees that no information about the keywords is leaked unless the respective trapdoors are available. Also other security definitions were introduced in the literature for SE and a detailed overview can be found in [38].

The schemes in [36, 57, 77, 99, 101, 105, 115, 139] make use of an index in order to enable the server to execute a search query over the encrypted data (documents). The work by Chai and Gong [48] introduced a verification mechanism into the setting of SSE, and thus enables a user to verify returned search results from a single keyword equality query. Liu et al. [107] extend the model of Boneh et al. [36] to support verification of

search results from a single keyword equality query, where the indexes are created using a public key. Sun et al. [135] and Wang et al. [141] detail VSE schemes with enhanced functionality, i.e. verifiable multi-keyword ranked search and verifiable fuzzy keyword search, respectively. Gajek [71] introduced a dynamic SSE scheme using a novel cryptographic tool called constrained functional encryption. Recently, Zhang et al. [147] show that file injection attacks on the query of single-keyword and conjunctive SE schemes can reveal the client's queries in their entirety.

2.5 Digital Signatures

Digital signatures provide a proof of message integrity and authenticate data's origin (since keys can be associated to particular users). We require a message to be signed using a private signing key owned by a particular entity, and in order to verify that the signature was actually generated using the given signing key a public verification key is used. If the verification process is successful then this shows that the contents of the message have not changed since the signature was computed. We will use this primitive (e.g. in Chapter 3) to provide a means of validating that the result of a computation was computed by the claimed server and that it has not been maliciously altered. We follow here the formalisation from [100].

Definition 2.27. *A* digital signature scheme SIG *comprises the following three polynomial-time algorithms:*

- $(sk, vk) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$*: this randomised algorithm takes as input the security parameter* λ *and generates a signing key* sk *and a verification key* vk*;*
- $\gamma \xleftarrow{\$} \mathsf{Sig.Sign}(m, sk)$*: this randomised algorithm takes as input a message* m *to be signed and the signing key* sk*, and outputs a signature* γ *of* m*;*
- $\delta \leftarrow \mathsf{Sig.Verify}(m, \gamma, vk)$*: this algorithm takes as input a message* m *and corresponding signature* γ *to be verified as well as the verification key* vk*, and outputs a decision* δ *which corresponds to* `accept` *if* γ *is a valid signature on* m *and* `reject` *otherwise.*

Definition 2.28. *A digital signature scheme is* correct *if for every security parameter* λ*, every key pair* (sk, vk) *outputted by* $\mathsf{KeyGen}(1^\lambda)$*, and every message* $m \in \mathcal{M}$ *it holds* $\mathsf{Sig.Verify}(m, \mathsf{Sig.Sign}(m, sk), vk) = \texttt{accept}$ *except possibly with negligible probability.*

We define a signature scheme to be *existentially unforgeable under an adaptive chosen message attack* (EUF-CMA) if an adversary, given polynomially many signatures on messages of its choice, cannot create a message $m^\star$ with a valid signature where $m^\star$ was not one of the messages the adversary was allowed to obtain a signature for. More formally, this is captured in Figure 2.7.

$\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}\left[\mathsf{SIG}, 1^{\lambda}\right]$

$Q \leftarrow \emptyset$
$(sk, vk) \leftarrow \mathsf{Sig.KeyGen}(1^{\lambda})$
$(m^{\star}, \gamma^{\star}) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sig.Sign}}(\cdot, sk)}(vk)$
if $\mathtt{accept} \leftarrow \mathsf{Sig.Verify}(m^{\star}, \gamma^{\star}, vk)$ **and** $m^{\star} \notin Q$ **then**
  return 1
else return 0

$\mathcal{O}^{\mathsf{Sig.Sign}}(m, sk)$

$Q \leftarrow Q \cup m$
return $\mathsf{Sig.Sign}(m, sk)$

Figure 2.7: The signature experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}\left[\mathsf{SIG}, 1^{\lambda}\right]$

Definition 2.29. *The* advantage *of an adversary $\mathcal{A}$ running in* PPT *is defined as:*

$$\mathbf{Adv}_{\mathcal{A},\mathsf{SIG}}^{\text{EUF-CMA}}(1^{\lambda}) = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}\left[\mathsf{SIG}, 1^{\lambda}\right] \rightarrow 1\right].$$

A digital signature scheme SIG *is* existentially unforgeable under an adaptive chosen-message attack (EUF-CMA)*, or just* secure*, if for all* PPT *adversaries $\mathcal{A}$,*

$$\mathbf{Adv}_{\mathcal{A},\mathsf{SIG}}^{\text{EUF-CMA}}(1^{\lambda}) \leq \text{negl}(\lambda).$$

2.6 One-way Functions

A *one-way function* g is characterised by having the properties of being easy to compute, but hard to invert. The first condition is given by the requirement that g is computable in polynomial time. The second condition is formalised by requiring that it is infeasible for any probabilistic polynomial-time algorithm to invert g (that is, to find a pre-image of a given value y) except with negligible probability. This requirement will be captured in the *inverting experiment* formally represented in Figure 2.8 where we consider the experiment for any algorithm $\mathcal{A}$, any value λ for the security parameter, and the function $g\colon \{0,1\}^{*} \rightarrow \{0,1\}^{*}$. Note that it suffices for $\mathcal{A}$ to find any value of w' for which $g(w') = z = g(w)$ in the experiment.
The following definition specifies what it means for a function g to be one-way [100].

Definition 2.30. *A function $g\colon \{0,1\}^{*} \rightarrow \{0,1\}^{*}$ is* one-way *if the following two conditions hold.*

- (Easy to compute.) *There exists a polynomial-time algorithm M_g computing g; i.e. $M_g(w) = g(w)$ for all w.*

$\mathbf{Exp}_{\mathcal{A}}^{\text{INVERT}}\left[g, 1^{\lambda}\right]$

1 : $w \leftarrow \{0,1\}^{\lambda}$
2 : $z = g(w)$
3 : $w' \leftarrow \mathcal{A}(1^{\lambda}, z)$
4 : **if** $g(w') = z$ **then**
5 : **return** 1
6 : **else return** 0

Figure 2.8: The inverting experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{INVERT}}\left[g, 1^{\lambda}\right]$

- (Hard to invert.) *For every* PPT *algorithm* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{INVERT}}\left[g, 1^{\lambda}\right] \rightarrow 1\right] \leq \text{negl}(\lambda).$$

2.7 Verifiable Outsourced Computation

Verifiable outsourced computation (VC) may be seen as a protocol between two polynomial-time parties, namely a client C and a server S. A successful execution of the protocol outputs a provably correct statement about the returned computational result $F(x)$ by the server on an input x provided by the client. The motivation for such protocols is based on the overwhelming success of resource constrained devices that need to perform computationally intensive tasks and as such outsource the task to a more powerful server which is likely to be untrusted. In many settings, the client wishes to (or even must) ensure the correctness of the computational result to ensure that no accidental or malicious errors have been introduced. A malicious server may try to convince the client to accept a malformed result while not being detected, in order to affect the client's future behaviour or the server tries to spare computational resources by simply returning a random result.

Some solutions have focused on an *audit*-based approach [25, 110] in which the client either requires to recalculate some portions of the computations or employs multiple servers to verify correctness. However, such an approach may be infeasible for the client given that she is resource-constrained. On the other hand, employing multiple servers is likely to increase the client's cost tremendously, in addition to require assurance that the servers do not collude. Other solutions may require specific secure computing environments such as relying on trusted platform modules [95] being employed by a

server or the client may receive a piece of secure hardware [50] that provides support with the expensive computation. However, such solutions raise trust issues as the client does not trust the hardware as well as increases the involved costs of employing them. Interactive proofs [20, 81, 83, 138] also provide a solution to VC as well as the more efficient notion of probabilistic checkable proofs (PCP) [13, 28] in which the client verifies the proof by only checking a small number of random locations. Yet another solution can be obtained using Micali's computational sound proofs [108].

2.7.1 Non-interactive Verifiable Outsourced Computation

The concept of *non-interactive verifiable outsourced computation* was introduced by Gennaro et al. [72]. In more detail, a VC scheme consists of the following three phases.

Pre-processing. The pre-processing is a one-time stage in which the client computes some public and private information associated with the function F she wishes to outsource. The client sends the public part to the server. This phase can take roughly the time comparable to computing the function from scratch, but it is performed only once, and its cost is amortised over all future executions.

Input preparation The client wishes the server to compute a function F on her input x to receive the result $F(x)$. Therefore, the client prepares some public and private information associated with her input x and sends the public part to the server.

Output computation and verification. Once the server has received the public information of F and x, it computes an encoded result which allegedly encodes the computational result $F(x)$ and returns it to the client. From this encoded value the client can compute the value $F(x)$ and verify its correctness.

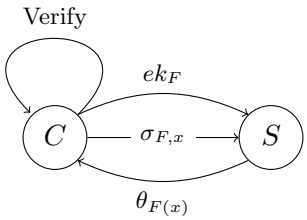

Figure 2.9: Basic operation of a verifiable outsourced computation scheme

The operation of a verifiable computation scheme is illustrated in Figure 2.9.

Note that this is a minimally interactive (non-interactive) protocol. This requires that there is only one round of interaction between the client and the server each time a computation is performed and thus rules out approaches based on repeated probabilistic challenge-response protocols. The crucial efficiency requirement considered in VC

is that input preparation and output verification must take less time for the client than computing the function from scratch (ideally linear time). Also, the output computation should take roughly the same amount as computing F itself.

Definition 2.31. *A* non-interactive verifiable outsourced computation (VC) scheme *consists of the following four algorithms*[3]*:*

- $(ek_F, sk_F) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, F)$: *this randomised algorithm takes as input the security parameter λ and the function F to be computed. It outputs a public evaluation key ek_F which the server will use to evaluate F and a secret key sk_F which is kept private by the client;*
- $(\sigma_{F,x}, vk_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, sk_F)$: *this randomised algorithm takes as input an input value x and the secret key sk_F to prepare a public encoded input $\sigma_{F,x}$ and a verification key $vk_{F,x}$ which is kept private by the client;*
- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, ek_F)$: *this randomised algorithm takes as input $\sigma_{F,x}$ and the evaluation key ek_F for F to compute an encoded version $\theta_{F(x)}$ of the function's output $y = F(x)$;*
- $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x}, sk_F)$: *this algorithm takes as input the encoded output $\theta_{F(x)}$, the verification key $vk_{F,x}$ and secret key sk_F. It outputs a result y which either corresponds to $F(x)$ if $\theta_{F(x)}$ is valid (i.e. $\theta_{F(x)}$ is a correct encoding of the output), or else corresponds to $\perp$ if the result is incorrect.*

Note that the function F used in the key generation algorithm is a function from the family of admissible functions $\mathcal{F}$ for the VC scheme. Different VC schemes are able to evaluate different families of functions $\mathcal{F}$, e.g. Boolean circuits and arithmetic circuits.

A verifiable computation scheme should be both correct and secure. A VC scheme is *correct* if the problem-generation algorithm produces values that allow an honest server to compute results that will verify successfully and correspond to the evaluation of F on those inputs. More formally this is captured as follows.

Definition 2.32. *A verifiable computation scheme for a family of functions $\mathcal{F}$ is* correct *if for any choice of function $F \in \mathcal{F}$, every key pair (ek_F, sk_F) outputted by $\mathsf{KeyGen}(1^\lambda, F)$ and for all input values $x \in \mathsf{Dom}(F)$, if $(\sigma_{F,x}, vk_{F,x}) \leftarrow_\$ \mathsf{ProbGen}(x, sk_F)$ and $\theta_{F(x)} \leftarrow_\$ \mathsf{Compute}(\sigma_{F,x}, ek_F)$ then $y = F(x) \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x}, sk_F)$.*

The main security notion considered in VC is the notion of *verifiability*. We say that a VC scheme is *secure* in the sense of verifiability if a malicious adversary cannot convince the verification algorithm to accept an incorrect output and we formally define this

[3]We change the notation relative to the literature and accommodate ours which we use throughout this thesis.

notion in Figure 2.10. In more detail, the game starts with the challenger running the KeyGen algorithm to generate a key pair for a function F in the family of admissible functions $\mathcal{F}$. The adversary is provided with the public evaluation key ek_F and access to the $\mathsf{ProbGen}$ oracle $\mathcal{O}^{\mathsf{ProbGen}}$ and returns an input x for which the challenger honestly prepares an encoded input $\sigma_{F,x}$ and verification key. This step is necessary to ensure that the challenger possesses a valid verification key corresponding to the computation. The adversary wins the game if it is able to produce an encoded output which is accepted by the verification algorithm Verify but in fact does not correspond to the correct result of $F(x)$.

$\mathbf{Exp}^{\text{VERIF}}_{\mathcal{A}}\left[\mathcal{VC}, 1^\lambda, F\right]$

1 : $(ek_F, sk_F) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda, F)$
2 : $x \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}}(\cdot, sk_F)}(ek_F)$
3 : $(\sigma_{F,x}, vk_{F,x}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x, sk_F)$
4 : $\theta_{F(x)} \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}}(\cdot, sk_F)}(\sigma_{F,x}, ek_F)$
5 : $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x}, sk_F)$
6 : **if** $(y \neq \perp)$ **and** $(y \neq F(x))$ **then**
7 : **return** 1
8 : **else return** 0

$\mathcal{O}^{\mathsf{ProbGen}}(z, sk_F)$

1 : $(\sigma_{F,z}, vk_{F,z}) \leftarrow\!\!\$\ \mathsf{ProbGen}(z, sk_F)$
2 : **return** $\sigma_{F,z}$

Figure 2.10: The verifiability experiment $\mathbf{Exp}^{\text{VERIF}}_{\mathcal{A}}\left[\mathcal{VC}, 1^\lambda, F\right]$

Definition 2.33. *The* advantage *of a PPT adversary in the* VERIF *game for a verifiable computation scheme $\mathcal{VC}$ is defined as:*

$$\mathbf{Adv}^{\text{VERIF}}_{\mathcal{A},\mathcal{VC}}(1^\lambda, F) = \Pr\left[\mathbf{Exp}^{\text{VERIF}}_{\mathcal{A}}\left[\mathcal{VC}, 1^\lambda, F\right] \rightarrow 1\right].$$

We say that the verifiable computation scheme $\mathcal{VC}$ is secure for a function F if for all PPT adversaries $\mathcal{A}$, it holds that

$$\mathbf{Adv}^{\text{VERIF}}_{\mathcal{A},\mathcal{VC}}(1^\lambda, F) \leq \text{negl}(\lambda).$$

Gennaro et al. [72] also consider the notions of input and output privacy. The former notion models that an adversary is not able to distinguish from the encoded input $\sigma_{F,x}$ which input value x was encoded. Output privacy assures that $\theta_{F(x)}$ does not reveal the actual output value $F(x)$.

A trivial solution to the problem of verifiable outsourced computation is to redundantly outsource the same computation to multiple servers and compare the returned results.

Then the majority of same results is assumed to be correct. Canetti et al. [43] follow the above scenario but show in contrast that a client can efficiently verify the computation as long as at least one server is honest. Most VC schemes [10, 22, 23, 30] try to improve on this solution to remove the redundancy, improve collusion resilience and to use only a single server per computation. Gennaro et al. [72] proposed a construction based on *Yao's garbled circuit* [144] construction which provides a "one-time" verifiable computation scheme allowing a client to outsource the evaluation of a function on a single input. However, this approach is insecure in case the circuit is reused and thus the cost of the pre-processing phase cannot be amortised as the cost of generating a new garbled circuit is approximately equal to the cost of evaluating the function itself. In order to overcome this problem, the authors propose using a fully homomorphic encryption scheme (FHE) [74] to re-randomise the garbled circuit for multiple evaluations on different inputs, but existing FHE scheme are expensive [42, 74, 131] and therefore are currently impractical. Some works have considered approaches to accommodate multiple clients who wish to send their respective inputs to the server that performs the computation over the jointly formed inputs. In such a scenario the notions such as input privacy become more important. Choi et al. [52] extended the garbled circuit approach [72] using a proxy oblivious transfer primitive [111] to achieve input privacy in a non-interactive multi-client VC scheme. Pham et al. [121] approach the problem of verifiable outsourced computation differently and provide a game-theoretical treatment in which they employ a weaker notion of security and develop optimal contracts for outsourcing computations via an appropriate use of rewards, punishments and auditing rate.

There is an issue with the verifiability in the above VC scheme [72] (which is also an inherent issue in other VC schemes [52] and delegation schemes [54, 81]) where the verifiability of the scheme holds only under the restriction that the cheating server does not learn whether the verifier accepted or rejected previous computations. This problem is denoted as the *rejection problem*. In more detail, as soon as the server learns that the client has rejected a computation then the server is able to deviate from the protocol and generate *improper* computational results such that by learning the decisions of the client as feedback, it can learn some information about secret verification keys that would enable future forgeries and therefore verifiability can no longer be claimed. Goldwasser et al. [82] overcome the rejection problem by using a designated verifier CS proof system and Fiore et al. [69] strengthen the VC scheme from [72] to avoid the rejection problem by providing the adversary with access to a verification oracle.

2.7.2 Publicly Verifiable Outsourced Computation

Parno et al. [118] introduced the notion of *publicly verifiable outsourced computation* (PVC) which enriches the functionality of VC with two new properties, namely *public delegation* and *public verification*. A large part of this thesis deals with extending the notion of PVC.

The PVC framework [118] is motivated in the setting of a scientific lab where the head of the research team specifies the function to be evaluated over the gathered data whilst the members of the research team decide on the specific inputs for each instance and verify the result. The advantage of being able to publicly verify also enables patients or other entities to ensure that the computational result is derived correctly.

The notion of PVC extends the prior notion of VC and includes multiple clients into the system model. In more detail, a single client C_1 performs the KeyGen algorithm to obtain an evaluation key ek_F which is given to the server, and publishing information pk_F are made available to *any* other client enabling them to encode inputs they wish to be evaluated by the server for the specific function F. This means that only one client needs to perform the expensive pre-processing phase. Each time the client prepares an encoded input using the $\mathsf{ProbGen}$ algorithm it may publish her verification key $vk_{F,x}$ enabling *any* other client to verify the computational output. Thus, outsourcing and verifying computations rely only on public information. Hence, the KeyGen algorithm needs to be only performed once per function rather than once per function and client.

Definition 2.34. *A* publicly verifiable outsourced computation (PVC) scheme *consists of the following four algorithms:*

- $(ek_F, pk_F) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, F)$: *this randomised algorithm takes as input the security parameter λ and the function F to be computed. It outputs a public evaluation key ek_F which the server will use to evaluate F and a public key pk_F which will be used for input delegation;*

- $(\sigma_{F,x}, vk_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, pk_F)$: *this randomised algorithm takes as input an input value x and the public key pk_F. It prepares an encoded input $\sigma_{F,x}$ and a verification key $vk_{F,x}$ which is used for verification;*

- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, ek_F)$: *this randomised algorithm takes as input the encoded input $\sigma_{F,x}$ and the evaluation key ek_F for F to compute an encoded version $\theta_{F(x)}$ of the function's output $y = F(x)$;*

- $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x})$: *this public algorithm takes as input the encoded output $\theta_{F(x)}$ and the verification key $vk_{F,x}$. It outputs a result y which either corresponds to $F(x)$ if $\theta_{F(x)}$ is valid, or else corresponds to $\perp$ if the result is incorrect.*

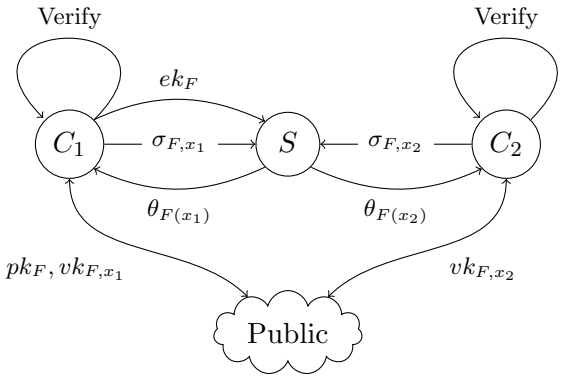

Figure 2.11: Basic operation of a publicly verifiable outsourced computation scheme

The operation of a PVC scheme is illustrated in Figure 2.11. Note that in both the definitions of VC and PVC the client is responsible to perform the KeyGen algorithm for a single function F which may be expensive (roughly the cost of executing the function F itself). However, the client is assumed to only possess limited resources and is generally interested to compute multiple functions which makes the definitions to some extent restrictive. The main changes in the definition of PVC relative to VC are that KeyGen generates a secret key sk_F which in PVC will be made public and thus enables any entity to outsource a computation of F, and that the verification key $vk_{F,x}$ generated in ProbGen is published to allow any entity to verify a computational result.

The correctness of a PVC scheme follows similarly to the one of a VC scheme. A PVC scheme is *correct* if an honest run of the protocol will verify successfully and correspond to the evaluation of F on those inputs. More formally this is captured as follows.

Definition 2.35. *A publicly verifiable outsourced computation scheme for a family of functions $\mathcal{F}$ is* correct *if for any choice of function $F \in \mathcal{F}$, every key pair (ek_F, pk_F) outputted by $\mathsf{KeyGen}(1^\lambda, F)$ and for all input values $x \in \mathsf{Dom}(F)$, if $(\sigma_{F,x}, vk_{F,x}) \leftarrow_\$ \mathsf{ProbGen}(x, pk_F)$ and $\theta_{F(x)} \leftarrow_\$ \mathsf{Compute}(\sigma_{F,x}, ek_F)$ then $y = F(x) \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x})$.*

The security notion of *public verifiability* for a PVC scheme is defined in Figure 2.12. Compared to Figure 2.10 which describes the notion of verifiability for a VC scheme, the public verifiable setting does not require a ProbGen oracle as outsourcing computations is solely based on public parameters and hence can be run by the adversary itself. The adversary wins the game if it is able to produce an encoded output in such a way that it is accepted by the verifier but in fact does not correspond to the correct result of $F(x)$.

Definition 2.36. *The* advantage *of a PPT adversary in the* PubVerif *game for a*

$\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{PVC}, 1^{\lambda}, F\right]$

1 : $(ek_F, pk_F) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^{\lambda}, F)$
2 : $x \leftarrow\!\!\$\ \mathcal{A}(ek_F, pk_F)$
3 : $(\sigma_{F,x}, vk_{F,x}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x, pk_F)$
4 : $\theta_{F(x)} \leftarrow\!\!\$\ \mathcal{A}(\sigma_{F,x}, vk_{F,x}, ek_F, pk_F)$
5 : $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x})$
6 : **if** $(y \neq \perp)$ **and** $(y \neq F(x))$ **then**
7 : **return** 1
8 : **else return** 0

Figure 2.12: The public verifiability experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{PVC}, 1^{\lambda}, F\right]$

publicly verifiable computation scheme $\mathcal{PVC}$ is defined as:

$$\mathbf{Adv}_{\mathcal{A},\mathcal{PVC}}^{\text{PubVerif}}(1^{\lambda}, F) = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{PVC}, 1^{\lambda}, F\right] \rightarrow 1\right].$$

We say that the public verifiable computation scheme $\mathcal{PVC}$ is secure for a function F if for all PPT adversaries $\mathcal{A}$, it holds that

$$\mathbf{Adv}_{\mathcal{A},\mathcal{PVC}}^{\text{PubVerif}}(1^{\lambda}, F) \leq \text{negl}(\lambda).$$

The presented notion of public verification allows a stronger notion of security since any entity, including the server, is able to perform verification. Since this enables the server to tell whether a result will be accepted or rejected by the client, it makes the rejection problem in the context of PVC invalid. From a more technical point of view this is the case as verification only depends on the public parameters and some instance specific randomness generated by the client and not any long-term secret. Thus, obtaining the result from a previous verification step on one instance does not provide any advantage in breaking the verifiability on a different instance.

Some prior work [33, 81, 82] already introduced the property of public delegation but are mostly in the random oracle model or rely on non-standard assumptions. Parno et al. [118] have been the first who considered public verifiability. Since then the concept has been widely used to publicly verify specific operations [68] and construct an almost practical scheme [117].

In Section 2.7.3, we provide a detailed discussion on the basic principles of how to construct a PVC scheme based on key-policy attribute-based encryption following

Parno et al. [118].

Multi-function Verifiable Outsourced Computation

Parno et al. [118] introduced the notion of *multi-function verifiable outsourced computation* (MFVC) which extents the original definition of VC [72] to efficiently enable servers to evaluate multiple functions on a single input. In other words, a client encodes a single input x independently from any functions, and then can request evaluations of multiple functions upon it. This notion is useful in case the client's data remains static and she wishes to use it multiple times.

In more detail, compared to [72] where the function to be evaluated is embedded in the system's parameters, MFVC separates the generation of system parameters and function parameters into different algorithms to allow the evaluation of multiple functions on the same input instance.

Definition 2.37. *A* multi-function verifiable outsourced computation (MFVC) scheme *consists of the following five algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$: *this randomised algorithm takes as input the security parameter λ. It produces the public parameters pp and secret parameters mk that do not depend on any function to be evaluated;*
- $(ek_F, sk_F) \xleftarrow{\$} \mathsf{KeyGen}(F, mk, pp)$: *this randomised algorithm takes as input the function F to be computed, the secret parameters mk and the public parameters pp. It outputs an evaluation key ek_F which the server will use to evaluate F and a secret key sk_F used for verification which will be kept private by the client;*
- $(\sigma_x, vk_x) \xleftarrow{\$} \mathsf{ProbGen}(x, mk, pp)$: *this randomised algorithm takes as input an input value x, the secret parameters mk and the public parameters pp, which are all independent of the function F. It produces an encoded input σ_x and a secret verification key vk_x which will be kept private by the user;*
- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_x, ek_F, pp)$: *this randomised algorithm takes as input the encoded input σ_x, the evaluation key ek_F and the public parameters pp. It computes an encoded version $\theta_{F(x)}$ of the function's output $y = F(x)$;*
- $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_x, sk_F)$: *this public algorithm takes as input the encoded output $\theta_{F(x)}$, the input specific verification key vk_x and function specific secret key sk_F. It outputs a result y which either corresponds to $F(x)$ if $\theta_{F(x)}$ is valid, or else corresponds to $\perp$ if the result is incorrect.*

A MFVC scheme is *correct* if an honest run of the protocol will verify successfully and correspond to the evaluation of F on those inputs. More formally this is captured as follows.

Definition 2.38. *A multi-function verifiable outsourced computation scheme for a family of functions $\mathcal{F}$ is* correct *if for the generated parameters (pp, mk) outputted by $\mathsf{Setup}(1^\lambda)$, for any choice of function $F \in \mathcal{F}$, every key pair (ek_F, sk_F) outputted by $\mathsf{KeyGen}(F, mk, pp)$ and for all input values $x \in \mathsf{Dom}(F)$, if $(\sigma_x, vk_x) \leftarrow_\$ \mathsf{ProbGen}(x, mk, pp)$ and $\theta_{F(x)} \leftarrow_\$ \mathsf{Compute}(\sigma_x, ek_F, pp)$ then $y = F(x) \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_x, sk_F)$.*

The security notion we consider for a MFVC scheme is *multi-function verifiability* which we formally define in Figure 2.13. It proceeds similar to the security of a VC scheme as represented in Figure 2.10. The main difference is that the adversary has additionally access to a KeyGen oracle representing its capability to request keys for any arbitrary functions it wishes to evaluate. The adversary wins the game if it is able to produce an encoded output which is accepted by the verifier but in fact does not correspond to the correct result of $F(x)$.

$\mathbf{Exp}^{\text{MultiVerif}}_{\mathcal{A}}\left[\mathcal{MFVC}, 1^\lambda\right]$

1 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
2 : $(x, F) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,mk,pp),\mathcal{O}^{\mathsf{ProbGen}}(\cdot,mk,pp)}(pp)$
3 : $(\sigma_x, vk_x) \leftarrow_\$ \mathsf{ProbGen}(x, mk, pp)$
4 : $\theta_{F(x)} \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,mk,pp),\mathcal{O}^{\mathsf{ProbGen}}(\cdot,mk,pp)}(pp)$
5 : $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_x, sk_F)$
6 : **if** $(y \neq \perp)$ **and** $(y \neq F(x))$ **then**
7 : **return** 1
8 : **else return** 0

$\mathcal{O}^{\mathsf{KeyGen}}(F, mk, pp)$

1 : $(ek_F, sk_F) \leftarrow_\$ \mathsf{KeyGen}(F, mk, pp)$
2 : **return** ek_F

$\mathcal{O}^{\mathsf{ProbGen}}(z, mk, pp)$

1 : $(\sigma_z, vk_z) \leftarrow_\$ \mathsf{ProbGen}(z, mk, pp)$
2 : **return** σ_z

Figure 2.13: The multi-function verifiability experiment $\mathbf{Exp}^{\text{MultiVerif}}_{\mathcal{A}}\left[\mathcal{MFVC}, 1^\lambda\right]$

Definition 2.39. *The* advantage *of a PPT adversary in the* MultiVerif *game for a multi-function verifiable computation scheme $\mathcal{MFVC}$ is defined as:*

$$\mathbf{Adv}^{\text{MultiVerif}}_{\mathcal{A},\mathcal{MFVC}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{MultiVerif}}_{\mathcal{A}}\left[\mathcal{MFVC}, 1^\lambda\right] \rightarrow 1\right].$$

We say that the multi-function verifiable computation scheme $\mathcal{MFVC}$ is secure for a function F if for all PPT adversaries $\mathcal{A}$, it holds that

$$\mathbf{Adv}^{\text{MultiVerif}}_{\mathcal{A},\mathcal{MFVC}}(1^\lambda) \leq \text{negl}(\lambda).$$

Parno et al. [118] provide a construction for MFVC based on *KP-ABE with outsourcing* as introduced by Green et al. [90]. However, the given construction is publicly delegable

but not publicly verifiable. This 'drawback' comes from the fact that in [90], giving out sk_F enables the server directly to cheat about the evaluated function, i.e. the server claims an output computed with F was the result of applying G. Parno et al. left it as an open problem to overcome this issue to construct a publicly verifiable multi-function VC scheme.

2.7.3 Construction of Publicly Verifiable Computation Schemes

Parno et al. [118] provide a PVC construction using key-policy attribute-based encryption (KP-ABE)[89] for outsourcing the family of monotone Boolean functions.[4] In this section we present the basic principles of their PVC construction as those principles will be used throughout this thesis as a building block for our extended PVC proposals.

Parno et al. use the idea that the KP-ABE decryption functionality provides a proof that a monotone Boolean function is satisfied (i.e. outputs 1) on a given input. We recall from Section 2.3.1 that in KP-ABE, the decryption keys are associated with access structures while ciphertexts are associated with attribute sets. Decryption is successful if and only if the attribute set in the ciphertext satisfies the access structure in the decryption key. This idea can be lifted into the setting of PVC by encoding the function to be outsourced as an access structure and issue the server with a respective decryption key. Input data can be encoded in terms of attribute sets and the encryption of messages result into ciphertexts which are associated with those attribute sets.

In order to outsource a computation $F(x)$, Parno et al. select a random message m_0 from the underlying message space of the KP-ABE scheme and encrypt it under the attribute representation A_x that encodes the input x. A server is issued an evaluation key to perform the computation in form of a KP-ABE decryption key for the access structure encoding the function F. The server succeeds in decrypting the ciphertext and recovering the message m_0 if and only if $F(x) = 1$, which indicates that the access structure is satisfied by the attribute set. In case that $F(x) = 0$ then this indicates that the access structure is *not* satisfied by the attribute set and the adversary cannot do better at finding the message m_0 than a random guess.[5] In other words, if the client receives the same message that she initially encrypted, she is fully convinced that decryption was successful and $F(x) = 1$. However, if the client receives no answer or $\perp$ from the server then this could be the case because the server was truly unable to decrypt and $F(x) = 0$, or because $F(x) = 1$ and the server intentionally refuses to decrypt. Thus, this proposal can be seen as a protocol with a one-sided error.

[4] If input privacy is required then a predicate encryption scheme could be used in place of the KP-ABE scheme.

[5] Note that in order for an ABE scheme to be secure, it is required that the message space is large enough such that the server is not able to randomly guess and return the correct message with a significant probability.

To overcome the problem with the one-sided error, Parno et al. suggest to initialise a second (independent) KP-ABE scheme and then following the same principles as above and encrypting a different random message m_1 under the same attribute set A_x corresponding to x using the parameters of the second system. The server is now issued an evaluation key in the form of a KP-ABE decryption key for the access structure encoding the *complement* function $\overline{F}(x) = F(x) \oplus 1$, which always outputs the opposite result to $F(x)$. Thus, exactly one of $F(x)$ or $\overline{F}(x)$ evaluates to 1 and therefore exactly one decryption was successful and the respective message is returned.[6] The client is able to observe which message was returned and therefore knows under which KP-ABE scheme the message was encrypted which enables the client to determine whether $F(x)$ or $\overline{F}(x)$ was satisfied and thus learns whether $F(x) = 1$ or 0 respectively. A well-formed and valid response from a server comprises the outputs (d_0, d_1) from both decryption procedures and therefore is of the following form:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } F(x) = 1; \\ (\perp, m_1) & \text{if } F(x) = 0. \end{cases} \tag{2.1}$$

Since KP-ABE is a public-key encryption primitive, this construction can be seen as the "public-key" version of the initial proposal by Gennaro et al. [72]. Furthermore, this construction enables *any* entity to create ciphertexts and hence the construction achieves public delegability. On the other hand the construction can achieve public verifiability by employing a one-way function g, e.g. a pre-image resistant hash function. In more detail, the client publishes a verification key for the outsourced computation comprising the results of applying the one-way function to each randomly chosen message m_0 and m_1. The server outputs a computational result (corresponding to exactly one message) and on receipt of such, any entity can apply g to the returned message and compare the result with the values in the verification key to verify correctness. We want to stress that even a malicious server does not gain any advantage from knowing the verification key since it cannot invert the one-way function g to recover either message. In Table 2.1, we provide an overview about the particular mapping between abstract PVC parameters and KP-ABE parameters used in the construction.

As mentioned above, we need a particular encoding procedure that defines input data for outsourced computations as attribute sets. However, Parno et al. did not specify any details about the encoding procedure they may use in their scheme. Therefore, we want to provide details about the particular procedure we use throughout this thesis which also applies to the Parno et al. construction. In more detail, we define a universe $\mathcal{U}$ of n attributes and associate $V \subseteq \mathcal{U}$ with a binary n-tuple where the i-th entry is 1 if and only if the ith attribute is in V. We call this tuple the *characteristic tuple* of

[6] Goldwasser et al. [80] call such a slightly modified ABE scheme *two-outcome attribute-based encryption scheme.*

Abstract PVC parameter	Parameter in KP-ABE instantiation
ek_F	$sk_{\mathbb{A}_F}$
pk_F	Master public key pp
$\sigma_{F,x}$	Encryption of m using pp and A_x
$\theta_{F(x)}$	m or $\perp$
$vk_{F,x}$	$g(m)$

Table 2.1: Mapping between PVC and KP-ABE parameters

V. Thus, there is a natural one-to-one correspondence between n-tuples and attribute sets and we write A_x to denote the attribute set corresponding to the input data x. An alternative way to view this is to let $\mathcal{U} = \{A_1, A_2, \ldots, A_n\}$. Then, a bit string $\bar{v}$ of length n is the characteristic tuple of the set $V \subseteq \mathcal{U}$ if $V = \{A_i : \overline{v_i} = 1\}$. A function $F : \{0,1\}^n \to \{0,1\}$ is monotonic if $x \leqslant y$ implies $F(x) \leqslant F(y)$, where $x = (x_1, \ldots, x_n)$ is less than or equal to $y = (y_1, \ldots, y_n)$ if and only if $x_i \leqslant y_i$ for all i. For a monotonic function $F : \{0,1\}^n \to \{0,1\}$ the set $\mathbb{A}_F = \{x \in \{0,1\}^n : F(x) = 1\}$ defines a monotonic access structure.

Throughout the remainder of this thesis, we mainly refer to monotonic Boolean functions. This is mainly due to the majority of the ABE literature considering monotonic access structures. However, a notable non-monotonic KP-ABE scheme was given by Ostrovsky et al. [112] which accommodates general Boolean functions and hence would enable us to outsource the NC^1 class of functions. The use of such a scheme in our construction as well as in the one of Parno et al. should be straightforward since we use the KP-ABE scheme as a black-box. However, we would need to slightly adjust our encoding procedure for the input data since 0 values in the bit strings can affect the outcome of the computation. In more detail, we define the universe of attributes $\mathcal{U}$ to consist of $2n$ attributes $\{A_i^0, A_i^1\}_{i=1}^n$. Then, a bit string $\bar{v}$ of length n is the characteristic tuple of the set $V \subseteq \mathcal{U}$ if $V = \{A_i^j : \overline{v_i} = j\}$. By applying De-Morgan's laws to a non-monotonic Boolean function it is possible to move any negations within the function such that they only apply to the input variables. Hence a non-monotonic Boolean function can be satisfied by choosing the 0 or 1 attribute appropriately in the input attribute set.

2.8 Proofs of Retrievability

Proofs of retrievability (PoR) may be seen as a protocol between a polynomial-time client and a polynomial-time server. A successful execution of the PoR protocol results in a concise proof produced by a server that the user's outsourced file F can be retrieved, that is, the server retains and reliably transmits file data sufficient for the user to recover the original file F in its entirety. The motivation for a PoR scheme is the overwhelming success of the cloud model which offers various benefits such as flexible

scalability and accessibility of different cloud services in a cost effective manner where in case of a PoR the client makes use of the cloud's storage instance (e.g. Amazon S3, Windows Azure) to which the client initially outsources her data. In this settings, the client wishes to assure the correctness of the outsourced data to ensure that no accidental or malicious errors have been introduced into the data. A malicious server may try to change, tamper or delete the data whilst not being detected in order to spare storage resources.

A simple solution to the above problem may be to download the whole file every time the client wishes to check the data. However, this approach stands in contrast to the goals of outsourcing the data in the first place and thus the client would lose all initial outsourcing benefits, as well as being very bandwidth intensive. The challenging problem is to enable verification of the file F without explicit knowledge of the full file. This problem was first described by Blum et al. [34] who explored the task of efficiently checking the correctness of a memory-management program. Another solution [103] may require a trusted *security provider* controlling the integrity-protection of files in untrusted cloud storage providers. A PoR can loosely be seen as a proof of knowledge (PoK) [27] conducted between a prover and verifier on a file F. The essential design goal in a PoK is to preserve the secrecy of some short secret. However, in PoR, the design is different as the verifier already learned the value F as the file was initially outsourced by the verifier. Thus, the main challenge is to prove knowledge of F with computational and communication costs substantially smaller than $|F|$. PoR are also akin to other proof systems such as proofs of computational ability [146] and proofs of work [96].

Proofs of retrievability were introduced by Juels and Kaliski [98]. Informally, PoR can be seen as a cryptographic proof showing that the outsourced file is retained and fully intact stored at a cloud service provider. In other words, a PoR aims to provide a mechanism of proving that the file is intact and the client can fully recover it. More formally this is captured in the following definition.

Definition 2.40. *A* proofs of retrievability (PoR) scheme *consists of the following procedures:*

- $(pk, sk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*: this randomised algorithm generates a public-private key pair* (pk, sk) *and takes as input the security parameter* λ*;*
- $(\mathcal{F}, \tau) \xleftarrow{\$} \mathsf{Store}(sk, F)$*: this randomised file storing algorithm takes as input a secret key* sk *and a file* $F \in \{0,1\}^*$*. The file gets processed and the algorithm outputs* $\mathcal{F}$ *which will be stored on the server as well as a file tag* τ*. The tag* τ *contains additional information (e.g. metadata) about* $\mathcal{F}$*;*
- $\delta \xleftarrow{\$} [\mathsf{Verify}(pk, sk, \tau) \rightleftharpoons \mathsf{Prove}(pk, \mathcal{F}, \tau)]$*: the randomised proving and verifying*

algorithms define a protocol for proving file retrievability. The prover algorithm takes as input the public key pk as well as the file tag τ and the processed file $\mathcal{F}$. The verifier algorithm takes the secret key sk, public key pk and the file tag τ. Algorithm Verify *outputs at the end of the protocol execution a binary value δ which corresponds to* `accept` *if the verification succeeds, indicating the file is being stored and retrievable on the server, and* `reject` *otherwise.*

Note that the above definition is adapted from Shacham and Waters [126] rather than the original one by Juels and Kaliski [98]. For simplicity, let us denote the challenge-response procedure $[\mathsf{Verify}(pk, sk, \tau) \rightleftharpoons \mathsf{Prove}(pk, \mathcal{F}, \tau)]$ as PoRP.

A proof of retrievability scheme should be both correct and secure. A PoR scheme is *correct* if the processed file $\mathcal{F}$ outputted by the store procedure will be accepted by the verification algorithm when interacting with a valid prover. More formally this is captured as follows.

Definition 2.41. *A PoR scheme is* correct *if there exists a negligible function* negl *such that for every security parameter λ, every key pair (pk, sk) generated by* Setup*, for all files $F \in \{0,1\}^*$, and for all $(\mathcal{F}, \tau)$ generated by* Store*, it holds that*

$$\Pr[(\mathsf{Verify}(pk, sk, \tau) \rightleftharpoons \mathsf{Prove}(pk, \mathcal{F}, \tau)) \twoheadrightarrow \texttt{accept}] = \mathrm{negl}(\lambda).$$

Security of a PoR scheme is defined in the usual terms of an experiment in which the adversary plays the role of the prover $\mathcal{P}$. Intuitively, a PoR scheme is secure if any cheating prover that convinces the verification algorithm that it stores a file F is indeed storing the file. In other words, we wish to guarantee that, whenever a malicious prover is in a position of successfully passing a PoRP instance, it must *know* the entire file content. As in a PoK, we need to formalise *knowledge* via the existence of an efficient *extractor* $\mathcal{E}$ that can recover the original file F given access to the malicious prover. We first formalise the notion of an extractor and then formally define the relevant security notion called *extractability* following the formalisation in [126].
An extractor algorithm $\mathcal{E}(pk, sk, \tau, \mathcal{P}')$ takes as input the generated key pair, the file tag τ as well as a description of the machine implementing the prover's role in the PoR scheme and its output is the file F. The extractor is given (non black-box) access to $\mathcal{P}'$ and in particular can rewind it. Furthermore, we require that the algorithm is efficient, i.e. $\mathcal{E}$'s running time needs to be polynomial in the security parameter.

Consider the following *extractability* game $\mathbf{Exp}^{\text{EXTRACT}}_{\mathcal{A},\epsilon}\left[\mathcal{POR}, 1^\lambda\right]$ between a malicious adversary $\mathcal{A}$, an extractor $\mathcal{E}$, and a challenger $\mathcal{C}$.

1. The challenger initialises the system by running Setup to generate the public and private key pair for all entities. The public key is provided to $\mathcal{A}$.

2. The adversary $\mathcal{A}$ is now able to interact with the challenger that takes the role of an honest client. $\mathcal{A}$ is allowed to request executions to a Store oracle by providing, for each query, some file F. The challenger runs $(\mathcal{F}, \tau) \xleftarrow{\$} \mathsf{Store}(sk, F)$ and returns both outputs to the adversary.

3. Likewise, $\mathcal{A}$ can request executions of the PoRP scheme for any file on which it previously made a Store query by specifying the corresponding tag τ. In the procedure, the challenger will play the role of the honest verifier $\mathcal{V}$ and the adversary the role of the corrupted prover, i.e. $\mathcal{V}(\tau, pk, sk) \rightleftharpoons \mathcal{A}$. In the end of the execution the adversary is provided with the output of the verifier. Furthermore, the Store oracle queries and executions of PoRP can be interleaved arbitrarily.

4. The adversary outputs a challenge tag τ' returned from some Store query and the description a prover $\mathcal{P}'$.

5. Run the extractor algorithm $F' \leftarrow \mathcal{E}(pk, sk, \tau', \mathcal{P}')$ inputting the challenge tag τ' and description $\mathcal{P}'$ where $\mathcal{E}$ gets black-box rewinding access to $\mathcal{P}'$, and attempts to extract the file content as F'.

6. If $\Pr[(\mathcal{V}(\tau, pk, sk) \rightleftharpoons \mathcal{P}') \rightarrow \mathtt{accept}] \geq \epsilon$ and $F' \neq F$ then output 1, else 0.

Note that we say a malicious prover $\mathcal{P}'$ is ϵ-*admissible* if the probability that it convincingly answers verification challenges is at least ϵ, i.e. if $\Pr[(\mathcal{V}(\tau, pk, sk) \rightleftharpoons \mathcal{P}') \rightarrow \mathtt{accept}] \geq \epsilon$. Here the probability is over the coins of the verifier and prover.

Definition 2.42. *We say that a PoR scheme, $\mathcal{POR}$, is* ϵ-extractable *(or* secure*) if there exists an efficient extraction algorithm $\mathcal{E}$ such that, for all PPT adversaries $\mathcal{A}$ it holds that* $\Pr\left[\mathbf{Exp}^{\text{EXTRACT}}_{\mathcal{A},\epsilon}\left[\mathcal{POR}, 1^\lambda\right] \rightarrow 1\right]$ *is negligible in the security parameter.*

Juels and Kaliski [98] present a PoR scheme which relies on so-called *sentinels*. In their model, a file consists of *blocks* and a sentinel itself is an indistinguishable block which will be hidden among regular file blocks in order to detect data modification by the server. In more detail, the client challenges the server by specifying the positions of a collection of sentinels and requests the server to return the associated sentinel values. In case the server has deleted or modified a *substantial* portion of the file F, then with high probability it will have also suppressed a number of sentinels. Thus, it is unlikely that the server will respond correctly to the verifier. In order to also protect against corruption of a *small* portion of F, Juels and Kaliski [98] propose to employ *erasure-correcting codes* (ECC). In other words, the usage of erasure-correcting codes can *amplify* errors in the stored file and thus it is more likely for the client to recover the file even if erasures were introduced. We discuss the necessity and importance as well as present a formal definition of ECC later in this section. A similar approach of using erasure-correcting codes was proposed in an early PoR-like protocol of Lillibridge et al. [106]. However, their main goal is slightly different compared to PoR as they

wish to achieve an assurance of the availability of an outsourced file that is distributed over a set of servers in a peer relationship. The Juels-Kaliski proposal only supports a bounded number of PoRP executions after which the server has learnt all embedded sentinels and thus requires the client to re-perform the initialisation procedure in case she wishes to further check the file via a PoR. The authors also propose a public PoR scheme using a Merkle-tree construction so that it can be verified by any external party without requiring a secret key.

Another notable PoR construction was proposed by Shacham and Waters [126]. They propose two PoR schemes, namely a private-key based and a public-key based scheme. The former scheme builds on pseudorandom functions and is secure in the standard model whilst the latter uses BLS signatures and is secure in the random oracle model. Both schemes, however, uses homomorphic authenticators to yield compact proofs. More details about the private-key scheme can be found in Chapter 7.
Following the initial PoR scheme, works have been developed that propose improvements compared to [98]. For example, Bowers et al. [40] construct a PoR scheme which tolerates a Byzantine attacker model. Dodis et al. [62] introduce the notion of PoR codes which combines concepts in PoR and hardness amplification. Paterson et al. [119] treat PoR schemes in the model of unconditional security where an adversary has unbounded computational power and they show that retrievability can be modelled as erasure-correction in a certain mode. Armknecht et al. [11] proposed the notion of an outsourced PoR scheme which enables a client to task an external auditor to perform and verify PoR procedures on her behalf. The above approaches, however, are limited in the sense of being only able to handle static files. In contrast to the above approaches, some proposals deal with the construction of dynamic schemes supporting efficient updates. Cash et al. [47] achieve dynamic updates using oblivious RAM whereas Shi et al. [128] improve the performance by relying on a Merkle hash tree. Stefanov et al. [134] consider updates where a trusted "portal" performs operations on the client's behalf. Recently, Guan et al. [92] explore the usage of indistinguishability obfuscation for building a PoR scheme that offers public verification while the encryption process is based on symmetric key primitives. Other contributions [123, 127, 145] deal with public verifiable PoR schemes.

Concurrently to the work of Juels and Kaliski [98], Ateniese et al. [15] proposed a close variant of PoR called *proofs of data possession* (PDP). The main difference between PoR and PDP is the notion of security they achieve. More precisely, a PoR provides stronger security guarantees than PDP as a PoR assures that the server maintains *full* knowledge of the client's data whereas a PDP only assures that most of the data is retained. Dynamic PDP solutions where proposed in [16] where the problem of dynamic writes/updates was considered and [63] uses authenticated dictionaries based on rank

information. In [58], Curtmola et al. propose a multi-replica PDP which enables a client to efficiently verify that a file is replicated at least across k replicas by the cloud. Some work appeared exploring the direction to extend works into the multi-server setting. Bowers et al. [39] introduce a system called HAIL which enables a set of servers to prove to a client that a stored file is intact and retrievable against a mobile adversary that can progressively corrupt the full set of storage providers. In [41], Bowers et al. present a scheme that enables the client to verify if her data is redundantly stored at multiple servers by measuring the time it takes the server to respond to a read request for a set of data blocks. Gritti et al. [91] introduce a third party enabling the client to efficiently check the integrity of the data.

Erasure-correcting codes

Erasure-correcting codes (ECC) play a crucial role in the functionality of a PoR scheme. As mentioned above, the use of erasure-correcting codes were introduced as a mechanism to handle corruption of a small number of file blocks. ECC enable to boost the detection probability and ensure that the server must possess sufficiently many blocks to pass a PoRP procedure. Typically, this procedure consists of checking the authenticity of λ random processed file blocks, where λ is the security parameter.

In basic terms, ECC is a process that adds redundant data to the original data in such a way that a receiver may recover the processed data even when a number of errors were introduced, either during the transmission of the file, or on storage. As a concrete example, let us suppose a client wishes to outsource a file consisting of n blocks which are erasure coded into $m = n/\rho$ blocks for some ECC rate $0 < \rho \leq 1$, such that knowledge of any n of m blocks suffice to recover the original file blocks. In other words, to recover the file we require a reception efficiency of 1, i.e. we need to check as many file blocks as the *original* file consists of. Thus, this means that the server needs to delete more than $(1-\rho)$ percent of the file blocks in order to incur actual data loss. However, if the server deletes that many blocks, it will fail the above PoRP with overwhelming probability and the adversary will be detected cheating with a probability of at least $(1-(1-\rho)^{\lambda})$. In more detail, the PoR scheme provides a probabilistic assurance about the authenticity and retrievability of the data and thus about the outcome of the PoR scheme itself.

We close this chapter with the formal definition of an erasure-correcting code following the standard literature [94].

Definition 2.43. *Let Σ denote a finite alphabet. An $(m, n, d)_{\Sigma}$ erasure-correcting code is defined to be a pair of algorithms* $\mathsf{encode}\colon \Sigma^n \to \Sigma^m$, *and* $\mathsf{decode}\colon \Sigma^{m-d+1} \to \Sigma^n$, *such that as long as the number of erasures is bounded by $d-1$, then* decode *can always recover the original data. A code is* maximum distance separable *(MDS), if*

$n + d = m + 1$.

Chapter 3

Revocation in Publicly Verifiable Outsourced Computation

Contents

This chapter deals with the setting of publicly verifiable outsourced computation (PVC) for which has been shown that key-policy attribute-based encryption can be used. We propose extensions and improvements to the current PVC proposal in order to accommodate a more practical framework. For this mode of computation, we introduce a distinguished entity that provides support to minimise the client's computational burden regarding the expensive pre-processing. We also investigate a simple mechanism to enable a server to compute multiple functions, and provide a method to revoke misbehaving servers from future evaluations within the system. The results of this chapter appear in [4].

3.1 Introduction

The cloud model has led to many benefits in various application domains such as storing and computing. Nowadays, it is also increasingly common for mobile devices being used as general computing devices. Since users produce a vast amount of data there is a natural desire to process and evaluate the data. However, since the mobile devices cannot handle this task sufficiently well there is a need to outsource the evaluation of the data to the cloud. Upon receiving a computational result from the cloud, the user wishes to obtain some assurance about the validity of the result.

This problem known as *verifiable outsourced computation* (VC) has recently attracted a lot of attention in the community and we summarised the concept in detail in Sec-

tion 2.7. Verifiable computation enables a computationally-limited client to outsource a computation to a computational powerful server and efficiently verify whether the returned result was evaluated correctly. *Publicly verifiable computation* [118], on the other hand, aims to provide a more practical scenario for VC. In PVC only one client needs to perform the expensive one-time pre-processing stage and is able to publish parameters (for a specific function F) in such a way that any other entity can use those parameters themselves in order to delegate the evaluation of F on their own input and also to verify results. Thus, PVC can be seen as a multi-client system but the system does not currently support *multiple servers* or *multiple functions*.

We believe that both requirements might be important to make current PVC schemes "more practical". In more detail, it may be desirable for a client to be able to choose from a set of available servers per computation within the system. Certain different individual computation requests may require different computational resources such as a certain amount of processor cores or a given amount of RAM. Such specific requirements may only be fulfilled by a small number of servers within the system and clients are flexible to choose the server depending on their needs related on the complexity of the computation. A client may also require that certain computations can only be evaluated on servers that are located within a specific geographical location. The motivation may be to minimise latency or the specific computation may be of sensitive nature and thus may not leave the country of origin and therefore the client may be even willing to pay a higher reward for using such a specific server. Having registered multiple servers offering a computation service to clients within a PVC system, it seems conceivable that the servers may compete against each other to reduce costs or may bid on computations depending on their available resources. Thus, such behaviour may be of particular interest for the client since this may reduce the involved cost of the system and rewards dramatically.

In current PVC schemes the system is only initialised to evaluate a single function. In case a client wishes to compute a new function, she is required to initialise a new independent PVC system. This requires the client to invest a lot of her own resources each time she needs to compute a new function. Since this is not acceptable in a practical environment, we aim to enhance the system to accommodate handling multiple functions in a publicly verifiable way to spare those resources for other tasks. Since PVC can be seen as a multi-client system, it is also conceivable that many clients are largely interested in outsourcing the same set of functions, albeit on different, client-specific input data. Currently, it is likely that within a group of clients (e.g. a set of employees within a company) there is a distinguished client responsible initialising the system by running the expensive pre-processing and distributing the respective delegation key and evaluation key to the other members of the group. However, given that the set of

functions for different groups of clients may overlap, it seems required that each distinguished client invests the same effort to initialise the respective system for their group. This effort is redundant as all distinguished clients just repeat each others work while the workload has already increased as the client initialised a system to accommodate multiple servers and multiple functions. The role of these clients become akin to an authority of entities within the system. Therefore, we propose the introduction of a separate trusted entity which we denote as the *key distribution centre* (KDC). The role of the key distribution centre is to initialise the system and issue the evaluation keys on behalf of *all* clients within the system.

Given that we have enrolled multiple untrusted servers within a PVC system, it may well be desirable that cheating servers are prevented from performing further computations and as such being *revoked* from the system, as they are deemed completely untrustworthy in case they were caught cheating. In the single-client VC scheme [72], the client can simply choose to no longer delegate computations to the server. In a multi-client setting, on the other hand, it is necessary that all clients within the system are aware that a server is known to be not trustworthy and stop using it to spare their resources. However, in both VC and PVC, to overcome the problem, we are required to initialise a new system which requires the client to invest again her valuable resources on the pre-processing. In our new PVC proposal, it is not desirable to initialise a new system as other (potentially trustworthy) servers are also enrolled and can be still used. Note that if any client would outsource a computation to a misbehaving server then the verification procedure would still ensure that the errors are detected. However, we wish future clients not to waste their limited resources by delegating to a 'bad' server, and wish to disincentivise servers from cheating in the first place, as they know they will be detected and revoked, and therefore incur a significant (financial) penalty from not receiving future work.

Our main contribution in this chapter is to introduce the new notion of *revocable publicly verifiable computation* (RPVC). Our scheme aims to achieve the following goals:

- we allow to enrol multiple servers within a PVC system;
- we allow the server to handle multiple outsourced functions within a single PVC system;
- we introduce the notion of a KDC that is responsible for handling the computational expensive part and to issue evaluation keys for the functions. In case a client detects a cheating server, the client may report to the KDC which in turn is able to revoke the misbehaving server without having to initialise a new PVC system.

We provide a rigorous definitional framework for RPVC, that we believe more ac-

curately reflects real environments than previously considered in the PVC literature. This new framework both removes redundancy and includes additional functionalities, leading to several new security notions.

Related Work

In independent and concurrent work, Carter et al. [45] introduced a trusted third party for a verifiable computation scheme as well. This more powerful entity is responsible to generate garbled circuits for such schemes. However, the solution requires the entity to be online throughout the computation and models the system as a secure multi-party computation protocol between the client, server and third party. We do not believe this solution is practical in general since it is conceivable that a trusted entity is not always available in order to take an active part in computations.
For example, following the battlefield communications scenario by Gennaro et al. [72], VC schemes are required where soldiers are deployed with lightweight computing devices which gather data from their surroundings. The soldiers then send the data to their regional servers for analysis and receive a result that needs to be verified. In this scenario, the trusted party could be located within a high security base or governmental building generating relevant keys which are provided to the soldiers before being deployed with the lightweight device. However, we believe it is rather infeasible for a soldier to contact and maintain a permanent communication link with the headquarters for the duration of a computation.
It seems a reasonable assumption and scenario that in a VC scheme there could be many available servers offering computations but only a single (or a small number) of trusted third parties. The third party could easily become here a bottleneck in the system and limit the number of computations that can take place at any one time.

Organisation of Chapter

In Section 3.2 we introduce our new system model framework for our revocable PVC scheme. This new model leads to new and extended security models which we discuss and analyse in detail in Section 3.3. This is followed, in Section 3.4, by a discussion about the technical challenges within our enhanced model and a concrete instantiation for a RPVC scheme based on the revocable KP-ABE scheme. In Section 3.5 we present detailed proofs of security for the achievable security notions. We conclude the chapter in Section 3.6.

3.2 Revocable Publicly Verifiable Outsourced Computation

The aim of this section is to enhance the existing PVC system model to accommodate a more practical system comprising multiple clients and multiple servers. We introduce a single trusted entity known as the *key distribution centre* (KDC) that acts as an

authority on entities enrolled in the system and thus makes entity management more straightforward. Our model allows multiple servers to compute multiple functions in a secure manner and we ensure that a server is not able to use an evaluation key for a function G to return a valid computational result for $F(x)$.

In the remainder of this section, we discuss in more detail the role of the key distribution centre and discuss two system architectures which later lead to several new security notions. We also provide a formal definition of revocable PVC.

3.2.1 Key Distribution Centre

Existing frameworks assume that a client or several clients run the expensive pre-processing of a VC scheme and that a single server performs the outsourced computation. We believe that this is not adequate for a number of reasons, irrespective of whether the client is sufficiently powerful to perform the required operations. First, in a real-world system, we may wish to outsource the setup phase to a trusted third party. In this setting, the third party would operate rather similarly to a certificate authority, providing a trust service to facilitate other operations of an organisation (in this case outsourced computation, rather than authentication). Second, we may wish to limit the functions that some clients can outsource.

A distinguished client with additional computational resources to perform the expensive pre-processing could act as the KDC. However, we consider the KDC to be a *separate* entity to illustrate separation of duty between the clients that request computations, and the KDC that is authoritative on the system and users, and we minimise its workload to key generation and revocation only.

It may be tempting to suggest that the KDC, as a trusted entity, performs all computations itself. However we believe that this is not a practical solution in many real-world scenarios, e.g. the KDC could be an authority within the organisation responsible for user authorisation that wishes to enable workers to securely use cloud-based Software-as-a-Service. As an entity within organisation boundaries, performing all computations would negate the achieved benefits from initially outsourcing computations to externally available servers.

We want to emphasise that the KDC is basically responsible to perform the expensive pre-processing to reduce the client's computational burden. The KDC is responsible for providing each server with a set of evaluation keys enabling them to perform computations for a set of functions. The client may request the computation of $F(x)$ from any server that is certified to compute F. The KDC also enables efficient revocation of misbehaving servers as the update key material can be easily generated and distributed.

3.2.2 Standard Model

The *standard model* is a natural extension of the PVC architecture of Parno et al. [118] with the addition of a trusted key distribution centre. The entity population comprises a set of clients, a set of servers and a KDC. The KDC initialises the system and generates keys to enable a verifiable computation service. It publishes the keys to delegate computations for the clients, whilst keys to evaluate specific functions are given to individual servers. Clients can submit computation requests to a particular server for their respective inputs, and publish some verification information. The servers receive the encoded input values and perform the computation to generate encoded results. Any party can verify the correctness of the server's output using the published verification information. If the output is incorrect, the verifier may report the misbehaving server to the KDC for revocation, which will prevent the server from performing any further computations within the system.

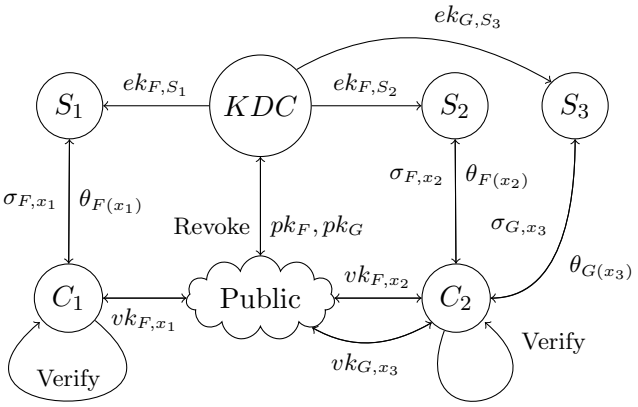

Figure 3.1: Operation of the standard model of a RPVC scheme

Note that the expensive KeyGen operation (pre-processing) is now run by the more capable KDC, and many servers are able to use the generated keys to evaluate the same function, whereas in previous PVC proposals each client would have been required to run KeyGen to set up a system with its choice of server.
Figure 3.1 illustrates the entity population and their respective interaction within the standard model of a PVC scheme.

3.2.3 Manager Model

The *manager model*, in contrast, employs an additional *manager* entity who "owns" a pool of computation servers. Clients submit jobs to the manager, who will select a server from the pool based on workload scheduling, available resources or as a result of some bidding process in case servers are to be rewarded per computation. A plausible scenario is that servers enlist with a manager to "sell" the use of spare resources, whilst

clients subscribe to utilise these through the manager. Encoded results are returned to the manager who should be able to verify the server's work. The manager forwards correct results to the client whilst a misbehaving server may be reported to the KDC for revocation, and the job assigned to another server. Due to public verifiability, any party with access to the output and the verification token can also verify the result. This also enables the client to check whether the manager performed its duties correctly.

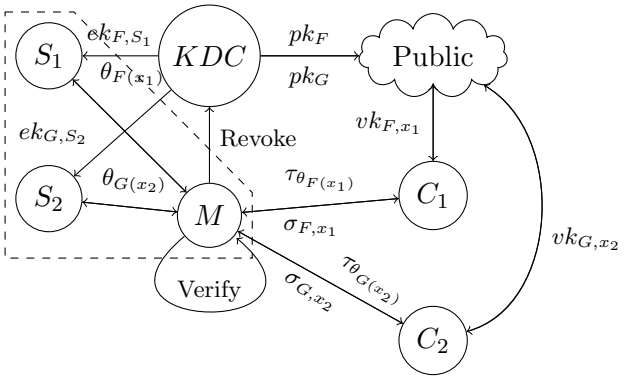

Figure 3.2: Operation of the manager model of a RPVC scheme

The interaction between entities in this model is illustrated in Figure 3.2. The manager and computational servers are shown within a dashed region to illustrate the boundaries of internal and external entities, i.e. the entities not within the dashed region could all be within an organisation that wish to use the external resources provided by the manager to outsource computations.
Notice that the manager performs the verification operation and if the computation is correct the manager forwards the result with an acceptance token to the client. In case the verification procedure fails the manager reports to the KDC for revocation and sends the client a rejection token expressing that within the current time period t the computation failed.

Both models, the standard model and manager model, aim to enhance the existing PVC model to reflect a more practical scenario. However, being in a publicly verifiable setting enables any entity to verify a computational result and thus learn the output. In some scenarios this may be acceptable whereas in others it is not desirable that external entities can access the result, yet there still remains legitimate reasons for specific entities, such as the manager, to verify *only* the correctness of the result without learning the actual value. Therefore, a notion of *blind verification* would be a desirable property such that the manager (or any other entity) may verify only the validity of the computation but is *not* able to determine the actual result of the computation. The delegating client would create an additional (secret) piece of information called the *retrieval token* which enables her to determine the result, and thus would provide a

notion of output privacy. This additional piece of information could also be shared with authorised (trustworthy) entities such that they may learn the result too. Parno et al. also envisage that such a property may be useful in the PVC setting and provide a one line remark that permuting keys and ciphertexts derived during ProbGen could provide output privacy. Unfortunately, our system model cannot provably accommodate the notion of blind verification but we believe that this is an interesting problem to investigate in future work.

In Table 3.1 we provide an overview which entities are responsible for running each algorithm in "normal" verifiable outsourced computation (VC), publicly verifiable outsourced computation (PVC), the standard model of RPVC, and finally RPVC in the Manager model.

Algorithm	**Run by**			
	VC	**PVC**	**RPVC Standard**	**RPVC Manager**
KeyGen	C_1	C_1	KDC	KDC
ProbGen	C_1	$C_1, C_2, \ldots$	$C_1, C_2, \ldots$	$C_1, C_2, \ldots$
Compute	S	S	$S_1, S_2, \ldots$	$S_1, S_2, \ldots$
Verify	C_1	$C_1, C_2, \ldots$	$C_1, C_2, \ldots$	$M, C_1, C_2, \ldots$

Table 3.1: Overview of entity population in various VC models

3.2.4 Formal Definition

We now present a formal definition of all necessary algorithms for a RPVC scheme.

Definition 3.1. *A* revocable publicly verifiable outsourced computation (RPVC) scheme *consists of the following algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$: *this randomised algorithm is run by the KDC to establish public parameters pp and a master secret key mk for a family of functions* $\mathcal{F}$*;*
- $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$: *this randomised algorithm is run by the KDC to generate a public delegation key* pk_F *enabling clients to outsource computations of a function* F*;*
- $sk_S \xleftarrow{\$} \mathsf{Register}(S, mk, pp)$: *this randomised algorithm is run by the KDC to enrol a computation server* S *within the system by generating a personalised signing key* sk_S *for* S*;*
- $ek_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, mk, pp)$: *this randomised algorithm is run by the KDC to certify a computation server by providing it with an evaluation key* $ek_{F,S}$ *for a function* F *and server* S*;*

- $(\sigma_{F,x}, vk_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, pk_F, pp)$: *this randomised algorithm is run by a client to delegate the computation of* $F(x)$ *to a server. The client inputs her input value* x*, the public delegation key and the public parameters. The algorithm outputs an encoded input* $\sigma_{F,x}$ *and a verification key* $vk_{F,x}$ *which can be later used for verifying the returned computational result;*

- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, ek_{F,S}, sk_S, pp)$: *this randomised algorithm is run by a server* S *using its evaluation key* $ek_{F,S}$*, a signing key* sk_S *and an encoded input* $\sigma_{F,x}$ *of* x*. It outputs an encoding* $\theta_{F(x)}$ *of* $F(x)$*;*

- $(y, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x}, pp)$: *this algorithm is run by any verifying party (standard model), or by the manager (manager model), in possession of* $vk_{F,x}$ *and an encoded output* $\theta_{F(x)}$*. It outputs the actual result* y*. If the result* y *corresponds to* $F(x)$ *it additionally creates a token* $\tau_{\theta_{F(x)}} = (\texttt{accept}, S)$ *indicating that the result was correctly computed. Otherwise, the result* y *corresponds to* $\perp$ *and it creates a token* $\tau_{\theta_{F(x)}} = (\texttt{reject}, S)$ *indicating that the result is malformed and* S *misbehaved;*

- $um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(x)}}, mk, pp)$: *this randomised algorithm is run by the KDC inputting the token from the verification process. If* $\tau_{\theta_{F(x)}} = (\texttt{reject}, S)$*, the algorithm revokes all evaluation keys* $ek_{\cdot,S}$ *of the server* S *thereby preventing* S *from performing any further evaluations within the current system. The update material* um *consists of a set of updated evaluation keys* $ek_{\cdot,S'}$ *which are issued to all non-revoked servers. Otherwise, in case* $\tau_{\theta_{F(x)}} = (\texttt{accept}, S)$ *then the algorithm outputs* $\perp$ *indicating that no update was necessary.*

In some instantiations of a PVC scheme, it may not be necessary to issue entirely new evaluation keys to each entity. For example, in our concrete instantiation (cf. Section 3.4), we only need to issue a partially updated key.

Although not explicitly stated, the KDC may update the public parameters pp during any algorithm in order to address any changes within the system. Those changes can encompass of servers being added or removed from the system or a server may be granted the ability to compute additional functions.

Note that in case we would employ the blind verification mechanism, as discussed in Section 3.2.3, into the formal definition of RPVC we would be required to split the verification algorithm into two sub-algorithms. Namely, we would divide it into a *blind verification* and *retrieve* algorithm. As discussed before, blind verification would enable one to check only the correctness of the output without learning the actual result, whereas the retrieve algorithm would then enable the client who initially prepared the encoded input to retrieve the actual result using her secret retrieval token.

We say that a RPVC scheme is *correct* if the verifying party almost certainly accepts the returned result generated by a non-revoked server using a valid generated verification key and encoded output given the non-revoked server used a valid generated encoded input and evaluation key. More formally this can be represented as follows.

Definition 3.2. *A revocable publicly verifiable outsourced computation (RPVC) scheme is* correct *for a family of functions $\mathcal{F}$ if for all functions $F \in \mathcal{F}$, all inputs $x \in \mathsf{Dom}(F)$ and all non-revoked servers S, the following holds:*

$$\begin{aligned}
&\Pr[(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F}),\\
&\quad pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp),\\
&\quad sk_S \xleftarrow{\$} \mathsf{Register}(S, mk, pp),\\
&\quad ek_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, mk, pp),\\
&\quad (\sigma_{F,x}, vk_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, pk_F, pp),\\
&\quad \theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, ek_{F,S}, sk_S, pp),\\
&\quad y \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x}, pp),\\
&\quad um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(x)}}, mk, pp)]\\
&\quad = 1 - \mathrm{negl}(\lambda).
\end{aligned}$$

3.3 Security Models

In this section, we introduce several security models capturing different requirements of a RPVC scheme. The introduction of the KDC, multiple servers and the ability to compute multiple functions, and the subsequent changes in operation give rise to several new security concerns. In more detail:

- **Public Verifiability**: since two (or more) servers may be certified to compute the same function, it is important to ensure that servers cannot collude in order to convince a client to accept an incorrect computational result as a correct result;
- **Revocation**: we must ensure that neither an uncertified nor a de-certified server can convince a client to accept an output;
- **Vindictive Server**: we must ensure that a malicious server cannot convince a client to believe that an honest server has produced an incorrect output;
- **Vindictive Manager**: we must ensure that, in the manager model, a malicious manager cannot convince a client of an incorrect result.

Given those concerns, we define four notions of security for our RPVC scheme where each is modelled as a cryptographic game. For the notions of public verifiability, revocation and vindictive manager, we need to define weaker notions of security which we term

selective and *selective semi-static*, respectively. This restriction arises from the particular IND-sHRSS indirectly revocable key-policy attribute-based encryption scheme we use in our construction that introduces similar constraints (cf. Section 3.4). In other words, given the current primitives we use in our construction we cannot achieve full security for all security notions, but we can achieve slightly weaker variants. However, as we employ the KP-ABE scheme in a black-box fashion, if stronger primitives are found it should be straightforward to change to them and finally achieve full security. In the remainder of this section, we first present the ideal notions of security we wish to achieve for a RPVC scheme. We then discuss the necessary modifications to define the selective and selective, semi-static notions we are currently able to achieve.

3.3.1 Ideal Security Properties

In this section we discuss the *ideal security* notions we wish to achieve in our RPVC framework. Even if it is not possible to achieve all of these notions with the current primitives, we include them for completeness.

3.3.1.1 Public Verifiability

We extend the *public verifiability* game of Parno et al. [118] to formalise that multiple servers should not be able to collude in order to gain an advantage in convincing *any* verifying party of an incorrect output. More formally, that is that the algorithm Verify returns `accept` on an encoded output $\theta^\star$ not corresponding to the correct output $F(x)$ of the computation.
Recall that Parno et al. [118] initially only considered the case where the adversary is limited to learn only one evaluation key and one encoded input. This stems from the fact that the system is initialised only for one server and one function. The motivation for this updated game is that there is a now a trusted party issuing keys to multiple servers that may collude. Here each server is able to request evaluation keys for multiple functions and must not be able to use an evaluation key for a function G to produce a valid looking result for a computation request for $F(x)$. Thus, in the formal description of our game, we allow the adversary to learn multiple evaluation keys for different functions and each is associated with different server identities (since evaluation keys are server-specific in our setting to enable per-server revocation). Furthermore, in our setting, it is likely that the set of servers evaluate many computations on behalf of multiple clients simultaneously, and so we allow the adversary to collect multiple encoded inputs by simply running the $\mathsf{ProbGen}$ algorithm.

In Figure 3.3, we present the ideal notion of public verifiability. The game begins with the challenger setting up the system. The adversary $\mathcal{A}$ receives the resulting parameters and is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm.

$\mathbf{Exp}^{\text{PubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

1 : $(pp, mk) \leftarrow\$ \mathsf{Setup}(1^\lambda, \mathcal{F})$
2 : $(F, x^\star) \leftarrow\$ \mathcal{A}^{\mathcal{O}}(pp)$
3 : $pk_F \leftarrow\$ \mathsf{FnInit}(F, mk, pp)$
4 : $(\sigma_{F,x^\star}, vk_{F,x^\star}) \leftarrow\$ \mathsf{ProbGen}(x^\star, pk_F, pp)$
5 : $\theta^\star \leftarrow\$ \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, vk_{F,x^\star}, pk_F, pp)$
6 : $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x^\star}, pp)$
7 : **if** $(y, \tau_{\theta^\star}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F(x^\star))$ **then**
8 : **return** 1
9 : **else return** 0

Figure 3.3: The ideal public verifiability experiment $\mathbf{Exp}^{\text{PubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

This step models the adversary observing an existing RPVC system and corrupting various servers to learn their evaluation keys.
Eventually, the adversary finishes its query phase to the oracles and outputs its choice of challenge pair consisting of the challenge function F and challenge input $x^\star$. Note that the adversary's underlying goal is to convince the client of accepting an incorrect result for the computation $F(x^\star)$. The challenger will then run FnInit to initialise the challenge function F and generate a challenge by running $\mathsf{ProbGen}$ on input $x^\star$, and give the resulting encoded input to $\mathcal{A}$. The adversary is again given oracle access and eventually outputs $\theta^\star$. It wins if the encoded output verifies correctly but does not encode the value $F(x^\star)$.

Definition 3.3. *The* advantage *of a PPT adversary in the* PubVerif *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{PubVerif}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{PubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$ is secure with respect to public verifiability *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{PubVerif}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

In practical environments, a server may be interacting with multiple clients simultaneously and it could be that having multiple simultaneous interactions could provide an advantage against any *one* of the computations. Thus, when modelling this scenario as a game, we may wish the adversary would choose a polynomially sized set of input

values to be challenged upon to model these simultaneous inputs and for the adversary to win against any one of the inputs. We formalise this for the case of public verifiability and present the notion of public verifiability with a polynomial sized set of input values in Figure 3.4. In the following theorem, we show that this notion is polynomially equivalent to the case where the adversary chooses a single challenge input as described in Figure 3.3.

$\mathbf{Exp}^{\text{mPubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

1 : $(pp, mk) \leftarrow\$ \mathsf{Setup}(1^\lambda, \mathcal{F})$
2 : $\{(F_i, x_i^\star)\}_{i\in[n]} \leftarrow\$ \mathcal{A}^{\mathcal{O}}(pp)$
3 : **for** $i = 1, \ldots, n$ **do**
4 : $\quad pk_{F_i} \leftarrow\$ \mathsf{FnInit}(F_i, mk, pp)$
5 : $\quad (\sigma_{F_i,x_i^\star}, vk_{F_i,x_i^\star}) \leftarrow\$ \mathsf{ProbGen}(x_i^\star, pk_{F_i}, pp)$
6 : $\theta^\star \leftarrow\$ \mathcal{A}^{\mathcal{O}}(\{\sigma_{F_i,x_i^\star}, vk_{F_i,x_i^\star}, pk_{F_i}\}, pp)$
7 : $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x_i^\star}, pp)$
8 : **if** $\exists i \in [n] : (y, \tau_{\theta^\star}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F_i(x_i^\star))$ **then**
9 : $\quad$ **return** 1
10 : **else return** 0

Figure 3.4: The public verifiability experiment with polynomial sized set of input values $\mathbf{Exp}^{\text{mPubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

Theorem 3.4. *Let λ be the security parameter, and $n(\lambda) \in \mathbb{N}$ a function polynomial in the security parameter. Then public verifiability where the adversary may target an arbitrary set of n challenge inputs (Figure 3.4) is polynomially equivalent to public verifiability where the adversary chooses a single challenge input (Figure 3.3).*

Proof. It is trivial to show that security with multiple choices implies security with a single choice, since an adversary with multiple choices could simply choose $n = 1$ and output a single choice.
To see that security with a single choice also implies security with multiple choices we can perform the following reduction. Suppose that $\mathcal{RPVC}$ is a secure RPVC scheme when the adversary $\mathcal{A}_{\text{S}}$ makes a single choice of challenge input. For contradiction, let $\mathcal{A}_{\text{M}}$ be an adversary with non-negligible advantage δ against $\mathcal{RPVC}$ when it can make multiple challenge choices. We show that $\mathcal{A}_{\text{S}}$ could use $\mathcal{A}_{\text{M}}$ as a sub-routine to gain a non-negligible advantage against $\mathcal{RPVC}$ even with just a single challenge choice. Let $\mathcal{C}$ be the challenger playing the public verifiability game with a single challenge as in Figure 3.3 with $\mathcal{A}_{\text{S}}$ that in turn acts as the challenger for $\mathcal{A}_{\text{M}}$ in the public verifiability game with multiple challenges as described in Figure 3.4. It follows:

1. $\mathcal{C}$ runs Setup and sends the resulting parameters to $\mathcal{A}_{\mathrm{S}}$ that simply forwards them to $\mathcal{A}_{\mathrm{M}}$.
2. $\mathcal{A}_{\mathrm{M}}$ makes oracle queries which $\mathcal{A}_{\mathrm{S}}$ passes to $\mathcal{C}$ and forwards the response to $\mathcal{A}_{\mathrm{M}}$.
3. $\mathcal{A}_{\mathrm{M}}$ will return a set of n challenge input pairs $\{(F_i, x_i^\star)\}_{i\in[n]}$.
4. $\mathcal{A}_{\mathrm{S}}$ chooses one of these input pairs at random, $(F, x^\star) \xleftarrow{\$} \{(F_i, x_i^\star)\}_{i\in[n]}$, and sends this to $\mathcal{C}$.
5. $\mathcal{C}$ returns the results of running FnInit and $\mathsf{ProbGen}$ on F and $x^\star$ and then provides further oracle access to $\mathcal{A}_{\mathrm{S}}$.
6. $\mathcal{A}_{\mathrm{S}}$ can query the FnInit oracle for all other public keys pk_{F_i} as well as query the $\mathsf{ProbGen}$ oracle for the remaining inputs $\{x_i^\star\}_{i\in[n]} \setminus x^\star$ and returns the whole set to $\mathcal{A}_{\mathrm{M}}$. Since no other query was made between $\mathcal{C}$ generating the challenge and these $\mathsf{ProbGen}$ queries, the system parameters have not changed and all challenges are consistent.
7. $\mathcal{A}_{\mathrm{M}}$ makes oracle queries which $\mathcal{A}_{\mathrm{S}}$ again forwards to $\mathcal{C}$, and eventually outputs a challenge output $\theta^\star$.
8. Let $x_j^\star$ be the challenge input corresponding to $\theta^\star$. In more detail, since $\mathcal{A}_{\mathrm{M}}$ is assumed to be successful, the algorithm $\mathsf{Verify}(\theta^\star, vk_{F,x_j^\star}, pp)$ does not return $(\bot, (\mathtt{reject}, \cdot))$ and hence $\theta^\star$ is a valid encoding of $F(x_j^\star)$ for some $x_j^\star$. If $x_j^\star = x^\star$ then $\mathcal{A}_{\mathrm{S}}$ forwards $\theta^\star$ to $\mathcal{C}$ as its result. Otherwise, $\mathcal{A}_{\mathrm{S}}$ stops.

Hence,

$$\mathbf{Adv}^{\textsc{PubVerif}}_{\mathcal{A}_{\mathrm{S}},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \Pr[x^\star = x_j^\star] \cdot \mathbf{Adv}^{\textsc{mPubVerif}}_{\mathcal{A}_{\mathrm{M}},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \frac{\delta}{n}.$$

Since we assumed δ being non-negligible, and as n is polynomial in λ, we conclude that $\mathbf{Adv}^{\textsc{PubVerif}}_{\mathcal{A}_{\mathrm{S}},\mathcal{RPVC}}(1^\lambda, \mathcal{F})$ is non-negligible. However, we assumed that $\mathcal{RPVC}$ was secure against a single challenge and hence the adversary making multiple challenges with non-negligible advantage may not exist. □

Similar arguments hold for the other games, and henceforth we shall only consider single challenges.

3.3.1.2 Revocation

The notion of *revocation* requires that, if a server is detected as misbehaving, meaning that a server S returns a result such that the verification algorithm Verify outputs $(\bot, (\mathtt{reject}, S))$, then any subsequent computations by S should be rejected, even if the result may be correct.

The motivation here is that even though the costly computation and pre-processing stages have been outsourced to the server and KDC respectively, there is still a cost involved for the client to delegate and verify a computation. Our underlying aim is to remove any incentive for a malicious server to attempt providing an outsourcing service since the server is aware that no result will be accepted. In addition, we may punish and further disincentive malicious servers by removing their ability to perform computations and thus earn rewards. Finally, from a privacy perspective, we may not wish to supply input data to a server that is known to be untrustworthy.

The ideal notion of revocation is defined in Figure 3.5. The game begins by declaring a Boolean flag `chall` which is initially set to `false` and a list Q_{Rev} in which servers will be added in case they are revoked and removed from when re-certified. The `chall` flag will be set to `true` when the challenge is created, and after this point Q_{Rev} is no longer updated. Thus Q_{Rev} will comprise all server identities that are revoked at the challenge time. The adversary wins if it can output a result 'from' one of these servers and have it accepted in the verification stage.

The game proceeds in a similar fashion to the notion of public verifiability with the challenger running Setup to initialise the system and providing the public parameters to the adversary. The adversary is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm except for the $\mathsf{Certify}$ and Revoke oracles which additionally maintain the list of revoked entities. The formal details are specified in the respective oracles in Figure 3.5. After the adversary has finished this query phase, it outputs a challenge function F and challenge input $x^\star$. The challenger runs FnInit for the challenge function and sets the `chall` flag to `true`. It then generates the challenge by running $\mathsf{ProbGen}$ on $x^\star$ and gives the resulting parameters to the adversary along with oracle access again. However we want to emphasise again that since `chall` is set to `true`, Q_{Rev} will no longer be updated. Eventually, the adversary outputs a result $\theta^\star$ and wins if Verify successfully accepts the result (either correct or malformed) for a server that was revoked when the challenge was generated.

Definition 3.5. *The* advantage *of a PPT adversary in the* Revoc *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{Revoc}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{Revoc}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$ is

$\mathbf{Exp}^{\text{REVOC}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

1 : $\texttt{chall} = \texttt{false}$
2 : $Q_{\text{Rev}} \leftarrow \epsilon$
3 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{F})$
4 : $(F, x^\star) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(pp)$
5 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
6 : $\texttt{chall} = \texttt{true}$
7 : $(\sigma_{F,x^\star}, vk_{F,x^\star}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x^\star, pk_F, pp)$
8 : $\theta^\star \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, vk_{F,x^\star}, pk_F, pp)$
9 : $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x^\star}, pp)$
10 : **if** $(y, \tau_{\theta^\star}) = (y, (\texttt{accept}, S))$ **and** $(S \in Q_{\text{Rev}})$ **then**
11 : **return** 1
12 : **else return** 0

$\mathcal{O}^{\mathsf{Certify}}(S, F', mk, pp)$

1 : **if** $(\texttt{chall} = \texttt{false})$ **then** $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S$
2 : **return** $\mathsf{Certify}(S, F', mk, pp)$

$\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(x)}}, mk, pp)$

1 : $um \leftarrow\!\!\$\ \mathsf{Revoke}(\tau_{\theta_{F'(x)}}, mk, pp)$
2 : **if** $(um \neq \perp$ **and** $\texttt{chall} = \texttt{false})$ **then** $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \cup S$
3 : **return** um

Figure 3.5: The ideal revocation experiment $\mathbf{Exp}^{\text{REVOC}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

secure with respect to revocation *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{REVOC}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

3.3.1.3 Vindictive Server

This security notion of *vindictive server* is primarily motivated in the context of the manager model where a pool of computational servers is available to accept a 'job' but they are distributed by the manager such that the client does not know a priori the server identity to which the job was assigned. Since an invalid result can lead to revocation, this reveals a new threat model (particularly if servers are rewarded per computation). A malicious server may return incorrect results but attribute them to a different server identity such that a potentially honest server is punished by being revoked on the basis of the malicious server's behaviour. The pool of available servers

for future computations is therefore reduced in size and thus leads to a likely increase in rewards for the malicious server since it may get assigned more work.

The notion of vindictive servers is formalised in Figure 3.6. The game starts with the challenger maintaining a list of currently registered entities Q_{Reg}, a list Q_{Server} of servers for which the adversary was able to learn the signing key as well as initialising $\overline{S}$ which represents the target server identity, initially set to $\perp$, until the adversary selects its target identity. The game proceeds from here on similarly to the previous notions except that, on line 8 and 10, the adversary chooses the target server identity $\overline{S}$ and then generates an encoded output that the adversary hopes will lead to the revocation of $\overline{S}$.
The adversary is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm except for the Certify and the two additionally defined oracles Register2 and Compute. The formal details are provided in the respective oracles represented in Figure 3.6. In general, those oracles must ensure that the adversary is never issued the signing key $sk_{\overline{S}}$ as the adversary would then be trivially able to act like $\overline{S}$ and win the game. For the same reason, the adversary loses the game in line 9 if it has previously learnt the signing key $sk_{\overline{S}}$ for the target server chosen in line 8 meaning that $\overline{S}$ is listed on Q_{Server}. In line 10, the adversary is additionally provided with access to a compute oracle $\mathcal{O}^{\mathsf{Compute}}(\cdot, ek_{\cdot,\overline{S}}, sk_{\overline{S}}, pp)$ which enables the adversary to observe evaluation results generated by the target server $\overline{S}$. This oracle basically models the adversary observing $\overline{S}$ before attacking.

In more detail, the Register oracle outputs $\perp$ if it is queried with the target identity $\overline{S}$. Otherwise, if the queried server identity is not already recorded on the list of registered entities Q_{Reg}, then the oracle adds the queried server identity to the lists Q_{Reg} and Q_{Server} as it will issue a signing key sk_S. On line 8, the adversary is additionally provided with a modified Register oracle. This Register2 oracle performs similarly to the Register oracle but *does not* return the resulting signing key sk_S. However, it may update the public parameters to reflect any additional registered entities within the system. The adversary may query *any* identity to Register2 including $\overline{S}$. The purpose of this oracle is to model the adversary observing uncorrupted servers within the RPVC system which it can target for revocation. Formally, this means the adversary is able to enrol servers within the system but unable to learn any server specific secrets. Those restrictions need to be considered in the oracles as if the adversary corrupts a server and learns its secret signing key, then the adversary would be able to trivially output an incorrect result and cause the server to be revoked. However, this would contradict the aim of this notion which is to cause an honest, uncorrupted server to be revoked.

$\mathbf{Exp}^{\text{VINDS}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

$Q_{\text{Reg}} \leftarrow \epsilon$
$Q_{\text{Server}} \leftarrow \epsilon$
$\overline{S} \leftarrow \perp$
$(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{F})$
$(F, x^\star) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O},\mathsf{Register2}}(pp)$
$pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
$(\sigma_{F,x^\star}, vk_{F,x^\star}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x^\star, pk_F, pp)$
$\overline{S} \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O},\mathsf{Register2}}(\sigma_{F,x^\star}, vk_{F,x^\star}, pk_F, pp)$
if $(\overline{S} \in Q_{\text{Server}})$ **then return** 0
$\theta^\star \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O},\mathsf{Register2},\mathsf{Compute}}(\sigma_{F,x^\star}, vk_{F,x^\star}, pk_F, pp)$
$(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x^\star}, pp)$
if $(y, \tau_{\theta^\star}) = (\perp, (\texttt{reject}, \overline{S}))$ **and** $(\perp \nleftarrow \mathsf{Revoke}(\tau_{\theta^\star}, mk, pp))$ **then**
  return 1
else return 0

$\mathcal{O}^{\mathsf{Register}}(S, mk, pp)$

if $(S = \overline{S})$ **then return** $\perp$
if $(S, \cdot) \notin Q_{\text{Reg}}$ **then**
  $sk_S \leftarrow\!\!\$\ \mathsf{Register}(S, mk, pp)$
  $Q_{\text{Reg}} \leftarrow Q_{\text{Reg}} \cup (S, sk_S)$
  $Q_{\text{Server}} \leftarrow Q_{\text{Server}} \cup S$
return sk_S

$\mathcal{O}^{\mathsf{Register2}}(S, mk, pp)$

if $(S, \cdot) \notin Q_{\text{Reg}}$ **then**
  $sk_S \leftarrow\!\!\$\ \mathsf{Register}(S, mk, pp)$
  $Q_{\text{Reg}} \leftarrow Q_{\text{Reg}} \cup (S, sk_S)$
  $Q_{\text{Server}} \leftarrow Q_{\text{Server}} \cup S$
return $\perp$

$\mathcal{O}^{\mathsf{Compute}}(\sigma_{F',x}, ek_{F',\overline{S}}, sk_{\overline{S}}, pp)$

if $(F' = F)$ **and** $(x = x^\star)$ **then**
  return $\perp$
else return $\mathsf{Compute}(\sigma_{F',x}, ek_{F',\overline{S}}, sk_{\overline{S}}, pp)$

Figure 3.6: The ideal vindictive server experiment $\mathbf{Exp}^{\text{VINDS}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

The oracles Register and Register2 are modified compared to the standard Register oracle. Both oracles first check whether the queried server identity has already been added to the list of registered entities Q_{Reg}. If not, then both algorithms run the usual Register algorithm and record the server identity and signing key to the list Q_{Reg}. In case that the server was already registered then the Register oracle returns the stored signing key whilst the Register2 oracle returns $\perp$. Thus, both oracles will generate together a single singing key per server and output identical responses.

In order to avoid a trivial win for the adversary in which the adversary simply forwards prior outputs that were actually generated by $\overline{S}$, we need to restrict queries to the Compute oracle. The adversary is restricted to not being able to ask for the evaluation key for the challenge computation $F(x^\star)$ from $\overline{S}$. Note that for all other servers than $\overline{S}$, the adversary is able to run Compute itself using all parameters learnt from the other queries.
Finally, the adversary wins if the challenger believes that $\overline{S}$ generated the encoded output which does not verify correctly and therefore revokes $\overline{S}$.

Definition 3.6. *The* advantage *of a PPT adversary in the* VINDS *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{VINDS}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{VINDS}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right].$$

We say that the revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$ is secure with respect to vindictive servers *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{VINDS}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

3.3.1.4 Vindictive Manager

The notion of *vindictive manager* is a natural extension of the public verifiability notion to the manager model where a vindictive manager may attempt to provide a client with an incorrect answer. In this notion, clients may subscribe to a pool of servers managed by a manager. However, the manager may not wish to be responsible for incorrect results in order to avoid losing business and also may not have the required resources to re-compute a malformed result. Thus, occasionally, the manager may try to send an incorrect result to the client on purpose claiming that it is correct.

We note that instantiations may vary depending on the level of trust given to the manager. For example, a completely trusted manager may simply return the result to a client, whilst an untrusted manager may have to provide the full output from the server so that the client can perform the full verification step as well. Note that in this case, security against vindictive managers will reduce to security against public verifiability since the manager would need to forge an encoded output that passes a full verification step. Here in our framework, we consider a middle ground where the manager is semi-trusted but the clients would still like a final, efficient check.

The ideal notion of security against vindictive manager is defined in Figure 3.7. The game starts with the challenger initialising the system as usual. The adversary receives the resulting parameters and is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$,

$\mathbf{Exp}_{\mathcal{A}}^{\text{VINDM}}\left[\mathcal{RPVC}, 1^{\lambda}, \mathcal{F}\right]$

1 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^{\lambda}, \mathcal{F})$
2 : $(F, x^{\star}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(pp)$
3 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
4 : $S \leftarrow\!\!\$\ \mathcal{U}_{\text{ID}}$
5 : $sk_S \leftarrow\!\!\$\ \mathsf{Register}(S, mk, pp)$
6 : $ek_{F,S} \leftarrow\!\!\$\ \mathsf{Certify}(S, F, mk, pp)$
7 : $(\sigma_{F,x^{\star}}, vk_{F,x^{\star}}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x^{\star}, pk_F, pp)$
8 : $\theta_{F(x^{\star})} \leftarrow\!\!\$\ \mathsf{Compute}(\sigma_{F,x^{\star}}, ek_{F,S}, sk_S, pp)$
9 : $(y, \tau_{\theta_{F(x^{\star})}}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^{\star}}, \theta_{F(x^{\star})}, vk_{F,x^{\star}}, pk_F, pp)$
10 : **if** $(y, \tau_{\theta_{F(x^{\star})}}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F(x^{\star}))$ **then**
11 : **return** 1
12 : **else return** 0

Figure 3.7: The ideal vindictive manager experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{VINDM}}\left[\mathcal{RPVC}, 1^{\lambda}, \mathcal{F}\right]$

$\mathsf{Certify}(\cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. Each oracle simply runs the relevant algorithms. After the adversary has finished its query phase, it outputs a challenge function F and challenge input $x^{\star}$. The challenger runs FnInit and outputs the public delegation key. In the next step, the challenger randomly selects a server identity from the space of all identities $\mathcal{U}_{\text{ID}}$ for which the challenge will be created. The challenger runs $\mathsf{Register}$ and $\mathsf{Certify}$ for this server, creates a problem instance by running $\mathsf{ProbGen}$ on $x^{\star}$ and finally runs $\mathsf{Compute}$ on the generated encoded input. The adversary is then given the encoded input, the verification key, the encoded output from $\mathsf{Compute}$, the usual parameters, as well as the usual oracle access as detailed above, and eventually outputs the result y which corresponds to $\theta_{F(x^{\star})}$ and an acceptance token $\tau_{\theta_{F(x^{\star})}}$. The adversary wins if the challenger accepts this output and $y \neq F(x^{\star})$.

Definition 3.7. *The* advantage *of a PPT adversary in the* VINDM *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}_{\mathcal{A},\mathcal{RPVC}}^{\text{VINDM}}(1^{\lambda}, \mathcal{F}) = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{VINDM}}\left[\mathcal{RPVC}, 1^{\lambda}, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$ is secure with respect to vindictive managers *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}_{\mathcal{A},\mathcal{RPVC}}^{\text{VINDM}}(1^{\lambda}, \mathcal{F}) \leq \text{negl}(\lambda).$$

3.3.2 Restricted Security Properties

In this section we present the actual *restricted security properties* which we can provably achieve with the current primitives. In more detail, using the current primitives, we cannot achieve the ideal notion of security for public verifiability, revocation and vindictive manager. We are therefore required to introduce slightly weaker versions of these security notions in order to reflect the similar restrictions placed on our construction by employing an IND-sHRSS indirectly revocable KP-ABE scheme as introduced in Section 2.3.2. Since we use this primitive in a black-box manner, any achievement with finding a fully secure primitive with the same functionality should easily lead to achieve ideal security for our suggested notions.

These restricted security variants require up to two additional restrictions on the adversary. Namely, a *selective* and a *semi-static* restriction. In more detail, firstly, a selective restriction requires the adversary to choose the set of inputs for the challenge stage *before* seeing the public parameters. This notion stands in contrast to the ideal notion where the adversary can declare the set of inputs after it has seen the public parameters as well as accessed the oracles to the system. This restriction has similarly been used in many ABE schemes throughout the literature to give a heuristic level of security when the ideal notion is difficult to achieve as it allows to initialise the system with a particular attack target in mind.
Secondly, a semi-static restriction requires the adversary to declare a challenge list $\overline{R}$ of servers that must be revoked from the system before $\mathsf{ProbGen}$ generates the challenge encoded inputs and before the adversary is provided with oracle access. This restriction is related to the revocation mechanism of the revocable KP-ABE scheme and means that oracles are able to refuse to perform oracle queries that would lead to trivial wins for the adversary, for example by issuing functional evaluation keys for servers that should be revoked during the challenge time period. To formally accommodate this semi-static restriction, we need to add some additional steps to each security notion. In more detail, the challenger defines the parameter t which models the underlying system time and is initially set to 1, and the second parameter is Q_{Rev} which is a list comprising all currently revoked server identities during the current time period. Note that Q_{Rev} is initialised to be empty when the system is set up describing that no server has been revoked yet.
Whenever a revocation query has been made to the system both parameters are usually updated. The time parameter is incremented for every revocation query indicating that keys from prior time periods may no longer be valid. For the parameter Q_{Rev}, every revocation query means that servers are added to the list if it was queried with a rejection token $(\mathtt{reject}, \cdot)$, and it can possibly mean that servers are removed from the list in case they are re-certified for a function. Note that unless a server is added or removed from Q_{Rev}, the revocation list remains consistent over all consecutive oracle

queries modelling a realistic system evolution. Recall that the semi-static restriction forces the adversary to choose a list of revoked servers $\overline{R}$ for the challenge time period. Note that if the current list of revoked servers Q_{Rev} for the challenge time period $t^\star$ is not a superset of $\overline{R}$, i.e. there exists a server that the adversary claimed would be revoked but actually is not revoked, then the adversary loses the game to avoid a trivial win since the adversary has not made a suitable sequence of oracle queries.
To avoid other trivial wins we need to restrict the adversary's oracle queries such that it cannot obtain both a secret key *and* an update key, which together form a valid evaluation key for a server that is revoked at the challenge time. Otherwise, if the adversary can form a functional evaluation key it can evaluate the encoded input and output a correct response. However, a revoked server would not have such an ability in practice.

In the IND-sHRSS game [17], update keys are associated with a particular time period and queries can be made for arbitrary time periods. However, in our RPVC setting, we consider an interactive protocol and as such require that time increases monotonically. The adversary in the IND-sHRSS game selects a time period for the challenge as well as the challenge input. However, in our security notions, we parametrise the adversary on the number of queries q it requests to the oracles and define security over all choices of q. In particular, we restrict the adversary to make $q_t \leq q$ queries to the Revoke oracle in its first oracle query phase. After q_t queries have been made the challenge needs to be generated. Since the time parameter is only incremented whenever a Revoke query has been made, the challenge occurs when $t^\star = q_t$, and hence the challenger may select $t^\star$ as its challenge time in a reductive proof.

In order to remove the above restrictions we would require, on one hand, a fully secure indirectly revocable KP-ABE scheme to overcome the selective restriction, and on the other hand, an adaptive notion of revocation to overcome the semi-static restriction. However, currently instantiating such a primitive is an open problem.[1]

3.3.2.1 Selective Public Verifiability

In Figure 3.8, we define a *selective notion of public verifiability*. The only difference between the selective notion and the ideal notion, represented in Figure 3.3, is that in the selective notion the adversary is required to provide its challenge inputs F and $x^\star$ *before* the challenger runs Setup. Otherwise, the game proceeds identical to the ideal notion.
Note that in this security notion, the semi-static restriction is not required since the revocation mechanism is not part of the winning condition.

[1] Attrapadung and Imai [17] defined a notion with adaptive queries but did not provide an instantiation.

$\mathbf{Exp}^{\text{sPubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

1 : $(F, x^\star) \leftarrow_\$ \mathcal{A}(1^\lambda, \mathcal{F})$
2 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mathcal{F})$
3 : $pk_F \leftarrow_\$ \mathsf{FnInit}(F, mk, pp)$
4 : $(\sigma_{F,x^\star}, vk_{F,x^\star}) \xleftarrow{\$} \mathsf{ProbGen}(x^\star, pk_F, pp)$
5 : $\theta^\star \leftarrow_\$ \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, vk_{F,x^\star}, pk_F, pp)$
6 : $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x^\star}, pp)$
7 : **if** $(y, \tau_{\theta^\star}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F(x^\star))$ **then**
8 : **return** 1
9 : **else return** 0

Figure 3.8: The selective public verifiability experiment $\mathbf{Exp}^{\text{sPubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

Definition 3.8. *The* advantage *of a PPT adversary in the* sPubVerif *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{sPubVerif}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{sPubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the revocable publicly verifiable computation outsourced scheme $\mathcal{RPVC}$ is secure with respect to selective public verifiability *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sPubVerif}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

3.3.2.2 Selective, Semi-static Revocation

In Figure 3.9, we consider the *selective, semi-static notion of revocation.* Recall from Section 3.3.1.2 that the winning condition against the security notion was formalised in terms of the challenger accepting *any* result formed by a revoked entity. Since revocation is a central requirement in the winning condition, we require here both the *selective* and *semi-static* restrictions to accommodate the IND-sHRSS game.

The game begins with the adversary selecting the challenge function F and challenge input $x^\star$ to be outsourced. The challenger now initialises an initially empty list of currently revoked entities Q_{Rev} as well as a time parameter t. Next, the challenger runs Setup and FnInit to initialise the system and to derive the public delegation key pk_F for the function F. The adversary receives all public parameters and is required to output a list $\overline{R}$ of servers that need to be revoked when the challenge is created. In the next

$\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}, q_t\right]$

1 : $(F, x^\star) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda, \mathcal{F}, q_t)$
2 : $Q_{\text{Rev}} \leftarrow \epsilon$
3 : $t \leftarrow 1$
4 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{F})$
5 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
6 : $\overline{R} \leftarrow \mathcal{A}(pk_F, pp)$
7 : $\mathcal{A}^{\mathcal{O}}(pk_F, pp)$
8 : **if** $(\overline{R} \not\subseteq Q_{\text{Rev}})$ **then return** 0
9 : $(\sigma_{F,x^\star}, vk_{F,x^\star}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x^\star, pk_F, pp)$
10 : $\theta^\star \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, vk_{F,x^\star}, pk_F, pp)$
11 : $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x^\star}, pp)$
12 : **if** $(y, \tau_{\theta^\star}) = (y, (\texttt{accept}, S))$ **and** $(S \in \overline{R})$ **then**
13 : **return** 1
14 : **else return** 0

$\mathcal{O}^{\mathsf{Certify}}(S, F', mk, pp)$

1 : **if** $(F' = F$ **and** $S \notin \overline{R})$ **or** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S)$ **then return** $\perp$
2 : $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S$
3 : **return** $\mathsf{Certify}(S, F', mk, pp)$

$\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(x)}}, mk, pp)$

1 : $t \leftarrow t + 1$
2 : **if** $(\tau_{\theta_{F'(x)}} = (\texttt{accept}, \cdot))$ **then return** $\perp$
3 : **if** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\text{Rev}} \cup S)$ **then return** $\perp$
4 : $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \cup S$
5 : **return** $\mathsf{Revoke}(\tau_{\theta_{F'(x)}}, mk, pp)$

Figure 3.9: The selective, semi-static revocation experiment $\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}, q_t\right]$

step, line 7, the adversary is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. The oracles simply run the usual algorithms except for queries to the Certify and Revoke oracles, which the challenger replies to accordingly as specified in Figure 3.9. The challenger needs to ensure that the revocation list Q_{Rev} is permanently kept up-to-date by adding and removing the queried entities. In case of revocation the challenger needs to ensure that the time parameter is incremented to the next step and requires that no issued keys

will lead to a trivial win. The latter part is handled by the Certify algorithm which does not issue evaluation keys $ek_{F,S}$ for the challenge function F and a server S that may not be revoked at the time the challenge is generated.

Recall that the adversary is, due to the semi-static restriction, parametrised to make exactly q_t revocation queries and the time parameter is only incremented during the revocation algorithm. Thus, the challenge time period occurs when $t = q_t$. Following the second restriction of the Certify oracle in Figure 3.9, an evaluation key for a server S should not be issued if requested during the challenge time period q_t and if there exists any server (other than S which is about to be certified) within the system that according to the adversary's chosen challenge revocation list $\overline{R}$ should be revoked but has not actually been revoked, i.e. not listed in Q_{Rev}. Intuitively, this restriction means that Certify issues a valid and functional evaluation key within the current time period and such a key can only be disabled by revoking the particular server which on the other hand then leads to the requirement of incrementing the time period. Incrementing the time may be a problem if the challenge time period is already reached after q_t Revoke queries. Our particular construction may reveal generated update material via Certify from the latest revocation procedure that enables the evaluation keys to be functional for the current time period. If such update material is issued then this leads to any non-revoked evaluation key being updated for the current challenge time period q_t and therefore can be used to perform computations and return valid results that are accepted by the challenger. Therefore, if such an updated key belongs to a server that was listed on $\overline{R}$, then this would count as a trivial win for the adversary since the adversary claimed that this server would be revoked for the challenge time period but was not revoked.

Whenever the Revoke oracle is queried in Figure 3.9, first the time parameter t will be incremented and the oracle returns $\perp$ if the queried token corresponds to an acceptance token $(\texttt{accept}, \cdot)$ which reflects that no server needs to be revoked. Since the time parameter t is incremented for each query made to the oracle, the adversary may query acceptance tokens to Revoke in order to progress the system time without altering the revocation list if desired. However, in case a query is made at the challenge time period $t = q_t$, the challenger must return $\perp$ if the chosen challenge revocation list $\overline{R}$ is not a subset of the current revocation list including the queried S as it is about to be revoked. In other words, the algorithm outputs $\perp$ if there exists a server on the challenge revocation list $\overline{R}$ that should be revoked, other than S, but is not on the list of currently revoked servers Q_{Rev}. This requirement is needed in order to avoid a trivial win for the adversary, since otherwise the adversary can request an updated evaluation key for a server listed on $\overline{R}$ which enables the adversary to create a valid result which will be accepted by the challenger as it was generated by a non-revoked

server.

The adversary finishes its oracles query phase (line 7) after making a polynomial number of queries q, including q_t many Revoke queries, and does not return a value other than signalling to the challenger that it may proceed with the remainder of the game. The challenger checks that all queries made by the adversary have indeed generated a list of currently revoked servers that is a superset of the challenge revocation list $\overline{R}$. If this is not true, the challenger aborts the game and the adversary loses as it was not able to choose its queries or the list $\overline{R}$ appropriately. Otherwise, the challenger continues with the game and generates the challenge by running ProbGen on $x^\star$ and provides the resulting encoded input to the adversary. The adversary is again provided with oracle access as above and eventually outputs its guess $\theta^\star$. The adversary wins if the challenger accepts *any* result $\theta^\star$, i.e. a correct or malformed response, as a valid result from any server that was revoked at the time of the challenge which were at least the ones chosen by the adversary on the list $\overline{R}$.

Definition 3.9. *The* advantage *of a PPT adversary in the* sssRevoc *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{sssRevoc}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}, q_t) = \Pr\left[\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}, q_t\right] \to 1\right].$$

We say that the revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$ is secure with respect to selective, semi-static revocation *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sssRevoc}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}, q_t) \leq \text{negl}(\lambda).$$

3.3.2.3 Selective Vindictive Manager

Recall that the security notion of *vindictive manager* is a natural extension of the public verifiability notion to the manager model where a vindictive manager may attempt to provide a client with an incorrect answer. Note that the winning condition of vindictive manager, as in the notion of public verifiability, does not rely on the revocation mechanism and therefore we only require the selective restriction for this security notion. In Figure 3.10, we formally consider the *selective notion of vindictive manager*.

The game starts with the adversary selecting its challenge function F and challenge input $x^\star$. The challenger initialises the system as usual by running Setup and FnInit for F. It then randomly selects a server from the space of server identities $\mathcal{U}_{\text{ID}}$ for which the challenger generates the challenge parameters for the adversary. The challenger runs Register and Certify for the chosen server and challenge function, runs ProbGen

$\mathbf{Exp}^{\text{sVIndM}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

$(F, x^\star) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda, \mathcal{F})$
$(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{F})$
$pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
$S \leftarrow\!\!\$\ \mathcal{U}_{\text{ID}}$
$sk_S \leftarrow\!\!\$\ \mathsf{Register}(S, mk, pp)$
$ek_{F,S} \leftarrow\!\!\$\ \mathsf{Certify}(S, F, mk, pp)$
$(\sigma_{F,x^\star}, vk_{F,x^\star}) \leftarrow\!\!\$\ \mathsf{ProbGen}(x^\star, pk_F, pp)$
$\theta_{F(x^\star)} \leftarrow\!\!\$\ \mathsf{Compute}(\sigma_{F,x^\star}, ek_{F,S}, sk_S, pp)$
$(y, \tau_{\theta_{F(x^\star)}}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, \theta_{F(x^\star)}, vk_{F,x^\star}, pk_F, pp)$
if $(y, \tau_{\theta_{F(x^\star)}}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F(x^\star))$ **then**
  return 1
else return 0

Figure 3.10: The selective vindictive manager experiment $\mathbf{Exp}^{\text{sVIndM}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$

on the challenge input, and finally runs $\mathsf{Compute}$ to output an encoded output $\theta_{F(x^\star)}$. The adversary is provided with all public parameters, the encoded input $\sigma_{F,x^\star}$ and its verification key $vk_{F,x^\star}$, the encoded output $\theta_{F(x^\star)}$ as well as access to the oracles $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. The adversary eventually outputs the result y which corresponds to $\theta_{F(x^\star)}$ and an acceptance token $\tau_{\theta_{F(x^\star)}}$. The adversary wins if the challenger accepts this output and $y \neq F(x^\star)$.

Definition 3.10. *The* advantage *of a PPT adversary in the* sVIndM *game for a revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{sVIndM}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{sVIndM}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the revocable publicly verifiable outsourced computation scheme $\mathcal{RPVC}$ is secure with respect to selective vindictive managers *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sVIndM}}_{\mathcal{A},\mathcal{RPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

3.4 Construction

In this section we provide an instantiation of a RPVC scheme. Our proposed construction follows the principles of Parno et al. [118] (summarised in Section 2.7.3) which uses key-policy attribute-based encryption (KP-ABE) in a black-box manner to outsource the computation of a monotone Boolean function. We restrict our attention to Boolean functions, and in particular the complexity class NC^1 which includes all circuits of depth $\mathcal{O}(\log n)$ [12] where n corresponds to the number of gates. Thus, functions we can outsource can be built from common operations such as AND gates, OR gates, NOT gates[2], equality and comparison operators, arithmetic operators and regular expressions. Note that our scheme only evaluates Boolean functions with single bit output and therefore seems slightly limited.[3] However, it is possible to outsource the evaluation of functions with n-bit outputs by outsourcing n different functions each returning a single bit in the ith position, where $1 \leq i \leq n$.

Let us recall from Section 2.7.3 the basic underlying principles required for a PVC construction. Only instantiating a single ABE scheme leads to a PVC protocol with a one-sided error since a computational server is able to return $\perp$ in response to a computational request and therefore the verifier is not able to determine whether $F(x) = 0$ or whether the server intentionally misbehaved and refused to decrypt. To overcome this issue, we need to restrict the possible set of functions we can evaluate to be the family of Boolean functions closed under complement $\mathcal{F}$ as well as we need to initialise a second ABE scheme. In more detail, if the function F belongs to $\mathcal{F}$ then the complement function $\overline{F}(x) = F(x) \oplus 1$ also belongs to $\mathcal{F}$. The client then encrypts two randomly chosen messages m_0 and m_1 under the same attribute representation of the input and the server must decrypt each ciphertext using the keys associated with access structures encoding F or $\overline{F}$ respectively. Since exactly one of F and $\overline{F}$ will be satisfied by any given input, exactly one message (plaintext) will be successfully returned during the decryption procedure which enables the client to determine whether $F(x)$ is 1 or 0 according to the order of the well-formed response as detailed in equation (2.1).

As discussed earlier in this chapter, we aim to enhance current PVC proposals and provide within our RPVC model the possibility to evaluate *multiple* functions in a secure manner. Recall that the original scheme from Parno et al. is only initialised for a single function. Therefore, it is necessary to initialise a *new* system in case the client wishes to outsource the computation of a different function. Parno et al. may partially overcome this limitation with introducing the notion of *multi-function VC* (Definition

[2]We are able to use NOT gates by swapping to the non-monotonic ABE scheme by Ostrovsky et al. [112]

[3]If we are interested to evaluate different function families then we will require different constructions from that presented here for Boolean functions.

2.38) offering the possibility to handle multiple functions. However, the drawback of this solution is that it only works in the non-publicly verifiable setting, and the public verifiable setting is left as an open problem.

Our proposed solution moves towards a solution for multi-function PVC but is ultimately only an intermediate step towards a solution to the above problem. In contrast, we take a slightly different point of view compared to Parno et al. and require that the clients encode their input per computation they outsource. We introduce a simple encoding trick which enables servers to be certified for multiple functions and this also prevents the server from misusing the evaluation keys. Furthermore, we also apply this trick in order to restrict the client's input only being evaluated for the specified function the client wishes to evaluate. We also wish to accommodate multiple servers within our model that are certified to compute multiple functions.

In the remainder of this section, we discuss the necessary technical details we use for our RPVC construction and provide details of our encoding trick and describe how we are able to handle multiple certified servers within the system. Finally, we provide the instantiation details and state our main theorem.

3.4.1 Technical Details

We use an *indirectly revocable KP-ABE scheme* for a class of monotone Boolean functions $\mathcal{F}$ closed under complement (cf. Section 2.3.2) which comprises of the algorithms $\mathsf{ABE.Setup}$, $\mathsf{ABE.KeyGen}$, $\mathsf{ABE.KeyUpdate}$, $\mathsf{ABE.Encrypt}$ and $\mathsf{ABE.Decrypt}$. We mainly chose this notion as our building block as it enables us to implement the revocation mechanism for our model and since the client's device is computationally weak, the indirect revocation mechanism reduces the client's workload as she is not required to maintain the revocation list. We also use a signature scheme with algorithms $\mathsf{Sig.KeyGen}$, $\mathsf{Sig.Sign}$ and $\mathsf{Sig.Verify}$, and a one-way function g. Let $\mathcal{U}$ be the universe of attributes for the indirectly revocable KP-ABE scheme which we need to slightly extend compared to the original definition as stated in Section 2.3.2. In more detail, let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\mathcal{F}}$ be the universe of attributes for the indirectly revocable KP-ABE scheme. It is formed as the union of the following sub-universes, where $\mathcal{U}_{\text{attr}}$ consists of the attributes that form characteristic tuples for input data, $\mathcal{U}_{\text{ID}}$ comprises attributes representing entity identifiers, $\mathcal{U}_{\text{time}}$ comprises attributes representing time periods issued by the time source $\mathbb{T}$ and finally $\mathcal{U}_{\mathcal{F}}$ comprises attributes that represent functions in $\mathcal{F}$.

3.4.1.1 Handling Multiple Servers

The scheme of Parno et al. required a one-key IND-CPA notion of security for the underlying KP-ABE scheme since it only permitted the evaluation of a single function.

This is a more relaxed notion than considered in the vast majority of the ABE literature where the adversary is limited to learning just one decryption key. Parno et al. could use this property due to their restricted system model where the client is certified for only a single function per set of public parameters which requires the client to set up a new ABE environment per function and server.

Within our system model, we aim to accommodate multiple computational servers being also certified for multiple functions and as such the adversary is provided with a KeyGen oracle which enables to request and learn polynomial many decryption keys for different servers and different functions. The scheme needs to ensure that collusions between servers holding different decryption keys can be prevented as well as that a malicious server is not able to use a decryption key for a particular function to claim a computational result for another function and have the result accepted by the client. Collusions can be prevented by the IND-CPA security of the ABE scheme. In order to prevent the misuse of decryption keys for multiple functions we introduce our simple encoding trick in the next section.

3.4.1.2 Handling Multiple Functions

Recall that Parno et al. introduced the notion of multi-function VC in the non-publicly verifiable setting but requires a somewhat more complex notion of KP-ABE with Outsourcing as introduced by Green et al. [90]. In this thesis, we take a different approach in order to handle multiple functions on the same input data within a single PVC system for multiple servers and only require a simple encoding trick. We believe that in practical environments it is unrealistic to expect a server to compute just a single function, and we also believe that it is a reasonable cost expectation to prepare an encoded input per computation, as long as the associated cost of doing so is relatively low, especially given that the input data to different functions may well differ. Thus, whereas Parno et al. use a more non-standard and complex primitive to enable this functionality, we require only a simple encoding trick which allows servers to possess different evaluation keys for multiple functions in the publicly verifiable setting and we only require the standard and well-studied multi-key notion of security considered in the literature.

First we remark that the PVC construction of Parno et al., as summarised in Section 2.7.3, suffers a straightforward attack in case one extends the scheme by providing the adversary with a KeyGen oracle to provide access to multiple evaluation keys. In more detail, the client would encrypt as usual two randomly sampled messages both under the attribute representation of the input within different ABE schemes. The malicious server must successfully decrypt one of these messages by using its evaluation key, which comprises the ABE decryption keys of some function G and its complement

$\overline{G}$, since either G or $\overline{G}$ is satisfied by the input data. Thus, even if the malicious server does not hold an evaluation key for F, it can still successfully decrypt one message and return the result of $G(x)$ claiming it to be the correct outcome of $F(x)$ which the client accepts since the verification check (equality check using a one-way function g) was successful.

To overcome this attack and to use more standard primitives when handling multiple functions in the publicly verifiable setting, we now describe our simple encoding trick. Let us define a bijective mapping $\Lambda\colon \mathcal{F} \to \mathcal{U}_{\mathcal{F}}$ that maps functions from the function family $\mathcal{F}$ of a RPVC scheme to attributes in the sub-universe $\mathcal{U}_{\mathcal{F}}$. Our encoding trick basically adds a conjunctive clause, i.e. we add an additional AND gate, to each monotone Boolean function $F \in \mathcal{F}$. This resulting function label also needs to be present in the input attribute set such that this input can *only* be used for a particular function in order to prevent a misuse of evaluation keys since using a different evaluation key would lead to the policy not being satisfied by the input attributes. In more technical terms, we encode the monotone Boolean function F in a decryption key for the policy $F \wedge \Lambda(F)$ and similarly encode the complement function $\overline{F}$ in a decryption key for the policy $\overline{F} \wedge \Lambda(F)$. Finally, the client wishes to restrict the evaluation of her input data x exclusively for F. Thus, we need to add the label $\Lambda(F) \in \mathcal{U}_{\mathcal{F}}$ also to the attribute set A_x encoding her input data x resulting in a representation of the input data as $A_x \cup \Lambda(F)$. Note that decryption will only be successful if and only if the policy is satisfied by the input data *and* additionally the same label is present in both the policy and input data. This function label prevents the above mentioned attack since a malicious server that is certified for multiple functions cannot use an evaluation key for some function G computing on some data intended for F. More precisely, the decryption components within the evaluation key are associated with $G \wedge \Lambda(G)$ and $\overline{G} \wedge \Lambda(G)$, whereas the client initially specified her input data being associated with the function label for F, i.e. $A_x \cup \Lambda(F)$. Neither policy is satisfied as the label $\Lambda(G)$ is not present in the attribute set representing the input and therefore the malicious server cannot return any correct message that convinces the client to accept the computation. In our setting, we also require the client to perform the ProbGen procedure per computation as the function label may differ in case the client wishes her input to be evaluated for a different function. On the other hand, if the client wishes to receive an evaluation on some new input data for the same function, she can just change the input data and keep the function label.
As a result of the above, and unlike the single function notion of Parno et al., we are able to provide the adversary with oracle access in our security games.

3.4.2 Instantiation Details

Our RPVC scheme operates in the following way.

1. Setup, presented in Algorithm 1, first forms the attribute universe as well as establishes public parameters and a master secret key by calling ABE.Setup twice. We require to establish two distinct ABE schemes to overcome the one-sided error in the evaluation and also to enable the security proofs to go through. Informally, one system will be linked to the function F and one to the complement function $\overline{F}$. The public parameters and master secret keys for the respective ABE systems are distinguished with a superscript 0 or 1 respectively.

   This Setup algorithm initialises a time source $\mathbb{T}$.[4] It also initialises a two-dimensional array of registered servers L_{Reg} indexed by server identities. The purpose of this array is to store required public information about the certified servers within the system. For each server S, $L_{\text{Reg}}[S][0]$ stores the respective signature verification key while $L_{\text{Reg}}[S][1]$ stores a list of functions the server is authorised to compute. The algorithm also initialises a list of revoked servers L_{Rev} that is initially empty as no server has been revoked from the system yet.

   Finally, the public parameters pp are defined to contain both sets of ABE parameters mpk^0_{ABE} and mpk^1_{ABE}, the array L_{Reg} and the time source $\mathbb{T}$ such that each entity within the system is able to check the current time period. The master secret key mk comprises the ABE master secret msk^0_{ABE} and msk^1_{ABE} as well as the list of revoked servers L_{Rev}.

   Note that the public parameters may be implicitly updated throughout the execution of all algorithms of a RPVC scheme accommodating any changes in the system population.

Algorithm 1 $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$

$\mathcal{U} \leftarrow \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\mathcal{F}}$
$(mpk^0_{\text{ABE}}, msk^0_{\text{ABE}}) \leftarrow_{\$} \mathsf{ABE.Setup}(1^\lambda, \mathcal{U})$
$(mpk^1_{\text{ABE}}, msk^1_{\text{ABE}}) \leftarrow_{\$} \mathsf{ABE.Setup}(1^\lambda, \mathcal{U})$
for $S \in \mathcal{U}_{\text{ID}}$ **do**
    $L_{\text{Reg}}[S][0] \leftarrow \epsilon$
    $L_{\text{Reg}}[S][1] \leftarrow \{\epsilon\}$
endfor
$L_{\text{Rev}} \leftarrow \epsilon$
Initialise $\mathbb{T}$
$pp \leftarrow (mpk^0_{\text{ABE}}, mpk^1_{\text{ABE}}, L_{\text{Reg}}, \mathbb{T})$
$mk \leftarrow (msk^0_{\text{ABE}}, msk^1_{\text{ABE}}, L_{\text{Rev}})$

[4] $\mathbb{T}$ can be seen as a counter that is maintained in the public parameters or a networked clock from which the time period may be efficiently sampled as $t \leftarrow \mathbb{T}$.

2. FnInit, presented in Algorithm 2, simply outputs the public parameters and is the same for all functions. This step is not required in our particular construction, but we retain the algorithm for consistency with prior definitions as well as for generality as other instantiations may require this step.

Algorithm 2 $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$

1 : $pk_F \leftarrow pp$

3. Register, presented in Algorithm 3, creates a public-private key pair by calling the KeyGen algorithm of the digital signature scheme. The algorithm provides the server with its own secret signing key and updates $L_{\text{Reg}}[S][0]$ to store the verification key for S. This ensures that a server is not imitated and maliciously revoked.

Algorithm 3 $sk_S \xleftarrow{\$} \mathsf{Register}(S, mk, pp)$

1 : $(sk_{\text{Sig}}, vk_{\text{Sig}}) \leftarrow_{\$} \mathsf{Sig.KeyGen}(1^\lambda)$

2 : $sk_S \leftarrow sk_{\text{Sig}}$

3 : $L_{\text{Reg}}[S][0] \leftarrow L_{\text{Reg}}[S][0] \cup vk_{\text{Sig}}$

4. Certify, presented in Algorithm 4, aims to generate an evaluation key $ek_{F,S}$ for a function F enabling a server to compute this function on behalf of the client. The algorithm first removes the server from the list of revoked entities and then updates the server's array and includes the function F to the list of functions the server is authorised to compute. It then checks the current time period t from the time source $\mathbb{T}$ and calls both the ABE.KeyGen algorithm and the ABE.KeyUpdate algorithm twice. Recall that we want to prevent a server that is certified for two different functions F and G (that differ on their output) from using the key for G to recover the plaintext and claiming it as a result for F. To prevent this, we use the encoding trick from Section 3.4.1.2 and add an additional function attribute label $\Lambda(\cdot)$ to the policy that corresponds to the function. Thus, we run the two ABE algorithms once with the policy for the function and function label $F \wedge \Lambda(F)$ using ABE system parameters with superscript 0, and once with the policy for the complement function and function label $\overline{F} \wedge \Lambda(F)$ using ABE system parameters with superscript 1.

   Finally, the evaluation key is formed by the decryption keys and two update keys for the current time period.

Algorithm 4 $ek_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, mk, pp)$

1 : $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \setminus S$
2 : $L_{\mathrm{Reg}}[S][1] \leftarrow L_{\mathrm{Reg}}[S][1] \cup F$
3 : $t \leftarrow \mathbb{T}$
4 : $sk^0_{\mathrm{ABE}} \leftarrow_{\$} \mathsf{ABE.KeyGen}(S, F \wedge \Lambda(F), msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
5 : $sk^1_{\mathrm{ABE}} \leftarrow_{\$} \mathsf{ABE.KeyGen}(S, \overline{F} \wedge \Lambda(F), msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
6 : $uk^0_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
7 : $uk^1_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
8 : $ek_{F,S} \leftarrow (sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, uk^1_{L_{\mathrm{Rev}},t})$

5. ProbGen, presented in Algorithm 5, aims to create an encoded problem instance $\sigma_{F,x}$ that the server can use to evaluate the function as well as preparing a verification key that enables anyone to verify the server's computational result. The algorithm first samples the current time period t from the time source $\mathbb{T}$ in the public parameters pp. It then samples two messages m_0 and m_1 of equal length uniformly at random from the message space. The algorithm calls the ABE.Encrypt algorithm twice each generating a ciphertext.

   The first ciphertext c_0 is formed by encrypting message m_0 under the attribute representation $A_x \cup \Lambda(F)$ of the input data and the function label $\Lambda(F) \in \mathcal{U}_{\mathcal{F}}$. The function label itself is an attribute representing the function for which the input may be evaluated. Furthermore, the encryption algorithm requires the current time period t and the public parameters mpk^0_{ABE} for the first ABE system for finally forming c_0. The second ciphertext c_1 is formed similarly by encrypting m_1 under the attributes $A_x \cup \Lambda(F)$, the current time period t and the public parameters mpk^1_{ABE} for the second ABE system. Both ciphertexts together form the problem instance (or encoded input) $\sigma_{F,x}$ and will be sent to the server.

   The algorithm also prepares a public verification key $vk_{F,x}$. The key is simply generated by applying a one-way function g to the each randomly sampled message and also includes a copy of L_{Reg} from the public parameters that enables to check whether the list is modified between the current time period and the time of verification (e.g. a server is revoked).

Algorithm 5 $(\sigma_{F,x}, vk_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, pk_F, pp)$

1 : $t \leftarrow \mathbb{T}$
2 : $(m_0, m_1) \leftarrow_{\$} \mathcal{M} \times \mathcal{M}$
3 : $c_0 \leftarrow_{\$} \mathsf{ABE.Encrypt}(m_0, (A_x \cup \Lambda(F)), t, mpk^0_{\mathrm{ABE}})$
4 : $c_1 \leftarrow_{\$} \mathsf{ABE.Encrypt}(m_1, (A_x \cup \Lambda(F)), t, mpk^1_{\mathrm{ABE}})$
5 : $\sigma_{F,x} \leftarrow (c_0, c_1)$
6 : $vk_{F,x} \leftarrow (g(m_0), g(m_1), L_{\mathrm{Reg}})$

6. Compute, presented in Algorithm 6, is performed by a server S and aims to return the result of the evaluation of a function on some input data. The server parses the problem instance $\sigma_{F,x}$ and attempts to decrypt each ciphertext individually by using ABE.Decrypt. The server tries to decrypt c_0 by using the appropriate material associated to the first ABE system in the public parameters and the evaluation key, i.e. it uses sk^0_{ABE}, $uk^0_{L_{\mathrm{Rev}},t}$ and mpk^0_{ABE}, and returns d_0. Similarly, the server attempts to decrypt c_1 by using the appropriate material related to the second ABE system, i.e. it uses sk^1_{ABE}, $uk^1_{L_{\mathrm{Rev}},t}$ and mpk^1_{ABE}, and returns d_1. Note that the two plaintexts will follow the principle of a well-formed computational response as detailed in equation (2.1). It returns $(d_0, d_1) = (m_0, \perp)$ if $F(x) = 1$ or $(d_0, d_1) = (\perp, m_1)$ if $F(x) = 0$. After the decryption procedure, the server uses its personal signing key and signs the plaintexts including its own identity.

   The server finally forms the computational result $\theta_{F(x)}$ comprising the two plaintexts, the server identity and the server's signature on the output.

Algorithm 6 $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, ek_{F,S}, sk_S, pp)$

1 : Parse $\sigma_{F,x}$ as (c_0, c_1)
2 : $d_0 \leftarrow \mathsf{ABE.Decrypt}\left(c_0, sk^0_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, mpk^0_{\mathrm{ABE}}\right)$
3 : $d_1 \leftarrow \mathsf{ABE.Decrypt}\left(c_1, sk^1_{\mathrm{ABE}}, uk^1_{L_{\mathrm{Rev}},t}, mpk^1_{\mathrm{ABE}}\right)$
4 : $\gamma \leftarrow_{\$} \mathsf{Sig.Sign}(d_0, d_1, S, sk_S)$
5 : $\theta_{F(x)} \leftarrow (d_0, d_1, S, \gamma)$

7. Verify, presented in Algorithm 7, determines whether the returned computational result is valid or not. The algorithm first parses the computational result $\theta_{F(x)}$ as (d_0, d_1, S, γ) and the verification key $vk_{F,x}$ as $(g(m_0), g(m_1), L_{\mathrm{Reg}})$. It checks whether the function F is listed in $L_{\mathrm{Reg}}[S][1]$ meaning that the server that generated the computational result is authorised to compute F. If this check fails, the result is immediately rejected.

If the check was successful, the verifier continues with verifying the server's signature on the computational result using Sig.Verify to check its validity which assures that the result was indeed generated by S. Those checks ensure whether the list L_{Reg} in the public parameters and the one stored in the verification key are identical and thus have not been modified throughout the execution of the scheme. In case the signature verification fails, the result is immediately rejected. Otherwise, the verifier continues with verifying whether the returned plaintext is correct. The verifier starts with applying the one-way function g to the first entry d_0 and compares the result with the first entry of the verification key. If both values match (i.e. $g(m_0) = g(d_0)$) then the verifier accepts the result and is able to determine following equation (2.1) that the computational result y corresponds to 1. If this is the case then it also creates an acceptance token $\tau_{\theta_{F(x)}} = (\texttt{accept}, S)$ indicating that the server indeed performed the computation correctly. If both values do not match (i.e. $g(m_0) \neq g(d_0)$) then the verifier applies the one-way function g to the second entry d_1 and compares the result with the second element of the verification key.[5] In case this is a match (i.e. $g(m_1) = g(d_1)$) then the verifier accepts the result and following equation (2.1) she is able to determine that the computational result y corresponds to 0 and also outputs an acceptance token $\tau_{\theta_{F(x)}} = (\texttt{accept}, S)$. If neither comparison is successful the verifier rejects the result and reports S for revocation by forming a rejection token $\tau_{\theta_{F(x)}} = (\texttt{reject}, S)$.

Note that this algorithm can be run by any entity since the computational result and verification key are publicly available.

Algorithm 7 $(y, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{Verify}(\theta_{F(x)}, vk_{F,x}, pp)$

```
1 : Parse θ_{F(x)} as (d_0, d_1, S, γ) and vk_{F,x} as (g(m_0), g(m_1), L_Reg)
2 : if F ∈ L_Reg[S][1] then
3 :   if accept ← Sig.Verify((d_0, d_1, S), γ, L_Reg[S][0])
4 :     if g(m_0) = g(d_0) return (y ← 1, τ_{θ_{F(x)}} ← (accept, S))
5 :     elseif g(m_1) = g(d_1) return (y ← 0, τ_{θ_{F(x)}} ← (accept, S))
6 :     else (y ←⊥, τ_{θ_{F(x)}} ← (reject, S))
7 :     endif
8 :   endif
9 : endif
10 : return (y ←⊥, τ_{θ_{F(x)}} ← (reject, S))
```

8. Revoke, presented in Algorithm 8, aims to revoke misbehaving servers by re-

[5] Note that $g(\perp) = \perp$.

distributing fresh evaluation keys to all non-revoked servers. If the algorithm receives as input a rejection token $\tau_{\theta_{F(x)}} = (\texttt{reject}, S)$ for a server S, the KDC first removes all functions from the list $L_{\text{Reg}}[S][1]$ such that the server is no longer authorised to perform any computations and additionally adds the server to the list of revoked entities L_{Rev}. The algorithm then refreshes the time source $\mathbb{T}$ and samples the new time period.[6] Next the ABE.KeyUpdate algorithm is run twice, i.e. it is run once for each ABE system. The algorithms generate new update key material for the current time period with respect to the revocation list L_{Rev}. Finally, for all non-revoked servers within the system, the KDC refreshes and redistributes the updated evaluation keys.

If the algorithm receives a token not specifying any server to be revoked it returns $\perp$ indicating that the algorithm did not update any keys.

Algorithm 8 $um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(x)}}, mk, pp)$

1 : **if** $\tau_{\theta_{F(x)}} = (\texttt{reject}, S)$ **then**
2 : $L_{\text{Reg}}[S][1] \leftarrow \{\epsilon\}$
3 : $L_{\text{Rev}} \leftarrow L_{\text{Rev}} \cup S$
4 : Refresh $\mathbb{T}$
5 : $t \leftarrow \mathbb{T}$
6 : $uk^0_{L_{\text{Rev}},t} \leftarrow_{\$} \mathsf{ABE.KeyUpdate}(L_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$
7 : $uk^1_{L_{\text{Rev}},t} \leftarrow_{\$} \mathsf{ABE.KeyUpdate}(L_{\text{Rev}}, t, msk^1_{\text{ABE}}, mpk^1_{\text{ABE}})$
8 : **for** $S' \in \mathcal{U}_{\text{ID}}$ **do**
9 : Parse $ek_{F,S'}$ as $(sk^0_{\text{ABE}}, sk^1_{\text{ABE}}, uk^0_{L_{\text{Rev}},t-1}, uk^1_{L_{\text{Rev}},t-1})$
10 : $ek_{F,S'} \leftarrow (sk^0_{\text{ABE}}, sk^1_{\text{ABE}}, uk^0_{L_{\text{Rev}},t}, uk^1_{L_{\text{Rev}},t})$
11 : **endfor**
12 : **return** $um \leftarrow \{ek_{F,S'}\}_{S' \in \mathcal{U}_{\text{ID}}}$
13 : **else**
14 : **return** $\perp$

Theorem 3.11. *Given an indirectly revocable KP-ABE scheme secure in the sense of indistinguishability against selective-target with semi-static query attacks (*IND-sHRSS*) for a class of monotone Boolean functions $\mathcal{F}$ closed under complement, an* EUF-CMA *secure signature scheme and a one-way function g. Let $\mathcal{RPVC}$ be the revocable publicly verifiable outsourced computation scheme as defined in Algorithms 1–8. Then $\mathcal{RPVC}$ is secure in the sense of selective public verifiability (Figure 3.8), selective semi-static revocation (Figure 3.9), vindictive servers (Figure 3.6) and selective vindictive managers (Figure 3.10).*

[6] If the time source is a counter then refreshing leads to incrementing the time period.

Informally, the proofs of selective public verifiability and selective vindictive managers rely on the IND-sHRSS security of the underlying indirectly revocable KP-ABE scheme and the one-wayness of the function g while the proof of revocation only relies on the IND-sHRSS security. Security against vindictive servers relies in the EUF-CMA security of the digital signature scheme such that a vindictive server cannot return an incorrect result with a forged signature claiming to be formed by an honest server. Note that the chosen message attack is required since the vindictive server could act like a client and submit computation requests to get a valid signature.

In the description of the Setup algorithm (cf. Algorithm 1) we already discussed that we require to initialise two distinct ABE systems. This is mainly in order to overcome the possible one-sided error in the protocol in case only a single system is initialised as well as to enable the security proofs to go through. Recall that we link one system with the function F and the other system with the complement function $\overline{F}$. To avoid the adversary to trivially win in the security proofs, we need to restrict the type of oracle queries the adversary can form when querying a KeyGen oracle. In particular, we require that the adversary cannot query a KeyGen oracle for a function that is satisfied by the challenge input (attribute set). Thus, we require to initialise two ABE systems where one is maintained by the challenger and hence requires oracle access and one is handled by the adversary itself. We need to choose these systems carefully such that the system maintained by the challenger is associated to the unsatisfied function F or $\overline{F}$ and hence appropriate keys can be obtained by querying the KeyGen oracle.

3.5 Proofs of Security

In this section we prove Theorem 3.11 by providing the full proofs of security for selective public verifiability, selective semi-static revocation, vindictive servers and selective vindictive managers.

3.5.1 Selective Public Verifiability

Lemma 3.12. *The $\mathcal{RPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective public verifiability (Figure 3.8) under the same assumptions as in Theorem 3.11.*

Proof. Suppose $\mathcal{A}_{\mathrm{RPVC}}$ is an adversary with non-negligible advantage against the selective public verifiability game (Figure 3.8) when instantiated by Algorithms 1–8. We begin by defining the following three games:

- **Game 0.** This is the selective public verifiability game as defined in Figure 3.8.
- **Game 1.** This is the same as **Game 0** with the modification that in ProbGen, we no longer return an encryption of m_0 and m_1. Instead, we choose another

equal length random message $m' \neq m_0, m_1$ and, if $F(x^\star) = 1$, we replace c_1 by the encryption of m', and otherwise we replace c_0. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.

- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function inversion game.

The proof partially follows in the fashion of Parno et al. [118] and we aim to show that from the point of view of the adversary **Game 2** is indistinguishable from **Game 0** except with negligible probability. Thus, this means that an adversary against the selective public verifiability game can be run against **Game 2**. We then finally show that if an adversary has a non-negligible advantage against **Game 2** then the adversary can invert a one-way function.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{\text{RPVC}}$ can distinguish the two games with non-negligible advantage δ. We then show that it is possible to construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{RPVC}}$ as a sub-routine to break the IND-sHRSS security of the indirectly revocable KP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-sHRSS game (Figure 2.3) with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ in turn acts as a challenger for $\mathcal{A}_{\text{RPVC}}$. Given the above parameters the entities interact in the following way.

1. $\mathcal{A}_{\text{RPVC}}$ declares its choice of challenge function F and challenge input $x^\star$.
2. $\mathcal{A}_{\text{ABE}}$ first computes $F(x^\star) = r$ and transforms the challenge input from $\mathcal{A}_{\text{RPVC}}$ into its own challenge input for the IND-sHRSS game. It sets $\overline{x^\star} = A_{x^\star} \cup \Lambda(F)$ where $\Lambda(F) \in \mathcal{U}_{\mathcal{F}}$ corresponds to the attribute representing the challenge function F. Finally, $\mathcal{A}_{\text{ABE}}$ also sets its challenge for the time period $t^\star = 1$ for the IND-sHRSS game and sends both challenge parameters $\overline{x^\star}$ and $t^\star$ to the challenger $\mathcal{C}$.
3. $\mathcal{C}$ runs the $\mathsf{ABE.Setup}$ algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.
4. $\mathcal{A}_{\text{ABE}}$ initialises its target revocation list $\overline{R}$ which is initially empty and sends it to $\mathcal{C}$. Next, $\mathcal{A}_{\text{ABE}}$ runs $\mathsf{RPVC.Setup}$ in such a way that the ABE system maintained by the challenger $\mathcal{C}$ is used as the ABE system with parameters $(mpk^r_{\text{ABE}}, msk^r_{\text{ABE}})$. The main reason for this is to avoid a trivial win for $\mathcal{A}_{\text{ABE}}$ in the IND-sHRSS game. In more detail, since $\mathcal{A}_{\text{ABE}}$ does not receive the secret parameters msk_{ABE} generated by $\mathcal{C}$, the adversary $\mathcal{A}_{\text{ABE}}$ is required to issue

queries to oracles handled by $\mathcal{C}$ in order to generate valid parameters for $\mathcal{A}_{\text{RPVC}}$. However, we require to restrict $\mathcal{A}_{\text{ABE}}$ from querying the KeyGen oracle for a function that evaluates to 1 on the challenge input $\overline{x^\star}$. This prevents the decryption of the challenge ciphertext and thus avoids a trivial win by the adversary.

Recall that in the RPVC.Certify algorithm (cf. Algorithm 4), one decryption key for a function F is generated using the parameters $(mpk^0_{\text{ABE}}, msk^0_{\text{ABE}})$ associated with the first initialised ABE system, while another decryption key is generated encoding the complement function $\overline{F}$ using the parameters $(mpk^1_{\text{ABE}}, msk^1_{\text{ABE}})$ related to the second ABE scheme.

Depending on the outcome of the initial computation $F(x^\star) = r$ we need to ensure that the challenger $\mathcal{C}$ and adversary $\mathcal{A}_{\text{ABE}}$ are in possession of the *correct* parameters. In more detail, if $F(x^\star) = 1$, then the policy $F \wedge \Lambda(F)$ will be satisfied by $\mathcal{A}_{\text{ABE}}$'s (transformed) challenge input $\overline{x^\star} = A_{x^\star} \cup \Lambda(F)$. Therefore, $\mathcal{A}_{\text{ABE}}$ cannot make any KeyGen oracle queries to $\mathcal{C}$ for the policy $F \wedge \Lambda(F)$. In this case, it is required that we ensure that the ABE system parameters maintained by the challenger corresponds to $(mpk^1_{\text{ABE}}, msk^1_{\text{ABE}})$ and the ones maintained by $\mathcal{A}_{\text{ABE}}$ correspond to $(mpk^0_{\text{ABE}}, msk^0_{\text{ABE}})$. Thus, $\mathcal{A}_{\text{ABE}}$ is able to generate a key for $F \wedge \Lambda(F)$ itself and queries to challenger's KeyGen oracle are not trivially satisfied by the input.

On the other hand, if $F(x^\star) = 0$, then the complement policy $\overline{F} \wedge \Lambda(F)$ may not be queried to the KeyGen oracle as this would lead to a trivial win. Following the RPVC.Certify algorithm (cf. Algorithm 4), then the decryption key associated with the complement function uses the parameters $(mpk^1_{\text{ABE}}, msk^1_{\text{ABE}})$ and as such we need to ensure that $\mathcal{A}_{\text{ABE}}$ maintains those parameters and the challenger maintains the parameters $(mpk^0_{\text{ABE}}, msk^0_{\text{ABE}})$.

Thus, we need to ensure that the challenger maintains the parameters $(mpk^r_{\text{ABE}}, msk^r_{\text{ABE}})$ where $r = F(x^\star)$.

Now $\mathcal{A}_{\text{ABE}}$ simulates running the RPVC.Setup algorithm (cf. Algorithm 1) with the exception that when ABE.Setup is called on line 2 and 3 it sets mpk^r_{ABE} to be the parameter provided by the challenger, and implicitly sets msk^r_{ABE} to be that held by the challenger.

5. $\mathcal{A}_{\text{ABE}}$ runs RPVC.FnInit as detailed in Algorithm 2.

6. $\mathcal{A}_{\text{ABE}}$ must generate a challenge problem instance for $\mathcal{A}_{\text{RPVC}}$ as the output of RPVC.ProbGen. To do so, $\mathcal{A}_{\text{ABE}}$ samples three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. $\mathcal{A}_{\text{ABE}}$ provides m_0 and m_1 as its choice of challenge for the IND-sHRSS game to $\mathcal{C}$, and receives back the encryption, $ct^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$, where $b^\star$ was chosen by the challenger), under attributes $\overline{x^\star}$ and time $t^\star$. More formally

this is $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{b^\star}, \overline{x^\star}, t^\star, mpk^r_{\mathrm{ABE}})$. It needs to assign $ct^\star$ to be one of the ciphertexts c or c' that form the challenge problem instance (encoded input) $\sigma_{F,x^\star}$ using the correct ABE system parameters. $\mathcal{A}_{\mathrm{ABE}}$ chooses a random bit $s \xleftarrow{\$} \{0,1\}$ which intuitively corresponds to its guess for the challenger's choice of $b^\star$. Therefore,

- If $r = 0$, then $\mathcal{A}_{\mathrm{ABE}}$ sets c to be $ct^\star$ and randomly generates the remaining ciphertext as

$$c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \overline{x^\star}, t^\star, mpk^1_{\mathrm{ABE}}).$$

  It sets $vk = g(m_s)$ and $vk' = g(m')$.

- If $r = 1$, then $\mathcal{A}_{\mathrm{ABE}}$ randomly generates the ciphertext as

$$c \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \overline{x^\star}, t^\star, mpk^0_{\mathrm{ABE}}),$$

  and sets c' to be $ct^\star$. It sets $vk = g(m')$ and $vk' = g(m_s)$.

Finally, $\mathcal{A}_{\mathrm{ABE}}$ sets $\sigma_{F,x^\star} = (c, c')$ and $vk_{F,x^\star} = (vk, vk', L_{\mathrm{Reg}})$.

7. $\mathcal{A}_{\mathrm{RPVC}}$ receives all outputs from the above $\mathsf{RPVC.ProbGen}$ algorithm, and then is provided with oracle access to which $\mathcal{A}_{\mathrm{ABE}}$ responds in the following way:

   - Queries to $\mathsf{RPVCFnInit}$ and $\mathsf{RPVC.Register}$ are performed as specified in Algorithms 2 and 3.
   - Queries of the form $\mathsf{RPVC.Certify}(S, F', mk, pp)$ are handled by $\mathcal{A}_{\mathrm{ABE}}$ running Algorithm 4 with the exception that the $\mathsf{ABE.KeyGen}$ and $\mathsf{ABE.KeyUpdate}$ algorithms for the ABE system with the parameters maintained by the challenger are replaced by queries to the respective oracles provided by $\mathcal{C}$. $\mathcal{A}_{\mathrm{ABE}}$ queries the $\mathsf{ABE.KeyGen}$ oracle in order to obtain a decryption key sk^r_{ABE}. Here we need to distinguish two cases.
     If $r = 0$, the query is formed over the parameters of the first ABE system and is of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, F' \wedge \Lambda(F'), msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$. If $r = 1$, then the query is formed over the parameters of the second ABE system and is of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, \overline{F}' \wedge \Lambda(F'), msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$. The challenger returns for both cases a valid decryption key unless the challenge input $\overline{x^\star}$ satisfies the queried policy. Note that in case the queried function F' does not correspond to the challenge function F then due to the bijective mapping Λ the attribute function labels do not coincide, i.e. $\Lambda(F') \neq \Lambda(F)$. Thus, neither of the possible queries to the KeyGen oracle will be satisfied. On the other hand, since we chose the ABE system maintained by the challenger to be unsatisfied for F, any query for F will be rejected. Hence, if $F' \neq F$,

then the first clause of the "if" statement in the KeyGen oracle of the IND-sHRSS game (Figure 2.3) will never be correct and therefore the challenger will always return a valid decryption key in response to a query.

In order to generate a valid update key $uk^r_{L_{\text{Rev}},t}$, $\mathcal{A}_{\text{ABE}}$ makes a query to the ABE.KeyUpdate oracle which are of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$. The challenger returns a valid update key if the current queried time period t does not coincide with the challenge time period $t^\star$ which $\mathcal{A}_{\text{ABE}}$ chose to be 1 *and* if the queried revocation list L_{Rev} contains the challenge target revocation list $\overline{R}$ which $\mathcal{A}_{\text{ABE}}$ chose initially to be empty. Since $\overline{R} = \epsilon$ is a subset of any revocation list L_{Rev}, the second clause of the "if" statement in the KeyUpdate oracle of the IND-sHRSS game (Figure 2.3) will not be satisfied and therefore the challenger will always return a valid update key.

- Queries of the form RPVC.Revoke$(\tau_{\theta_{F(x)}}, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ running Algorithm 8 with the exception of generating an update key $uk^r_{L_{\text{Rev}},t}$ on line 6 and 7 which will be replaced with respective queries to the KeyUpdate oracle provided by $\mathcal{C}$. $\mathcal{A}_{\text{ABE}}$ queries the oracle for $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$ and the challenger returns a valid key if the queried time period t does not correspond to the challenge time period, i.e. $t \neq 1$, and if the queried revocation list does not contain the challenge target revocation list $\overline{R}$. Since $\mathcal{A}_{\text{ABE}}$ chose $\overline{R}$ to be empty, the second clause of the "if" statement in the KeyUpdate oracle will not be satisfied and the challenger returns a valid update key.

8. Eventually $\mathcal{A}_{\text{RPVC}}$ finishes its query phase and outputs a guess $\theta^\star$. Let Y be the non-$\perp$ plaintext contained in $\theta^\star$. If $g(Y) = g(m_s)$, $\mathcal{A}_{\text{ABE}}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{\text{ABE}}$ guesses $b' = 1 - s$.

Note that if $s = b^\star$ (the challenge bit chosen by $\mathcal{C}$ in the IND-sHRSS game in step 6), then the distribution of the above coincides with **Game 0** since the verification key comprises $g(m')$ and $g(m_s)$ where m' and m_s are the two plaintexts corresponding to the ciphertexts of the encoded input for which $\mathcal{A}_{\text{RPVC}}$ recovers exactly one. Otherwise, if $s = 1 - b^\star$ then the distribution coincides with **Game 1** since the verification key comprises the one-way function g applied to a legitimate message m' and a random message m_{1-s} that is unrelated to both ciphertexts.

Now, we consider the advantage of this constructed adversary $\mathcal{A}_{\text{ABE}}$ playing the IND-sHRSS game for the revocable KP-ABE scheme. Recall that by assumption, $\mathcal{A}_{\text{RPVC}}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1**, that

is

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right| \geqslant \delta$$

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$ denotes the output of running $\mathcal{A}_{\mathrm{RPVC}}$ in **Game i**.

Now we derive the probability of $\mathcal{A}_{\mathrm{ABE}}$ guessing $b^\star$ correctly and it follows:

$$\begin{aligned}
\Pr[b' = b^\star] &= \Pr[s = b^\star]\Pr[b' = b^\star | s = b^\star] + \Pr[s \neq b^\star]\Pr[b' = b^\star | s \neq b^\star] \\
&= \frac{1}{2}\Pr[g(Y) = g(m_s) | s = b^\star] + \frac{1}{2}\Pr[g(Y) \neq g(m_s) | s \neq b^\star] \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + \frac{1}{2}\left(1 - \Pr[g(Y) = g(m_s) | s \neq b^\star]\right) \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] \\
&\quad + \frac{1}{2}\left(1 - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right. \\
&\quad \left. - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\mathrm{ABE}}} &\geqslant \left|\Pr[b' = b^\star] - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&= \frac{\delta}{2}
\end{aligned}$$

Since δ is assumed to be non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\mathrm{RPVC}}$ has advantage δ at distinguishing these games then $\mathcal{A}_{\mathrm{ABE}}$ can win the IND-sHRSS game with non-negligible probability. Thus since we assumed the ABE scheme to be IND-sHRSS secure, we conclude that $\mathcal{A}_{\mathrm{RPVC}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is to simply set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game (Figure 2.8). We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system except the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{\mathrm{RPVC}}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{\mathrm{ABE}}$).

Final Proof. We now show that using $\mathcal{A}_{\text{RPVC}}$ in **Game 2**, $\mathcal{A}_{\text{ABE}}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ $\mathcal{A}_{\text{ABE}}$ can recover w. Specifically, during $\mathsf{ProbGen}$, $\mathcal{A}_{\text{ABE}}$ chooses the messages as follows:

- if $F(x^\star) = 1$, we implicitly set m_1 to be w and the corresponding verification key component to be $z = g(w)$. We randomly choose m_0 from the message space and compute the remainder of the verification key as usual.
- if $F(x^\star) = 0$, we implicitly set m_0 to be w and set the verification key component to $z = g(w)$. m_1 is chosen randomly from the message space and the remainder of the verification key is computed as usual.

Now, since $\mathcal{A}_{\text{RPVC}}$ is assumed to be successful, it will output a forgery comprising the plaintext that was encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{\text{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\text{RPVC}}$ has against the selective public verifiability game.

We conclude that if the ABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $\mathcal{RPVC}$ as defined by Algorithms 1–8 is secure in the sense of selective public verifiability. □

3.5.2 Selective, Semi-static Revocation

Lemma 3.13. *The $\mathcal{RPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective, semi-static revocation (Figure 3.9) under the same assumptions as in Theorem 3.11.*

Proof. In this proof, we aim to perform a reduction from the the selective, semi-static revocation game (Figure 3.9) to the IND-sHRSS security of the underlying revocable KP-ABE scheme (Figure 2.3). We wish the prove this reduction by achieving a contradiction and therefore we assume that $\mathcal{A}_{\text{RPVC}}$ is an adversary with non-negligible probability against the selective, semi-static revocation game when instantiated by Algorithms 1–8. We show that we can construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{RPVC}}$ as a sub-routine to break the IND-sHRSS security of the indirectly revocable KP-ABE scheme. Let $\mathcal{C}$ be a challenger playing the IND-sHRSS game with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ acts as a challenger for $\mathcal{A}_{\text{RPVC}}$. Given the security parameter λ, the function family $\mathcal{F}$ and the number of queries q_t the adversary $\mathcal{A}_{\text{RPVC}}$ makes to the Revoke oracle, the entities interact in the following way.

1. $\mathcal{A}_{\text{RPVC}}$ declares its choice of challenge function F and challenge input $x^\star$.
2. $\mathcal{A}_{\text{ABE}}$ initialises an (empty) list Q_{Rev} of currently revoked entities and sets the current time period to $t = 1$. Next, $\mathcal{A}_{\text{ABE}}$ transforms the challenge input from

$\mathcal{A}_{\text{RPVC}}$ into its own challenge input for the IND-sHRSS game. It sets $A^\star = A_{x^\star} \cup \Lambda(F)$ where $A_{x^\star}$ is the attribute encoding of the challenge input and $\Lambda(F) \in \mathcal{U}_{\mathcal{F}}$ corresponds to the attribute representing the function F. Finally, $\mathcal{A}_{\text{ABE}}$ also sets its challenge for the time period $t^\star = q_t$ for the IND-sHRSS game and sends both challenge parameters to the challenger $\mathcal{C}$.

3. $\mathcal{C}$ runs ABE.Setup to create $(mpk_{\text{ABE}}, msk_{\text{ABE}})$ and sends the public parameter mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.

4. $\mathcal{A}_{\text{ABE}}$ simulates running the RPVC.Setup algorithm (cf. Algorithm 1) as specified with the exception in line 2 where the first ABE system is initialised. There, $\mathcal{A}_{\text{ABE}}$ sets mpk^0_{ABE} to be mpk_{ABE} which was generated by the challenger, and msk^0_{ABE} is implicitly set to be the secret parameters held by $\mathcal{C}$. Since $\mathcal{A}_{\text{ABE}}$ does not possess msk_{ABE}, it will make use of oracle queries to $\mathcal{C}$ whenever msk^0_{ABE} is required.

5. $\mathcal{A}_{\text{ABE}}$ runs RPVC.FnInit as detailed in Algorithm 2.

6. $\mathcal{A}_{\text{RPVC}}$ receives all public parameters pk_F and pp and chooses a challenge target revocation list $\overline{R}$ which $\mathcal{A}_{\text{ABE}}$ receives and forwards to its challenger $\mathcal{C}$.

7. $\mathcal{A}_{\text{RPVC}}$ may now perform oracle queries which $\mathcal{A}_{\text{ABE}}$ handles as follows:

   - Queries to RPVC.FnInit and RPVC.Register are run as written in Algorithms 2 and 3.
   - Queries of the form RPVC.Certify(S, F', mk, pp) are handled by $\mathcal{A}_{\text{ABE}}$ by running the Certify oracle as specified in Figure 3.9. If the queried function F' corresponds to the challenge function F *and* if the queried identity S is not listed on the challenge target revocation list, i.e. $S \notin \overline{R}$, then $\mathcal{A}_{\text{ABE}}$ returns $\perp$ to $\mathcal{A}_{\text{RPVC}}$. Note that otherwise, the adversary would be issued with an evaluation key that will not be revoked at the time of the challenge and therefore would trivially win. $\mathcal{A}_{\text{ABE}}$ also returns $\perp$ if the current time period t is equal to the challenge time period q_t *and* if there is a server (other than S) that is not currently revoked but should be revoked in accordance with $\mathcal{A}_{\text{RPVC}}$'s challenge target revocation list $\overline{R}$. Otherwise, i.e. if this check failed and the oracle has not returned $\perp$, it removes S form the list of currently revoked entities Q_{Rev} and simulates running Certify. This is simulated by running Algorithm 4 as detailed above with the exception of line 4 and 6 where the query needs to be passed to $\mathcal{C}$ since it only possesses the valid parameters.

     In more detail, in order to simulate line 4, $\mathcal{A}_{\text{ABE}}$ queries $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S, F' \wedge \Lambda(F'), msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. $\mathcal{C}$ returns the decryption key *unless* the policy $F' \wedge \Lambda(F')$ is satisfied by $A^\star$ *and* $S \notin \overline{R}$. Observe that the policy will never

be satisfied unless $F' = F$ (since Λ is a bijective mapping and $\Lambda(F')$ needs to be present such that the policy is satisfied). Hence, $\mathcal{C}$ will always return a valid key if $F' \neq F$.

On the other hand, if the queried function $F' = F$, then by the checks performed by $\mathcal{A}_{\text{ABE}}$ at the beginning of the Certify oracle (Figure 3.9), S is included on $\overline{R}$ (else $\perp$ would have been returned prior to this point). Therefore, even if the challenge function is queried, $\mathcal{C}$ will return a key. In particular, note that $\mathcal{C}$ never returns $\perp$ in a manner inconsistent with that expected by $\mathcal{A}_{\text{RPVC}}$ in accordance with the Certify oracle.

In order to simulate line 6, $\mathcal{A}_{\text{ABE}}$ makes a query to $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* the current time period t is the challenge time period q_t *and* the queried revocation list does not contain the challenge target revocation list $\overline{R}$. However, if this was the case then $\mathcal{A}_{\text{ABE}}$ would already have returned $\perp$ by the second clause of the "if" statement in the Certify oracle. Therefore, $\mathcal{C}$ shall always return an update key which $\mathcal{A}_{\text{ABE}}$ can use in the execution of Certify.

- Queries of the form $\mathsf{RPVC.Revoke}(\tau_{\theta_{F(x)}}, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the Revoke oracle as specified in Figure 3.9. Whenever a call is made to the Revoke oracle, $\mathcal{A}_{\text{ABE}}$ first increments the time period t. If the token does not identify a server to revoke, it outputs $\perp$ (as would the Revoke algorithm). Following line 3 of the oracle specification, in case the current time period corresponds to q_t (i.e. the maximal number to Revoke queries), then $\mathcal{A}_{\text{ABE}}$ returns $\perp$ if Q_{Rev} does not contain all servers listed on the challenge target revocation list $\overline{R}$. Otherwise, S is added to Q_{Rev}. $\mathcal{A}_{\text{ABE}}$ now simulates running the RPVC.Revoke algorithm by running Algorithm 8 as specified with the exception of line 6. To simulate this line, $\mathcal{A}_{\text{ABE}}$ needs to make a query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* $t = q_t$ *and* the queried revocation list does not contain the challenge target revocation list $\overline{R}$. However, if this would be the case, then $\mathcal{A}_{\text{ABE}}$ would have returned $\perp$ above, and so a valid update key is returned which $\mathcal{A}_{\text{ABE}}$ can forward to $\mathcal{A}_{\text{RPVC}}$.

8. Eventually (after q_t Revoke queries), $\mathcal{A}_{\text{RPVC}}$ finishes its query phase. $\mathcal{A}_{\text{ABE}}$ checks if $\mathcal{A}_{\text{RPVC}}$ has made suitable Revoke queries. If there exists an entity in $\overline{R}$ that is not currently revoked (listed in Q_{Rev}), it returns 0 and aborts immediately.

9. $\mathcal{A}_{\text{ABE}}$ must now generate a challenge for $\mathcal{A}_{\text{RPVC}}$. $\mathcal{A}_{\text{ABE}}$ chooses three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. It then sends m_0 and m_1 to $\mathcal{C}$ as its choice of challenge for the IND-sHRSS game. $\mathcal{C}$ chooses a random bit $b^\star \xleftarrow{\$} \{0,1\}$ and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{b^\star}, A^\star, q_t, mpk^0_{\text{ABE}})$. $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$, generates $c' \xleftarrow{\$}$

$\mathsf{ABE.Encrypt}(m', A^\star, q_t, mpk^1_{\mathrm{ABE}})$ and forms the challenge problem instance $\sigma_{F,x^\star} = (c, c')$. $\mathcal{A}_{\mathrm{ABE}}$ selects a bit $s \xleftarrow{\$} \{0,1\}$ and forms the verification key as $vk_{F,x^\star} = (g(m_s), g(m'), L_{\mathrm{Reg}})$. Note that s intuitively corresponds to $\mathcal{A}_{\mathrm{ABE}}$'s guess for $b^\star$.

10. $\mathcal{A}_{\mathrm{RPVC}}$ receives the resulting parameters from $\mathsf{ProbGen}$ and is again provided with oracle access. These queries are handled in the same way as before, and $\mathcal{A}_{\mathrm{RPVC}}$ eventually outputs its guess $\theta^\star$.

11. Let Y be the non-$\perp$ plaintext returned in $\theta^\star$. If $g(Y) = g(m_s)$, $\mathcal{A}_{\mathrm{ABE}}$ guesses $b' = s$. Else, $\mathcal{A}_{\mathrm{ABE}}$ guesses $b' = 1 - s$.

    If $g(Y) = g(m')$, $\mathcal{A}_{\mathrm{ABE}}$ makes a random guess $b' = \tilde{b} \xleftarrow{\$} \{0,1\}$ since $\mathcal{A}_{\mathrm{RPVC}}$ did not forge a result for either m_0 or m_1 and therefore is of no use for $\mathcal{A}_{\mathrm{ABE}}$ in order to break the IND-sHRSS game.

Now we consider the advantage of $\mathcal{A}_{\mathrm{ABE}}$ playing the IND-sHRSS game. By assumption, $\mathcal{A}_{\mathrm{RPVC}}$ has a non-negligible advantage δ against the selective, semi-static revocation game. It follows

$$
\begin{aligned}
\Pr[b' = b^\star] &= \Pr[b' = b^\star | s = b^\star] \Pr[s = b^\star] + \Pr[b' = b^\star | 1 - s = b^\star] \Pr[1 - s = b^\star] \\
&\quad + \Pr[b' = b^\star | \tilde{b} = b^\star] \Pr[\tilde{b} = b^\star] \\
&= \Pr[g(Y) = g(m_s) | s = b^\star] \Pr[s = b^\star] \\
&\quad + \Pr[g(Y) \neq g(m_s) | 1 - s = b^\star] \Pr[1 - s = b^\star] \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \Pr[\tilde{b} = b^\star] \\
&= \frac{1}{2} \Pr[g(Y) = g(m_s) | s = b^\star] + \frac{1}{2} \Pr[g(Y) \neq g(m_s) | 1 - s = b^\star] \\
&\quad + \frac{1}{2} \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \\
&= \frac{1}{2} \Big(\Pr[g(Y) = g(m_s) | s = b^\star] + (1 - \Pr[g(Y) = g(m_s) | 1 - s = b^\star]) \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \Big) \\
&= \frac{1}{2} \Big(\Pr[g(Y) = g(m_s) | s = b^\star] - \Pr[g(Y) = g(m_s) | 1 - s = b^\star] \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] + 1 \Big) \\
&= \frac{1}{2}(\delta + 1).
\end{aligned}
$$

Hence,

$$
\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\mathrm{ABE}}} &\geqslant \left| \Pr[b' = b^\star] - \frac{1}{2} \right| \\
&\geqslant \left| \frac{1}{2}(\delta + 1) - \frac{1}{2} \right| \\
&= \frac{\delta}{2}.
\end{aligned}
$$

Since δ is non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\text{RPVC}}$ has advantage δ at breaking the selective, semi-static revocation game then $\mathcal{A}_{\text{ABE}}$ can win the IND-sHRSS game with non-negligible probability. However, since the ABE scheme was assumed IND-sHRSS secure, such an adversary $\mathcal{A}_{\text{RPVC}}$ cannot exist. Therefore, we conclude that that if the ABE scheme is IND-sHRSS secure then $\mathcal{RPVC}$ as instantiated by Algorithms 1–8 is secure in the sense of selective, semi-static revocation. □

3.5.3 Vindictive Servers

Lemma 3.14. *The $\mathcal{RPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of vindictive server (Figure 3.6) under the same assumptions as in Theorem 3.11.*

Proof. For a contradiction let us assume that $\mathcal{A}_{\text{RPVC}}$ is an adversary with non-negligible advantage against the vindictive server game (Figure 3.6) when instantiated by Algorithms 1–8. We show that an adversary $\mathcal{A}_{\text{Sig}}$ with non-negligible advantage δ in the EUF-CMA signature game (Figure 2.7) can be constructed using $\mathcal{A}_{\text{RPVC}}$ as a subroutine. $\mathcal{A}_{\text{Sig}}$ interacts with the challenger $\mathcal{C}$ in the EUF-CMA security game and itself acts as the challenger for $\mathcal{A}_{\text{RPVC}}$ in the security game for vindictive servers for a function F as follows.
The basic idea is that $\mathcal{A}_{\text{Sig}}$ can create a RPVC instance and play the vindictive server game with $\mathcal{A}_{\text{RPVC}}$ by executing Algorithms 1–8 itself. $\mathcal{A}_{\text{Sig}}$ will guess a server identity that it thinks the adversary $\mathcal{A}_{\text{RPVC}}$ will select to vindictively revoke. The signature signing key that would be generated during the Register algorithm for this server will be implicitly set to be the signing key in the EUF-CMA game and any Compute oracle queries for this identity will be forwarded to the challenger to compute. Then, assuming that $\mathcal{A}_{\text{Sig}}$ guessed the correct server identity, $\mathcal{A}_{\text{RPVC}}$ will output a forged signature that $\mathcal{A}_{\text{Sig}}$ may output as its guess in the EUF-CMA game. Given the security parameter and the the function family $\mathcal{F}$, the entities interact in the following way.

1. The challenger $\mathcal{C}$ (in the EUF-CMA game) initialises an initially empty list Q of messages queried to the Sig.Sign oracle. It runs $\mathsf{Sig.KeyGen}(1^\lambda)$ to generate a challenge signing key $\overline{SK}$ and verification key $\overline{VK}$. $\mathcal{C}$ sends $\overline{VK}$ to $\mathcal{A}_{\text{Sig}}$.

2. $\mathcal{A}_{\text{Sig}}$ starts with initialising two empty lists. The fist list Q_{Reg} records all registered entities within the system as well as a list Q_{Server} which records all entities for which the adversary has learnt a signing key. $\mathcal{A}_{\text{Sig}}$ also initialises the target server identity $\overline{S}$ to be $\perp$. Furthermore, $\mathcal{A}_{\text{Sig}}$ chooses a server identity from $\mathcal{U}_{\text{ID}}$ denoted as $\widetilde{S}$. This identity is $\mathcal{A}_{\text{Sig}}$'s guess of the (final) target server identity $\overline{S}$ that $\mathcal{A}_{\text{RPVC}}$ will choose at some later point in the game. Note, if this guess will be correct, then any signing operations related to $\widetilde{S}$ can be performed using the oracle provided by $\mathcal{C}$ and hence when $\mathcal{A}_{\text{RPVC}}$ attacks this server identity, its output can be used to break the EUF-CMA game.

3. $\mathcal{A}_{\text{Sig}}$ runs RPVC.Setup and provides the resulting public parameters to $\mathcal{A}_{\text{RPVC}}$. $\mathcal{A}_{\text{RPVC}}$ is also provided with oracle access which $\mathcal{A}_{\text{Sig}}$ can respond in the following way:

   - All queries to the oracles FnInit, Certify, Revoke and the additional oracle Register2 are handled as specified in the respective algorithm;
   - Queries made for servers that do not correspond to $\mathcal{A}_{\text{Sig}}$'s guess, i.e. $S \neq \widetilde{S}$, to the Register oracle are handled by $\mathcal{A}_{\text{Sig}}$ running the Register oracle as specified in Figure 3.6. A query to the oracle returns $\perp$ if the query is for the target server identity $\overline{S}$. If, on the other hand, queries are made for servers that match to $\mathcal{A}_{\text{Sig}}$'s guess, i.e. $S = \widetilde{S}$, then $\mathcal{C}$ aborts the game to avoid a trivial win for $\mathcal{A}_{\text{Sig}}$. This is the case because $\mathcal{A}_{\text{RPVC}}$ may not choose its target server $\overline{S}$ to be a server for which it previously learnt the signing key and which are recorded on the list Q_{Server}. Therefore, $\mathcal{A}_{\text{RPVC}}$ cannot choose $\overline{S} = S = \widetilde{S}$ and hence, $\mathcal{A}_{\text{Sig}}$'s choice of target server identity was wrong and thus the EUF-CMA challenge parameters were embedded incorrectly in the reduction.

4. Eventually, $\mathcal{A}_{\text{RPVC}}$ finishes its query phase and outputs its choice of challenge function F and challenge input $x^{\star}$.

5. $\mathcal{A}_{\text{Sig}}$ runs RPVC.FnInit as detailed in Algorithm 2. It also runs RPVC.ProbGen on the challenge input $x^{\star}$ as specified in Algorithm 5.

6. $\mathcal{A}_{\text{RPVC}}$ is provided with the resulting parameters and is given again oracle access which is handled as above.

   $\mathcal{A}_{\text{RPVC}}$ eventually outputs its choice for a target server identity $\overline{S}$. $\mathcal{A}_{\text{Sig}}$ returns 0 if $\mathcal{A}_{\text{RPVC}}$ has previously queried this identity in the Register oracle and thus the identity was listed on Q_{Server}.

7. If $\overline{S} \neq \widetilde{S}$, then $\mathcal{A}_{\text{Sig}}$ outputs $\perp$ since it has guessed incorrectly the server identity. Otherwise, $\mathcal{A}_{\text{RPVC}}$ continues with access to the provided oracles as before and additionally receives access to a Compute oracle as specified in Figure 3.6. $\mathcal{A}_{\text{RPVC}}$ submits queries to the Compute oracle of form $\mathcal{O}^{\mathsf{Compute}}(\sigma_{F',x}, ek_{F',\overline{S}}, sk_{\overline{S}}, pp)$ for some computation $F'(x)$ which does not correspond to $F(x^{\star})$. Recall that this oracle enables $\mathcal{A}_{\text{RPVC}}$ to observe $\overline{S}$ before attacking. If $S \neq \widetilde{S}$ then $\mathcal{A}_{\text{Sig}}$ simply follows Algorithm 6 using the decryption and signing keys generated during the oracle queries. Otherwise, if $S = \widetilde{S}$ then $\mathcal{A}_{\text{Sig}}$ does not have access to the signing key $SK_{\overline{S}}$. Thus, it runs the ABE.Decrypt operations correctly to generate plaintexts d_0 and d_1, and submits $m = (d_0, d_1, \widetilde{S})$ as a Sig.Sign oracle query to $\mathcal{C}$. The challenger adds m to the list Q and returns $\gamma \leftarrow \mathsf{Sig.Sign}(m, \overline{SK})$, which $\mathcal{A}_{\text{Sig}}$ uses to return $\theta_{F(x)} = (d_0, d_1, \widetilde{S}, \gamma)$.

8. $\mathcal{A}_{\text{RPVC}}$ finally outputs $\theta^\star = (d_0^\star, d_1^\star, \overline{S}, \gamma)$ which appears to be an invalid result computed by $\overline{S}$. Thus, RPVC.Verify will output a `reject` token for $\overline{S}$, i.e. $\tau_{\theta^\star} = (\texttt{reject}, \overline{S})$, and $\texttt{accept} \leftarrow \mathsf{Sig.Verify}((d_0^\star, d_1^\star, \overline{S}), \gamma, \overline{VK})$. Thus, γ is a valid signature under key $\overline{SK}$.

9. $\mathcal{A}_{\text{Sig}}$ outputs $m^\star = (d_0^\star, d_1^\star, \overline{S})$ and $\gamma^\star = \gamma$ to $\mathcal{C}$.

Note that the Compute oracle was not simulated for the computation $F(x^\star)$ due to the "if" statement (line 1) in the oracle (cf. Figure 3.6) since $\mathcal{A}_{\text{Sig}}$ did not make any query to the Sig.Sign oracle provided by $\mathcal{C}$. Thus the forgery $(m^\star, \gamma^\star)$ outputted by $\mathcal{A}_{\text{Sig}}$ satisfies the requirement in the EUF-CMA game (Figure 2.7) that $m^\star \notin Q$.
We argue that, assuming $\overline{S} = \widetilde{S}$ (i.e. $\mathcal{A}_{\text{Sig}}$ correctly guessed the challenge identity) then $\mathcal{A}_{\text{Sig}}$ succeeds with the same non-negligible advantage δ as $\mathcal{A}_{\text{RPVC}}$. We assume that $n = |\mathcal{U}_{\text{ID}}|$ is polynomial in the security parameter (else the KDC could not efficiently search the list L_{Reg}). The probability that $\mathcal{A}_{\text{Sig}}$ correctly guesses $\overline{S} = \widetilde{S}$ is $\frac{1}{n}$ and

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}_{\text{Sig}}} &\geq \frac{1}{n}\mathbf{Adv}_{\mathcal{A}_{\text{RPVC}}} \\ &\geq \frac{\delta}{n}.\end{aligned}$$

We conclude that, since n is polynomial in the security parameter, if $\mathcal{A}_{\text{RPVC}}$ has a non-negligible advantage in the vindictive servers game (Figure 3.6) then $\mathcal{A}_{\text{Sig}}$ has the same advantage in the EUF-CMA game, but since the signature scheme is assumed EUF-CMA secure, $\mathcal{A}_{\text{RPVC}}$ may not exist. □

We note that we lose a polynomial factor in the advantage due to having to guess the server $\widetilde{S}$ that the adversary will attempt to revoke. This factor could be removed if we formulated the security model in a selective fashion such that $\mathcal{A}_{\text{RPVC}}$ must declare up front which server it will target, and then $\mathcal{A}_{\text{Sig}}$ can implicitly set the signing key for that server (in the Register step) to be the challenge key in the EUF-CMA game and forward any Compute oracle requests to the challenger.

3.5.4 Selective Vindictive Manager

Lemma 3.15. *The $\mathcal{RPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective vindictive manager (Figure 3.10) under the same assumptions as in Theorem 3.11.*

Proof. Since the notion of vindictive manager is a natural extension of the public verifiability notion in the framework of the manager model, the following security proof follows similar to the proof of Lemma 3.12. We begin by defining the following three games:

- **Game 0.** This is the selective vindictive manager game as defined in Figure 3.10.
- **Game 1.** This is the same as **Game 0** with the modification that in ProbGen, we no longer return an encryption of m_0 and m_1. Instead, we choose another random message $m' \neq m_0, m_1$ and, if $F(x^\star) = 1$, we replace c_1 by the encryption of m', and otherwise we replace c_0. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.
- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

We aim to show that from the the adversary's point of view **Game 2** is indistinguishable from **Game 0** except with negligible probability. Thus, this means that an adversary against the selective vindictive manager game can be run against **Game 2**. We then finally show that if an adversary has a non-negligible advantage against **Game 2** then the adversary can invert a one-way function.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{\text{RPVC}}$ can distinguish the two games with non-negligible advantage δ. We then show that it is possible to construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{RPVC}}$ as a sub-routine to break the IND-sHRSS security of the indirectly revocable KP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-sHRSS game (Figure 2.3) with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ in turn acts as a challenger for $\mathcal{A}_{\text{RPVC}}$. Given the above parameters the entities interact in the following way.

1. $\mathcal{A}_{\text{RPVC}}$ declares its choice of challenge function F and challenge input $x^\star$.
2. $\mathcal{A}_{\text{ABE}}$ first transforms the challenge input from $\mathcal{A}_{\text{RPVC}}$ into its own challenge input for the IND-sHRSS game. It sets $\overline{x^\star} = A_{x^\star} \cup \Lambda(F)$ where $\Lambda(F) \in \mathcal{U}_{\mathcal{F}}$ corresponds to the attribute representing the challenge function F. Finally, $\mathcal{A}_{\text{ABE}}$ also sets its challenge for the time period $t^\star = 1$ for the IND-sHRSS game and sends both challenge parameters to the challenger $\mathcal{C}$. $\mathcal{A}_{\text{ABE}}$ also computes $r = F(x^\star)$ which will determine which of the two ABE systems will be used for functions and which for the complement functions. This is required since $\mathcal{C}$ will not issue a decryption key for a function satisfied by the challenge input and so $\mathcal{A}_{\text{ABE}}$ must be sure that it will only be queried for the non-satisfied function. In the following, let us use the notation F_r as follows:
   - If $r = 0$ then $F_r = F$ and $F_{1-r} = \overline{F}$

- If $r = 1$ then $F_r = \overline{F}$ and $F_{1-r} = F$.

That is, we choose r such that $F_r(x^\star) = 0$.

3. $\mathcal{C}$ runs the ABE.Setup algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.

4. $\mathcal{A}_{\text{ABE}}$ initialises an empty challenge target revocation list $\overline{R}$.

5. $\mathcal{A}_{\text{ABE}}$ then simulates running RPVC.Setup by running Algorithm 1 as written, with the exception that one of the sets of ABE system parameters is assigned to be those generated by the challenger. Recall that $r = F(x^\star)$. $\mathcal{A}_{\text{ABE}}$ sets mpk^r_{ABE} to be the public parameters issued by $\mathcal{C}$ and msk^r_{ABE} is implicitly set to be that held by $\mathcal{C}$. It runs ABE.Setup to generate $mpk^{1-r}_{\text{ABE}}, msk^{1-r}_{\text{ABE}}$ as usual.

6. $\mathcal{A}_{\text{ABE}}$ runs RPVC.FnInit as detailed in Algorithm 2. Later $\mathcal{A}_{\text{ABE}}$ needs to output a challenge encoded output and therefore it needs to simulate a computation server. To do so, $\mathcal{A}_{\text{ABE}}$ first needs to pick a server identity from the space $\mathcal{U}_{\text{ID}}$ uniformly at random and runs the Register algorithm as detailed in Algorithm 3 to register S. $\mathcal{A}_{\text{ABE}}$ then needs to simulate running the Certify algorithm for S and challenge function F. However, $\mathcal{A}_{\text{ABE}}$ does not hold the full master secret key mk and therefore is required to make appropriate oracle calls to the challenger.

   $\mathcal{A}_{\text{ABE}}$ will run Algorithm 4 as specified with exceptions to queries to KeyGen and KeyUpdate oracles for which the challenger's parameters are necessary. In more detail:

   - sk^r_{ABE} is generated by issuing an oracle query to the ABE.KeyGen oracle of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, F_r \wedge \Lambda(F), msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$. $\mathcal{C}$ will return a valid decryption key unless $\overline{x^\star} \in F_r \wedge \Lambda(F)$ *and* $S \notin \overline{R}$. It is clear that S is never listed in $\overline{R}$ as the list was chosen to be empty and therefore the second clause in the "if" statement in the KeyGen oracle (Figure 2.3) is always satisfied. However, r was specifically chosen such that $F_r(x^\star) = 0$. Hence, $x^\star \notin F_r$ and $\overline{x^\star} = A_{x^\star} \cup \Lambda(F) \notin F_r \wedge \Lambda(F)$. Thus, $\mathcal{C}$ will always be able to return a valid decryption key sk^r_{ABE}.
   - sk^{1-r}_{ABE} is generated by $\mathcal{A}_{\text{ABE}}$ running ABE.KeyGen using msk^{1-r}_{ABE} for the function $F_{1-r} \wedge \Lambda(F)$ as usual.
   - $uk^r_{L_{\text{Rev}},t}$ is generated by making a query to the ABE.KeyUpdate oracle of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$. $\mathcal{C}$ will return a valid update key unless the current time period t is the challenge time period $t^\star$ and $\overline{R} \not\subseteq L_{\text{Rev}}$. Note that $t^\star$ was chosen to be 1 and since no Revoke queries were requested, the time period t indeed corresponds to $t^\star$. On the other hand, the list $\overline{R}$ was initially chosen to be empty and therefore it is indeed a subset of L_{Rev}

(both lists are empty). Therefore, the second clause in the "if" statement in the KeyUpdate oracle (Figure 2.3) is never satisfied and thus $\mathcal{C}$ will always return a valid update key $uk^r_{L_{\text{Rev}},t}$.

- $uk^{1-r}_{L_{\text{Rev}},t}$ is generated by $\mathcal{A}_{\text{ABE}}$ running ABE.KeyUpdate using msk^{1-r}_{ABE} as usual.

7. $\mathcal{A}_{\text{ABE}}$ must generate a challenge problem instance for $\mathcal{A}_{\text{RPVC}}$ as the output of RPVC.ProbGen for either **Game 0** or **Game 1**. To do so, $\mathcal{A}_{\text{ABE}}$ samples three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. $\mathcal{A}_{\text{ABE}}$ provides m_0 and m_1 as its choice of challenge to $\mathcal{C}$, and receives back the encryption, $ct^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$, where $b^\star$ is chosen by the challenger), under attributes $\overline{x^\star}$, time $t^\star$ and public parameters mpk^r_{ABE}. It needs to assign $ct^\star$ to be one of the ciphertexts c or c' that form the challenge problem instance (encoded input) $\sigma_{F,x^\star}$ using the correct ABE system parameters. $\mathcal{A}_{\text{ABE}}$ chooses a random bit $s \xleftarrow{\$} \{0,1\}$ which intuitively corresponds to its guess for the challenger's choice of $b^\star$. Therefore,

   - If $r = 0$, then $\mathcal{A}_{\text{ABE}}$ sets c to be $ct^\star$ and randomly generates the remaining ciphertext as

     $$c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \overline{x^\star}, t^\star, mpk^1_{\text{ABE}}).$$

     It sets $vk = g(m_s)$ and $vk' = g(m')$.
   - If $r = 1$, then $\mathcal{A}_{\text{ABE}}$ randomly generates the ciphertext as

     $$c \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \overline{x^\star}, t^\star, mpk^0_{\text{ABE}}),$$

     and sets c' to be $ct^\star$. Further it sets $vk = g(m')$ and $vk' = g(m_s)$.

   Finally, $\mathcal{A}_{\text{ABE}}$ sets $\sigma_{F,x^\star} = (c, c')$ and $vk_{F,x^\star} = (vk, vk', L_{\text{Reg}})$.

8. $\mathcal{A}_{\text{ABE}}$ now simulates the server S performing the computation to output $\theta_{F(x^\star)}$ by running algorithm 6 as specified, since valid keys have been generated for S in the preceding steps.

9. The resulting values $\sigma_{F,x^\star}$, $\theta_{F(x^\star)}$, $vk_{F,x^\star}$, pk_F and pp are sent to $\mathcal{A}_{\text{RPVC}}$ that is also additionally provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds as follows:

   - Queries to RPVC.FnInit and RPVC.Register are performed as specified in algorithms 2 and 3.
   - Queries of the form $\mathsf{RPVC.Certify}(S, F', mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ running algorithm 4 with the exception that the ABE.KeyGen and

ABE.KeyUpdate algorithms for the ABE system with the parameters maintained by the challenger are replaced by queries to the respective oracles provided by $\mathcal{C}$.

$\mathcal{A}_{\text{ABE}}$ queries the ABE.KeyGen oracle in order to obtain a decryption key sk^r_{ABE} with queries of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, F'_r \wedge \Lambda(F'), msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$. The challenger returns a valid decryption key unless the challenge input $\overline{x^\star}$ satisfies the queried policy.

Note that in case the queried function F' does not correspond to the challenge function F then due to the bijective mapping Λ the attribute function labels do not coincide, i.e. $\Lambda(F') \neq \Lambda(F)$. Thus, neither of the possible queries to the KeyGen oracle will be satisfied. On the other hand, since we chose the ABE system maintained by the challenger to be unsatisfied for F, any query for F will be rejected. Hence, if $F' \neq F$, then the first clause of the "if" statement in the KeyGen oracle of the IND-sHRSS game (Figure 2.3) will never be correct and therefore the challenger will always return a valid decryption key in response to a query.

In order to generate a valid update key $uk^r_{L_{\text{Rev}},t}$, $\mathcal{A}_{\text{ABE}}$ makes a query to the ABE.KeyUpdate oracle which are of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$. The challenger returns a valid update key if the current queried time period t does not coincide with the challenge time period which $\mathcal{A}_{\text{ABE}}$ chose to be 1 *and* the queried revocation list L_{Rev} contains the challenge target revocation list $\overline{R}$ which $\mathcal{A}_{\text{ABE}}$ chose initially to be empty. Since $\overline{R} = \epsilon$ is a subset of any revocation list L_{Rev}, the second clause of the "if" statement in the KeyUpdate oracle of the IND-sHRSS game (Figure 2.3) will not be satisfied and therefore the challenger will always return a valid update key.

- Queries of the form RPVC.Revoke$(\tau_{\theta_{F(x)}}, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ running Algorithm 8 with the exception of generating an update key $uk^r_{L_{\text{Rev}},t}$ on line 6 and 7 which will be replaced with respective queries to the KeyUpdate oracle provided by $\mathcal{C}$. $\mathcal{A}_{\text{ABE}}$ queries for $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, msk^r_{\text{ABE}}, mpk^r_{\text{ABE}})$ and the challenger returns a valid key if the queried time period t does not correspond to the challenge time period, i.e. $t \neq 1$, and if the queried revocation list does not contain the challenge target revocation list $\overline{R}$. Since $\mathcal{A}_{\text{ABE}}$ chose $\overline{R}$ to be empty, the second clause of the "if" statement in the KeyUpdate oracle will not be satisfied and the challenger returns a valid update key.

10. Eventually $\mathcal{A}_{\text{RPVC}}$ outputs its guess for the non-$\perp$ value Y and $\tau_{\theta_{F(x)}}$. If $g(Y) = g(m_s)$, $\mathcal{A}_{\text{ABE}}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{\text{ABE}}$ guesses $b' = 1 - s$.

Notice that if $s = b^\star$ (the challenge bit chosen by $\mathcal{C}$ in the IND-sHRSS game), then the distribution of the above coincides with **Game 0** since the verification key comprises $g(m')$ and $g(m_s)$ where m' and m_s are the two plaintexts corresponding to the ciphertexts of the encoded input for which $\mathcal{A}_{\text{RPVC}}$ recovers exactly one. Otherwise, if $s = 1 - b^\star$ then the distribution coincides with **Game 1** since the verification key comprises the one-way function g applied to a legitimate message m' and a random message $m_{1-b^\star}$ that is unrelated to both ciphertexts.

Now, we consider the advantage of this constructed $\mathcal{A}_{\text{ABE}}$ playing the IND-sHRSS game for the revocable KP-ABE scheme. Recall that by assumption, $\mathcal{A}_{\text{RPVC}}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1** – that is

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right| \geqslant \delta$$

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right]$ denotes the output of running $\mathcal{A}_{\text{RPVC}}$ in **Game i**.

Now we derive the probability of $\mathcal{A}_{\text{ABE}}$ guessing $b^\star$ and it follows:

$$\begin{aligned}
\Pr[b' = b^\star] &= \Pr[s = b^\star]\Pr[b' = b^\star | s = b^\star] + \Pr[s \neq b^\star]\Pr[b' = b^\star | s \neq b^\star] \\
&= \frac{1}{2}\Pr[g(Y) = g(m_s) | s = b^\star] + \frac{1}{2}\Pr[g(Y) \neq g(m_s) | s \neq b^\star] \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + \frac{1}{2}\left(1 - \Pr[g(Y) = g(m_s) | s \neq b^\star]\right) \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] \\
&\quad + \frac{1}{2}\left(1 - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right. \\
&\quad \left. - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{RPVC}}}\left[\mathcal{RPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\text{ABE}}} &\geqslant \left|\Pr[b' = b^\star] - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&= \frac{\delta}{2}.
\end{aligned}$$

Since δ is assumed to be non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\text{RPVC}}$ has advantage δ at distinguishing these games then $\mathcal{A}_{\text{ABE}}$ can win the IND-sHRSS game with non-

negligible probability. Thus since we assumed the ABE scheme to be IND-sHRSS secure, we conclude that $\mathcal{A}_{\text{RPVC}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is to simply set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game (Figure 2.8). We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{\text{RPVC}}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{\text{ABE}}$).

Final Proof. We now show that using $\mathcal{A}_{\text{RPVC}}$ in **Game 2**, $\mathcal{A}_{\text{ABE}}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ $\mathcal{A}_{\text{ABE}}$ can recover w. Specifically, during $\mathsf{ProbGen}$, $\mathcal{A}_{\text{ABE}}$ chooses the messages as follows:

- if $F(x^\star) = 1$, we implicitly set m_1 to be w and the corresponding verification key component to be $z = g(w)$. We randomly choose m_0 from the message space and compute the remainder of the verification key as usual.
- if $F(x^\star) = 0$, we implicitly set m_0 to be w and set the verification key component to $z = g(w)$. m_1 is chosen randomly from the message space and the remainder of the verification key computed as usual.

Now, since $\mathcal{A}_{\text{RPVC}}$ is assumed to be successful, it will output a forgery comprising the plaintext that was encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{\text{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\text{RPVC}}$ has against the selective vindictive manager game.

We conclude that if the ABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $\mathcal{RPVC}$ as defined by Algorithms 1–8 is secure in the sense of selective vindictive manager. □

Overall Theorem 3.11 follows as a Corollary of Lemmas 3.12–3.15.

3.6 Conclusion

In this chapter, we have introduced the new notion of revocable publicly verifiable outsourced computation (RPVC) and provided a rigorous framework that we believe to be more realistic than the purely theory oriented models of prior work, especially when the KDC is an entity responsible for user authorisation within a organisation. We

believe our model more accurately reflects practical environments and the necessary interaction between entities for PVC. Each server may provide services for many different functions and for many different clients. Additionally, in our model, any clients may submit multiple requests to any available servers, whereas prior work considered just the single server case.

The consideration of this new model leads to new functionalities as well as new security threats. We have shown that by using a revocable KP-ABE scheme we can revoke misbehaving servers such that they receive a penalty for misbehaving. We have extended previous notions of security to fit our new definitional framework, introduced new models to capture additional threats (e.g. vindictive servers using revocation to remove competing servers), and provided a provably secure construction. We believe that this work is a useful step towards making PVC practical and provides a natural set of baseline definitions from which to add future functionality.

As mentioned throughout the chapter, we believe that for future work it would be interesting and beneficial to investigate the construction of a fully secure indirectly revocable KP-ABE scheme with adaptive queries. Such a scheme would allow to overcome the limitations of the selective and semi-static notions which are forced upon our security models given the current primitives.
Furthermore, achieving a notion of output privacy via the informally discussed mechanism of blind verification is yet another interesting problem which would enhance the usability of our RPVC framework in practice.

Chapter 4

Publicly Verifiable Delegable Computation

Contents

In this chapter, we investigate a different mode of publicly verifiable computation. We study the problem in which an untrusted server holds a data set in such a way that any client can ask the server to compute a function on any input portion of the data set. We show that ciphertext-policy attribute-based encryption can provide this novel mode of computation and see that this setting has natural applications such as verifiable queries on remote data and verifiable MapReduce operations. The results of this chapter appear in the full online version of [6].

4.1 Introduction

With the emergence of cloud computing, where clients and businesses rent computation and storage resources from powerful cloud service providers, it is of great importance to ensure the correctness of computations and the integrity of stored data. Let us consider the following example scenario. A server possesses a static (authenticated) data set D consisting of data points. For instance, each data point could correspond to a measured temperature at fixed time intervals, e.g. every day at midday. The server now wishes to specify a list of functions $\mathcal{F}$ for which it is willing to let entities query the data set. Here, for example, the server could specify one function to compute the average temperature over a portion of the data set, whereas another function could evaluate the standard deviation. A client could now request the evaluation of a function from $\mathcal{F}$ on the server's data set in order to receive a result for which the client wants to ensure correctness. However, as the data owner wishes to keep the data set confidential the

crucial task is to ensure that verification of the result is successful even without the client knowing the data.

In this chapter, we model the above scenario and introduce the notion of *publicly verifiable delegable computation* which we simply abbreviate as VDC. This proposal can be considered as a reversed model to the problem of publicly verifiable outsourced computation (PVC), where clients outsource requests for computations on their own input data, as discussed in the previous chapter. In VDC, servers play the role of the data owner possessing a static database over which any client can request computations/queries to be performed. Hence, compared to PVC, the entity relationship is more akin to the traditional client-server model. In the previous chapter, we have seen that key-policy attribute-based encryption can be used to provide a mechanism of proving correctness of an outsourced computation. Here, in this chapter, we study whether ciphertext-policy attribute-based encryption (CP-ABE) lends itself in any meaningful way to the setting of outsourced computation. Note that CP-ABE is a reversed notion of KP-ABE where the association of attribute sets and policies to ciphertexts and keys are simply reversed. We answer in the affirmative and show that CP-ABE can be employed to build such a VDC scheme. We may embed the static data set in a server's secret key whilst the computation of functions can be requested by creating ciphertexts using public information. In order to keep the data set confidential, as well as enabling the client to query on any subsets of the data set, we let the server publish a unique description of each data point in form of a label. It suffices for the client to select servers and data based *only* on the knowledge of these labels. The scheme enables clients to publicly verify the computational results. Referring back to the battlefield example from Chapter 3, local servers within a coalition may offer soldiers to query their stored data. However, it is of great importance that soldiers in the proximity of the querying soldier may also verify the returned result since the outcome may also influence their own decisions. In case the results are of sensitive nature the querier needs to employ a VC scheme that provides output privacy. Furthermore, we discuss that VDC has some natural applications. For example, VDC enables to construct verifiable MapReduce operations for parallel computing problems, and can also be used to perform verifiable queries on remote databases without accessing the data itself.

The *efficiency* requirement for VDC is also very different from PVC: unlike PVC, outsourcing a computation is not merely an attempt to gain efficiency as the client never possesses the input data and therefore cannot execute the computation herself (even with the necessary resources). Thus, VDC does not have the stringent efficiency requirement present as in PVC. Recall that in PVC outsourcing and verifying computations are required to be more efficient than performing the computation itself in order to make outsourcing computations worthwhile. The efficiency requirement is simply that

verification of a result is more efficient than computing it. We believe that CP-ABE behaves reasonably well in this setting. More precisely, our solution achieves constant time public verification and the size of the query depends on the function F, while the size of the server's response depends only on the size of the result itself and *not* on the input size which may be large, particularly when querying remote databases.

Related Work

Work from the realm of authenticated data lends itself to the framework of verifiable computations over outsourced data (albeit for specific functions only). Backes et al. [21] introduce a framework using privacy-preserving proofs over authenticated data outsourced by a trusted client that enables other entities to execute computations over the outsourced data. In more detail, a trusted source produces and authenticates some data which is given to the (untrusted) server. Other entities are able to request computations on this authenticated data and also efficiently verify the results whereas they should not learn more than the computational result. In particular, the client should not learn the data itself. The solution presented in [21] makes use of homomorphic MACs and succinct non-interactive arguments (SNARGs) [75, 108]. Similar results are presented in [137] using public logs. It is notable that the solutions in [21], [33] and [114] achieve public verifiability, and we note that the scheme in [114] only achieves it for specific set operations.

Chung et al. [53] introduce the concept of *memory delegation* where a client uploads her memory to a server that can update and compute a function F over the entire memory. In contrast, the model we propose in this chapter does not consider the client to be the data owner and we can enable computations to be performed on any subset of the input data, but are restricted to the static data case. Backes et al. [22] consider a client that outsources data and requests computations on a data portion. The client can efficiently verify the correctness of the result without holding the input data. Most works require the client to know the data in order to verify [73, 29, 33, 113]. *Verifiable oblivious storage* [9] ensures data confidentiality, access pattern privacy, integrity and freshness of data accesses. Etemad and Küpçü [64] show how to handle different types of queries on hierarchical authenticated data structures. Ahn et al. [3] present a framework for computing on authenticated data using the notion of P-homomorphic signatures.

Organisation of Chapter

In Section 4.2, we describe and provide a formal definition of our model of publicly verifiable delegable computation and we discuss some example applications, e.g. MapReduce, in which our model can be applied. Next, in Section 4.3, we define security in terms of public verifiability for our model and provide in Section 4.4 an example construction of VDC based on ciphertext-policy attribute-based encryption. In Section 4.5, we show

that our given construction is secure according to the introduced security model. We conclude the chapter in Section 4.6.

4.2 Publicly Verifiable Delegable Computation

In this section, we introduce a new PVC mode of computation called *publicly verifiable delegable computation* (VDC). The model introduces a change in roles regarding the data ownership. In more detail, in VDC the server initially holds a data set (gathered over time or given by a trusted source) which it makes available for querying of specific functions and the client is able to efficiently verify the result without ever having possessed the data set herself. Thus, this model reverses the roles of data owner compared to Chapter 3.

We now first informally describe our VDC model. The scheme uses similarly to Chapter 3 the notion of a key distribution centre (KDC) that handles the expensive preprocessing operation and distributes the keys but is not responsible for performing any computations itself. In more detail, a VDC scheme for a family of functions $\mathcal{F}$ comprises n computational servers S_i, $i \in [n]$. Each server owns a static data set D_i consisting of k_i elements ($D_i = \{x_{i,j}\}_{j=1}^{k_i}$) and also specifies a list of functions $\mathcal{F}_i \subseteq \mathcal{F}$ that it is willing to compute on (specified) portions of the data set. To enable clients to query specific portions of the data, S_i publishes a unique descriptive label $l(x_{i,j})$ of each data point $x_{i,j} \in D_i$. Note that, to preserve confidentiality of the server's data set from the client, the labels should *not* reveal the value of the data, but may reveal the semantic meaning of such data. The KDC authenticates the server's data set by providing an evaluation key ek_{D_i,S_i} to S_i. This key enables S_i to compute any function F in $\mathcal{F}_i$ on its data set D_i. Clients can request computations of any function $F \in \mathcal{F}_i$ on any set of data points $X \subseteq D_i$ (as long as $X \in \mathsf{Dom}(F)$) by specifying the respective set of labels $\{l(x_{i,j})\}_{x_{i,j} \in X}$. Note that the function has $|X|$ inputs, e.g. if $X = \{x_1, x_2, x_5\}$ then the server computes the function $F(x_1, x_2, x_5)$.

One could suggest that a server S_i just simply caches the results of computations $F \in \mathcal{F}_i$ on its data set. However, this is an unattractive solution as the choice of specific data points X that are acceptable for each computation may vary and as such the number of results that need to be cached could be large.

In Figure 4.1, we illustrate the entity population and respective interaction between the entities within the VDC model.

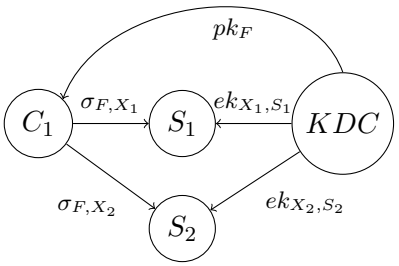

Figure 4.1: Operation of a VDC scheme

4.2.1 Formal Definition

A VDC scheme for a family of functions $\mathcal{F}$ begins with a key distribution centre (KDC) running Setup to produce public parameters and a master secret key.[1] Furthermore, the KDC also registers each server S_i, by providing an individual private signing key sk_{S_i}, and publishes a public delegation key pk_F for each function of interest $F \in \mathcal{F}$. Each server S_i registers their interest in performing computations on their data set D_i. This enrolment process is done by the KDC using the $\mathsf{Certify}$ algorithm to issue a single evaluation key ek_{D_i,S_i} enabling S_i to perform computations on D_i. Note that this implicitly also enables S_i to compute on any subsets of D_i. Each data owner (server) specifies a list of functions $\mathcal{F}_i \subseteq \mathcal{F}$ that they are willing to evaluate on their data sets. However, not all data points in the data set $x_{i,j} \in D_i$ may be appropriate for each function. Therefore, we define the set of functions $\mathcal{F}_i$ to consist of elements represented in the form $(F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$ listing the function and the associated permissible inputs.

The client executes the $\mathsf{ProbGen}$ algorithm to request a computational result from a server. The client specifies a function $F \in \mathcal{F}_i$ and a set of data points $X \subseteq \mathsf{Dom}(F)$ represented by their unique labels $\{l(x_{i,j})\}_{x_{i,j} \in X}$. The algorithm outputs an encoded input $\sigma_{F,X}$ and a verification key $vk_{F,X}$ that enables *anyone* to verify the computational result later. In the $\mathsf{Compute}$ algorithm a server S_i uses its evaluation key ek_{D_i,S_i} and the encoded input to output an encoded result $\theta_{F(X)}$ corresponding to the computational result $F(X)$.

Finally any entity can verify the correctness of $\theta_{F(X)}$ using $vk_{F,X}$. Verification outputs the result $y = F(X)$ indicating that the computation was performed correctly, or else $y = \perp$ showing that the computational response is malformed. More formally this is captured in the following definition.

Definition 4.1. *A* publicly verifiable delegable computation (VDC) scheme *comprises*

[1] Backes et al. [21] make use of a trusted source which can be thought of as the trusted KDC in our model.

the following algorithms:

1. $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$: *this randomised algorithm is run by the KDC to initialise the system. The inputs are the security parameter λ and the family of functions $\mathcal{F}$;*

2. $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$: *this randomised algorithm is run by the KDC to generate a public delegation key pk_F allowing clients to request computations of a function $F \in \mathcal{F}$;*

3. $sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp)$: *this randomised algorithm is run by the KDC to enrol a server S_i within the system. It generates the server's personalised signing key sk_{S_i};*

4. $ek_{D_i,S_i} \xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{l(x_{i,j})\}_{x_{i,j} \in D_i}, \mathcal{F}_i, mk, pp)$: *this randomised algorithm is run by the KDC to generate an evaluation key ek_{D_i,S_i} enabling the certified server S_i to perform computations of all functions $F \in \mathcal{F}_i \subseteq \mathcal{F}$, chosen by the server. Those computations can be performed on the server's input data $D_i = \{x_{i,j}\}_{j=1}^{k_i}$ consisting of k_i data points each uniquely labelled by $l(x_{i,j})$;*

5. $(\sigma_{F,X}, vk_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(F, \{l(x_{i,j})\}_{x_{i,j} \in X}, pk_F, pp)$: *this randomised algorithm is run by the client to request a computation of the function F evaluated on a set of data points $X \subseteq D_i$ owned by the server S_i. The inputs are the function F, a set of labels identifying each data point $x_{i,j} \in X$ that the client wishes F to be evaluated on, as well as the public delegation key pk_F and the public parameters pp. The algorithm outputs an encoded input $\sigma_{F,X}$ and a public verification key $vk_{F,X}$.*

6. $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,X}, ek_{D_i,S_i}, sk_{S_i}, pp)$: *this randomised algorithm is run by the server S_i to compute $F(X)$. The inputs are the encoded input $\sigma_{F,X}$, an evaluation key ek_{D_i,S_i} enabling S_i to compute on its data set D_i, the server's signing key sk_{S_i} and the public parameters pp. The algorithm outputs an encoded output $\theta_{F(X)}$ representing $F(X)$.*

7. $y \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{F,X}, pp)$: *this algorithm can be run by any entity. The inputs are the encoded output $\theta_{F(X)}$ produced by S_i, the verification key $vk_{F,X}$ and the public parameters pp. The algorithm produces an output $y = F(X)$ if the result was computed correctly, or else $y = \perp$ indicating that the computation was performed incorrectly.*

In our VDC scheme we do not consider the mechanism of revocation but observe that an indirectly revocable CP-ABE scheme could be employed in a similar fashion to the indirectly revocable KP-ABE scheme as presented in Chapter 3. Although not explicitly mentioned, the KDC may update the public parameters pp during any algorithm

and execution in order to reflect any changes in the entity population as e.g. servers may be added or granted the ability to perform computations for additional functions. Note that in the algorithm $\mathsf{ProbGen}$, the data itself is not needed in order to form a computational request for the server. The respective unique labels suffice as the client never owned the data in the first place herself.

A VDC scheme is *correct* if verification succeeds with overwhelming probability when all algorithms are run honestly. More formally this can be represented as follows.

Definition 4.2. *A publicly verifiable delegable computation scheme is* correct *for a family of functions $\mathcal{F}$ if, for all functions $F \in \mathcal{F}$, servers S_i and respective input data sets D_i, and all computational inputs $X \subseteq D_i$, where $X \in \mathsf{Dom}(F)$, the following holds:*

$$
\begin{aligned}
&\Pr[(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F}),\\
&\quad pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp),\\
&\quad sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp),\\
&\quad ek_{D_i,S_i} \xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{l(x_{i,j})\}_{x_{i,j} \in D_i}, \mathcal{F}_i, mk, pp),\\
&\quad (\sigma_{F,X}, vk_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(F, \{l(x_{i,j})\}_{x_{i,j} \in X}, pk_F, pp),\\
&\quad \theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,X}, ek_{D_i,S_i}, sk_{S_i}, pp),\\
&\quad F(X) \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{F,X}, pp)]\\
&\quad = 1 - \mathrm{negl}(\lambda).
\end{aligned}
$$

4.2.2 Possible Applications of VDC

We briefly discuss two example applications where our notion of VDC could be applied.

MapReduce [61] (or Hadoop) is a programming model for the parallel processing of large computations using a cluster or grid of computers (nodes) which can take advantage of the locality of data to decrease transmission costs. Each worker node computes a sub-problem on a portion of the data and reports to a manager who combines the results. VDC enables verifiable MapReduce such that only *valid* results are combined. The manager acts as the KDC to distribute evaluation keys for partitions of the data to workers, and then requests multiple sub-problems to be solved over this partitioning.

Note that in the setting of MapReduce, the set of permissible functions $\mathcal{F}_i$ could indeed correspond to the full set $\mathcal{F}$ since the client is the original data owner of the entire input data and provides each server with only a static portion of input data.

Verifiable queries on remote databases. Servers may also act as remote database providers and register with a KDC to provide a verifiable querying service. Any

client may use public information to query any function allowed by the server on these databases. Data is remotely stored and a client sees nothing more than the results of queries, which they are assured to be correct. Alternatively, in this setting, the data owner could act as the KDC to outsource its data to an untrusted server. Due to the public delegation and verification properties, other data users can query the outsourced data and verify the correctness of the results. The data owner does not need to retain any knowledge of the data after it has been outsourced.

4.3 Security Model

In the context of VDC we consider security in the sense of *public verifiability* represented as a game-based notion in Figure 4.2. This notion ensures that a server is not able to return a malformed response without being detected even if it has corrupted other servers and holds verification keys, as well as even if the client or verifier never possessed the input data themselves. The notion can be seen as a natural extension to the one discussed in Section 3.3 and since we do not consider revocation in this framework, we do not require the other security notions as in Section 3.3.

4.3.1 Public Verifiability

In Figure 4.2, we present the security notion of public verifiability. The game begins with the challenger $\mathcal{C}$ initialising the system and providing the resulting public parameters pp to the adversary $\mathcal{A}$. The adversary is provided with oracle access to the functions $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$ and $\mathsf{Certify}(\cdot, \cdot, \cdot, \cdot, mk, pp)$, which we denote by $\mathcal{O}$. The adversary $\mathcal{A}$ selects the challenge inputs consisting of the challenge function F, the challenge data set $X^\star$ and the respective labels $l(x_j)$ for each data point $x_j \in X^\star$. Note that as the adversary owns the challenge data set $X^\star$ it may additionally query the $\mathsf{Certify}$ oracle for a data set $D \supseteq X^\star$. Next the challenger initialises the challenge function F by running FnInit outputting the respective key pk_F. The challenger $\mathcal{C}$ then outputs a challenge by executing $\mathsf{ProbGen}$ on the challenge function and input labels. The adversary receives the resulting parameters from the challenger and is again provided with oracle access as above. $\mathcal{A}$ wins if it produces an encoded output that verifies correctly but does not correspond to the actual result $F(X^\star)$.

Definition 4.3. *The* advantage *of a PPT adversary in the* PubVerif *game for a $\mathcal{VDC}$ construction, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{PubVerif}}_{\mathcal{A},\mathcal{VDC}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{PubVerif}}_{\mathcal{A}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right].$$

A VDC scheme, $\mathcal{VDC}$, is secure *against public verifiability if for all PPT adversaries*

$\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{VDC}, 1^{\lambda}, \mathcal{F}\right]$

1 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^{\lambda}, \mathcal{F})$
2 : $(F, X^{\star}, \{l(x_j)\}_{x_j \in X^{\star}}) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(pp)$
3 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
4 : $(\sigma_{F,X^{\star}}, vk_{F,X^{\star}}) \leftarrow\!\!\$\ \mathsf{ProbGen}(F, \{l(x_j)\}_{x_j \in X^{\star}}, pk_F, pp)$
5 : $\theta^{\star} \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma_{F,X^{\star}}, vk_{F,X^{\star}}, pk_F, pp)$
6 : $y \leftarrow \mathsf{Verify}(\theta^{\star}, vk_{F,X}, pp)$
7 : **if** $(y \neq \perp)$ **and** $(y \neq F(X^{\star}))$ **then**
8 : **return** 1
9 : **else return** 0

Figure 4.2: The public verifiability experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{VDC}, 1^{\lambda}, \mathcal{F}\right]$

$\mathcal{A}$, it holds that

$$\mathbf{Adv}_{\mathcal{A},\mathcal{VDC}}^{\text{PubVerif}}(1^{\lambda}) \leq \text{negl}(\lambda).$$

4.4 Construction

4.4.1 Overview

In this section, we provide a construction of a VDC scheme. Our VDC scheme is based upon ciphertext-policy attribute-based encryption and enables the computation of the family of (monotone) Boolean formulas closed under complement, operating similarly to the RPVC scheme as introduced in Chapter 3.

The client will choose a random message from the message space $\mathcal{M}$ to act as a verification token and encrypt this using a CP-ABE scheme under the Boolean function F to be evaluated. Each server S_i receives a decryption key for a set of attributes encoding the data D_i that they hold. Note that the following decryption procedure proceeds similarly to the basic PVC principles as summarised in Section 2.7.3. In more detail, the server attempts to decrypt the ciphertext and learns the chosen message if and only if $F(D_i) = 1$. By the security of the CP-ABE scheme, the server learns nothing about the message if $F(D_i) = 0$ since this corresponds to an access structure that is not satisfied. Thus, if the correct message is returned, the client is convinced that $F(D_i) = 1$. If, however, $F(D_i) = 0$, the decryption will return $\perp$. This is insufficient for verification since any server can return $\perp$ to convince a client of a false negative result and therefore the protocol suffers the same one-sided error problem as the scheme in Section 2.7.3.

To overcome this problem, we produce two CP-ABE ciphertexts. As in the previous chapter, one ciphertext corresponds to F, whilst the other ciphertext corresponds to the complement function $\overline{F} = F(X) \oplus 1$ which always outputs the opposite result to F for Boolean functions. Thus, if $F(D_i) = 0$ then, necessarily, $\overline{F}(D_i) = 1$. Hence, the server's key for data D_i will decrypt *exactly one* ciphertext and the returned message will distinguish whether F or $\overline{F}$ was satisfied, and therefore the value of $F(X)$, where $X \subseteq D_i$. Note that each function is encoded in terms of attributes, and is specific to each input, i.e. the encoding of a function F will differ to compute $F(X)$ and $F(X')$ as the input data set comprises different unique labels specifying data points. However, if the function F is encoded in terms of the data set X but is then evaluated on an input set $X' \supset X$ the server simply evaluates the function only on data points in X and discards all other data points that belong to X'. Thus, the computation remains the same and outputs $F(X)$. Similarly to equation (2.1), a well-formed response (d_0, d_1) in VDC satisfies the following:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } F(D_i) = 1; \\ (\perp, m_1), & \text{if } F(D_i) = 0. \end{cases} \tag{4.1}$$

If the returned plaintext does not match one of the randomly chosen random messages then the server has returned an incorrect result. Note that if both returned results correspond to $\perp$ also indicates that the server returned an incorrect result but a rational malicious server would never return this.

Public verifiability is achieved by publishing a token comprising the output of a one-way function g applied to each sampled plaintext. Any entity can apply g to the server's response and compare this result with the published token to check correctness. The public parameters contain next to the public key a list about registered entities within the system. That is a two-dimensional array L_{Reg} (indexed by server identities) where the first dimension, $L_{\text{Reg}}[S_i][0]$, contains a signature verification key, whilst the second dimension, $L_{\text{Reg}}[S_i][1]$, lists all functions and labels for which S_i is certified.

Let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}}$ be the universe of attributes for the CP-ABE scheme. It is formed over the union of three sub-universes, where $\mathcal{U}_{\text{attr}}$ consists of the attributes that form characteristic tuples for input data, $\mathcal{U}_l$ is a disjoint (from $\mathcal{U}_{\text{attr}}$) universe representing unique labels $l(x_{i,j})$ for each attribute, and $\mathcal{U}_{\text{ID}}$ comprises server identities. We assume in our instantiation that the algorithms check, in case it is applicable, that all functions and input data are formed over $\mathcal{U}_{\text{attr}}$ and each additionally contains exactly one attribute/clause over the label universe $\mathcal{U}_l$. To encode an n-bit binary input string $x = x_1 x_2 \ldots x_n$ as an attribute set A_x, we define $\mathcal{U}_{\text{attr}} = \{\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}\}$ of n attributes and let $\mathtt{x_i} \in A_x$ if and only if the ith bit of the input string is 1, i.e. $A_x = \{\mathtt{x_i} : x_i = 1\}$.

Note that the inputs to our computations are sets of data points $X \subseteq D_i$ owned by the server. We can think of the set X being the concatenation of the bit string representation of each data point $x_{i,j} \in X$ and so we can encode X in the same way as above.

We consider adversaries that have access to multiple keys and therefore must ensure that a key for different data sets cannot produce a valid looking response. We do this, similarly to the previous chapter and label each data point $x_{i,j}$ with a unique label $l(x_{i,j})$ that is valid across the entire system, and assign each label with an attribute in $\mathcal{U}_l$. Note that we refer for simplicity to these attributes as $l(x_{i,j})$ too. The decryption key for a data set D_i is formed over the attribute set $(A_x \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j}))$. During ProbGen for a computation $F(X)$ on a subset of the data set $X \subseteq D_i$, we first need to sample two random messages m_0 and m_1 of equal length. The employed CP-ABE encryption algorithm uses the access structure encoding of the conjunction $(F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j}))$ to encrypt m_0 obtaining the first ciphertext c_0. The same procedure can be applied for the complement function $\overline{F}$, i.e. a second CP-ABE encryption algorithm is executed using the access structure encoding of the conjunction $(\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j}))$ to encrypt m_1 obtaining the second ciphertext c_1. Thus, the CP-ABE decryption algorithm only succeeds if $F(D_i) = 1$ *and* all labels $l(x_{i,j})$ for $x_{i,j} \in X$ are matched in the key and ciphertext. Note that since we wish to evaluate a subset X of a data set D_i, $F(D_i) = 1$ implies that $F(X) = 1$ where the decryption procedure ignores the remaining $x_{i,j} \notin X$. Note that a generated secret key for a different data set D_j which does not contain X as a subset will not include the correct labels and thus cannot be used for an attempt to compute $F(X)$.

Our instantiation of VDC is somewhat more efficient than that for PVC since we do not require to setup two independent ABE systems and thus need to perform two (expensive) key generations. The client needs to perform roughly the amount of work of executing the computation twice by herself in order to prepare the encoded input. That is, she needs to prepare the access structure encodings $\left(F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})\right)$ and $\left(\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})\right)$ which will be used in the encryption algorithms to prepare the encoded input. However, we believe that this is an acceptable workload for the client as she *never* possessed the input data sets herself and wishes to learn the computational result $F(X)$ from the server.

4.4.2 Instantiation Details

Let $\mathcal{CP}\text{-}\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$ define a CP-ABE encryption scheme over the universe $\mathcal{U}$ for a class of Boolean functions $\mathcal{F}$ closed under complement. We also make use of a signature scheme with algorithms Sig.KeyGen, Sig.Sign and Sig.Verify, and a one-way function g. Then Algorithms 1–7 define a VDC scheme for the class of functions $\mathcal{F}$ which works as follows:

1. Setup, presented in Algorithm 1, first forms the attribute universe $\mathcal{U}$ for the function family $\mathcal{F}$. The attribute universe comprises of all input labels and server identities. The algorithm then calls the ABE.Setup algorithm in order to initialise the CP-ABE scheme. Next it initialises a two-dimensional array L_{Reg} indexed by the server identities. For each server S_i, the first dimension of the array $L_{\text{Reg}}[S_i][0]$ stores a signature verification key for S_i and the second dimension $L_{\text{Reg}}[S_i][1]$ stores a list of functions that S_i is willing to compute. Note that L_{Reg} is initially empty. The output of the algorithm consists of the public parameters pp and the master secret mk for the VDC system. The public parameters consist of the master public key mpk_{ABE} of the CP-ABE scheme and the array L_{Reg}. The master key comprises of the master secret key msk_{ABE} of the CP-ABE scheme.

Algorithm 1 $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$

$\mathcal{U} \leftarrow \mathcal{U}_{\text{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}}$
$(mpk_{\text{ABE}}, msk_{\text{ABE}}) \leftarrow_{\$} \mathsf{ABE.Setup}(1^\lambda, \mathcal{U})$
for $S_i \in \mathcal{U}_{\text{ID}}$
  $L_{\text{Reg}}[S_i][0] \leftarrow \epsilon$
  $L_{\text{Reg}}[S_i][1] \leftarrow \{\epsilon\}$
endfor
$pp \leftarrow (mpk_{\text{ABE}}, L_{\text{Reg}})$
$mk \leftarrow msk_{\text{ABE}}$

2. FnInit, presented in Algorithm 2, simply outputs the public parameters for the VDC scheme and is therefore the same for all functions. This step is not particularly required in our construction as we make use of a public-key CP-ABE scheme but we retain the algorithm for consistency with prior definitions as well as for generality as other instantiations may require this step.

Algorithm 2 $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$

$pk_F \leftarrow pp$

3. Register, presented in Algorithm 3, registers a potential server's interest in offering computations. This is achieved by creating a key pair using the digital signature Sig.KeyGen algorithm. The resulting signing key sk_{Sig} is issued to S_i while the verification key vk_{Sig} is added to the first dimension of the public array L_{Reg} such that anyone can verify signatures produced by S_i.

Algorithm 3 $sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp)$

1 : $(sk_{\mathrm{Sig}}, vk_{\mathrm{Sig}}) \leftarrow_{\$} \mathsf{Sig.KeyGen}(1^{\lambda})$

2 : $sk_{S_i} \leftarrow sk_{\mathrm{Sig}}$

3 : $L_{\mathrm{Reg}}[S_i][0] \leftarrow L_{\mathrm{Reg}}[S_i][0] \cup vk_{\mathrm{Sig}}$

4. Certify, presented in Algorithm 4, certifies a server S_i for offering to execute computations of functions in the set $\mathcal{F}_i \subseteq \mathcal{F}$ on its data set D_i. The data set contains k_i data points $x_{i,j}$ and each element is uniquely expressed by an attribute $l(x_{i,j}) \in \mathcal{U}_l$. The algorithm first adds the pair $(F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$ to the array $L_{\mathrm{Reg}}[S_i][i]$ for each function $F \in \mathcal{F}_i$. This publicises to prospective clients what functions F the server S_i is willing to compute on any set of its data points $x_{i,j} \in D_i \subseteq \mathsf{Dom}(F)$. The algorithm outputs a CP-ABE decryption key sk_{ABE,D_i} for the data set D_i. This key is formed over the attribute set $(A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j}))$ and the evaluation key ek_{D_i,S_i} is set to be the CP-ABE decryption key.

Algorithm 4 $ek_{D_i,S_i} \xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{l(x_{i,j})\}_{x_{i,j} \in D_i}, \mathcal{F}_i, mk, pp)$

1 : **for** $F \in \mathcal{F}_i$

2 : $\quad L_{\mathrm{Reg}}[S_i][1] \leftarrow L_{\mathrm{Reg}}[S_i][1] \cup (F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$

3 : **endfor**

4 : $sk_{\mathrm{ABE},D_i} \leftarrow_{\$} \mathsf{ABE.KeyGen}((A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j})), msk_{\mathrm{ABE}}, mpk_{\mathrm{ABE}})$

5 : $ek_{D_i,S_i} \leftarrow sk_{\mathrm{ABE},D_i}$

5. ProbGen, presented in Algorithm 5, chooses two equal length messages m_0 and m_1 uniformly at random from the message space. For a computational request the algorithm needs to form two CP-ABE ciphertexts c_0 and c_1 that encrypt the chosen messages under the policies $\left(F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})\right)$ and $\left(\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})\right)$ respectively, and $X \subseteq D_i$. Those two ciphertexts form the encoded input $\sigma_{F,X}$. The verification key $vk_{F,X}$ is created by applying a one-way function g to each message and g allows the key to be published.

Algorithm 5 $(\sigma_{F,X}, vk_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(F, \{l(x_{i,j})\}_{x_{i,j} \in X}, pk_F, pp)$

1 : $(m_0, m_1) \leftarrow_{\$} \mathcal{M} \times \mathcal{M}$

2 : $c_0 \leftarrow_{\$} \mathsf{ABE.Encrypt}(m_0, (F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})), mpk_{\mathrm{ABE}})$

3 : $c_1 \leftarrow_{\$} \mathsf{ABE.Encrypt}(m_1, (\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})), mpk_{\mathrm{ABE}})$

4 : $\sigma_{F,X} \leftarrow (c_0, c_1)$

5 : $vk_{F,X} \leftarrow (g(m_0), g(m_1), L_{\mathrm{Reg}})$

6. Compute, presented in Algorithm 6, first attempts to decrypt both ciphertexts using the issued evaluation key ek_{D_i,S_i}. Decryption succeeds only if the function evaluates to 1 on the input data set X, i.e. the policy is satisfied. Since F and $\overline{F}$ output opposite results on the input data set, this ensures that exactly one plaintext will correspond to a failure symbol $\perp$. It then signs the resulting plaintexts using the server's signing key sk_{S_i}. Finally the algorithm outputs the encoded output containing the recovered plaintexts, the server's identity and its signature.

Algorithm 6 $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,X}, ek_{D_i,S_i}, sk_{S_i}, pp)$

1 : Parse $\sigma_{F,X}$ as (c_0, c_1)

2 : $d_0 \leftarrow \mathsf{ABE.Decrypt}\,(c_0, ek_{D_i,S_i}, mpk_{\mathrm{ABE}})$

3 : $d_1 \leftarrow \mathsf{ABE.Decrypt}\,(c_1, ek_{D_i,S_i}, mpk_{\mathrm{ABE}})$

4 : $\gamma \leftarrow_{\$} \mathsf{Sig.Sign}(d_0, d_1, S_i, sk_{S_i})$

5 : $\theta_{F(X)} \leftarrow (d_0, d_1, S_i, \gamma)$

7. Verify, presented in Algorithm 7, first parses the encoded output $\theta_{F(X)}$ as (d_0, d_1, S_i, γ) and the verification key $vk_{F,X}$ as $(g(m_0), g(m_1), L_{\mathrm{Reg}})$. The algorithm first checks whether the function F is listed in $L_{\mathrm{Reg}}[S_i][1]$, i.e. the respective server that generated the computational result is authorised to compute F. If this check fails, the result is immediately rejected.

   If the check was successful, the verifier continues with verifying the signature using the signature verification key vk_{Sig} stored in L_{Reg}. In case the signature is accepted, it applies the one-way function g to each plaintext in $\theta_{F(X)}$ and compares the results to the components in the verification key. If either comparison is successful, this indicates that the server has indeed recovered a message. Otherwise, it shows that the server provided a malformed response. Note, if m_0

was returned then $F(X) = 1$, and otherwise if m_1 was returned then $F(X) = 0$ following equation (4.1).

Algorithm 7 $y \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{F,X}, pp)$

```
1 : Parse θ_{F(X)} as (d_0, d_1, S_i, γ) and vk_{F,X} as (g(m_0), g(m_1), L_Reg)
2 : if F ∈ L_Reg[S_i][1] then
3 :   if accept ← Sig.Verify(d_0, d_1, S_i, γ, L_Reg[S_i][0])
4 :     if g(m_0) = g(d_0) return y ← 1
5 :     elseif g(m_1) = g(d_1) return y ← 0
6 :     else y ←⊥
7 :     endif
8 :   endif
9 : endif
10 : return y ←⊥
```

Theorem 4.4. *Given an* IND-CPA *secure CP-ABE scheme for a class of Boolean functions $\mathcal{F}$ closed under complement, a one-way function g, and a signature scheme secure against* EUF-CMA*. Let $\mathcal{VDC}$ be the verifiable delegable computation scheme as defined in Algorithms 1–7. Then $\mathcal{VDC}$ is secure in the sense of public verifiability (Figure 4.2).*

Informally, the proof relies on the IND-CPA property of the underlying CP-ABE encryption scheme and the one-wayness of g. It proceeds by showing that an adversary is not able to observe whether the plaintext has been altered for the unsatisfied function F or $\overline{F}$. Thus, the verification key can be the one-way function challenge $g(w)$ and we can set the plaintext implicitly to be w. A successful adversary finally returns w to break the one-wayness of g.

4.5 Proof of Security

In this section we present the full proof of Theorem 4.4. The proof partially follows the same principles as the proof of Lemma 3.12 in Chapter 3. The main proof differences occur by using CP-ABE here rather than KP-ABE.

Proof. Suppose $\mathcal{A}_{\mathrm{VDC}}$ is an adversary with non-negligible advantage against the public verifiability experiment (Figure 4.2) when instantiated with Algorithms 1–7. We begin by defining the following three games:

- **Game 0.** This is the public verifiability game as defined in Figure 4.2.

- **Game 1.** This is the same as **Game 0** with the modification that in ProbGen, we no longer return an encryption of m_0 and m_1. Instead, we choose another equal length random message $m' \neq m_0, m_1$ and, if $F(X^\star) = 1$, we replace c_1 by $\mathsf{ABE.Encrypt}(m', (\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})), mpk_{\text{ABE}})$. Otherwise, we replace c_0 by $\mathsf{ABE.Encrypt}(m', (F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})), mpk_{\text{ABE}})$. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.

- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game (Figure 2.8).

We show that an adversary with non-negligible advantage against the public verifiability game can be used to construct an adversary that may invert the one-way function g.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{\text{VDC}}$ can distinguish the two games with non-negligible advantage δ. We then construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{VDC}}$ as a sub-routine to break the IND-CPA security of the CP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-CPA game (Figure 2.4) with attribute universe $\mathcal{U}$ with $\mathcal{A}_{\text{ABE}}$, that in turn acts as a challenger in the public verifiability game for $\mathcal{A}_{\text{VDC}}$:

1. $\mathcal{C}$ runs the $\mathsf{ABE.Setup}$ algorithm on the security parameter and universe $\mathcal{U}$ to generate mpk_{ABE} and msk_{ABE}. The challenger gives mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.

2. $\mathcal{A}_{\text{ABE}}$ now simulates running $\mathsf{VDC.Setup}$ such that the outcome is consistent with mpk_{ABE}. It initialises the list L_{Reg} and sets $pp = (mpk_{\text{ABE}}, L_{Reg})$. The master key mk is implicitly set to msk_{ABE}.

3. $\mathcal{A}_{\text{VDC}}$ is provided with the public parameters pp and oracle access to the following functionalities, which are handled by $\mathcal{A}_{\text{ABE}}$.

   - $\mathsf{VDC.FnInit}(\cdot, mk, pp)$ and $\mathsf{VDC.Register}(\cdot, mk, pp)$ can be executed as specified in Algorithms 2 and 3.
   - $\mathsf{VDC.Certify}(\cdot, \cdot, \cdot, \cdot, mk, pp)$ can be run in order to generate the evaluation key ek_{D_i,S_i} for a queried data set D_i. To do so, $\mathcal{A}_{\text{ABE}}$ makes use of the KeyGen oracle $\mathcal{O}^{\mathsf{KeyGen}}$ in the CP-ABE game. First, it updates L_{Reg} according to the specified procedure in the first two lines of the $\mathsf{Certify}$ algorithm. $\mathcal{A}_{\text{ABE}}$ then sets the attribute data set $\widetilde{D}$ to be $(A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j}))$ and makes an oracle query to the challenger $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyGen}}(\widetilde{D}, mk, pp)$. The challenger generates a CP-ABE decryption key $sk_{\text{ABE},\widetilde{D}}$ if and only if $\widetilde{D} \notin \mathbb{A}^\star$. $\mathcal{C}$ may

generate the key ek_{D_i,S_i} since $\mathbb{A}^\star$ is still $\{\emptyset\}$, and provide $\mathcal{A}_{\text{ABE}}$ with the key.

4. $\mathcal{A}_{\text{VDC}}$ outputs the function F, its input data set $X^\star$ and unique labels $\{l(x_j)\}$ corresponding to the data points x_j in $X^\star$ as its challenge parameters.

5. $\mathcal{A}_{\text{ABE}}$ also runs $\mathsf{VDC.FnInit}$ as given in the construction, cf. Algorithm 2.

6. To generate the challenge input, $\mathcal{A}_{\text{ABE}}$ begins by choosing three random equal length messages m_0, m_1 and m' from the message space.

   Now $\mathcal{A}_{\text{ABE}}$ needs to choose its challenge access structure $\mathbb{A}^\star$ for the CP-ABE IND-CPA game. First, it computes $r = F(X^\star)$. If $r = 0$, $\mathcal{A}_{\text{ABE}}$ sets $\mathbb{A}^\star = (F \wedge \bigwedge_{x_j \in X^\star} l(x_j))$. Else it sets $\mathbb{A}^\star = (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j))$. Next it sends $\mathbb{A}^\star$ and the messages m_0 and m_1 to $\mathcal{C}$ as its challenge parameters for the CP-ABE game. We note that $\mathbb{A}^\star$ is a valid challenge access structure as the only queries made to the KeyGen oracle $\mathcal{O}^{\mathsf{KeyGen}}$ of the CP-ABE IND-CPA game were initiated by queries to the $\mathsf{Certify}$ oracle $\mathcal{O}^{\mathsf{Certify}}$ handled by $\mathcal{A}_{\text{ABE}}$. Note that due to the unique labels $\{l(x_j)\}_{x_j \in X^\star}$ present in the access structure, no requests to the $\mathsf{Certify}$ oracle for input data points in $X' \subset X^\star$ would result in a KeyGen query for attributes that satisfy $\mathbb{A}^\star$. If the oracle is queried for $X' \supseteq X^\star$ then we can observe that $\mathbb{A}^\star$ was chosen specifically such that it is unsatisfied on this input. Thus, KeyGen is never queried for an attribute set that satisfies $\mathbb{A}^\star$, and therefore the challenge is valid.

   $\mathcal{C}$ chooses a random bit b and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_b, \mathbb{A}^\star, mpk_{\text{ABE}})$. $\mathcal{A}_{\text{ABE}}$ samples a random bit t which intuitively corresponds to its guess for the the challenger's choice of b.

   - If $r = 1$ (that is, $\mathbb{A}^\star = (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j))$), $\mathcal{A}_{\text{ABE}}$ generates

     $$c \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', (F \wedge \bigwedge_{x_j \in X^\star} l(x_j)), mpk_{\text{ABE}})$$

     and sets $c' = ct^\star$ (formed over $\mathbb{A}^\star$ by $\mathcal{C}$). It also sets $vk = g(m')$ and $vk' = g(m_t)$.
   - Else $r = 0$, and $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and computes

     $$c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j)), mpk_{\text{ABE}}).$$

     It sets $vk = g(m_t)$ and $vk' = g(m')$.

   Finally, $\mathcal{A}_{\text{ABE}}$ sets $\sigma_{F,X^\star} = (c, c')$ and $vk_{F,X^\star} = (vk, vk', L_{\text{Reg}})$.

7. $\mathcal{A}_{\text{ABE}}$ sends the output from $\mathsf{ProbGen}$ along with the public information to $\mathcal{A}_{\text{VDC}}$ that is also given oracle access to which $\mathcal{A}_{\text{ABE}}$ responds as follows.

- $\mathsf{VDC.FnInit}(\cdot, mk, pp)$ and $\mathsf{VDC.Register}(\cdot, mk, pp)$ can be executed as detailed in the construction, cf. Algorithm 2 and Algorithm 3.
- $\mathsf{VDC.Certify}(\cdot, \cdot, \cdot, \cdot, mk, pp)$: To generate the evaluation key for the queried attribute set D_i, $\mathcal{A}_{\mathrm{ABE}}$ follows the same procedure as specified in step 3. However, we note that by the definition of the access structure $\mathbb{A}^\star$, $\widetilde{D}$ satisfies $\mathbb{A}^\star$ only if $X^\star \subseteq D_i$. This follows from the uniqueness of the labels within the whole system as it holds that $\{l(x_j)\}_{x_j \in X^\star} \subseteq \{l(x_{i,j})\}_{x_{i,j} \in D_i}$. Additionally it follows that D_i must satisfy either F or $\overline{F}$ as chosen in $\mathbb{A}^\star$. However, this was chosen specifically such that $X^\star$ (and therefore D_i, as F will simply select the elements of $X^\star$ to evaluate on) does not satisfy the function, and therefore $\widetilde{D} \notin \mathbb{A}^\star$ and $\mathcal{C}$ may generate the key, which $\mathcal{A}_{\mathrm{ABE}}$ will receive as ek_{D_i,S_i}.

8. Eventually, $\mathcal{A}_{\mathrm{VDC}}$ outputs $\theta_{F(X^\star)}$ which it believes is a valid forgery, i.e. that the output will be accepted yet does not correspond to the correct value of $F(X^\star)$.

9. $\mathcal{A}_{\mathrm{ABE}}$ parses $\theta_{F(X^\star)}$ as (d, d', S_i, γ). One of d and d' will be $\perp$ (by construction) and we denote the other value (non-$\perp$) by Y. Observe that, since $\mathcal{A}_{\mathrm{VDC}}$ is assumed to be a successful adversary against public verifiability, the non-$\perp$ value, Y, that it will return will be the plaintext m_t since the challenge access structure was always set to be unsatisfied on the challenge input. Thus, if $g(Y) = g(m_t)$, $\mathcal{A}_{\mathrm{ABE}}$ outputs a guess $b' = t$ and otherwise guesses $b' = (1 - t)$.

If $t = b$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game 0** since the verification key comprises $g(m')$ where m' is the message a legitimate server could recover, and $g(m_b)$ where m_b is the other plaintext.
Alternatively, $t = 1 - b$ and the distribution of the above experiment is identical to **Game 1** since the verification key comprises the legitimate message and a random message m_{1-t} that is unrelated to the ciphertext.

Now, we consider the advantage of this constructed adversary $\mathcal{A}_{\mathrm{ABE}}$ playing the IND-CPA game for CP-ABE. Recall that by assumption, $\mathcal{A}_{\mathrm{VDC}}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1**, that is

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right| \geqslant \delta$$

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{\mathrm{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right]$ denotes the output of running $\mathcal{A}_{\mathrm{VDC}}$ in **Game i**.

The probability of $\mathcal{A}_{\text{ABE}}$ guessing b correctly is:

$$\begin{aligned}
\Pr[b' = b] &= \Pr[t = b]\Pr[b' = b | t = b] + \Pr[t \neq b]\Pr[b' = b | t \neq b] \\
&= \frac{1}{2}\Pr[g(Y) = g(m_t) | t = b] + \frac{1}{2}\Pr[g(Y) \neq g(m_t) | t \neq b] \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + \frac{1}{2}\left(1 - \Pr[g(Y) = g(m_t) | t \neq b]\right) \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right] \\
&\quad + \frac{1}{2}\left(1 - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right. \\
&\quad \left. - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{VDC}}}\left[\mathcal{VDC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\text{ABE}}} &\geqslant \left|\Pr[b = b'] - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&= \frac{\delta}{2}.
\end{aligned}$$

Therefore, if $\mathcal{A}_{\text{VDC}}$ has advantage δ at distinguishing these games then $\mathcal{A}_{\text{ABE}}$ can win the IND-CPA game for CP-ABE with non-negligible probability. Thus since we assumed the CP-ABE scheme to be secure, we conclude that $\mathcal{A}_{\text{VDC}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is simply to set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game. We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{\text{VDC}}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{\text{ABE}}$).

Final Proof. We now show that using $\mathcal{A}_{\text{VDC}}$ in **Game 2**, $\mathcal{A}_{\text{ABE}}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ we can recover w. Specifically, during ProbGen, we choose the messages as follows:

- if $F(X^\star) = 1$, we implicitly set m_1 to be w and set the verification key component $vk' = z = g(w)$. We choose m_0 randomly from the message space and compute

the remainder of the verification key as usual.

- if $F(X^\star) = 0$, we implicitly set m_0 to be w and set the verification key component $vk = z = g(w)$. We choose m_1 randomly from the message space and compute the remainder of the verification key as usual.

Now, since $\mathcal{A}_{\text{VDC}}$ is assumed to be successful, it will output a forgery comprising the plaintext encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w and the adversary's view is consistent since the verification key is simulated correctly using z. $\mathcal{A}_{\text{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\text{VDC}}$ has against the public verifiability game.

We conclude that if the CP-ABE scheme is IND-CPA secure and the one-way function is hard-to-invert, then $\mathcal{VDC}$ as defined by Algorithms 1–7 is secure in the sense of public verifiability. □

4.6 Conclusion

In this chapter, we have introduced the notion of publicly verifiable delegable computation as a new mode of computation within the PVC setting. In particular, we have shown that ciphertext-policy attribute-based encryption can be used to achieve VDC in a similar fashion to how we used revocable KP-ABE to construct RPVC in Chapter 3. The obtained system model reverses the role of the client and server as the data owner compared to the previous mode. In RPVC, the data owner is the client who employs a server to evaluate a function on her data since she does not have the necessary resources herself. In VDC, on the other hand, the server acts as the data owner making the data available for specific queries, and the client can request a computation and verify the correctness of the result even if the client never possessed the data herself.

We have presented a rigorous framework and a provably secure construction within our security model. Furthermore, we discussed relevant natural applications of VDC and showed that the use of alternative attribute-based encryption primitives can be also used to provide a proof method.

In future work, we would like to compare the solution of Backes et al. [21] using SNARGs and homomorphic MACs as a proof method of correct computation to ours more closely and investigate whether ABE provides SNARG functionality. Another interesting direction would be to investigate methods to allow dynamic updates of the data owned by the servers as a result of computations. In our model, the cost of outsourcing a computation depends on the functions and thus it would be interesting to focus on reducing this cost.

Chapter 5

Hybrid Publicly Verifiable Outsourced Computation

Contents

In this chapter we present a scheme that unifies the introduced proposals of the previous chapters of this thesis capturing the modes of RPVC and VDC within a single instantiated system. Furthermore, we introduce another mode of publicly verifiable computation that enables us to enforce (graph-based) access control policies over the delegators, servers and verifiers. We make use of a novel variant of dual-policy attribute-based encryption to instantiate the unified umbrella scheme called hybrid publicly verifiable computation. The results of this chapter appear in [6].

5.1 Introduction

Throughout this thesis so far, we enriched the achievable functionalities for publicly verifiable outsourced computation schemes. In Chapter 3, we have extended the PVC construction of Parno et al. [118] to accommodate a revocation mechanism into the realm of PVC to revoke misbehaving servers from the system built on a revocable KP-ABE scheme [17]. In Chapter 4, we changed the setting and considered a *reversed* system model, compared to Chapter 3, wherein servers hold some data sets and make them available for public, verifiable querying. Technically, this was achieved by switching the attribute-based encryption scheme, which acts as a proof mechanism, from KP-ABE to CP-ABE. Thus, the VC schemes that arise can be seen as large, multi-user systems comprising many servers and delegators. However, in such systems, the individual user requirements may be diverse and thus require different forms of outsourced

computations.

Let us consider the following scenarios: (i) employees with limited resources (e.g. using mobile devices when out of the office) need to delegate computations to more powerful servers. The workload of the employee may also involve responding to computation requests to perform tasks for other employees or to respond to inter-departmental queries over restricted databases; (ii) entities that invest heavily in outsourced computations could find themselves with a valuable, processed data set that is of interest to other parties, and hence want to selectively share this information by allowing others to query the data set in a verifiable fashion; (iii) database servers that allow public queries may become overwhelmed with requests, and need to enlist additional servers to help (essentially the server acts as a delegator to outsource queries with relevant data). Finally, (iv) consider a form of peer-to-peer network for sharing computational resources – as individual resource availability varies, entities can sell spare resources to perform computations for other users or make their own data available to others, whilst making computation requests to other entities when resources run low.

Current PVC solutions (including the ones introduced in the previous chapters) do not handle these *flexible* requirements particularly well; although there are several different proposals in the literature that realise some of the requirements described above, each requires an independent (potentially expensive) setup stage. Thus, in this chapter, we unify these previous notions under an umbrella framework which we call *hybrid publicly verifiable computation* (HPVC) which is a single mechanism with the associated costs of a single setup operation and a single set of system parameters to be published and maintained that simultaneously satisfies all of the above requirements. In other words, we provide a system that is configured to support the modes RPVC and VDC depending on the individual user requirements. We show that yet another form of attribute-based encryption, namely dual-policy attribute-based encryption (DP-ABE) [19] (cf. Section 2.3.4), can be used to instantiate our HPVC system. DP-ABE has not attracted much attention in the literature, which we believe is mainly due to applications for this primitive being less obvious than for the single-policy schemes.

We also introduce another mode of computation, namely *RPVC with access control*. Motivated in the above setting of multi-user VC systems, we may wish to (i) restrict the computations that may be outsourced by delegators; and (ii) restrict the computations a server may perform as it is generally unlikely that all participating entities within the system have uncontrolled access to all functionalities. We show that by using the full power of DP-ABE, i.e. using both forms of access policies simultaneously, we can also enforce access control policies over the computations a server may perform in our HPVC system.

Related Work

In independent and concurrent work, Shi et al. [129] considered a similar use of DP-ABE to combine keyword search on encrypted data with enforcing access control policies. Two more notable works that have considered access control related to the VC setting are the following. Clear and McGoldrick [55] considered access control policies over delegators only and in a non-verifiable, multi-input outsourced computation setting using homomorphic ciphertext-policy ABE and fully homomorphic encryption. Xu and Tang [143] also addressed the necessity for access control in the setting of verifiable computation, but limited their scope to non-public verifiable computation (i.e. not the full multi-user setting) enforcing access control on delegators only.

Organisation of Chapter

In Section 5.2, we describe and provide a formal definition of our hybrid publicly verifiable computation model and we discuss the possible different modes of computation the scheme supports. Next, in Section 5.3, we define the main security notions for our model. In order to implement the revocation functionality, we first develop a new form of DP-ABE including the ability to revoke misbehaving entities and prevent them to decrypt further ciphertexts. Thus, in Section 5.4, we combine the revocation techniques from the indirectly revocable KP-ABE scheme (cf. Section 2.3.2) we used in Chapter 3 with the DP-ABE scheme [19] as summarised in Section 2.3.4 to define and construct a new primitive called *revocable key DP-ABE* scheme. In the same section, we provide all relevant details including a definition, security model, as well as a concrete construction and full proof of security. In Section 5.5, we use the recently constructed primitive in a black-box manner and provide an example construction of HPVC. This is followed, in Section 5.6, by showing that our given construction is secure according to the introduced security models. We conclude this chapter in Section 5.7.

5.2 Hybrid Publicly Verifiable Outsourced Computation

In this section, we define our umbrella framework called *hybrid publicly verifiable outsourced computation* (HPVC). This is a single system that supports multiple modes of computation with the associated cost of a single setup operation and system parameters. We follow the previous chapters and enable a single KDC to initialise an HPVC system that provides different functionalities for many clients with diverse requirements. The presented unified (umbrella) construction, as detailed in Section 5.5, is based on a novel use of DP-ABE (cf. Section 2.3.4). Given that DP-ABE conjunctively combines KP-ABE and CP-ABE, we observe that by using special ("dummy") attribute tokens, we can also only implement KP-ABE and CP-ABE if necessary. Entities within an HPVC scheme may play the role of both delegators and servers dynamically as required depending on the activated mode of computation. In more detail, our HPVC

scheme captures the following modes of computations:

- **Revocable PVC:** in this setting, clients with limited resources outsource computations on data of their choice to more powerful, untrusted servers using only public information. This framework may accommodate multiple servers simultaneously within the system offering computations-as-a-service and they are able to compute multiple different functions. Servers may try to cheat to persuade verifiers of incorrect information or to avoid using their own resources. Misbehaving servers can be detected and revoked so that further results will be rejected and they will not be rewarded for their effort. This model was introduced in Chapter 3.

- **Publicly Verifiable Delegable Computation:** in this setting, servers are the data owners and make a static data set available to clients for verifiable querying. Clients request computations on subsets of the data set using public, descriptive labels and are able to efficiently verify the results even if the clients never owned the data themselves. This model was introduced in Chapter 4.

- **RPVC with Access Control:** in this setting, we wish to restrict the set of servers that may perform a particular computation. The set of servers that may evaluate a computation for a client can therefore be restricted based on factors such as sensitivity of input data or physical server location.

The last framework of RPVC with access control has not yet been formally introduced within this thesis. We believe that such a model is highly beneficial in many situations to limit the sets of entities that can view input data and results. Additionally, it is also highly unlikely that all clients within a system (or an organisation) have identical and uncontrolled access to all functionalities. A more detailed discussion about the individual modes, and in particular the third mode of computation, is provided in Section 5.2.3.

Note that such an HPVC scheme provides us with a flexible solution handling diverse client requirements within a large system where individual workflows may require different forms of outsourced computation. Thus, it is possible within the HPVC framework that entities may play the role of both delegators and servers as required. For example, let us consider a large company with many different departments that have contracted cloud service providers offering computation-as-a-service. Employees can now use those providers to outsource large computations but they may themselves need to provide results from inter-departmental queries on local databases. Thus, an employee who basically acts as a server in the VDC mode and is overwhelmed with local queries can then change to the RPVC mode to contract a more powerful server to perform the computation on their behalf. Such diverse requirements can be easily handled within our HPVC system.

5.2.1 Informal Overview

An HPVC system for a family of functions $\mathcal{F}$ begins (as before) with the key distribution centre (KDC) initialising the system by producing public parameters and a master secret key. For each function of interest F, the KDC derives the appropriate delegation key pk_F. Then the KDC registers entities S_i that wish to act as a server within the system by providing each with an individual private signing key sk_{S_i}.

In the next step, the server receives an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ that, depending on the mode the algorithm is run in, either enables the evaluation of the function $\mathbb{O}$ or the evaluation of input data ψ. Recall from the previous chapters that we aimed to provide the server with the possibility to evaluate multiple functions and wished to restrict specific input data to only being evaluated by specific functions. Thus, we have introduced a simple encoding trick by issuing specific attribute labels. Now, in the Certify algorithm where the server is issued with an evaluation key, the set of labels L_i the server chooses depends on the chosen mode. In more detail, the set of labels L_i in the modes RPVC and RPVC-AC uniquely represent the function F that a server is certified to evaluate, and in the mode VDC the set of labels uniquely represents data points contained in the data set D_i held by S_i. In either mode, the server also specifies a list of computations $\mathcal{F}_i$ that it is willing to compute.

A delegator now runs the ProbGen algorithm where depending on the mode the respective inputs slightly differ. The delegator provides a set of labels $L_{F,X} \subseteq L_i$ where, as in the previous algorithm, those labels represent in the modes RPVC and RPVC-AC the function F that should be computed on the provided input, and in the mode VDC the labels represent the data points $X \subseteq D_i$ ($X \subseteq \mathsf{Dom}(F)$) that should be computed on. The algorithm finally outputs an encoded input $\sigma_{(\omega,\mathbb{S})}$ and the respective verification key $vk_{(\omega,\mathbb{S})}$.

In the Compute algorithm, a server may use the encoded input and its evaluation key to compute an encoded output corresponding to the computational result $F(X)$. The verification algorithm Verify inputs the encoded output, verification key and public parameters and outputs a value y and a token $\tau_{\theta_{F(X)}}$. The token indicates whether the result is correct and adds the server identity. If verification failed, the token is sent to the KDC which then revokes the server to prevent it from performing further computations within the system and hence incurs a penalty. Otherwise, the returned value y is accepted as the correct result.

5.2.2 Formal Definition

We now present a formal definition of all necessary algorithms for an HPVC scheme.

Definition 5.1. *An* hybrid publicly verifiable outsourced computation (HPVC) scheme *for a family of functions $\mathcal{F}$ comprises the following algorithms:*

1. $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$: *this randomised algorithm is run by the KDC to establish public parameters pp and a master secret key mk for the system. The inputs are the security parameter λ, and the family of functions $\mathcal{F}$ that may be computed;*

2. $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$*: this randomised algorithm is run by the KDC to generate a public delegation key, pk_F, allowing entities to outsource, or request, computations of F;*

3. $sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp)$*: this randomised algorithm is run by the KDC to enrol an entity S_i within the system to act as a server. It generates a personalised signing key sk_{S_i};*

4. $ek_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$*: this randomised algorithm is run by the KDC to generate an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ enabling the entity S_i to compute on the pair $(\mathbb{O}, \psi)$. The algorithm also takes as input the mode in which it should operate, a set of labels L_i, a set of functions $\mathcal{F}_i$, the master secret key as well as the public parameters;*

5. $(\sigma_{(\omega,\mathbb{S})}, vk_{(\omega,\mathbb{S})}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, pk_F, pp)$*: this randomised algorithm is run by an entity to request a computation of $F(X)$ from S_i. The inputs are the mode, the pair $(\omega, \mathbb{S})$ representing the computation request, a set of labels $L_{F,X} \subseteq L_i$, the delegation key for F and the public parameters. The algorithm outputs an encoded input $\sigma_{(\omega,\mathbb{S})}$ and a verification key $vk_{(\omega,\mathbb{S})}$;*

6. $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, ek_{(\mathbb{O},\psi),S_i}, sk_{S_i}, pp)$*: this randomised algorithm is run by an entity S_i to compute $F(X)$. The inputs are the mode, an encoded input $\sigma_{(\omega,\mathbb{S})}$, an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ and a signing key for S_i. The algorithm outputs an encoded output $\theta_{F(X)}$ representing $F(X)$;*

7. $(y, \tau_{\theta_{F(X)}}) \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{(\omega,\mathbb{S})}, pp)$*: this algorithm is run by any entity that wants to verify whether the result was computed correctly or not. The inputs are the encoded output $\theta_{F(X)}$, the verification key $vk_{(\omega,\mathbb{S})}$ and the public parameters. The algorithm outputs the actual result y. If the result y corresponds to $F(x)$ it additionally creates a token $\tau_{\theta_{F(x)}} = (\mathtt{accept}, S_i)$ indicating that the result was correctly computed. Otherwise, the result y corresponds to $\perp$ and it creates a token $\tau_{\theta_{F(x)}} = (\mathtt{reject}, S_i)$ indicating that the result is malformed and S_i misbehaved;*

8. $um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(X)}}, mk, pp)$*: this randomised algorithm is run by the KDC inputting the token from the verification process, the master secret key and public parameters. If $\tau_{\theta_{F(X)}} = (\mathtt{reject}, S_i)$, the algorithm revokes all evaluation keys*

$ek_{(\cdot,\cdot),S_i}$ of the server S_i by rendering them non-functional and thereby preventing S_i from performing any further evaluations within the current system. The update material um consists of a set of updated evaluation keys $\{ek_{(\mathbb{O},\psi),S'}\}$ which are issued to all servers. Otherwise, in case $\tau_{\theta_{F(X)}} = (\texttt{accept}, S_i)$ then the algorithm outputs $\perp$ indicating that no update was necessary.

Although not explicitly stated, the KDC may update the public parameters *pp* during any algorithm in order to address any changes in the entity population.

We say that an HPVC scheme is *correct* if the verifying party almost certainly accepts the returned result generated by a non-revoked server using a valid generated verification key and encoded output given the non-revoked server used a valid generated encoded input and evaluation key in each of the respective modes. More formally this can be represented as follows.

Definition 5.2. *An hybrid publicly verifiable outsourced computation (HPVC) scheme is* correct *for a family of functions $\mathcal{F}$ if, for all attribute sets ω and ψ, for all access structures $\mathbb{O}$ and $\mathbb{S}$ defined for a computation $F(X)$, for $F \in \mathcal{F}$ and $X \in \mathsf{Dom}(F)$, if $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$, and for all non-revoked servers S_i, then the following holds:*

$$
\begin{aligned}
&\Pr[(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F}),\\
&\quad pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp),\\
&\quad sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp),\\
&\quad ek_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp),\\
&\quad (\sigma_{(\omega,\mathbb{S})}, vk_{(\omega,\mathbb{S})}) \xleftarrow{\$} \mathsf{ProbGen}(\texttt{mode}, (\omega, \mathbb{S}), L_{F,X}, pk_F, pp),\\
&\quad \theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\texttt{mode}, \sigma_{(\omega,\mathbb{S})}, ek_{(\mathbb{O},\psi),S_i}, sk_{S_i}, pp),\\
&\quad y \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{(\omega,\mathbb{S})}, pp),\\
&\quad um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(X)}}, mk, pp)]\\
&\quad = 1 - \mathrm{negl}(\lambda).
\end{aligned}
$$

5.2.3 Modes of Computation

In this section, we discuss in more detail the requirements and approaches needed to enable a specific mode of computation. As mentioned before, our HPVC scheme is generically defined in terms of objective and subjective policies ($\mathbb{O}$ and $\mathbb{S}$ respectively) as well as objective and subjective attribute sets (ω and ψ respectively). The value of these parameters determine in which mode the algorithms are executed. In Table 5.1, we specify these parameters and relate them to their respective mode. We define two additional parameters T_{O} and T_{S} which are required in case we run the HPVC system in a specific mode. For the moment it suffices to interpret them as "dummy" objects

mode	$\mathbb{O}$	ψ	ω	$\mathbb{S}$
RPVC	F	$\{T_{\mathrm{S}}\}$	X	$\{\{T_{\mathrm{S}}\}\}$
VDC	$\{\{T_{\mathrm{O}}\}\}$	D_i	$\{T_{\mathrm{O}}\}$	F
RPVC-AC	F	s	X	P

mode	L_i	$L_{F,X}$	$\mathcal{F}_i$
RPVC	$\{l(F)\}$	$\{l(F)\}$	$\{F\}$
VDC	$\{l(x_{i,j})\}_{x_{i,j}\in D_i}$	$\{l(x_{i,j})\}_{x_{i,j}\in X}$	$\{(F, \{l(x_{i,j})\}_{x_{i,j}\in \mathsf{Dom}(F)})\}_{F\in\mathcal{F}}$
RPVC-AC	$\{l(F)\}$	$\{l(F)\}$	$\{F\}$

Table 5.1: Parameter definitions for different modes of computation

which are able to satisfy $\{T_{\mathrm{O}}\} \in \{\{T_{\mathrm{O}}\}\}$. With this interpretation it is possible that we set the objective policy $\mathbb{O} = \{\{T_{\mathrm{O}}\}\}$ which is always trivially satisfied by the objective attribute set $\omega = \{T_{\mathrm{O}}\}$ as above. This similarly holds for T_{S}. We use these parameters in order to "disable" either the objective or subjective part within our HPVC system to run the system in one of the possible modes. More details about these parameters can be found in Section 5.5.

5.2.3.1 RPVC

In order to run the system in the RPVC mode (as introduced in Chapter 3), we only require to input the objective policy $\mathbb{O}$ and the objective attribute set ω. Therefore, the objective policy is set to be the function F and the objective attribute set corresponds to the input X to outsource the computation of $F(X)$. Note that the input set X only contains a single input data point x which is chosen specifically per computation. Since we wish here to run the system in the RPVC mode, the unneeded subjective parameters $\mathbb{S}$ and ψ correspond to the dummy parameter T_{S} such that $\mathbb{S}$ is trivially satisfied by ψ. Furthermore, the set of functions $\mathcal{F}_i$ chosen in Certify corresponds in this mode simply to the function F, and the sets of labels L_i and $L_{F,X}$ both just comprise a single element labelling F. In more detail, this label $l(F)$ corresponds to the label derived from the bijective mapping $\Lambda\colon \mathcal{F} \to \mathcal{U}_{\mathcal{F}}$ defined in Section 3.4.1.2.

5.2.3.2 VDC

In order to run the system in the VDC mode (as introduced in Chapter 4), we only require to input the subjective policy $\mathbb{S}$ and the subjective attribute set ψ. Here, the subjective policy $\mathbb{S}$ represents the function F whilst the subjective attribute set ψ corresponds to the data set D_i held by server S_i comprising k_i data points. Similarly to the previous mode, the unneeded objective parameters $\mathbb{O}$ and ω are set to correspond to the dummy parameter T_{O} such that $\mathbb{O}$ is trivially satisfied by ω. In this mode, the set of functions $\mathcal{F}_i$ chosen in Certify corresponds to the set of functions a server is willing to let a client query on its data set along with the labels of each permissible input for the function. The set L_i contains labels that label each data point $x_{i,j} \in D_i$

held by the server, whilst the set $L_{F,X}$ specifically contains labels of data points for a particular computation.

5.2.3.3 RPVC-AC

Before explicitly detailing the requirements to run the system in the RPVC-AC mode, let us first motivate the necessity for an access control framework in the realm of publicly verifiable outsourced computation.

In the previous chapters of this thesis, we have extended the system model to accommodate a large pool of servers and delegators. As with any multi-client setting, we may wish to control access to resources in a verifiable computation system. In [5], we show that not only is this setting of multi-user VC well-suited to the cryptographic enforcement of access control policies, but that such policies fulfil a natural and vital role in protecting outsourced computations. Specifically in the setting of multi-user VC, we may wish on one hand to restrict the computations that may be outsourced by delegators, and on the other hand to restrict the computations a server may perform. The first need stems from separation of duties and the observation that, within an organisation, it is extremely unlikely that all users have equal, uncontrolled access to all functionalities. We may restrict the set of delegators that may outsource a computation to those that are authorised to compute it (if given sufficient resources) by the organisation's policies. The second requirement arises from the fact delegators may not authenticate servers beforehand and have less control over which servers may operate on their data. The sensitivity of the data or other requirements, such as the physical location or resources of the server, may limit the servers that should be permitted to perform the computation.
Another motivation for access control in the VC setting is that computational services may be charged for (e.g. in subscription-based utility computing [85, 122]) and that service providers may offer different levels of services to different clients (e.g. different levels may provide access to different functions or computational resources). We must ensure that only valid subscribers may access each tier of service.

In many multi-user settings for access control to stored data [32, 125], servers enforce access control policies by authenticating users and granting or denying access based on *access control lists* or *capability lists*. This approach is not appropriate in the multi-user VC setting since the servers are assumed to be untrusted and may have a huge interest in violating the policies. We instead use a cryptographic enforcement mechanism for access control policies where cryptographic keys are used to protect objects and restrict access, thus here the access control mechanism reduces to the appropriate distribution of keys to authorised entities. In [5], we use the trusted key distribution centre (KDC) as introduced in Chapter 3 for our RPVC model and extend its duties to instantiate

the access control mechanism, i.e. we additionally require it to issue keys appropriate to the access control policy that enable read and write access to certain components of the system. For example, input data for particular functions may be protected such that only authorised servers may read the data and hence perform the computation.

Note that in our RPVC scheme in Chapter 3, the KDC already implicitly provides some access control in the sense that servers are certified to perform specific functions through the generation of evaluation keys. However, no access control is applied to delegators – *any* entity can outsource an evaluation of *any* function for which the KDC has published delegation information (essentially due to the use of asymmetric cryptographic primitives). Cryptographic enforcement mechanisms are particularly appropriate when the objects and policies are relatively static such that additional keys need not be generated and objects need not be re-encrypted as those are rare events. In the context of VC, we may assume that the set of functions that may be evaluated is fixed (a given VC construction can implement a specified family of functions) and that the input data to each function is also static (limited to the set of 'valid' inputs to that function). Thus, the set of objects (function evaluations in VC) is static, and policies will primarily be specified in terms of these computations. Thus multi-user VC is a very natural setting in which to use cryptographic access control.

We consider in [5] graph-based policies where "objects" to be protected are not data files, as in traditional access control policies, but outsourced computations and their results. Here the underlying entity population comprises the sets of delegators $\mathcal{C}$, computational servers $\mathcal{S}$, and verifiers $\mathcal{V}$. In detail, we discuss and specify concrete policies that restrict (i) which functions a delegator may delegate, (ii) which computations a server may evaluate, and (iii) which outputs a verifier may read. In our context, to enforce policies restricting the computations that may be delegated to a server, a delegator must use an appropriate key to encrypt the input data. Without the correct encryption, the input will be just discarded by the server. The enforcement of policies for performing computations is achieved by distributing keys to servers that can be used to decrypt encrypted inputs. Without decryption, the server will be unable to read the input data and evaluate the function. The enforcement of (read) policies on outputs uses cryptographic access control in a more conventional fashion, i.e. results are published and protected via encryption with an appropriate key.

In order to realise the policies, we require a security labelling function. We define a security labelling function in terms of a mapping $\lambda\colon \mathcal{C} \cup \mathcal{S} \cup \mathcal{V} \cup \mathcal{O} \to L$ where $\mathcal{C}, \mathcal{S}$ and $\mathcal{V}$ are the respective sets from the entity population, $\mathcal{O}$ is the family of computations that may be outsourced and $(L, \leqslant)$ is a poset of security labels. Note that in this context we refer to computations as $o = (F, X, \mathtt{aux}) \in \mathcal{O}$ where each computation o specifies all

required information in order to formulate the access control policy, i.e. the function F to be computed, the input data X, and any other relevant contextual informations are denoted by $\mathtt{aux}$.[1] This security labelling function assigns a label from L, representing the security classification (clearance), to each delegator, server and computation in the system. The access control policy requires that $\lambda(E) \geq \lambda(o)$ for an entity $E \in \mathcal{C} \cup \mathcal{S} \cup \mathcal{V}$ attempting to evaluate a computation $o \in \mathcal{O}$. Note that different types of policy can be achieved by different choices for the sets $\mathcal{O}$ and L. In [5], we discuss in detail delegation and computation policies, i.e. policies over functions and policies over function inputs, as well as verification policies.

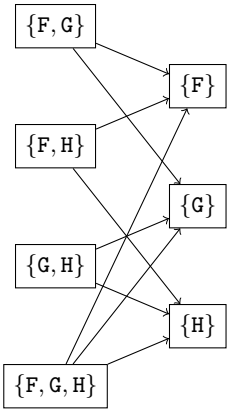

Figure 5.1: Example poset $\mathtt{L} = 2^{\{\mathtt{F},\mathtt{G},\mathtt{H}\}}$ for RPVC with access control

Let us provide here an example discussion about an access control policy over functions, i.e. we examine the case where policies are formulated purely in terms of the functions being computed. Thus, in this example objects correspond to functions $\mathcal{O} = \mathcal{F}$ and security labels are represented as sets of functions $L = 2^{\mathcal{F}}$. In simple terms, we associate each delegator C and server S with a set of functions $\lambda(C) \subseteq \mathcal{F}$ and $\lambda(S) \subseteq \mathcal{F}$ respectively. We define a *correctness criterion* that states that C should be able to prepare inputs for all functions $F \in \lambda(C)$ (and similarly for S), i.e. entities should be able to perform all operations that they are authorised for. The *security criterion* requires $\lambda(C), \lambda(S) \supseteq \lambda(o)$ in order to delegate or compute F respectively, and that a set of unauthorised entities cannot collude to perform an operation that any of them could not perform alone.

More formally, we define the set of security labels L to be $2^{\mathcal{F}}$ (the power set of all considered functions). Then, $\lambda(C) \subseteq \mathcal{F}$ defines the set of functions that a delegator C may outsource an evaluation of, $\lambda(S) \subseteq \mathcal{F}$ denotes the functions a server S may compute, and $\lambda(o) = \{F\}$ labels the computation of $F \in \mathcal{F}$. Then, for any $X, Y \in L$ we define an order relation $<$ such that $X < Y$ if and only if $X \in \mathcal{F}, Y \subseteq \mathcal{F}$ and $X \in Y$. The

[1] This notation for computations was chosen in reference to their role as protected objects in the access control system.

corresponding Hasse diagram with $\mathcal{F} = \{F, G, H\}$ is shown in Figure 5.1.[2] Any entity E authorised for $\lambda(E)$ is, by the correctness criterion, authorised to operate on all computations such that $\lambda(o) < \lambda(E)$. For example, in Figure 5.1, an entity authorised for the set $\lambda(E) = \{F, G\}$ is authorised for the functions F and G as expected. Each label $l \in L$ will be associated with a key κ_l. To outsource a computation o of $F(X)$, C prepares the encoding of X using the key $\kappa_o = \{F\}$ which, by the security criterion, C knows if and only if $\lambda(C) \geqslant \lambda(o)$, i.e. if and only if $\{F\} \in \lambda(C)$. To compute $F(X)$, S uses the corresponding key κ_o. As before, S may do this if and only if $\{F\} \in \lambda(S)$.

Furthermore, we presented in [5], a definition of a RPVC scheme with access control as well as relevant novel security notions for the scheme. Finally, we provided a concrete example instantiation based on the RPVC scheme as introduced in Chapter 3. In order to enforce the policies in the instantiation, we make use of a *key assignment scheme* (KAS) that assigns an appropriate key to each label. Thus, each entity is provided with a key corresponding to their respective label, and they may derive keys for all descendants.

Now we turn our attention to the case of running our HPVC scheme in the RPVC-AC mode. In contrast to the previous discussion on access control, we are able to slightly relax the access control framework within HPVC. The framework retains public verifiability and public delegability but we limit the use of access control policies to be restricted on the set of servers that may compute a given outsourced computation. Informally, we use the objective policy $\mathbb{O}$ to evaluate an outsourced computation as in the RPVC mode whilst we additionally use the subjective policy $\mathbb{S}$ to enforce access control on the server. Servers receive an evaluation key $ek_{(\mathbb{O},\psi),S}$ in which the objective policy corresponds to a function F and the subjective attributes correspond to a set of descriptive attributes describing their authorisation rights s, where $s \subseteq \mathcal{U}_{\text{auth}}$ and $\mathcal{U}_{\text{auth}}$ is an attribute universe that is solely used to authorisation. Using the above terminology of the security labelling function, a server receives the label $\lambda^C(S) = (F, s)$ where $\lambda^C(\cdot)$ denotes a computation policy.[3] The ProbGen algorithm for HPVC inputs the objective attributes which correspond here to the input data X and the subjective policy corresponds to an authorisation policy $P \subseteq 2^{\mathcal{U}_{\text{auth}}} \setminus \{\emptyset\}$ which specify the necessary required authorisation attributes to perform the computation. In terms of the security labelling function, the label for the computation o corresponds to $\lambda^C(o) = (X, P)$. A server may produce a valid output that will be accepted by the verification algorithm if and only if $s \in P$, i.e. the server satisfies the authorisation policy.

[2] Nodes for empty sets are excluded from the figure.

[3] In [5], we differentiate between computation policies over delegators and servers, denoted by $\lambda^C(\cdot)$, and verification policies over delegators only, denoted by $\lambda^V(\cdot)$.

5.3 Security Models

In this section we discuss the security notions we wish to achieve in our HPVC framework. In more detail, we can achieve security in the sense of *public verifiability*, *revocation* and *authorised computation*. Those notions follow the same motivations as discussed in the previous chapters. We require, as in Chapter 3, to include some additional restrictions on the games that are placed from our current rkDPABE primitive (which we introduce in Section 5.4) which acts as our main building block for our HPVC construction in this chapter. For brevity, we do not discuss further the respective ideal notions of security but it is straightforward to adapt those notions from Chapter 3 by accommodating some additional HPVC parameters.

5.3.1 Selective Public Verifiability

In Figure 5.2, we define a *selective notion of public verifiability*. This notion is a combination of the public verifiability notions introduced in Chapters 3 and 4, to formalise in the HPVC model that no server is able to return a malformed result for a computation without being detected.

$\mathbf{Exp}_{\mathcal{A}}^{\text{sPubVerif}}\left[\mathcal{HPVC}, 1^{\lambda}, \mathcal{F}\right]$

1 : $(\omega^{\star}, \mathbb{O}^{\star}, \psi^{\star}, \mathbb{S}^{\star}, L_{F,X^{\star}}, \texttt{mode}) \leftarrow\!\!\$\ \mathcal{A}(1^{\lambda}, \mathcal{F})$
2 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^{\lambda}, \mathcal{F})$
3 : **if** $\texttt{mode} = \text{VDC}$ **then** $(F \leftarrow \mathbb{S}^{\star}, X^{\star} \leftarrow \psi^{\star})$
4 : **else** $(F \leftarrow \mathbb{O}^{\star}, X^{\star} \leftarrow \omega^{\star})$
5 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
6 : $(\sigma^{\star}, vk^{\star}) \leftarrow\!\!\$\ \mathsf{ProbGen}(\texttt{mode}, (\omega^{\star}, \mathbb{S}^{\star}), L_{F,X^{\star}}, pk_F, pp)$
7 : $\theta^{\star} \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma^{\star}, vk^{\star}, pk_F, pp)$
8 : $(y, \tau_{\theta^{\star}}) \leftarrow \mathsf{Verify}(\theta^{\star}, vk^{\star}, pp)$
9 : **if** $(y, \tau_{\theta^{\star}}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F(X^{\star}))$ **then**
10 : **return** 1
11 : **else return** 0

Figure 5.2: The selective public verifiability experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{sPubVerif}}\left[\mathcal{HPVC}, 1^{\lambda}, \mathcal{F}\right]$

The game begins with the adversary first selecting its challenge parameters. Note that the adversary chooses the mode it wishes the challenge to be generated in and the respective labels necessary for this mode. Furthermore, the adversary outputs choices for $\omega^{\star}$, $\mathbb{O}^{\star}$, $\psi^{\star}$ and $\mathbb{S}^{\star}$, despite only $\omega^{\star}$ and $\mathbb{S}^{\star}$ are used to form the challenge input. This notation was used mainly for notational convenience to allow us to define the challenge

computation in terms of F and $X^\star$ in line 3 or 4 depending on the mode. However, we want to stress that this information can also be learnt from the set of labels $L_{F,X^\star}$ and the chosen mode of computation. Thus, this notational convenience does not weaken the game since the information has been already determined by the adversary's choices.

After the adversary has chosen the parameters, the game proceeds with the challenger running Setup to initialise the system and FnInit to return the public delegation key pk_F for the chosen challenge function. The challenger continues with running ProbGen on the challenge inputs to output a challenge for the adversary. The adversary receives the challenge and public information and is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm. Finally, the adversary wins the game if it is able to create an encoded output that verifies correctly but does not encode the correct value $F(X)$.

Definition 5.3. *The* advantage *of a PPT adversary in the* sPubVerif *game for an hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{sPubVerif}}_{\mathcal{A},\mathcal{HPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{sPubVerif}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$ is secure with respect to selective public verifiability *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sPubVerif}}_{\mathcal{A},\mathcal{HPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

5.3.2 Selective, Semi-static Revocation

The notion of *revocation* requires that, if a server is detected as misbehaving, meaning that a server S_i returns a result such that the verification algorithm Verify outputs $(\perp, (\texttt{reject}, S_i))$, then any subsequent computations by S_i should be rejected, even if the result may be correct.

In Figure 5.3, we define a *selective, semi-static notion of revocation* which is similarly defined to the notion in Section 3.3.2.2. As in the previous game, this notion starts with the adversary choosing its challenge parameters which the challenger can parse to determine F and $X^\star$. The challenger maintains a (initially empty) list Q_{Rev} of currently revoked entities as well as the current time period t which can be incremented during Revoke oracle queries. The game proceeds with the challenger running Setup to initialise the system and FnInit to return the public delegation key pk_F for the chosen challenge function. After this, on line 8, the adversary needs to declare (before receiving oracle

$\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}, q_t\right]$

1 : $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \mathtt{mode}) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda, \mathcal{F}, q_t)$
2 : **if** $\mathtt{mode} = \text{VDC}$ **then** $(F \leftarrow \mathbb{S}^\star, X^\star \leftarrow \psi^\star)$
3 : **else** $(F \leftarrow \mathbb{O}^\star, X^\star \leftarrow \omega^\star)$
4 : $Q_{\text{Rev}} \leftarrow \epsilon$
5 : $t \leftarrow 1$
6 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{F})$
7 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
8 : $\overline{R} \leftarrow \mathcal{A}(pk_F, pp)$
9 : $\mathcal{A}^{\mathcal{O}}(pk_F, pp)$
10 : **if** $(\overline{R} \not\subseteq Q_{\text{Rev}})$ **then return** 0
11 : $(\sigma^\star, vk^\star) \leftarrow\!\!\$\ \mathsf{ProbGen}(\mathtt{mode}, (\omega^\star, \mathbb{S}^\star), L_{F,X^\star}, pk_F, pp)$
12 : $\theta^\star \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma^\star, vk^\star, pk_F, pp)$
13 : **if** $((y, (\mathtt{accept}, S)) \leftarrow \mathsf{Verify}(\theta^\star, vk_{F,x^\star}, pp)$ **and** $(S \in \overline{R}))$ **then**
14 : **return** 1
15 : **else return** 0

$\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$

1 : **if** $(L_{F,X^\star} \subseteq L_i$ **and** $S_i \notin \overline{R})$ **or** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S_i)$ **then return** $\perp$
2 : $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S_i$
3 : **return** $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$

$\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(X)}}, mk, pp)$

1 : $t \leftarrow t + 1$
2 : **if** $(\tau_{\theta_{F'(x)}} = (\mathtt{accept}, \cdot))$ **then return** $\perp$
3 : **if** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\text{Rev}} \cup S_i)$ **then return** $\perp$
4 : $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \cup S_i$
5 : **return** $\mathsf{Revoke}(\tau_{\theta_{F'(X)}}, mk, pp)$

Figure 5.3: The selective, semi-static revocation experiment $\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}, q_t\right]$

access) a list $\overline{R}$ of servers to be revoked at the time period where the challenge will be generated which we assume will be at time period q_t. The adversary is then provided with oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. Certify and Revoke oracle queries are handled as specified in Figure 5.3. The adversary finishes its oracles query phase (line 9) after making a polynomial number of queries q, including q_t many Revoke queries,

and does not return a value other than signalling to the challenger that it may proceed with the remainder of the game. The challenger checks that all queries made by the adversary have indeed generated a list of currently revoked servers that is a superset of the challenge revocation list $\overline{R}$. If this is not true, the challenger aborts the game and the adversary loses as it was not able to choose its queries or the list $\overline{R}$ appropriately. Otherwise, the challenger continues with the game and generates the challenge by running ProbGen. The adversary wins the game if it outputs any result, i.e. a correct or malformed response, as a valid result from any server that was revoked at the time of the challenge.

Definition 5.4. *The* advantage *of a PPT adversary in the* sssRevoc *game for an hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{sssRevoc}}_{\mathcal{A},\mathcal{HPVC}}(1^\lambda, \mathcal{F}, q_t) = \Pr\left[\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}, q_t\right] \rightarrow 1\right].$$

We say that the hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$ is secure with respect to selective, semi-static revocation *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sssRevoc}}_{\mathcal{A},\mathcal{HPVC}}(1^\lambda, \mathcal{F}, q_t) \leq \text{negl}(\lambda).$$

5.3.3 Selective Authorised Computation

In Figure 5.4, we define a *selective notion of authorised computation*. This notion ensures that only a server that satisfies an additional authorisation policy in the encoded input should be able to perform a given computation on this encoded input. Thus, in contrast, a result generated by an unauthorised server should always be rejected even if the result is correct.

The game begins with explicitly setting the mode of computation to RPVC-AC and the adversary chooses the parameters for the game accordingly as otherwise the parameters would not be meaningful. Those parameters consist of a challenge function F, challenge input $X^\star$, the authorisation policy P and the respective function labels for this mode. The game proceeds with the challenger running Setup to initialise the system and FnInit to return the public delegation key pk_F for the chosen challenge function. It continues with the challenger running ProbGen to create the challenge for the adversary. The adversary receives the challenge and public information and is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm with the exception of Certify queries which are separately specified in Figure 5.4. The Certify oracle returns $\perp$ if the queried attribute set ψ satisfies the authorisation policy P, as otherwise the

$\mathbf{Exp}^{\text{sAuthComp}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right]$

1 : $\texttt{mode} = \text{RPVC-AC}$
2 : $(F, X^\star, P, \{l(F)\}) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda, \mathcal{F})$
3 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mathcal{F})$
4 : $pk_F \leftarrow\!\!\$\ \mathsf{FnInit}(F, mk, pp)$
5 : $(\sigma^\star, vk^\star) \leftarrow\!\!\$\ \mathsf{ProbGen}(\text{RPVC-AC}, (X^\star, P), \{l(F)\}, pk_F, pp)$
6 : $\theta^\star \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\sigma^\star, vk^\star, pk_F, pp)$
7 : $(y, \tau^\star) \leftarrow \mathsf{Verify}(\theta^\star, vk^\star, pp)$
8 : **if** $\tau^\star \neq (\texttt{reject}, \cdot)$ **then**
9 : **return** 1
10 : **else return** 0

$\mathcal{O}^{\mathsf{Certify}}(\text{RPVC-AC}, S_i, (F, \psi), \{l(F)\}, \mathcal{F}_i, mk, pp)$

1 : **if** $(\psi \in P)$ **then return** $\perp$
2 : **return** $\mathsf{Certify}(\text{RPVC-AC}, S_i, (F, \psi), \{l(F)\}, \mathcal{F}_i, mk, pp)$

Figure 5.4: The selective authorised computation experiment $\mathbf{Exp}^{\text{sAuthComp}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right]$

adversary would be able to trivially produce a valid response as an authorised entity. The adversary wins the game if it outputs a result and token that is accepted by a verifier.

Definition 5.5. *The* advantage *of a PPT adversary in the* sAuthComp *game for an hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$\mathbf{Adv}^{\text{sAuthComp}}_{\mathcal{A},\mathcal{HPVC}}(1^\lambda, \mathcal{F}) = \Pr\left[\mathbf{Exp}^{\text{sAuthComp}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \rightarrow 1\right].$$

We say that the hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$ is secure with respect to selective authorised computation *if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{sAuthComp}}_{\mathcal{A},\mathcal{HPVC}}(1^\lambda, \mathcal{F}) \leq \text{negl}(\lambda).$$

5.4 Revocable Dual-policy Attribute-based Encryption

Dual-policy attribute-based encryption, as described in Section 2.3.4, was introduced by Attrapadung and Imai [19] and conjunctively combines KP-ABE and CP-ABE such

that *both* the secret decryption key and the ciphertext comprise an access structure and an attribute set. The same authors [17] have also introduced the notion of revocation in ABE schemes (cf. Section 2.3.2) which we already have seen in Chapter 3 leads to a PVC scheme which can revoke misbehaving servers from the system. Recall that the notion of revocation supports two different modes, namely direct revocation and indirect revocation. The former notion enables a client to specify a revocation list at the point of encryption such that periodic re-keying is not necessary but the encryptors must have the knowledge of the specific (current) revocation list. On the other hand, indirect revocation requires a time period to be specified at the point of encryption and needs an authority that issues update key material at each time period in order to enable entities to update their key to stay functional during the time period. Throughout this thesis we have focused on the mode of indirect revocation, mainly as it minimises the client's workload as she is not required to maintain a synchronised revocation list. This mode is implemented in the KP-ABE setting by amending the policy including an entity identifier and by embedding the current time period into the ciphertext. Update keys are issued only to non-revoked entities at each time period. Note that only the combination of a secret key with an update key for a time period forms a functional evaluation key that is able to decrypt a ciphertext formed using the time period.

In this section, we aim to implement a revocation mechanism for a DP-ABE scheme. However, in this context, we are able to embed the revocation mechanism into the KP-ABE functionality *or* the CP-ABE functionality. Recall that decryption in DP-ABE is only successful if and only if *both* attribute sets satisfy their corresponding access structure. Thus, in order to prevent the decryption functionality to be successful, it suffices that at least one attribute set does not satisfy the corresponding access structure. Here we present a formal definition of a revocable DP-ABE scheme using indirect revocation in the key-policy.

5.4.1 Formal Definition

In the following we provide a formal definition of a *revocable key dual-policy attribute-based encryption scheme*. Recall from Section 2.3.4 that we refer to the access structure associated to a decryption key as an *objective* access structure, denoted as $\mathbb{O}$, and the attribute set associated with a ciphertext is referred to as an *objective* attribute set, denoted as ω. Both the objective access structure and attribute set are associated with the KP-ABE functionality. Similarly, we refer to the access structure associated with a ciphertext as a *subjective* access structure, denoted as $\mathbb{S}$, whereas we refer to the attribute set associated to a decryption key as a *subjective* attribute set, denoted as ψ. Thus, both the subjective access structure and attribute set are associated with the CP-ABE functionality.

An indirectly revocable key DP-ABE scheme defines the universe of attributes to be $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup T_{\text{O}} \cup T_{\text{S}}$. In more detail, $\mathcal{U}_{\text{attr}}$ is the "normal" attribute universe for describing ciphertexts and forming access policies, $\mathcal{U}_l$ contains a set of attributes (disjoint from $\mathcal{U}_{\text{attr}}$) that uniquely label each function and each data item, $\mathcal{U}_{\text{time}}$ comprises attributes for time periods, and $\mathcal{U}_{\text{ID}}$ contains attributes encoding entity identities. T_{O} and T_{S} are additional ("dummy") attributes that efficiently enable the DP-ABE scheme to either function as a KP-ABE scheme or CP-ABE scheme. In Section 5.5, we discuss in more detail how those dummy attributes influence the execution of the different modes of computations within our unified HPVC construction.

More formally, an indirectly revocable key DP-ABE scheme is presented in the following definition.

Definition 5.6. *A* revocable key dual-policy attribute-based encryption (rkDP-ABE) scheme *consists of the following algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm takes as input the security parameter and the universe of attributes* $\mathcal{U}$ *and outputs public parameters* pp *and master secret key* mk*;*
- $ct_{(\omega,\mathbb{S}),t} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, pp)$: *this randomised encryption algorithm inputs a message* m*, an objective attribute set* ω*, a subjective policy* $\mathbb{S}$*, the current time period* $t \in \mathcal{U}_{\text{time}}$ *and the public parameters* pp*. It outputs a ciphertext* $ct_{(\omega,\mathbb{S}),t}$ *that is valid for time* t*;*
- $sk_{\mathsf{id},(\mathbb{O},\psi)} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp)$: *this randomised key generation algorithm takes as input an identity* $\mathsf{id} \in \mathcal{U}_{\text{ID}}$*, an objective access structure* $\mathbb{O}$*, a subjective attribute set* ψ*, as well as the master secret key* mk *and public parameters* pp*. It outputs a secret decryption key* $sk_{\mathsf{id},(\mathbb{O},\psi)}$*;*
- $uk_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, mk, pp)$: *this randomised algorithm takes a revocation list* $R \subseteq \mathcal{U}_{\text{ID}}$ *containing the identities of revoked entities, the current time period* t*, as well as the master secret key* mk *and public parameters* pp*. It outputs updated key material* $uk_{R,t}$ *which makes the decryption keys* $sk_{\mathsf{id},(\mathbb{O},\psi)}$*, for all non-revoked identities* $\mathsf{id} \notin R$*, functional to decrypt ciphertexts encrypted for the time period* t*;*
- $pt \leftarrow \mathsf{Decrypt}(ct_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), sk_{\mathsf{id},(\mathbb{O},\psi)}, (\mathbb{O}, \psi), uk_{R,t}, pp)$: *this decryption algorithm takes as input a ciphertext* $ct_{(\omega,\mathbb{S}),t}$ *formed for the time period* t *and the associated pair* $(\omega, \mathbb{S})$*, a secret decryption key* $sk_{\mathsf{id},(\mathbb{O},\psi)}$ *for an entity* id *and the associated pair* $(\mathbb{O}, \psi)$*, an update key* $uk_{R,t}$ *for the current time period* t *and the public parameters* pp*. The algorithm outputs a plaintext* pt *which corresponds to the correct message* m*, if and only if the objective attributes* ω *satisfy the objective access structure* $\mathbb{O}$ and *the subjective attributes* ψ *satisfy the subjective access*

structure $\mathbb{S}$ and *the value of* t *in the update key matches the one specified during encryption. If not,* pt *outputs* $\perp$.

Correctness of a revocable key DP-ABE scheme is defined as follows.

Definition 5.7. *A revocable key DP-ABE scheme is* correct *if for all* $m \in \mathcal{M}$*, all* $\mathsf{id} \in \mathcal{U}_{\mathrm{ID}}$*, all* $R \subseteq \mathcal{U}_{\mathrm{ID}}$*, all access structures* $\mathbb{O}, \mathbb{S} \subseteq 2^{\mathcal{U}_{\mathrm{attr}}} \setminus \{\emptyset\}$*, all attribute sets* $\omega, \psi \subseteq \mathcal{U}_{\mathrm{attr}}$ *and all* $t \in \mathcal{U}_{\mathrm{time}}$*, if* $\omega \in \mathbb{O}$ *and* $\psi \in \mathbb{S}$ *and* $\mathsf{id} \notin R$*, it holds that*

$$\begin{aligned}
&\Pr[(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U}),\\
&\quad ct_{(\omega,\mathbb{S}),t} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, pp),\\
&\quad sk_{\mathsf{id},(\mathbb{O},\psi)} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp),\\
&\quad uk_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, mk, pp),\\
&\quad m \leftarrow \mathsf{Decrypt}(ct_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), sk_{\mathsf{id},(\mathbb{O},\psi)}, (\mathbb{O}, \psi), uk_{R,t}, pp)]\\
&\quad = 1 - \mathrm{negl}(\lambda).
\end{aligned}$$

5.4.2 Security Model

The security model for a rkDP-ABE scheme is a natural extension of the IND-sHRSS security notion for an indirectly revocable KP-ABE scheme (cf. Section 2.3.2) and the security notion is presented in Figure 5.5.

Definition 5.8. *The* advantage *of a PPT adversary in the* IND-sHRSS *game for a revocable key DP-ABE construction* $\mathcal{RKDPABE}$ *is defined as:*

$$\mathbf{Adv}^{\text{IND-sHRSS}}_{\mathcal{A},\mathcal{RKDPABE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{IND-sHRSS}}_{\mathcal{A}}\left[\mathcal{RKDPABE}, 1^\lambda, \mathcal{U}\right] \to 1\right] - \frac{1}{2}.$$

We say that the revocable key DP-ABE scheme is secure in the sense of indistinguishability against selective-target with semi-static query attack (IND-sHRSS) *if for all PPT adversaries* $\mathcal{A}$*, it holds that*

$$\mathbf{Adv}^{\text{IND-sHRSS}}_{\mathcal{A},\mathcal{RKDPABE}}(1^\lambda) \leq \mathrm{negl}(\lambda).$$

5.4.3 Construction of a rkDP-ABE scheme

Our revocable DP-ABE scheme will be based on a combination of DP-ABE [19], which itself is a combination of CP-ABE [142] and KP-ABE [89], and an ABE scheme supporting revocation [17]. We represent a subjective access structure $\mathbb{S}$ by a linear secret sharing scheme (LSSS) which we denote by (M, ρ) and represent an objective access structure $\mathbb{O}$ as a LSSS denoted by (N, π).

Let $\mathcal{U}_{\mathrm{s}}$ and $\mathcal{U}_{\mathrm{o}}$ be the universe of subjective and objective attributes respectively. The objective attribute universe comprises disjoint sub-universes $\mathcal{N}, \mathcal{T}, \mathcal{M}$ and $\mathcal{U}_{\mathrm{ID}}$ referring

$\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{RKDPABE}, 1^{\lambda}, \mathcal{U}\right]$

$(t^{\star}, (\omega^{\star}, \mathbb{S}^{\star})) \leftarrow_{\$} \mathcal{A}(1^{\lambda}, \mathcal{U})$
$(pp, mk) \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, \mathcal{U})$
$\overline{R} \leftarrow_{\$} \mathcal{A}(pp)$
$(m_0, m_1) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,mk,pp)^{\mathsf{KeyUpdate}}(\cdot,\cdot,mk,pp)}(\overline{R}, pp)$
if $(|m_0| \neq |m_1|)$ **then return** 0
$b \leftarrow_{\$} \{0, 1\}$
$ct^{\star} \leftarrow_{\$} \mathsf{Encrypt}(m_b, t^{\star}, (\omega^{\star}, \mathbb{S}^{\star}), pp)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,(\cdot,\cdot)mk,pp),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,mk,pp)}(ct^{\star}, \overline{R}, pp)$
if $b' = b$ **then**
  return 1
else return 0

$\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp)$

if $((\omega^{\star} \in \mathbb{O})$ **and** $(\psi \in \mathbb{S}^{\star})$ **and** $(\mathsf{id} \notin \overline{R}))$ **then**
  return $\perp$
else $\mathsf{KeyGen}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp)$
  return $sk_{\mathsf{id},(\mathbb{O},\psi)}$

$\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, mk, pp)$

if $(t = t^{\star})$ **and** $(\overline{R} \not\subseteq R)$ **then**
  return $\perp$
else $\mathsf{KeyUpdate}(R, t, mk, pp)$
  return $uk_{R,t}$

Figure 5.5: The IND-sHRSS experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{RKDPABE}, 1^{\lambda}, \mathcal{U}\right]$

to standard ABE attributes, time periods, messages and entity identities respectively. $\mathcal{U}_{\text{ID}}$ is set to be the set of leaves in a complete binary tree $\mathcal{X} = \{1, \ldots, n\}$. Without loss of generality, we assume that $\mathcal{T} \cap \mathcal{X} = \emptyset$ (e.g. by using a collision resistant hash function and using distinct prefixes to map elements from $\mathcal{T}$ and $\mathcal{X}$). The attribute set for the rkDP-ABE scheme is defined to be $\mathcal{U} = \mathcal{U}_{\text{s}} \cup \mathcal{U}_{\text{o}}$. Let us define m to be the maximum size of a subjective attribute set assigned to a key, i.e. we restrict $|\psi| \leqslant m$, and similarly define n to be the maximum size of an objective attribute set associated with a ciphertext, i.e. $|\omega| \leqslant n$. Furthermore, we denote the maximum number of rows of a subjective access structure matrix M to be $l_{s,\mathbf{max}}$. Now let $m' = m + l_{s,\mathbf{max}} - 1$ and $n' = n - 1$. Finally, let d be the maximum of $|\mathsf{Cover}(R)|$ for all $R \subseteq \mathcal{U}_{\text{ID}}$, where $\mathsf{Cover}(R)$ is defined as in Section 2.3.5.3. We construct each algorithm of the rkDPABE

scheme as follows:

1. $\mathsf{Setup}(1^\lambda, \mathcal{U})$: The algorithm picks random exponents $\gamma, \alpha \in \mathbb{Z}_p$ and a generator $g \in \mathbb{G}$. It defines three functions $F_s \colon \mathbb{Z}_p \to \mathbb{G}$, $F_o \colon \mathbb{Z}_p \to \mathbb{G}$ and $P \colon \mathbb{Z}_p \to \mathbb{G}$ by randomly choosing $h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d$ and setting

$$F_s(x) = \prod_{j=0}^{m'} h_j^{x^j}, \quad F_o(x) = \prod_{j=0}^{n'} q_j^{x^j}, \quad P(x) = \prod_{j=0}^{d} u_j^{x^j}. \tag{5.1}$$

   The public parameters are defined as

$$pp = (g, e(g,g)^\gamma, g^\alpha, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d).$$

   For each node label $x \in \mathcal{X}$ in the tree, it randomly chooses $a_x \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ to define a first degree polynomial $f_x(z) = a_x z + \alpha r_x + \gamma$. The master key is $mk = (\gamma, \alpha, \{a_x, r_x\}_{x \in \mathcal{X}})$.

2. $\mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, pp)$: The encryption algorithm takes as input a LSSS access structure (M, ρ) for the subjective policy $\mathbb{S}$ and an objective attribute set $\omega \subset \mathcal{U}_o$. Denote the dimensions of M as $l_s \times k_s$ matrix. The algorithm randomly chooses values $s, y_2, \ldots, y_{k_s} \in \mathbb{Z}_p$ and sets $\boldsymbol{u} = (s, y_2, \ldots, y_{k_s})$. It computes $\lambda_i = \boldsymbol{M_i} \cdot \boldsymbol{u}$ (for $i = 1, \ldots, l_s$), where $\boldsymbol{M_i}$ is the vector corresponding to the ith row of M. The ciphertext is then computed as $ct_{(\omega,\mathbb{S}),t} = (C, C^{(1)}, \{C_k^{(2)}\}_{k \in \omega}, \{C_i^{(3)}\}_{i=1,\ldots,l_s}, C^{(4)})$, where

$$\begin{aligned} C &= m \cdot (e(g,g)^\gamma)^s, & C^{(1)} &= g^s, \\ C_k^{(2)} &= F_o(k)^s, & C_i^{(3)} &= g^{\alpha\lambda_i} F_s(\rho(i))^{-s}, \\ C^{(4)} &= P(t)^s. \end{aligned}$$

   Intuitively, C masks the message by a group element in the target group of the bilinear map formed from the master secret γ and an encryption secret s (to randomise the encryption procedure). Decryption will have to compute this mask to recover the message.

   $C^{(1)}$ provides the encryption secret s. $C_k^{(2)}$ embeds each attribute in the objective set ω into the ciphertext, incorporating the encryption secret s such that attributes from prior ciphertexts cannot be combined with this encryption. Similarly, $C_i^{(3)}$ embeds the subjective policy $\mathbb{S}$ into the ciphertext using the shares of s divided according to $\mathbb{S}$, i.e. s is shared over the set of attributes such that any set of attributes that satisfies $\mathbb{S}$ can reconstruct the encryption secret s. Finally, $C^{(4)}$ links the encryption secret (and hence this particular ciphertext) to the specified time period t such that an update key for t is required to decrypt the ciphertext; this enables the revocation mechanism.

3. $\mathsf{KeyGen}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp)$: The key generation algorithm takes as input a LSSS access structure (N, π) for the objective policy $\mathbb{O}$ and a subjective attribute set $\psi \subset \mathcal{U}_\mathrm{s}$. Let the dimensions of N be denoted by $l_o \times k_o$. The algorithm also takes an identity $\mathsf{id} \in \mathcal{U}_\mathrm{ID}$ which is a leaf in the binary tree.

   For all $x \in \mathsf{Path}(\mathsf{id})$, the algorithm shares $f_x(1)$ using the LSSS (N, π). To do so, it randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and sets $\boldsymbol{v_x} = (f_x(1), z_{x,2}, \ldots, z_{x,k_o})$. For $i = 1, \ldots, l_o$, it calculates the share $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$, where $\boldsymbol{N_i}$ is the vector corresponding to the ith row of N.

   The algorithm then randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ for all $x \in \mathsf{Path}(\mathsf{id})$, and outputs the private key

$$sk_{\mathsf{id},(N,\pi)} = ((D^{(1)}_{x,i}, D^{(2)}_{x,i})_{x \in \mathsf{Path}(\mathsf{id}), i=1,\ldots,l_o}, (D_x, \{D^{(3)}_k\}_{k \in \psi})_{x \in \mathsf{Path}(\mathsf{id})}),$$

   where

$$D_x = g^{r_x}, \qquad D^{(1)}_{x,i} = g^{r_{x,i}},$$
$$D^{(2)}_{x,i} = g^{\sigma_{x,i}} F_o(\pi(i))^{r_{x,i}}, \qquad D^{(3)}_k = F_s(k)^{r_x}.$$

   Intuitively, r_x and $r_{x,i}$ for each $x \in \mathsf{Path}(\mathsf{id})$ randomises the key for the user id so that users may not collude. D_x and $D^{(1)}_{x,i}$ allow use of these random key values during decryption. $D^{(2)}_{x,i}$ embeds the shares of $f_x(1) = a_x + \alpha r_x + \gamma$ such that only the authorised sets according to $\mathbb{O}$ may reconstruct $f_x(1)$. Finally, $D^{(3)}_k$ embeds the attributes in ψ with the randomness chosen for this particular key. By linking these parameters to the path in a tree, only users for whom a valid update key has been issued (i.e. the non-revoked users) will be able to make use of these parameters to compute $f_x(1)$ for a node x; $f_x(1)$ is required as it contains the master secret γ which is used to cancel with the ciphertext component C to recover the message.

4. $\mathsf{KeyUpdate}(R, t, mk, pp)$: The algorithm first computes $\mathsf{Cover}(R)$ to find a minimal node set that covers $\mathcal{U} \setminus R$. For each $x \in \mathsf{Cover}(R)$, it randomly chooses $r_x \in \mathbb{Z}_p$ and sets the update key as $uk_{R,t} = \left\{U^{(1)}_x, U^{(2)}_x\right\}_{x \in \mathsf{Cover}(R)}$, where

$$U^{(1)}_x = g^{f_x(t)} P(t)^{r_x}, \quad U^{(2)}_x = g^{r_x}.$$

   Intuitively, each update key component is randomised by r_x and linked to a particular node x in the tree (covering only non-revoked users). $P(t)$ embeds the current time period which will match with the ciphertext component $C^{(4)}$. We also embed a point of the polynomial $f_x(t)$; given this point, and the point $f_x(1)$ (which can be recovered from the decryption key components $D^{(2)}_{x,i}$ given a satisfying set of objective attributes ω), one can perform Lagrange interpolation to recover the point $f_x(0)$ which will yield use of the master secret γ to cancel with the ciphertext component

C.

5. $\mathsf{Decrypt}(ct_{(\omega,\mathbb{S}),t}, (\omega,\mathbb{S}), sk_{\mathsf{id},(\mathbb{O},\psi)}, (\mathbb{O},\psi), uk_{R,t}, pp)$: The decryption algorithm takes as an input the ciphertext $ct_{(\omega,\mathbb{S}),t}$ which contains a subjective access structure (M,ρ) for $\mathbb{S}$ and a set of objective attributes ω, and a decryption key $sk_{\mathsf{id},(N,\pi)}$ which contains a set of subjective attributes ψ and an objective access structure (N,π) for $\mathbb{O}$. Suppose that ψ satisfies (M,ρ), the set ω satisfies (N,π), and that $\mathsf{id} \notin R$ (so that decryption is possible).

   Let $I_s = \{i : \rho(i) \in \psi\}$ and $I_o = \{i : \pi(i) \in \omega\}$. The algorithm computes sets of reconstruction constants $\{(i,\mu_i)\}_{i\in I_s}$ and $\{(i,\nu_i)\}_{i\in I_o}$ using the LSSS reconstruction algorithm. Since $\mathsf{id} \notin R$, the algorithm also finds a node x such that $x \in \mathsf{Path}(\mathsf{id}) \cap \mathsf{Cover}(R)$. Finally, it computes the following

$$C \cdot \frac{\prod_{i\in I_s}\left(e\left(C_i^{(3)}, D_x\right)\cdot e\left(C^{(1)}, D_{\rho(i)}^{(3)}\right)\right)^{\mu_i}}{\left(\prod_{j\in I_o}\left(\frac{e\left(D_{x,j}^{(2)}, C^{(1)}\right)}{e\left(C_{\pi(j)}^{(2)}, D_{x,j}^{(1)}\right)}\right)^{\nu_j}\right)^{\frac{t}{t-1}}\left(\frac{e\left(U_x^{(1)}, C^{(1)}\right)}{e\left(C^{(4)}, U_x^{(2)}\right)}\right)^{\frac{1}{1-t}}} = m.$$

We verify the correctness of the decryption as follows. Let us write the decryption computation as $C\cdot\frac{C'}{K}$, where $K = (K')^{\frac{t}{t-1}}(K'')^{\frac{1}{1-t}}$, and then consider each part in turn. Intuitively, C' is similar to a standard ABE decryption operation to match attributes to policies, whilst K' and K'' combine the two components of a functional decryption key (namely, a secret key and an update key) and perform a Lagrange interpolation to form a group element $e(g,g)^{s(\gamma+\alpha r_x)} = e(g,g)^{s\gamma}\cdot e(g,g)^{s\alpha r_x}$. The second part of this product will be the result of computing C' whilst the first will cancel with C to leave only m.

$$\begin{aligned}
C' &= \prod_{i\in I_s}\left(e\left(C_i^{(3)}, D_x\right)\cdot e\left(C^{(1)}, D_{\rho(i)}^{(3)}\right)\right)^{\mu_i}\\
&= \prod_{i\in I_s}\left(e\left(g^{\alpha\lambda_i}F_s(\rho(i))^{-s}, g^{r_x}\right)\cdot e\left(g^s, F_s(\rho(i))^{r_x}\right)\right)^{\mu_i}\\
&= \prod_{i\in I_s}\left(e\left(g,g\right)^{\alpha\lambda_i r_x}\cdot e\left(g, F_s(\rho(i))\right)^{-r_x s}\cdot e\left(g, F_s(\rho(i))\right)^{r_x s}\right)^{\mu_i}\\
&= e(g,g)^{\alpha r_x\sum_{i\in I_s}\mu_i\lambda_i}\\
&= e(g,g)^{\alpha r_x s}.
\end{aligned}$$

In the above expression, the second equality follows by substituting the values from the construction; the third equality follows from the properties of bilinear maps; the fourth equality simply moves the product into the exponent; and the final equality follows

from the reconstruction constants of the LSSS, namely that $\sum_{i \in I_s} \mu_i \lambda_i = s$.

$$\begin{aligned} K' &= \prod_{j \in I_o} \left(\frac{e\left(D^{(2)}_{x,j}, C^{(1)}\right)}{e\left(C^{(2)}_{x,\pi(j)}, D^{(1)}_{x,j}\right)} \right)^{\nu_j} = \prod_{j \in I_o} \left(\frac{e\left(g^{\sigma_{x,j}} F_o(\pi(j))^{r_{x,j}}, g^s\right)}{e\left(F_o(\pi(j))^s, g^{r_{x,j}}\right)} \right)^{\nu_j} \\ &= \prod_{j \in I_o} \left(\frac{e\left(g,g\right)^{\sigma_{x,j} s} \cdot e\left(g, F_o(\pi(j))\right)^{r_{x,j},s}}{e\left(g, F_o(\pi(j))\right)^{r_{x,j},s}} \right)^{\nu_j} \\ &= e\left(g,g\right)^{s \sum_{j \in I_o} \nu_j \sigma_{x,j}} = e(g,g)^{s f_x(1)}. \end{aligned}$$

In the above expression, the second equality follows directly from the construction; the third one follows from the properties of bilinear maps; the fourth equality stems from moving the product into the exponent; and the last one follows from the set of LSSS reconstruction constants with $\sum_{j \in I_o} \nu_j \sigma_{x,j} = f_x(1) = a_x + \alpha r_x + \gamma$.

$$\begin{aligned} K'' &= \frac{e\left(U^{(1)}_x, C^{(1)}\right)}{e\left(C^{(4)}, U^{(2)}_x\right)} = \frac{e\left(g^{f_x(t)} P(t)^{r_x}, g^s\right)}{e\left(P(t)^s, g^{r_x}\right)} = \frac{e\left(g,g\right)^{f_x(t)s} \cdot e\left(g, P(t)^{r_x s}\right)}{e\left(g, P(t)^{r_x s}\right)} \\ &= e(g,g)^{f_x(t)s} \end{aligned}$$

Then, it follows

$$\begin{aligned} K &= (K')^{\frac{t}{t-1}} (K'')^{\frac{1}{1-t}} = \left(e(g,g)^{s f_x(1)}\right)^{\frac{t}{t-1}} \left(e(g,g)^{f_x(t)s}\right)^{\frac{1}{1-t}} \\ &= (e(g,g)^s)^{f_x(1)\frac{t}{t-1} + f_x(t)\frac{1}{1-t}} \end{aligned}$$

Notice that $f_x(1)\frac{t}{t-1} + f_x(t)\frac{1}{1-t}$ is in fact a Lagrange interpolation for the two points $(1, f_x(1)), (1, f_x(t))$ for the first degree polynomial f_x. Thus, $f_x(1)\frac{t}{t-1} + f_x(t)\frac{1}{1-t} = f_x(0) = \alpha r_x + \gamma$. Hence, $K = e(g,g)^{s(\alpha r_x + \gamma)}$. Combining all of these results, we obtain the result of the decryption operation

$$C \cdot \frac{C'}{K} = m \cdot e(g,g)^{s\gamma} \cdot \frac{e(g,g)^{\alpha s r_x}}{e(g,g)^{s(\alpha r_x + \gamma)}} = m \cdot e(g,g)^{s\gamma} \cdot \frac{e(g,g)^{\alpha s r_x}}{e(g,g)^{s\gamma} \cdot e(g,g)^{\alpha s r_x}} = m.$$

5.4.4 Security Proof

Theorem 5.9. *The* rkDPABE *construction is secure with respect to indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*), as specified in Figure 5.5, assuming that the decisional q-BDHE problem is hard.*

The proof follows from a combination of [17] and [18] with some adjustment in the simulation of the private keys. We show that if an adversary can win the IND-sHRSS game with advantage ϵ with a challenge subjective access structure matrix of size $l^\star_s \times k^\star_s$, then a simulator with advantage ϵ in solving the decisional q-BDHE problem can be constructed, where $m + k^\star_s \leqslant q$.

Proof. Suppose, to achieve a contradiction with Theorem 5.9, that there exists an adversary $\mathcal{A}$ that has an advantage ϵ in attacking the rkDPABE scheme. We build a simulator $\mathcal{B}$ that solves the decisional q-BDHE problem (see Definition 2.24) in $\mathbb{G}$. Recall that we abbreviate g^{a^j} by g_j. The simulator $\mathcal{B}$ is given a random q-BDHE challenge $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ where $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ and Z is either $e(g_{q+1}, h)$ or a random element in $\mathbb{G}_1$. $\mathcal{B}$ acts as the challenger for $\mathcal{A}$ in the IND-sHRSS game as follows.

1. $\mathcal{A}$ begins by selecting its challenge parameters $(t^\star, \omega^\star, \mathbb{S}^\star)$ where $\mathbb{S}^\star$ is represented by a LSSS $(M^\star, \rho^\star)$. Let the matrix $M^\star$ be of size $l_s^\star \times k_s^\star$, where $m + k_s^\star \leq q$ and let $l_s^\star = l_{s,\mathbf{max}}$ and $|\omega^\star| = n$.

2. $\mathcal{B}$ now simulates running Setup for the rkDPABE scheme, and embeds the challenge policy into the public parameters. It first chooses $\gamma' \xleftarrow{\$} \mathbb{Z}_p$, sets $g^\alpha = g_1 = g^a$, and implicitly defines $\gamma = \gamma' + a^{q+1}$ by defining

$$\begin{aligned} e(g,g)^\gamma &= e(g_1, g_q) \cdot e(g,g)^{\gamma'} = e\left(g^a, g^{a^q}\right) \cdot e\left(g,g\right)^{\gamma'} \\ &= e(g,g)^{\gamma' + a^{q+1}}. \end{aligned}$$

   It then must define the polynomials F_s, F_o and P (as in [17] and [18]). To define F_s, $\mathcal{B}$ begins by defining $F_s(x) = g^{p(x)}$, where p is a polynomial in $\mathbb{Z}_p[x]$ of degree $m + l_s^\star - 1$ which is implicitly defined in the following manner. It chooses $k_s^\star + m + 1$ polynomials $p_0, \ldots, p_{k_s^\star + m}$ in $\mathbb{Z}_p[x]$, each of degree $m + l_s^\star - 1$, such that for all $x = \rho^\star(i)$ for some i (i.e. all x in the image of $\rho^\star$, of which there are exactly $l_s^\star$ since $\rho^\star$ is an injective mapping):

$$p_j(x) = \begin{cases} M^\star_{i,j} & \text{for } j \in [1, k_s^\star] \\ 0 & \text{for } j \in [k_s^\star + 1, k_s^\star + m] \end{cases} \tag{5.2}$$

   The polynomial p_0 is chosen randomly, and for all other x (not in the image of $\rho^\star$), p_j is defined randomly by randomly choosing values at m other points. By writing the coefficients of each polynomial as $p_j(x) = \sum_{i=0}^{m + l_s^\star - 1} p_{j,i} \cdot x^i$, one can define the polynomial $p(x)$ to be

$$p(x) = \sum_{j=0}^{k_s^\star + m} p_j(x) a^j. \tag{5.3}$$

   Then, $\mathcal{B}$ sets $h_i = \prod_{j=0}^{k_s^\star + m} g_j^{p_{j,i}}$ for $i \in [0, m + l_s^\star - 1]$. Finally, as we assumed

$l_s^\star = l_{s,\mathbf{max}}$, note that $m' = m + l_{s,\mathbf{max}} - 1 = m + l_s^\star - 1$,

$$\begin{aligned}
F_s(x) &= \prod_{i=0}^{m'} h_i^{x^i} \\
&= \prod_{i=0}^{m'} \left(\prod_{j=0}^{k_s^\star+m} g_j^{p_{j,i}} \right)^{x^i} \\
&= \prod_{i=0}^{m'} \left(\prod_{j=0}^{k_s^\star+m} g^{p_{j,i}a^j} \right)^{x^i} \\
&= g^{\sum_{j=0}^{k_s^\star+m} \sum_{i=0}^{m'} p_{j,i}x^i a^j} \\
&= g^{\sum_{j=0}^{k_s^\star+m} p_j(x)a^j} \\
&= g^{p(x)}.
\end{aligned}$$

The first equality in the above expression follows from equation (5.1) whilst the second follows by the above definition of h_i. The third equality is obtained by definition of $g_j = g^{a^j}$ and the last one follows by equation (5.3).

To define F_o, $\mathcal{B}$ randomly picks a polynomial $f'(x) = \sum_{j=0}^{n-1} f'_j x^j$ in $\mathbb{Z}_p[x]$ of degree $n-1$. It then defines $f(x) = \prod_{k\in\omega^\star}(x-k) = \sum_{j=0}^{n-1} f_j x^j$ (which can be computed entirely from $\omega^\star$); note that $f(x) = 0$ if and only if $x \in \omega^\star$. It defines $q_j = g_q^{f_j} g^{f'_j}$ for $j = [0, n-1]$. Using the above we can finally compute

$$\begin{aligned}
F_o(x) &= \prod_{j=0}^{n-1} q_j^{(x^j)} \\
&= \left(\prod_{j=0}^{n-1} g_q^{f_j} \cdot g^{f'_j} \right)^{x^j} \\
&= g_q^{\sum_{j=0}^{n-1} f_j x^j} \cdot g^{\sum_{j=0}^{n-1} f'_j x^j} \\
&= g_q^{f(x)} g^{f'(x)}.
\end{aligned}$$

To define P, $\mathcal{B}$ defines

$$\hat{p}(y) = y^{d-1} \cdot (y - t^\star) = \sum_{j=0}^{d} \hat{p}_j y^j.$$

This ensures $\hat{p}(t) = 0$ if and only if $t = t^\star$ for $t \in \mathcal{T}$, and that for $x \in \mathcal{X}$, $\hat{p}(x) \neq 0$ since we assumed $\mathcal{T} \cap \mathcal{X} = \emptyset$.

$\mathcal{B}$ then randomly picks a degree d polynomial $\rho(y) = \sum_{j=0}^{d} \rho_j y^j$ in $\mathbb{Z}_p[x]$ and lets

$u_j = (g^a)^{\hat{p}_j} g^{\rho_j}$ for $j = 0, \ldots, d$. Thus we can compute

$$\begin{aligned} P(y) &= \prod_{j=0}^{d} u_j^{y^j} \\ &= \left(\prod_{j=0}^{d} (g^a)^{\hat{p}_j} g^{\rho_j} \right)^{y^j} \\ &= (g^a)^{\sum_{j=0}^{d} \hat{p}_j y^j} \, g^{\sum_{j=0}^{d} \rho_j y^j} \\ &= (g^a)^{\hat{p}(y)} g^{\rho(y)}. \end{aligned} \tag{5.4}$$

The public key pk for the rkDP-ABE scheme is defined to be

$$pk = (g, e(g,g)^{\gamma}, g^{\alpha}, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d),$$

which is given to $\mathcal{A}$. Note that the randomness of the q-BDHE challenge $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ and the independently chosen randomness used in the construction of the polynomials p_j, f', and ρ ensure the public parameters are distributed as expected.

3. $\mathcal{A}$ declares its list $\overline{R}$ and is then given oracle access to the KeyGen and $\mathsf{KeyUpdate}$ functions. Let $\mathcal{X}_{\overline{R}} = \{x \in \mathsf{Path}(\mathsf{id}) : \mathsf{id} \in \overline{R}\}$. For each node label $x \in \mathcal{X}$ in the tree, $\mathcal{B}$ randomly chooses $a'_x \in \mathbb{Z}_p$ and implicitly defines

$$a_x = \begin{cases} a'_x - \alpha r_x - \gamma & \text{if } x \in \mathcal{X}_{\overline{R}} \\ a'_x - \frac{\alpha r_x - \gamma}{t^\star} & \text{if } x \notin \mathcal{X}_{\overline{R}} \end{cases} \tag{5.5}$$

Hence,

$$f_x(1) = a_x + \alpha r_x + \gamma = a'_x - \alpha r_x - \gamma + \alpha r_x + \gamma = a'_x \qquad \text{if } x \in \mathcal{X}_{\overline{R}} \tag{5.6}$$

$$f_x(t^\star) = a_x t^\star + \alpha r_x + \gamma = (a'_x - \frac{\alpha r_x - \gamma}{t^\star}) t^\star + \alpha r_x + \gamma = a'_x t^\star \quad \text{if } x \notin \mathcal{X}_{\overline{R}} \tag{5.7}$$

To simulate KeyGen queries for an objective access structure (N, π), a subjective attribute set ψ and an identity id, we consider the following cases:

- $(\omega^\star \in \mathbb{O})$ and $(\mathsf{id} \in \overline{R})$:
  For each $x \in \mathsf{Path}(\mathsf{id})$, note that since $\mathsf{id} \in \overline{R}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (5.6), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathsf{id})$. $\mathcal{B}$ can therefore compute the key components precisely as in the construction by sharing the value of $f_x(1)$.
- $(\omega^\star \notin \mathbb{O})$ and $(\mathsf{id} \in \overline{R})$:
  For each $x \in \mathsf{Path}(\mathsf{id})$, note that, since $\mathsf{id} \in \overline{R}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (5.6), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathsf{id})$.

$\mathcal{B}$ randomly chooses $r_x \in \mathbb{Z}_p$. It then lets $D_x = g^{r_x}$, and for all $k \in \psi$ lets $D_k^{(3)} = F_s(k)^{r_x}$ as in the construction. Recall that the dimensions of N are $l_o \times k_o$. Since $\omega^\star$ does not satisfy N for this case of the query, and by Proposition 2.21, there exists a vector $\boldsymbol{a_x} = (a_1, \ldots, a_{k_o}) \in \mathbb{Z}_p^{k_o}$ such that $a_1 = -1$ and $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$ for all i where $\pi(i) \in \omega^\star$.

$\mathcal{B}$ randomly chooses $z'_{x,2}, \ldots, z'_{x,k_o} \in \mathbb{Z}_p$ and defines $\boldsymbol{v'_x} = (0, z'_{x,2}, \ldots, z'_{x,k_o})$. It then implicitly defines a vector $\boldsymbol{v_x} = -(a'_x)\boldsymbol{a_x} + \boldsymbol{v'_x}$ (by using (5.2)) which will be used for creating the share of $f_x(1) = \gamma + \alpha r_x + a_x$ (note that the first element of $\boldsymbol{v_x}$ is indeed $f_x(1)$ by (5.6)), as in our construction.

Now, for all i such that $\pi(i) \in \omega^\star$, $\mathcal{B}$ randomly chooses $r_{x,i} \in \mathbb{Z}_p$ and computes $D_{x,i}^{(1)} = g^{r_{x,i}}$ and

$$\begin{aligned} D_{x,i}^{(2)} &= g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} \\ &= g^{\boldsymbol{N_i} \cdot (\boldsymbol{v_x} + (a'_x)\boldsymbol{a_x})} F_o(\pi(i))^{r_{x,i}} \\ &= g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}}, \end{aligned}$$

where the last equality holds because $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$. Note that $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$ in our construction and hence $D_{x,i}^{(2)}$ is of valid form.

For all other i, where $\pi(i) \notin \omega^\star$, $\mathcal{B}$ randomly chooses $r'_{x,i} \in \mathbb{Z}_p$. Observe that

$$\begin{aligned} \boldsymbol{N_i} \cdot \boldsymbol{v_x} &= \boldsymbol{N_i} \cdot (-(a'_x)\boldsymbol{a_x} + \boldsymbol{v'_x}) \\ &= \boldsymbol{N_i} \cdot (\boldsymbol{v'_x} - (a'_x)\boldsymbol{a_x}) \end{aligned}$$

Note that, unlike [18], due to our definition of a_x, we do not have a term in a^{q+1} here, and $\mathcal{B}$ can generate $D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}}$ and $D_{x,i}^{(1)} = g^{r_{x,i}}$.

- $(\psi \notin \mathbb{S}^\star)$ and $(\mathsf{id} \notin \overline{R})$:

  For each $x \in \mathsf{Path}(\mathsf{id})$, $\mathcal{B}$ does the following. Since ψ does not satisfy $M^\star$, by Proposition 2.21, there exists a vector $\boldsymbol{w_x} = (w_1, \ldots, w_{k_s^\star}) \in \mathbb{Z}_p^{k_s^\star}$ such that $w_1 = -1$ and $\boldsymbol{M_i} \cdot \boldsymbol{w_x} = 0$ for all i where $\rho(i) \in \psi^\star$. Now, by our definition of $p_j(x)$ in (5.2), we have that $(p_1(x), \ldots, p_{k_s^\star}(x)) \cdot (w_1, \ldots, w_{k_s^\star}) = 0$.

  $\mathcal{B}$ then computes one possible solution of variables $w_{k_s^\star+1}, \ldots, w_{k_s^\star+m}$ for the system of $|\psi|$ equations: for all $x \in \psi$

  $$(p_1(x), \ldots, p_{k_s^\star+m}(x)) \cdot (w_1, \ldots, w_{k_s^\star+m}) = 0,$$

  which is possible as $|\psi| \leqslant m$.

$\mathcal{B}$ then randomly chooses $r'_x \in \mathbb{Z}_p$ and implicitly defines

$$r_x = r'_x + w_1 \left(\frac{t^\star}{t^\star - 1}\right) \cdot \alpha^q + w_2 \left(\frac{t^\star}{t^\star - 1}\right) \cdot \alpha^{q-1} + \cdots + \\ + w_{k_s^\star + m} \left(\frac{t^\star}{t^\star - 1}\right) \cdot \alpha^{q-(k_s^\star+m)+1}$$

by setting the key $D_x = g^{r'_x} \prod_{k=1}^{k_s^\star+m} (g_{q+1-k})^{w_k \left(\frac{t^\star}{t^\star-1}\right)} = g^{r_x}$. Then, since $\gamma = \gamma' + \alpha^{q+1}$ and as $x \notin \mathcal{X}_{\overline{R}}$, we have

$$\begin{aligned}
f_x(1) &= \gamma + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a'_x - \frac{\alpha r_x - \gamma}{t^\star} \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + (\alpha(\frac{t^\star - 1}{t^\star}))r_x \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + \left(\alpha \left(\frac{t^\star - 1}{t^\star}\right) \left(r'_x + w_1 \left(\frac{t^\star}{t^\star - 1}\right) \cdot \alpha^q \right.\right. \\
&\quad \left.\left. + w_2 \left(\frac{t^\star}{t^\star - 1}\right) \cdot \alpha^{q-1} + \cdots + w_{k_s^\star+m} \left(\frac{t^\star}{t^\star - 1}\right) \cdot \alpha^{q-(k_s^\star+m)+1}\right)\right) \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + \alpha \left(\frac{t^\star - 1}{t^\star}\right) r'_x + w_1 \alpha^{q+1} + w_2 \alpha^q \\
&\quad + \cdots + w_{k_s^\star+m} \alpha^{q-(k_s^\star+m)+2} \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha \left(\frac{t^\star - 1}{t^\star}\right) r'_x w_2 \alpha^q + \cdots + w_{k_s^\star+m} \alpha^{q-(k_s^\star+m)+2},
\end{aligned}$$

where the α^{q+1} term in γ has cancelled out using Proposition 2.21 and the third equality followed from using equation (5.5). The simulator now randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and implicitly lets the vector $\boldsymbol{v_x} = (\gamma + \alpha r_x + a_x, z_{x,2}, \ldots, z_{x,k_o})$ as in the construction.

$\mathcal{B}$ also randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and computes for $i = 1, \ldots, l_o$ the key $D_{x,1}^{(1)} = g^{r_{x,i}}$. The other keys are computed in the following way. We have

$$D_{x,i}^{(2)} = \left(g^{\gamma' + a'_x + \frac{\gamma}{t^\star}} \cdot g_1^{r'_x} \prod_{k=2}^{k_s^\star+m} (g_{q-k+2})^{w_k}\right)^{N_{i,1}} \cdot \prod_{j=2}^{k_o} g^{N_{i,j} z_j} F_o(\pi(i))^{r_{x,i}}$$

which can be computed since g_{q+1} is not required and, by collecting the exponents, it can be verified that $D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} \cdot F_o(\pi(i))^{r_i}$.

Recall that $(p_1(k), \ldots, p_{k_s^\star+m}(k)) \cdot (w_1, \ldots, w_{k_s^\star+m}) = 0$ for all $k \in \psi$.

$$\begin{aligned} D_k^{(3)} &= D_x^{p_0(k)} \prod_{j=1}^{k_s^\star+m} \left(g_j^{r'_x} \prod_{k \in [1,k_s^\star+m], k \neq j} (g_{q+1-k+j})^{w_k} \right)^{p_j(k)} \\ &= (g^{r_x})^{p_0(k)} \prod_{j=1}^{k_s^\star+m} (g^{r_x})^{\alpha^j p_j(k)} \\ &= \prod_{j=0}^{k_s^\star+m} (g^{r_x})^{p_j(k)\alpha^j} = (g^{r_x})^{\sum_{j=0}^{k_s^\star+m} p_j(k)\alpha^j} \\ &= (g^{r_x})^{p(k)} = F_s(k)^{r_x}, \end{aligned}$$

where the second equality holds by observing that

$$D_k^{(3)} = D_k^{(3)} (g_{q+1})^{(p_1(k),\ldots,p_{k_s^\star+m}(k)) \cdot (w_1,\ldots,w_{k_s^\star+m})}$$

since $(g_{q+1})^{(p_1(k),\ldots,p_{k_s^\star+m}(k)) \cdot (w_1,\ldots,w_{k_s^\star+m})} = (g_{q+1})^0 = 1$ (see [18]).

- $(\omega^\star \notin \mathbb{O})$ and $(\psi \in \mathbb{S}^\star)$ and $(\mathsf{id} \notin \overline{R})$:

  For each $x \in \mathsf{Path}(\mathsf{id})$, $\mathcal{B}$ randomly chooses $r_x \in \mathbb{Z}_p$. It then lets $D_x = g^{r_x}$, and for all $k \in \psi$ lets $D_k^{(3)} = F_s(k)^{r_x}$ as in the construction. Recall that the dimensions of N are $l_0 \times k_0$. Since $\omega^\star$ does not satisfy N for this case of the query, and by Proposition 2.21, there exists a vector $\boldsymbol{a_x} = (a_1, \ldots, a_{k_o}) \in \mathbb{Z}_p^{k_o}$ such that $a_1 = -1$ and $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$ for all i where $\pi(i) \in \omega^\star$.

  $\mathcal{B}$ randomly chooses $z'_{x,2}, \ldots, z'_{x,k_o} \in \mathbb{Z}_p$ and defines $\boldsymbol{v'_x} = (0, z'_{x,2}, \ldots, z'_{x,k_o})$. It then implicitly defines a vector $\boldsymbol{v_x} = -(a'_x - \frac{\alpha r_x - \gamma}{t^\star} + \alpha r_x + \gamma)\boldsymbol{a_x} + \boldsymbol{v'_x}$ which will be used to create the share of $f_x(1) = \gamma + \alpha r_x + a_x$ (note that the first element of $\boldsymbol{v_x}$ is indeed $f_x(1)$ by (5.5)), as in our construction.

  Now, for all i such that $\pi(i) \in \omega^\star$, $\mathcal{B}$ randomly chooses $r_{x,i} \in \mathbb{Z}_p$ and computes $D_{x,i}^{(1)} = g^{r_{x,i}}$ and

$$D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}},$$

  where the last equality holds because $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$. Note that $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$ in our construction and hence $D_{x,i}^{(2)}$ is of the valid form.

For all other i, where $\pi(i) \notin \omega^\star$, $\mathcal{B}$ randomly chooses $r'_{x,i} \in \mathbb{Z}_p$. Observe that

$$\begin{aligned}
\boldsymbol{N_i} \cdot \boldsymbol{v_x} &= \boldsymbol{N_i} \cdot \left(- \left(a'_x - \frac{\alpha r_x - \gamma}{t^\star} + \alpha r_x + \gamma \right) \boldsymbol{a_x} + \boldsymbol{v'_x} \right) \\
&= \boldsymbol{N_i} \cdot \left(- \left(a'_x - \frac{\alpha r_x - (\gamma' + a^{q+1})}{t^\star} + \alpha r_x + (\gamma' + a^{q+1}) \right) \boldsymbol{a_x} + \boldsymbol{v'_x} \right) \\
&= \boldsymbol{N_i} \cdot \left(\boldsymbol{v'_x} - \left(a'_x + \gamma' \left(\frac{1}{t^\star} + 1 \right) \right) \boldsymbol{a_x} \right) + \left(r_x \left(\frac{1}{t^\star} - 1 \right) \boldsymbol{N_i} \cdot \boldsymbol{a_x} \right) \alpha \\
&\quad - \left(\left(\frac{1}{t^\star} + 1 \right) \boldsymbol{N_i} \cdot \boldsymbol{a_x} \right) a^{q+1}
\end{aligned}$$

contains a term in a^{q+1} and hence we cannot compute this value (as a^{q+1} is the gap in the q-BDHE game). Instead, we will use the $r_{x,i}$ term in $F_o(\pi(i))^{r_{x,i}}$ to cancel the unknown value a^{q+1}. $\mathcal{B}$ implicitly defines $r_{x,i} = r'_{x,i} - \frac{a(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}$. To do so, it defines

$$\begin{aligned}
D^{(2)}_{x,i} = g_1^{\left(r_x\left(\frac{1}{t^\star}-1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x} - \left(\frac{1}{t^\star}+1\right)\frac{\boldsymbol{N_i}\cdot\boldsymbol{a_x} f'(\pi(i))}{f(\pi(i))} \right)} \\
\cdot\, g^{\boldsymbol{N_i}\cdot\left(\boldsymbol{v'_x} - \left(a'_x + \gamma'\left(\frac{1}{t^\star}+1\right)\right)\right)\boldsymbol{a_x}} F_o(\pi(i))^{r'_{x,i}}.
\end{aligned}$$

To see that $D^{(2)}_{x,i}$ is valid, we observe

$$\begin{aligned}
D^{(2)}_{x,i} &= g_{q+1}^{\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}} \cdot D^{(2)}_{x,i} \cdot g_{q+1}^{-\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}} \\
&= g_{q+1}^{\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}} \cdot g_1^{r_x\left(\frac{1}{t^\star}-1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}} \cdot g^{\boldsymbol{N_i}\cdot\left(\boldsymbol{v'_x} - \left(a'_x + \gamma'\left(\frac{1}{t^\star}+1\right)\right)\right)\boldsymbol{a_x}} \\
&\quad \cdot \left(g_{q+1}^{-\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}} g_1^{-\left(\frac{1}{t^\star}+1\right)\frac{\boldsymbol{N_i}\cdot\boldsymbol{a_x} f'(\pi(i))}{f(\pi(i))}} \right) \cdot F_o(\pi(i))^{r'_{x,i}} \\
&= g^{\boldsymbol{N_i}\cdot\boldsymbol{v_x}} \left(g_q^{f(\pi(i))} g^{f'(\pi(i))} \right)^{\frac{-a\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}} \cdot F_o(\pi(i))^{r'_{x,i}} \\
&= g^{\boldsymbol{N_i}\cdot\boldsymbol{v_x}} \cdot F_o(\pi(i))^{\frac{-a\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}} \cdot F_o(\pi(i))^{r'_{x,i}} \\
&= g^{\boldsymbol{N_i}\cdot\boldsymbol{v_x}} \cdot F_o(\pi(i))^{r_{x,i}},
\end{aligned}$$

where the second last equality follows from equation (5.4).

$\mathcal{B}$ also defines

$$D^{(1)}_{x,i} = g^{r'_{x,i}} g_1^{\frac{-\left(\frac{1}{t^\star}+1\right)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}} = g^{r_{x,i}}.$$

Note that $f(\pi(i)) \neq 0$ since $\pi(i) \notin \omega^\star$, and so $D^{(1)}_{x,i}$ and $D^{(2)}_{x,i}$ are well defined.

To simulate KeyUpdate queries for time period t and revocation list R, we consider the following cases:

- $t = t^\star$ and $\overline{R} \subseteq R$:

  For each $x \in \mathsf{Cover}(R)$, $\mathcal{B}$ chooses a random $r_x \in \mathbb{Z}_p$ and computes $U_x^{(1)} = (g^{a'_x t^\star}) P(t^\star)^{r_x}$ and $U_x^{(2)} = g^{r_x}$. Both keys are valid since $\overline{R} \subseteq R$ and thus for all $x \in \mathsf{Cover}(R)$ we have $x \notin \mathcal{X}_{\overline{R}}$. Hence, by (5.7), $f_x(t^\star) = a'_x t^\star$.

- $t \neq t^\star$:

  For each $x \in \mathsf{Cover}(R)$, $\mathcal{B}$ chooses a random $r'_x \in \mathbb{Z}_p$

  – If $x \in \mathsf{Cover}(R) \cap \mathcal{X}_{\overline{R}}$, it defines

$$U_x^{(1)} = (g^{a'_x})^t (g^{\gamma'})^{(1-t)} (g_1^{r'_x})^{(1-t)} g_q^{-\frac{\rho(t)(1-t)}{\hat{p}(t)+1-t}} P(t)^{r'_x}$$

$$U_x^{(2)} = (g^{r'_x})(g_q)^{-\frac{1-t}{\hat{p}(t)+1-t}}$$

  Note that $\hat{p}(t) \neq 0$ for $t \neq t^\star$ so this is well defined. We claim that these keys look valid according to the construction with implicit randomness $r_x = r'_x - \frac{a^q(1-t)}{\hat{p}(t)+1-t}$.
  Note that, in this case, $x \in \mathcal{X}_{\overline{\mathcal{R}}}$ and hence by (5.5)

$$\begin{aligned} f_x(t) &= a_x t + \alpha r_x + \gamma = (a'_x - \alpha r_x - \gamma)t + \alpha r_x + \gamma \\ &= a'_x t + \alpha r_x (1-t) + \gamma'(1-t) + a^{q+1}(1-t). \end{aligned}$$

  Then,

$$\begin{aligned} U_x^{(1)} &\overset{(1)}{=} g^{f_x(t)} P(t)^{r_x} \\ &\overset{(2)}{=} g^{a'_x t + a r_x (1-t) + \gamma'(1-t) + a^{q+1}(1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a r_x (1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \\ &\overset{(3)}{=} g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{a\hat{p}(t) r'_x} \\ &\qquad g^{-a\hat{p}(t)B} g^{\rho(t) r'_x} g^{-\rho(t)B} \\ &\overset{(4)}{=} g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{-a\hat{p}(t)B} g^{-\rho(t)B} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)B} g^{-Ba((1-t)+a\hat{p}(t))} \\ &\overset{(5)}{=} g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} \\ &\qquad g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} (g^a)^{-\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)((1-t)+\hat{p}(t))} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} (g^a)^{-(a^q(1-t))} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} g^{-a^{q+1}(1-t)} \\ &= g^{a'_x t} g^{\gamma'(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} \\ &= (g^{a'_x})^t (g^{\gamma'})^{(1-t)} (g_1^{r'_x})^{(1-t)} g_q^{-\frac{\rho(t)(1-t)}{\hat{p}(t)+1-t}} P(t)^{r'_x}. \end{aligned}$$

Note that equality (1) follows by construction and (2) uses $f_x(t)$ from above and equation (5.4). Equality (3) follows by replacing r_x with $r'_x - B$ and equality (4) follows from using (5.4). Equality (5) is valid by using $B = \frac{a^q(1-t)}{\hat{p}(t)+1-t}$.
Then,

$$\begin{aligned} U_x^{(2)} &= g^{r_x} = g^{r'_x} g^{-\frac{a^q(1-t)}{\hat{p}(t)+1-t}} \\ &= (g^{r'_x})(g_q)^{-\frac{1-t}{\hat{p}(t)+1-t}}. \end{aligned}$$

Hence, these keys look valid according to the construction.

– If $x \in \mathsf{Cover}(R) \setminus \mathcal{X}_{\overline{R}}$, it defines

$$U_x^{(1)} = (g^{a'_x})^t (g^{\gamma'})^{(\frac{t}{t^\star}+1)} (g_1^{r'_x})^{(1-\frac{t}{t^\star})} g_q^{-\frac{\rho(t)(1+\frac{t}{t^\star})}{\hat{p}(t)+1-\frac{t}{t^\star}}} P(t)^{r'_x}$$

$$U_x^{(2)} = (g^{r'_x})(g_q)^{-\frac{1+\frac{t}{t^\star}}{\hat{p}(t)+1-\frac{t}{t^\star}}}$$

In this case, by (5.5), $a_x = a'_x - \frac{\alpha r_x - \gamma}{t^\star}$. By a similar argument as above, these keys look valid according to the construction with implicit randomness $r_x = r'_x - -\frac{a^q(1+\frac{t}{t^\star})}{\hat{p}(t)+1-\frac{t}{t^\star}}$.

4. $\mathcal{A}$ selects two messages m_0 and m_1. $\mathcal{B}$ chooses $b \xleftarrow{\$} \{0,1\}$ and creates a ciphertext $C = m_b \cdot Z \cdot e(h, g^{\gamma'}), C^{(1)} = h$, and for $k \in \omega^\star$ we write $C_k^{(2)} = h^{f'(x)}$. We write $h = g^s$ for some unknown s. The simulator then chooses random elements $y'_2, \ldots, y'_{k_s^\star} \in \mathbb{Z}_p$ and lets $\boldsymbol{y'} = (0, y'_2, \ldots, y'_{k_s^\star})$. It defines $C_i^{(3)} = (g_1)^{\boldsymbol{M_i^\star} \cdot \boldsymbol{y'}} \cdot (g^s)^{-p_0(\rho^\star(i))}$ for $i = 1, \ldots, l'_s$ and $C^{(4)} = (g^s)^{\rho(t^\star)}$, to implicitly share the secret s via the vector

$$\boldsymbol{v_x} = (s, s\alpha + y'_2, s\alpha^2 + y'_3, \ldots, s\alpha^{k'_s - 1} + y'_{k'_s}).$$

We claim that if $Z = e(g_{q+1}, h)$ then the created ciphertext is a valid challenge. The validity of $C^{(1)} = h = g^s$ comes from the implicit definition of h. To see that C is valid, recall that $\gamma = \gamma' + a^{q+1}$. Then,

$$\begin{aligned} C &= m_b \cdot Z \cdot e(h, g^{\gamma'}) = m_b \cdot e(g_{q+1}, h) \cdot e(h, g^{\gamma'}) = m_b \cdot e(g,g)^{sa^{q+1}} \cdot e(g,g)^{s\gamma'} \\ &= m_b \cdot e(g,g)^{s(\gamma' + a^{q+1})} = m_b \cdot e(g,g)^{s\gamma}. \end{aligned}$$

For all $k \in \omega^\star$, we defined $f(k)$ such that $f(k) = 0$, and hence

$$C_k^{(2)} = h^{f'(k)} = (g^s)^{f'(k)} = (g_q^{f(k)} g^{f'(k)})^s = F_o(k)^s.$$

For $i = 1, \ldots, l'_s$, we have

$$\begin{aligned} C_i^{(3)} &= (g_1)^{\boldsymbol{M}_i^\star \cdot \boldsymbol{y}'} \cdot (g^s)^{-p_0(\rho^\star(i))} \\ &= (g^\alpha)^{\boldsymbol{M}_i^\star \cdot \boldsymbol{y}'} \prod_{j=1}^{k_s^\star} g^{M_{i,j}^\star s \alpha^j} \cdot (g^s)^{-p_0(\rho^\star(i))} \prod_{j=1}^{k_s^\star} (g^s)^{-M_{i,j}^\star \alpha^j} \\ &= g^{\alpha \boldsymbol{M}_i^\star \cdot \boldsymbol{v}_{\boldsymbol{x}}} \cdot (g^s)^{-p(\rho^\star(i))} = g^{\alpha \boldsymbol{M}_i^\star \cdot \boldsymbol{v}_{\boldsymbol{x}}} \cdot F_s(\rho^\star(i))^{-s}, \end{aligned}$$

Finally, since $\hat{p}(t^\star) = 0$, we have $C^{(4)} = (g^s)^{\rho(t^\star)} = ((g^a)^{\hat{p}(t^\star)} g^{\rho(t^\star)})^s = P(t^\star)^s$.

5. The challenge ciphertext is given to $\mathcal{A}$ along with oracle access which is handled as in Step 3.

6. $\mathcal{A}$ eventually outputs $b' \in \{0, 1\}$ as its guess of b. If $b = b'$ then $\mathcal{B}$ outputs 1 to guess that $Z = e(g_{q+1}, h)$. Otherwise, $\mathcal{B}$ outputs 0 to guess that Z is random.

If $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{R}_{BDHE}$ then $\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) \to 0] = \frac{1}{2}$ since $\mathcal{A}$ was given a malformed challenge and hence can only guess the value of b. On the other hand if $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{P}_{BDHE}$ then we formed a valid challenge ciphertext and, as $\mathcal{A}$ is assumed to have non-negligible advantage ϵ in the IND-sHRSS game, $|\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) \to 0] - \frac{1}{2}| \geq \epsilon$. It follows that $\mathcal{B}$ has advantage at least ϵ in solving q-BDHE problem in $\mathbb{G}$. However, we assumed that this problem is hard, so an adversary with non-negligible advantage in the IND-sHRSS game cannot exist. □

5.5 Construction

In this section we provide a construction of an HPVC scheme for a family $\mathcal{F}$ of monotone Boolean functions closed under complement using a revocable key dual-policy ABE scheme $\mathcal{RKDPABE}$ in a black box manner comprising the algorithms DPABE.Setup, DPABE.Encrypt, DPABE.KeyGen, DPABE.KeyUpdate and DPABE.Decrypt. We also use a signature scheme with algorithms Sig.KeyGen, Sig.Sign and Sig.Verify, and a one-way function g. Let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup T_{\text{O}} \cup T_{\text{S}}$ be the universe of attributes acceptable by the revocable key dual-policy ABE scheme, formed as the union of the following sub-universes, where $\mathcal{U}_{\text{attr}}$ consists of the attributes that form characteristic tuples for input data, $\mathcal{U}_l$ be a set of attributes (disjoint from $\mathcal{U}_{\text{attr}}$) that uniquely label each function and each data item, $\mathcal{U}_{\text{ID}}$ comprises attributes representing entity identifiers, $\mathcal{U}_{\text{time}}$ comprises attributes representing time periods issued by the time source $\mathbb{T}$ and finally T_{O} and T_{S} represent the objective dummy attribute and subjective dummy attribute respectively.

We encode as usual Boolean functions in terms of access structures over $\mathcal{U}_{\text{attr}}$. Computations with n-bit outputs can be built from n Boolean functions returning each bit

in turn. We can handle negations by either building rkDPABE from non-monotonic ABE [112] or by adding negated attributes to the universe [142]. We choose to use the latter approach and add negated attributes to $\mathcal{U}_{\text{attr}}$. Thus, for the ith bit of a binary input string $X = x_1 \dots x_n$, we define attributes $A^0_{X,i}$ and $A^1_{X,i} \in \mathcal{U}_{\text{attr}}$ and X is encoded as $A_X = \{A^j_{X,i} \in \mathcal{U}_{\text{attr}} : x_i = j\}$.

In more detail, the dummy attributes T_{O} and T_{S} play generally a crucial role in a DP-ABE scheme as they efficiently enable a DP-ABE scheme to function as either KP-ABE or CP-ABE [19]. For KP-ABE, the subjective policy corresponds to $\mathbb{S} = \{\{T_{\text{S}}\}\}$ and is satisfied by the subjective attribute ψ containing the special attribute T_{S}. Thus, $\mathbb{S}$ is trivially satisfied and decryption (in KP-ABE) only depends on the objective policy and attributes. Similarly, the same holds for CP-ABE where the objective policy corresponds to $\mathbb{O} = \{\{T_{\text{O}}\}\}$ that is trivially satisfied by the objective attribute ω containing the special attribute T_{O}. As discussed in Section 2.7.3 and Section 3.4, we require to establish two distinct ABE schemes to overcome a possible one-sided error in the verification stage. Thus, we initialise two distinct rkDP-ABE systems over $\mathcal{U}$ and hence we define a total of four additional dummy attributes where $T^0_{\text{O}}, T^0_{\text{S}}$ relate to the first rkDP-ABE system, and $T^1_{\text{O}}, T^1_{\text{S}}$ relate to the second rkDP-ABE system. As summarised in Table 5.1, the function corresponds in the modes RPVC and RPVC-AC to $\mathbb{O} = F$ and $\mathbb{S} = \{\{T^0_{\text{S}}\}\}$. Thus, the complement function for those modes can be defined as $\overline{\mathbb{O}} = \overline{F}$ and $\overline{\mathbb{S}} = \{\{T^1_{\text{S}}\}\}$. Similarly, it follows for the mode VDC that $\mathbb{O} = \{\{T^0_{\text{O}}\}\}$ and $\mathbb{S} = F$, and the the complement can be defined as $\overline{\mathbb{O}} = \{\{T^1_{\text{O}}\}\}$ and $\overline{\mathbb{S}} = \overline{F}$. Each mode operates by encrypting a pair of randomly chosen messages and issuing keys such that the recovery of one message implies whether the encryption of a message was linked to F or $\overline{F}$, and thus whether $F(X) = 1$ or 0. Ciphertext indistinguishability ensures that an adversary cannot cheat by returning the other message.

Our HPVC scheme operates in the following way.

1. Setup, presented in Algorithm 1, first forms the attribute universe $\mathcal{U}$ and initialises two rkDPABE schemes over the universe. It further creates an empty two-dimensional array L_{Reg} to list registered entities, a (empty) list of revoked entities L_{Rev} as well as a time source $\mathbb{T}$ (e.g. a networked clock or counter) to index update keys. The algorithm finally outputs the public parameters pp and master secret key mk comprising of public and secret rkDPABE parameters respectively. Furthermore, the public parameters also contain L_{Reg} and the dummy attributes enabling a client to flexibly switch between the modes of computations by disabling certain parts of the rkDPABE scheme while the master secret key additionally contains the list of revoked entities L_{Rev}. Note that the public parameters may be implicitly updated throughout the execution of all algorithms of an HPVC scheme accommodating any changes in the system population.

Algorithm 1 $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$

1 : $\mathcal{U} \leftarrow \mathcal{U}_{\text{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup T_{\text{O}} \cup T_{\text{S}}$
2 : $(mpk^0_{\text{ABE}}, msk^0_{\text{ABE}}, T^0_{\text{O}}, T^0_{\text{S}}) \leftarrow_{\$} \mathsf{DPABE.Setup}(1^\lambda, \mathcal{U})$
3 : $(mpk^1_{\text{ABE}}, msk^1_{\text{ABE}}, T^1_{\text{O}}, T^1_{\text{S}}) \leftarrow_{\$} \mathsf{DPABE.Setup}(1^\lambda, \mathcal{U})$
4 : **for** $S_i \in \mathcal{U}_{\text{ID}}$ **do**
5 : $\quad L_{\text{Reg}}[S_i][0] \leftarrow \epsilon$
6 : $\quad L_{\text{Reg}}[S_i][1] \leftarrow \{\epsilon\}$
7 : **endfor**
8 : $L_{\text{Rev}} \leftarrow \epsilon$
9 : Initialise $\mathbb{T}$
10 : $pp \leftarrow (mpk^0_{\text{ABE}}, mpk^1_{\text{ABE}}, T^0_{\text{O}}, T^0_{\text{S}}, T^1_{\text{O}}, T^1_{\text{S}} L_{\text{Reg}}, \mathbb{T})$
11 : $mk \leftarrow (msk^0_{\text{ABE}}, msk^1_{\text{ABE}}, L_{\text{Rev}})$

2. FnInit, presented in Algorithm 2, sets the public delegation key pk_F (for all functions F) to be the public parameters for the system (since we use public key primitives). This step is not required in our particular construction, but we retain the algorithm for consistency with prior definitions as well as for generality as other instantiations may require this step.

Algorithm 2 $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$

1 : $pk_F \leftarrow pp$

3. Register, presented in Algorithm 3, creates a public-private key pair by calling the KeyGen algorithm of the digital signature scheme. The algorithm provides the server with its own secret signature key and updates $L_{\text{Reg}}[S_i][0]$ to store the verification key for S_i. These prevent servers being impersonated and wrongly revoked.

Algorithm 3 $sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp)$

1 : $(sk_{\text{Sig}}, vk_{\text{Sig}}) \leftarrow_{\$} \mathsf{Sig.KeyGen}(1^\lambda)$
2 : $sk_{S_i} \leftarrow sk_{\text{Sig}}$
3 : $L_{\text{Reg}}[S_i][0] \leftarrow L_{\text{Reg}}[S_i][0] \cup vk_{\text{Sig}}$

4. Certify, presented in Algorithm 4, aims to generate an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ for a server S_i. The algorithm first adds an element $(F, \bigcup_{l \in L_i} l)$ to the list $L_{\text{Reg}}[S_i][1]$

for each $F \in \mathcal{F}_i$. This publicises the computations that S_i can perform (either functions in RPVC and RPVC-AC modes, or functions and data labels in VDC). The algorithm removes S_i from the revocation list, gets the current time period from $\mathbb{T}$ and generates a decryption key for $(\mathbb{O}, A_\psi \cup \bigcup_{l \in L_i} l)$ in the first DP-ABE system and A_ψ is the attribute set encoding ψ. The additional attributes for the labels $l \in \mathcal{U}_l$ ensure that a key cannot be used to evaluate computations that do not correspond to these labels. In RPVC and RPVC-AC, this means that a key for a function G cannot evaluate a computation request for $F(X)$. In VDC, it means that an evaluation key must be issued for a dataset D_i that includes (at least) the specified input data X. It is sufficient to include labels only on the subjective attribute set without also adding them to the objective policy. As these labels are a security measure against a misbehaving server, we amend the servers key but need not take similar measures against the delegator. Delegators are then able to specify the required labels in their created subjective policy. Those labels need to be present in the server's key for a successful evaluation (decryption). The KDC should check that the label corresponds to the input to ensure that a server does not advertise data he does not own. It also generates an update key for the current time period to prove that S_i is not currently revoked. In RPVC and RPVC-AC modes, another pair of keys is generated using the second DP-ABE system for the complement inputs.

Algorithm 4 $ek_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O},\psi), L_i, \mathcal{F}_i, mk, pp)$

1 : **for** $F \in \mathcal{F}_i$ **do**
2 : $\quad L_{\mathrm{Reg}}[S_i][1] \leftarrow L_{\mathrm{Reg}}[S_i][1] \cup (F, \bigcup_{l \in L_i} l)$
3 : **endfor**
4 : $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \setminus S_i$
5 : $t \leftarrow \mathbb{T}$
6 : $sk^0_{\mathrm{ABE}} \leftarrow_{\$} \mathsf{DPABE.KeyGen}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l \in L_i} l), msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
7 : $uk^0_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
8 : **if** ($\mathtt{mode} = \mathrm{RPVC}$) **or** ($\mathtt{mode} = \mathrm{RPVC\text{-}AC}$) **then**
9 : $\quad sk^1_{\mathrm{ABE}} \leftarrow_{\$} \mathsf{DPABE.KeyGen}(S_i, (\overline{\mathbb{O}}, A_\psi \cup \bigcup_{l \in L_i} l), msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
10 : $\quad uk^1_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
11 : **else**
12 : $\quad sk^1_{\mathrm{ABE}} \leftarrow \perp$
13 : $\quad uk^1_{L_{\mathrm{Rev}},t} \leftarrow \perp$
14 : **endif**
15 : $ek_{F,S} \leftarrow (sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, uk^1_{L_{\mathrm{Rev}},t})$

5. ProbGen, presented in Algorithm 5, aims to create a problem instance $\sigma_{(\omega,\mathbb{S})}$ that the server can use to evaluate the computation as well as preparing a verification key that enables anyone to verify the server's computational result. The algorithm starts with choosing messages m_0 and m_1 randomly from the message space. The message m_0 is encrypted with $(A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l)$ in the first rkDPABE system, whilst m_1 is encrypted with the complement policy under either the first rkDPABE system for VDC or the second one for RPVC and RPVC-AC depending on the chosen mode of computation. Note that the attributes remain the same as it is the same attribute T^0_{O} or input data X respectively. The algorithm also prepares a public verification key $vk_{(\omega,\mathbb{S})}$. The key is simply generated by applying a one-way function g to each randomly chosen message and also includes a copy of L_{Reg} from the public parameters in case the list is modified between the current time period and the time of verification.

Algorithm 5 $(\sigma_{(\omega,\mathbb{S})}, vk_{(\omega,\mathbb{S})}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega,\mathbb{S}), L_{F,X}, pk_F, pp)$

$(m_0, m_1) \leftarrow_{\$} \mathcal{M} \times \mathcal{M}$
$t \leftarrow \mathbb{T}$
$c_0 \leftarrow_{\$} \mathsf{DPABE.Encrypt}(m_0, (A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l), t, mpk^0_{\mathrm{ABE}})$
if ($\mathtt{mode} = \mathrm{VDC}$) **then**
  $c_1 \leftarrow_{\$} \mathsf{DPABE.Encrypt}(m_1, (A_\omega, \overline{\mathbb{S}} \wedge \bigwedge_{l \in L_{F,X}} l), t, mpk^0_{\mathrm{ABE}})$
else
  $c_1 \leftarrow_{\$} \mathsf{DPABE.Encrypt}(m_1, (A_\omega, \overline{\mathbb{S}} \wedge \bigwedge_{l \in L_{F,X}} l), t, mpk^1_{\mathrm{ABE}})$
endif
$\sigma_{(\omega,\mathbb{S})} \leftarrow (c_0, c_1)$
$vk_{(\omega,\mathbb{S})} \leftarrow (g(m_0), g(m_1), L_{\mathrm{Reg}})$

6. Compute, presented in Algorithm 6, is performed by a server S_i and aims to return the result of the evaluation of a function on some input data. The algorithm attempts to decrypt both ciphertexts of the problem instance $\sigma_{(\omega,\mathbb{S})}$, ensuring that different modes of computation use the correct parameters. Decryption succeeds only if the function evaluates to 1 on the input data X, i.e. the policy is satisfied. Since F and $\overline{F}$ output opposite results on X, exactly one plaintext will correspond to a failure symbol $\perp$. The server signs the results using its personal signing key. Finally, the algorithm outputs the computational result $\theta_{F(X)}$ comprising the two plaintexts, the server id and the server's signature on the output.

Algorithm 6 $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, ek_{(\mathbb{O},\psi),S_i}, sk_{S_i}, pp)$

Parse $\sigma_{(\omega,\mathbb{S})}$ as (c_0, c_1) and $ek_{(\mathbb{O},\psi),S_i}$ as $(sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, uk^1_{L_{\mathrm{Rev}},t})$
$d_0 \leftarrow \mathsf{DPABE.Decrypt}\left(c_0, sk^0_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, mpk^0_{\mathrm{ABE}}\right)$
if ($\mathtt{mode} = \mathrm{VDC}$) **then**
  $d_1 \leftarrow \mathsf{DPABE.Decrypt}\left(c_1, sk^0_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, mpk^0_{\mathrm{ABE}}\right)$
else
  $d_1 \leftarrow \mathsf{DPABE.Decrypt}\left(c_1, sk^1_{\mathrm{ABE}}, uk^1_{L_{\mathrm{Rev}},t}, mpk^1_{\mathrm{ABE}}\right)$
endif
$\gamma \leftarrow_{\$} \mathsf{Sig.Sign}(d_0, d_1, S_i, sk_{S_i})$
$\theta_{F(X)} \leftarrow (d_0, d_1, S_i, \gamma)$

7. Verify, presented in Algorithm 7, determines whether the returned computational result is valid or not. The algorithm first checks whether the function F is listed in $L_{\text{Reg}}[S][1]$ to ensure that the server that generated the computational result is authorised to compute F. If this check fails, the result is immediately rejected. Next, the algorithm verifies the signature using the verification key for S_i stored in L_{Reg}. If correct, it applies the one-way function g to each plaintext in $\theta_{F(X)}$ and compares the results to the components of the verification key. If either comparison results in a match (i.e. the server successfully recovered a message), the algorithm creates an acceptance token $\tau_{\theta_{F(X)}} = (\texttt{accept}, S_i)$ indicating that the server indeed performed the computation correctly. Otherwise the result is rejected, and the algorithm creates a rejection token $\tau_{\theta_{F(X)}} = (\texttt{reject}, S_i)$ and S_i is reported for revocation. If m_0 was returned then $F(X) = 1$ as m_0 was encrypted for the non-complemented inputs. Otherwise m_1 was returned and thus $F(X) = 0$. Note that this algorithm can be run by any entity since the computational result and verification key are publicly available.

Algorithm 7 $(y, \tau_{\theta_{F(X)}}) \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{(\omega,\mathbb{S})}, pp)$

Parse $\theta_{F(x)}$ as (d_0, d_1, S_i, γ) and $vk_{(\omega,\mathbb{S})}$ as $(g(m_0), g(m_1), L_{\text{Reg}})$
if $F \in L_{\text{Reg}}[S_i][1]$ **then**
  if $\texttt{accept} \leftarrow \mathsf{Sig.Verify}((d_0, d_1, S_i), \gamma, L_{\text{Reg}}[S_i][0])$
    if $g(m_0) = g(d_0)$ **return** $(y \leftarrow 1, \tau_{\theta_{F(X)}} \leftarrow (\texttt{accept}, S_i))$
    elseif $g(m_1) = g(d_1)$ **return** $(y \leftarrow 0, \tau_{\theta_{F(X)}} \leftarrow (\texttt{accept}, S_i))$
    else return $(y \leftarrow \perp, \tau_{\theta_{F(x)}} \leftarrow (\texttt{reject}, S_i))$
    endif
  endif
endif
return $(y \leftarrow \perp, \tau_{\theta_{F(x)}} \leftarrow (\texttt{reject}, S_i))$

8. Revoke, presented in Algorithm 8, aims to revoke misbehaving servers by redistributing fresh update keys to all non-revoked servers. The algorithm first checks whether a server S_i should in fact be revoked, i.e. whether it received as input a rejection token $\tau_{\theta_{F(X)}} = (\texttt{reject}, S_i)$. If so, it deletes the list $L_{\text{Reg}}[S_i][1]$ of computations that S_i may perform such that the server is no longer authorised to perform any computations within the system. Additionally, it also adds S_i to the revocation list L_{Rev}, and refreshes the time source $\mathbb{T}$ and samples the new time period. The algorithm then generates new update keys for all non-revoked entities such that non-revoked keys are still functional in the new time period and distributes them accordingly. If the algorithm receives as input an acceptance token indicating that there is no need to revoke any server since computations

were performed correctly, it outputs $\perp$.

Algorithm 8 $um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(X)}}, mk, pp)$

1: **if** $\tau_{\theta_{F(X)}} = (\texttt{reject}, S_i)$ **then**
2: $L_{\mathrm{Reg}}[S_i][1] \leftarrow \{\epsilon\}$
3: $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \cup S_i$
4: Refresh $\mathbb{T}$
5: $t \leftarrow \mathbb{T}$
6: $uk^0_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
7: **if** ($\texttt{mode} = \mathrm{RPVC}$) **or** ($\texttt{mode} = \mathrm{RPVC\text{-}AC}$) **then**
8: $uk^1_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
9: **endif**
10: **for** $S' \in \mathcal{U}_{\mathrm{ID}}$ **do**
11: Parse $ek_{F,S'}$ as $(sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t-1}, uk^1_{L_{\mathrm{Rev}},t-1})$
12: $ek_{F,S'} \leftarrow (sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, uk^1_{L_{\mathrm{Rev}},t})$
13: **endfor**
14: **return** $um \leftarrow \{ek_{F,S'}\}_{S' \in \mathcal{U}_{\mathrm{ID}}}$
15: **else**
16: **return** $\perp$

Theorem 5.10. *Given an* IND-sHRSS *secure* rkDPABE *scheme for a class of monotone Boolean functions $\mathcal{F}$ closed under complement, an* EUF-CMA *secure signature scheme and a one-way function g. Let $\mathcal{HPVC}$ be the hybrid publicly verifiable outsourced computation scheme as defined in Algorithms 1–8. Then $\mathcal{HPVC}$ is secure in the sense of selective public verifiability (Figure 5.2), and selective semi-static revocation (Figure 5.3) and selective authorised computation (Figure 5.4).*

5.6 Proofs of Security

In this section we present the full proof of Theorem 5.10 by providing proofs of security for the notions of selective public verifiability, selective semi-static revocation and selective authorised computations.

5.6.1 Selective Public Verifiability

Lemma 5.11. *The $\mathcal{HPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective public verifiability (Figure 5.2) under the same assumptions as in Theorem 5.10.*

Proof. Note that this proof is a combination of the proofs presented in Sections 3.5.1 and 4.5. The reduction follows similarly and we basically update the notation to accommodate an $\mathcal{HPVC}$ scheme.

Suppose $\mathcal{A}_{\text{HPVC}}$ is an adversary with non-negligible advantage against the selective public verifiability game (Figure 5.2) when instantiated by Algorithms 1–8. We begin by defining the following three games:

- **Game 0.** This is the selective public verifiability game as defined in Figure 5.2.
- **Game 1.** This is the same as **Game 0** with the modification that in ProbGen, we no longer return an encryption of m_0 and m_1. Instead, we choose another random message $m' \neq m_0, m_1$ and, if $F(X^\star) = 1$, we replace c_1 by the encryption of m', and otherwise we replace c_0. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.
- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

We show that from the adversary's point of view **Game 2** is indistinguishable from **Game 0** except with negligible probability. This means that an adversary against the selective public verifiability game can be run against **Game 2**. We then finally show that if an adversary has a non-negligible advantage against **Game 2** then the adversary can invert a one-way function.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**, both with parameters $(\mathcal{HPVC}, 1^\lambda, \mathcal{F})$. Suppose otherwise, that $\mathcal{A}_{\text{HPVC}}$ can distinguish the two games with non-negligible advantage δ. We then show that it is possible to construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{HPVC}}$ as a subroutine to break the IND-sHRSS security of the (indirectly) revocable key DP-ABE scheme formalised in Figure 5.5. Note that we only focus on the modes RPVC and VDC, and the mode RPVC-AC can be seen as a special case of the mode RPVC as we can assume the adversary being authorised to evaluate a challenge computation. We consider a challenger $\mathcal{C}$ playing the IND-sHRSS game (Figure 5.5) with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ in turn acts as a challenger for $\mathcal{A}_{\text{HPVC}}$. Given the above parameters the entities interact in the following way.

1. $\mathcal{A}_{\text{HPVC}}$ declares its choice of challenge parameters $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode})$ including a set of labels $L_{F,X^\star}$ and the mode of computation `mode` detailing in which mode the challenge needs to be generated.

2. $\mathcal{A}_{\text{ABE}}$ must send a challenge attribute set and policy $(\widetilde{\omega}, \widetilde{\mathbb{S}})$, and a challenge time period $\widetilde{t}$ to the challenger as $\mathcal{A}_{\text{ABE}}$'s challenge input for the IND-sHRSS game of the rkDP-ABE scheme. Recall from Table 5.1 that in case `mode` = VDC the challenge subjective policy $\mathbb{S}^\star$ corresponds to the function F and the subjective attribute set ψ corresponds to the challenge input data $X^\star \subseteq D_i$. Also following Table 5.1, in case `mode` = RPVC the challenge objective policy $\mathbb{O}^\star$ corresponds to the function F and the objective attribute set ω corresponds to the challenge input data $X^\star$. In either mode, the other challenge input parameters correspond to either dummy attributes or dummy policies, and these dummy policies are trivially satisfied by the dummy attributes (cf. Section 5.5). As usual, $\mathcal{A}_{\text{ABE}}$ computes $r = F(X^\star)$.

   - If `mode` = VDC, we need to set the challenge input pair to the IND-sHRSS game of the rkDP-ABE scheme such that the pair is not satisfied by the challenge input $X^\star$ and thus need to set $\widetilde{\mathbb{S}}$ to be unsatisfied.
     - If $r = 1$: we set

       $$\widetilde{\omega} = A_{\omega^\star} = \{T_{\text{O}}\},$$

       and

       $$\widetilde{\mathbb{S}} = \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j = \overline{F} \wedge \{l(x_{i,j})\}_{x_{i,j} \in X^\star}.$$

     - If $r = 0$: we set

       $$\widetilde{\omega} = A_{\omega^\star} = \{T_{\text{O}}\},$$

       and

       $$\widetilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j = F \wedge \{l(x_{i,j})\}_{x_{i,j} \in X^\star}.$$

   - If `mode` = RPVC, then we set

     $$\widetilde{\omega} = A_{\omega^\star} = A_{X^\star},$$

     and

     $$\widetilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j = \{\{T_{\text{S}}\}\} \wedge \{l(F)\}.$$

   Finally, $\mathcal{A}_{\text{ABE}}$ also sets its challenge $(\widetilde{\omega}, \widetilde{\mathbb{S}})$ for the time period $\widetilde{t} = 1$ for the IND-sHRSS game and sends all challenge parameters to the challenger $\mathcal{C}$.

3. $\mathcal{C}$ runs the DPABE.Setup algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.

4. $\mathcal{A}_{\text{ABE}}$ initialises its target revocation list $\overline{R}$ which is initially empty and sends it to $\mathcal{C}$, and simulates running HPVC.Setup such that the outcome is consistent with the previously generated mpk_{ABE}. If mode = VDC, it sets $mpk^0_{\text{ABE}} \leftarrow mpk_{\text{ABE}}$ as provided by the challenger and implicitly sets $msk^0_{\text{ABE}} \leftarrow msk_{\text{ABE}}$. Note that any use of msk^0_{ABE} will be simulated using oracle calls to the challenger. If mode = RPVC, it sets $mpk^r_{\text{ABE}} \leftarrow mpk_{\text{ABE}}$ as issued by $\mathcal{C}$, and implicitly sets $msk^r_{\text{ABE}} \leftarrow msk_{\text{ABE}}$ to be the key held by the challenger. In either case, $\mathcal{A}_{\text{ABE}}$ executes DPABE.Setup itself to generate a second DP-ABE system.

5. $\mathcal{A}_{\text{ABE}}$ runs HPVC.FnInit as detailed in Algorithm 2.

6. $\mathcal{A}_{\text{ABE}}$ must generate a challenge problem instance for $\mathcal{A}_{\text{HPVC}}$ as the output of HPVC.ProbGen. To do so, $\mathcal{A}_{\text{ABE}}$ samples three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. $\mathcal{A}_{\text{ABE}}$ provides m_0 and m_1 as its choice of challenge to $\mathcal{C}$, and receives back the encryption, $ct^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$, where $b^\star$ is chosen by the challenger), under the challenge attribute set and policy $(\widetilde{\omega}, \widetilde{\mathbb{S}})$ and challenge time period $\widetilde{t}$. More formally, $ct^\star \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_{b^\star}, (\widetilde{\omega}, \widetilde{\mathbb{S}}), \widetilde{t}, mpk_{\text{ABE}})$. It needs to assign $ct^\star$ to be one of the ciphertexts c or c' that form the challenge problem instance (encoded input) $\sigma_{F,X^\star}$ using the correct ABE system parameters. $\mathcal{A}_{\text{ABE}}$ chooses a random bit $s \xleftarrow{\$} \{0,1\}$ which intuitively corresponds to its guess for the challenger's choice of $b^\star$.

   - If mode = VDC, we need to distinguish the following cases.
     - If $r = 1$, $\mathcal{A}_{\text{ABE}}$ generates

$$c \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m', \widetilde{\omega}, \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\text{ABE}})$$

       and sets $c' = ct^\star$. It also sets $vk = g(m')$ and $vk' = g(m_s)$.
     - If $r = 0$, $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and generates

$$c' \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\text{ABE}}).$$

       It also sets $vk = g(m_s)$ and $vk' = g(m')$.
   - If mode = RPVC, $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and generates

$$c' \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l \in L_{F,X^\star}} l, \widetilde{t}, mpk^1_{\text{ABE}}).$$

It also sets $vk = g(m')$ and $vk' = g(m_s)$.

Finally, $\mathcal{A}_{\text{ABE}}$ sets $\sigma_{F,X^\star} = (c, c')$ and $vk_{F,X^\star} = (vk, vk', L_{\text{Reg}})$.

7. $\mathcal{A}_{\text{HPVC}}$ receives all outputs from the above HPVC.ProbGen algorithm, and then is provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds in the following way:

   - HPVC.FnInit$(\cdot, mk, pp)$ and HPVC.Register$(\cdot, mk, pp)$ are executed as specified in Algorithms 2 and 3.
   - HPVC.Certify$(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$: in order to generate an evaluation key for the queried parameters, $\mathcal{A}_{\text{ABE}}$ needs to request queries to the KeyGen oracle in the rkDP-ABE game formalised in Figure 5.5. $\mathcal{A}_{\text{ABE}}$ updates first the usual list entries and then sets $\mathbb{O}' = \mathbb{O}$ and $\psi' = A_\psi \cup \bigcup_{l_j \in L_i} l_j$ and requests an oracle query to the challenger for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}', \psi'), mk, pp)$ as specified in Figure 5.5. The challenger shall generate a rkDP-ABE decryption key if and only if $\widetilde{\omega} \notin \mathbb{O}'$ or $\psi' \notin \widetilde{\mathbb{S}}$ or $S_i \in \overline{R}$. Note that $S_i \notin \overline{R}$ is fulfilled since we chose $\overline{R}$ to be empty. By construction, the condition $\psi' \in \widetilde{\mathbb{S}}$ is satisfied only if the labels $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$. As specified above, if the labels do not satisfy this relation then $\psi' \notin \widetilde{\mathbb{S}}$ and the challenger may generate the key, which $\mathcal{A}_{\text{ABE}}$ will receive as sk^0_{ABE}.

     If, on the other hand, the labels do satisfy this relation then we have the following cases depending on the chosen mode.

     - If $\texttt{mode}$ = VDC, then from the above relation $\{l_k\}_{l_k \in L_{F,X^\star}} \subseteq \{l_j\}_{l_j \in L_i}$ it follows that $\{l(x_{i,k})\}_{x_{i,k} \in X^\star} \subseteq \{l(x_{i,j})\}_{x_{i,j} \in D_i}$ and thus it follows that $X^\star \subseteq D_i$. Thus, this means that by the uniqueness of the labels within the system, $\mathcal{A}_{\text{HPVC}}$ has requested an evaluation key for a superset of the challenge input set $X^\star$, i.e. the set D_i that contains the challenge input set $X^\star$ and possibly some more additional data points. If $X^\star \subseteq D_i$, then the data set D_i must satisfy either F or $\overline{F}$ in order to satisfy $\widetilde{\mathbb{S}}$. However, $\widetilde{\mathbb{S}}$ was chosen in such a way that it is not satisfied by $X^\star$ and thus also not by D_i. Hence, the challenger may generate a valid key which $\mathcal{A}_{\text{ABE}}$ stores as sk^0_{ABE}.
     - If $\texttt{mode}$ = RPVC, then (as specified in Table 5.1) both sets L_i and $L_{F,X^\star}$ are singleton sets. Thus, from $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$ it follows that $L_i = L_{F,X^\star} = \{l(F)\}$. By the uniqueness of the labels within the system, it then follows that $\mathbb{O} = \mathbb{O}^\star$ which means that $\mathcal{A}_{\text{HPVC}}$ has requested an evaluation key for the challenge function F. However, in step 4, the challenger got assigned the ABE system with master secret key msk^r_{ABE} such that $\mathbb{O}^\star$ is not satisfied by the challenge input $\widetilde{\omega}$. Therefore, also $\mathbb{O}'$ is not satisfied either by the challenge input $\widetilde{\omega}$ and

hence the challenger may generate a valid key which $\mathcal{A}_{\text{ABE}}$ stores as sk^r_{ABE}.

$\mathcal{A}_{\text{ABE}}$ needs further to make queries to a KeyUpdate oracle $\mathcal{O}^{\mathsf{KeyUpdate}}$ to the challenger in order to obtain an update key. The challenger returns a valid key if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Observe that the second condition is satisfied since $\overline{R} = \epsilon$ and hence is a subset of Q_{Rev}. Hence a challenger may generate a valid update key.

Also if mode = RPVC, then $\mathcal{A}_{\text{ABE}}$ additionally generates a secret key sk^{1-r}_{ABE} by itself using the parameters of the second DP-ABE system which it owns for the pair $(\overline{\mathbb{O}}, \overline{\psi})$.

- HPVC.Revoke$(\tau_{\theta_{F(X)}}, mk, pp)$: whenever a Revoke query is requested, $\mathcal{A}_{\text{ABE}}$ executes Algorithm 8 as specified except it requires to make a KeyUpdate oracle query to the challenger for the update key that relates to the ABE system owned by $\mathcal{C}$. If mode = RPVC, this is the key $uk^r_{L_{\text{Rev}},t}$, and if mode = VDC, this is the key $uk^0_{L_{\text{Rev}},t}$. The challenger may create a valid update key if and only if $t \neq q_t$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Since $\overline{R}$ was defined to be an empty list and hence is a subset of Q_{Rev} the challenger may always return a valid update key.

Eventually $\mathcal{A}_{\text{HPVC}}$ finishes its query phase and outputs a guess $\theta^\star$ which it believes to be a valid forgery.

8. $\mathcal{A}_{\text{ABE}}$ parses the guess $\theta^\star$ as (d, d', S_i, γ). One of the values d and d' will be $\perp$ (by construction) and we denote the other value (non-$\perp$) by Y. Observe that, since $\mathcal{A}_{\text{HPVC}}$ is assumed to be a successful adversary against selective public verifiability, the non-$\perp$ value, Y, that it will return will be the plaintext m_s since the challenge access structure was always set to be unsatisfied on the challenge input. Thus, if $g(Y) = g(m_s)$, $\mathcal{A}_{\text{ABE}}$ outputs a guess $b' = s$ and otherwise guesses $b' = (1 - s)$.

Notice that if $s = b^\star$ (the challenge bit chosen by $\mathcal{C}$ in the IND-sHRSS game in Figure 5.2), then the distribution of the above coincides with **Game 0** since the verification key comprises $g(m')$ and $g(m_s)$ where m' and m_s are the two plaintexts corresponding to the ciphertexts of the encoded input for which $\mathcal{A}_{\text{HPVC}}$ recovers exactly one. Otherwise, if $s = 1 - b^\star$ then the distribution coincides with **Game 1** since the verification key comprises the one-way function g applied to a legitimate message m' and a random message $m_{1-b^\star}$ that is unrelated to both ciphertexts.

Now, we consider the advantage of this constructed $\mathcal{A}_{\text{ABE}}$ playing the IND-sHRSS game for the revocable key DP-ABE scheme. Recall that by assumption, $\mathcal{A}_{\text{HPVC}}$ has

a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1**, that is

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right| \geqslant \delta$$

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right]$ denotes the output of running $\mathcal{A}_{\mathrm{HPVC}}$ in **Game i**.

Now we derive the probability of $\mathcal{A}_{\mathrm{ABE}}$ guessing $b^\star$ correctly. It follows:

$$\begin{aligned}
\Pr[b' = b^\star] &= \Pr[s = b^\star]\Pr[b' = b^\star | s = b^\star] + \Pr[s \neq b^\star]\Pr[b' = b^\star | s \neq b^\star] \\
&= \frac{1}{2}\Pr[g(Y) = g(m_s) | s = b^\star] + \frac{1}{2}\Pr[g(Y) \neq g(m_s) | s \neq b^\star] \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] \\
&\quad + \frac{1}{2}\left(1 - \Pr[g(Y) = g(m_s) | s \neq b^\star]\right) \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] \\
&\quad + \frac{1}{2}\left(1 - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right]\right. \\
&\quad \left. - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{HPVC}}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right] \to 1\right] + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\mathrm{ABE}}} &\geqslant \left|\Pr[b' = b^\star] - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&= \frac{\delta}{2}
\end{aligned}$$

Since δ is assumed non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\mathrm{HPVC}}$ has advantage δ at distinguishing these games then $\mathcal{A}_{\mathrm{ABE}}$ can win the IND-sHRSS game with non-negligible probability. Thus since we assumed the ABE scheme to be IND-sHRSS secure, we conclude that $\mathcal{A}_{\mathrm{HPVC}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is to simply set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game (Figure 2.8). We argue that the adversary has no distinguishing advantage between these games since the new value is independent of

anything else in the system except the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{\text{HPVC}}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{\text{ABE}}$).

Final Proof. We now show that using $\mathcal{A}_{\text{HPVC}}$ in **Game 2**, $\mathcal{A}_{\text{ABE}}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ $\mathcal{A}_{\text{ABE}}$ can recover w. Specifically, during HPVC.ProbGen, $\mathcal{A}_{\text{ABE}}$ chooses the messages as follows:

- if $F(X^{\star}) = 1$, we implicitly set m_1 to be w and the corresponding verification key component to be $z = g(w)$. We randomly choose m_0 from the message space and compute the remainder of the verification key as usual.
- if $F(X^{\star}) = 0$, we implicitly set m_0 to be w and set the verification key component to $z = g(w)$. m_1 is chosen randomly from the message space and the remainder of the verification key computed as usual.

Now, since $\mathcal{A}_{\text{HPVC}}$ is assumed to be successful, it will output a forgery comprising the plaintext that was encrypted under the unsatisfied function (F or $\overline{F}$) that evaluates to 0. By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{\text{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\text{HPVC}}$ has against the selective public verifiability game.

We conclude that if the rkDP-ABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $\mathcal{HPVC}$ as defined by Algorithms 1–8 is secure in the sense of selective public verifiability. □

5.6.2 Selective, Semi-static Revocation

Lemma 5.12. *The $\mathcal{HPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective, semi-static revocation (Figure 5.3) under the same assumptions as in Theorem 5.10.*

Proof. Note that this proof follows in a similar manner to the proof presented in Section 3.5.2 and we mainly update the notation to accommodate an $\mathcal{HPVC}$ scheme.

In this proof, we aim to perform a reduction from the the selective, semi-static revocation game (Figure 5.3) to the IND-sHRSS security of the underlying revocable key DP-ABE scheme (Figure 5.5). We wish the prove this reduction by achieving a contradiction and therefore we assume that $\mathcal{A}_{\text{HPVC}}$ is an adversary with non-negligible probability against the selective, semi-static revocation game when instantiated by Algorithms 1–8, and making q_t Revoke queries. We show that we can construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{HPVC}}$ as a sub-routine to break the IND-sHRSS security

of the indirectly revocable key DP-ABE scheme. Note that as in the previous proof, we only focus on the modes RPVC and VDC, and the mode RPVC-AC can be seen as a special case of the mode RPVC as we can assume the adversary being authorised to evaluate a challenge computation. Let $\mathcal{C}$ be a challenger playing the IND-sHRSS game with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ acts as a challenger for $\mathcal{A}_{\text{HPVC}}$.

1. $\mathcal{A}_{\text{RPVC}}$ declares its choice of challenge input parameters $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode})$ for a challenge computation $F(X^\star)$ including a set of labels $L_{F,X^\star}$ and the mode of computation `mode` detailing in which mode the challenge needs to be generated.

2. $\mathcal{A}_{\text{ABE}}$ initialises an (empty) list Q_{Rev} of currently revoked entities and sets the current time period $t = 1$. Next, $\mathcal{A}_{\text{ABE}}$ needs to form its own challenge input for the IND-sHRSS game. $\mathcal{A}_{\text{ABE}}$ sets its challenge for the time period $\widetilde{t} = q_t$, and it forms $\widetilde{\omega} = A_{\omega^\star}$ and $\widetilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j$. Finally, it sends $(\widetilde{t}, (\widetilde{\omega}, \widetilde{\mathbb{S}}))$ to the challenger.

3. $\mathcal{C}$ runs the $\mathsf{DPABE.Setup}$ algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.

4. $\mathcal{A}_{\text{ABE}}$ simulates running $\mathsf{HPVC.Setup}$ such that the outcome is consistent with the previously generated mpk_{ABE} from $\mathcal{C}$. It executes the algorithm as detailed with the exception of line 2, since msk^0_{ABE} and mpk^0_{ABE} were already generated by the challenger.

5. $\mathcal{A}_{\text{ABE}}$ runs $\mathsf{HPVC.FnInit}$ as detailed in Algorithm 2.

6. $\mathcal{A}_{\text{HPVC}}$ chooses a challenge revocation list $\overline{R}$, which $\mathcal{A}_{\text{ABE}}$ forwards to $\mathcal{C}$.

7. $\mathcal{A}_{\text{HPVC}}$ is provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds in the following way:

   - $\mathsf{HPVC.FnInit}(\cdot, mk, pp)$ and $\mathsf{HPVC.Register}(\cdot, mk, pp)$ are executed as specified in Algorithms 2 and 3.
   - Queries of the form $\mathsf{HPVC.Certify}(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the $\mathsf{Certify}$ oracle as specified in Figure 5.3. $\mathcal{A}_{\text{ABE}}$ executes Algorithm 4 as detailed except lines 6 and 7, as these rely on the master secret key msk^0_{ABE} held by the challenger. In order to simulate line 6, $\mathcal{A}_{\text{ABE}}$ requires to make a KeyGen oracle query of the form $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_k \in L_i} l_k), msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. The challenger responds by running the KeyGen oracle as detailed in Figure 5.5 which returns a valid key if and only if $\widetilde{\omega} \notin \mathbb{O}$ or $A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \widetilde{\mathbb{S}}$ or $S_i \in \overline{R}$. Now if $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \widetilde{\mathbb{S}})$ is fulfilled then the challenger can return a valid

decryption key. By construction, we observe that the condition $\psi \in \widetilde{\mathbb{S}}$ is satisfied only if $\{l_k\}_{l_k \in L_i} \supseteq \{l_j\}_{l_j \in L_{F,X^\star}}$. By the uniqueness of the label within the system this implies $L_{F,X^\star} \subseteq L_i$. However, in this case, the first condition in the "if" statement in the Certify oracle in Figure 5.3 is satisfied and thus $\mathcal{A}_{\text{ABE}}$ would have returned $\perp$ without querying KeyGen if $S_i \notin \overline{R}$ to avoid certifying $\mathcal{A}_{\text{HPVC}}$ for the challenge computation. If $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \widetilde{\mathbb{S}})$ is satisfied at the point of making a KeyGen query, then $S_i \in \overline{R}$, and thus the challenger can respond to all queries made to it during this phase with a valid key.

In order to simulate line 7, $\mathcal{A}_{\text{ABE}}$ makes a query to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. Here the challenger responds as detailed in Figure 5.5 which returns a valid update key if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Recall that $\mathcal{A}_{\text{ABE}}$ chose $\widetilde{t} = q_t$, and at the point of calling the KeyUpdate oracle, the list of currently revoked entities corresponds to $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S_i$. Therefore, if the challenger returns $\perp$ in response to this query, then $\mathcal{A}_{\text{ABE}}$ would already have returned $\perp$ as a response to the Certify oracle (Figure 5.3) as a result of the second condition in the "if" statement. Hence, for all queries made to the challenger, a valid update key is returned.

- Queries of the form $\mathsf{HPVC.Revoke}(\tau_{\theta_{F(X^\star)}}, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the Revoke oracle as specified in Figure 5.3. In order to simulate running the algorithm, $\mathcal{A}_{\text{ABE}}$ executes Algorithm 8 with the exception of line 6. Here $\mathcal{A}_{\text{ABE}}$ is required to make KeyUpdate oracle calls to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. Note that the Revoke oracle in Figure 5.3 returns $\perp$ if $t = q_t$ and $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S_i$. This corresponds directly to the conditions that $\mathcal{C}$ cannot form a valid update key through a KeyUpdate oracle call (Figure 5.5) since $t = \widetilde{t}$ and $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, since S_i was already removed from the list of currently revoked entities Q_{Rev}, $\mathcal{C}$ can form a valid update key and $\mathcal{A}_{\text{ABE}}$ can simulate the remainder of the algorithm.

8. Eventually (after q_t Revoke queries), $\mathcal{A}_{\text{HPVC}}$ finishes the query phase. $\mathcal{A}_{\text{ABE}}$ checks if $\mathcal{A}_{\text{HPVC}}$ has made suitable Revoke queries. If there exists an entity in $\overline{R}$ that is not currently revoked (listed in Q_{Rev}), it returns 0 and aborts immediately.

9. $\mathcal{A}_{\text{ABE}}$ must now generate a challenge for $\mathcal{A}_{\text{HPVC}}$. $\mathcal{A}_{\text{ABE}}$ chooses three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. It then sends m_0 and m_1 to $\mathcal{C}$ as its choice of challenge for the IND-sHRSS game. $\mathcal{C}$ chooses a random bit $b^\star \xleftarrow{\$} \{0,1\}$ and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{b^\star}, \widetilde{\omega}, \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\text{ABE}})$. $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and generates depending on the chosen mode the second ciphertext. If $\texttt{mode} = \text{VDC}$, then $c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\text{ABE}})$. In case $\texttt{mode} =$

RPVC, then $c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^1_{\mathrm{ABE}})$. Finally, $\mathcal{A}_{\mathrm{ABE}}$ forms the challenge problem instance $\sigma^\star = (c, c')$. $\mathcal{A}_{\mathrm{ABE}}$ selects a bit $s \xleftarrow{\$} \{0,1\}$ and forms the verification key as $vk^\star = (g(m_s), g(m'), L_{\mathrm{Reg}})$. Note that s intuitively corresponds to $\mathcal{A}_{\mathrm{ABE}}$'s guess for $b^\star$.

10. $\mathcal{A}_{\mathrm{HPVC}}$ receives the resulting parameters from $\mathsf{ProbGen}$ and is again provided with oracle access. These queries are handled in the same way as previously, and eventually $\mathcal{A}_{\mathrm{HPVC}}$ outputs its guess $\theta^\star$.

11. Let Y be the non-$\perp$ plaintext returned in $\theta^\star$. If $g(Y) = g(m_s)$, $\mathcal{A}_{\mathrm{ABE}}$ guesses $b' = s$. Else, $\mathcal{A}_{\mathrm{ABE}}$ guesses $b' = 1 - s$.

    If $g(Y) = g(m')$, $\mathcal{A}_{\mathrm{ABE}}$ makes a random guess $b' = \tilde{b} \xleftarrow{\$} \{0,1\}$ since $\mathcal{A}_{\mathrm{HPVC}}$ did not forge a result for either m_0 or m_1 and therefore is of no use for $\mathcal{A}_{\mathrm{ABE}}$ in order to break the IND-sHRSS game.

Now we consider the advantage of $\mathcal{A}_{\mathrm{ABE}}$ playing the IND-sHRSS game. By assumption, $\mathcal{A}_{\mathrm{HPVC}}$ has a non-negligible advantage δ against the selective, semi-static revocation game. It follows

$$
\begin{aligned}
\Pr[b' = b^\star] &= \Pr[b' = b^\star | s = b^\star] \Pr[s = b^\star] + \Pr[b' = b^\star | 1 - s = b^\star] \Pr[1 - s = b^\star] \\
&\quad + \Pr[b' = b^\star | \tilde{b} = b^\star] \Pr[\tilde{b} = b^\star] \\
&= \Pr[g(Y) = g(m_s) | s = b^\star] \Pr[s = b^\star] \\
&\quad + \Pr[g(Y) \neq g(m_s) | 1 - s = b^\star] \Pr[1 - s = b^\star] \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \Pr[\tilde{b} = b^\star] \\
&= \frac{1}{2} \Pr[g(Y) = g(m_s) | s = b^\star] + \frac{1}{2} \Pr[g(Y) \neq g(m_s) | 1 - s = b^\star] \\
&\quad + \frac{1}{2} \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \\
&= \frac{1}{2} \Big(\Pr[g(Y) = g(m_s) | s = b^\star] + (1 - \Pr[g(Y) = g(m_s) | 1 - s = b^\star]) \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \Big) \\
&= \frac{1}{2} \Big(\Pr[g(Y) = g(m_s) | s = b^\star] - \Pr[g(Y) = g(m_s) | 1 - s = b^\star] \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] + 1 \Big) \\
&= \frac{1}{2}(\delta + 1).
\end{aligned}
$$

Hence,

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}_{\mathrm{ABE}}} &\geqslant \left|\Pr[b' = b^\star] - \frac{1}{2}\right| \\ &\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\ &= \frac{\delta}{2}.\end{aligned}$$

Since δ is non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\mathrm{HPVC}}$ has advantage δ at breaking the selective, semi-static revocation game then $\mathcal{A}_{\mathrm{ABE}}$ can win the IND-sHRSS game with non-negligible probability. However, since the indirectly revocable key DP-ABE scheme was assumed to be IND-sHRSS secure, such an adversary $\mathcal{A}_{\mathrm{HPVC}}$ cannot exist. Therefore, we conclude that if the revocable key DP-ABE scheme is IND-sHRSS secure then $\mathcal{HPVC}$ as instantiated by Algorithms 1–8 is secure in the sense of selective, semi-static revocation. □

5.6.3 Selective Authorised Computation

Lemma 5.13. *The $\mathcal{HPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective authorised computation (Figure 5.4) under the same assumptions as in Theorem 5.10.*

Proof. In this proof, we aim to perform a reduction from the selective authorised computation game (Figure 5.4) to the IND-sHRSS security of the underlying revocable key DP-ABE scheme (Figure 5.5). We wish to prove this reduction by achieving a contradiction and therefore we assume that $\mathcal{A}_{\mathrm{HPVC}}$ is an adversary with non-negligible probability against the selective authorised computation game when instantiated by Algorithms 1–8. We show that we can construct an adversary $\mathcal{A}_{\mathrm{ABE}}$ that uses $\mathcal{A}_{\mathrm{HPVC}}$ as a subroutine to break the IND-sHRSS security of the indirectly revocable key DP-ABE scheme. Note that the notion of selective authorised computation is only meaningful as long as the system is run in the RPVC-AC mode. Let $\mathcal{C}$ be a challenger playing the IND-sHRSS game with $\mathcal{A}_{\mathrm{ABE}}$, and $\mathcal{A}_{\mathrm{ABE}}$ acts as a challenger for $\mathcal{A}_{\mathrm{HPVC}}$.

1. $\mathcal{A}_{\mathrm{HPVC}}$ begins by declaring its choice of challenge input parameters for the RPVC-AC mode consisting of F, $X^\star$, the authorisation policy P and the function label $\{l(F)\}$.

2. $\mathcal{A}_{\mathrm{ABE}}$ needs to form its own challenge input for the IND-sHRSS game. Thus, $\mathcal{A}_{\mathrm{ABE}}$ sets its challenge for the time period $\widetilde{t} = 1$, and it forms $\widetilde{\omega} = A_{X^\star}$ and $\widetilde{\mathbb{S}} = P \wedge \{l(F)\}$. Finally, it sends $(\widetilde{t}, (\widetilde{\omega}, \widetilde{\mathbb{S}}))$ to the challenger.

3. $\mathcal{C}$ runs the DPABE.Setup algorithm to generate $mpk_{\mathrm{ABE}}, msk_{\mathrm{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\mathrm{ABE}}$.

4. $\mathcal{A}_{\text{ABE}}$ simulates running $\mathsf{HPVC.Setup}$ such that the outcome is consistent with the previously generated mpk_{ABE} from $\mathcal{C}$. It executes the algorithm as detailed with the exception of line 2, since msk^0_{ABE} and mpk^0_{ABE} were already generated by the challenger. $\mathcal{A}_{\text{ABE}}$ chooses an empty list of currently revoked entities $\overline{R}$ and sends it to the challenger.

5. $\mathcal{A}_{\text{ABE}}$ runs $\mathsf{HPVC.FnInit}$ as detailed in Algorithm 2.

6. $\mathcal{A}_{\text{ABE}}$ must now generate a challenge for $\mathcal{A}_{\text{HPVC}}$. $\mathcal{A}_{\text{ABE}}$ chooses three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. It then sends m_0 and m_1 to $\mathcal{C}$ as its choice of challenge for the IND-sHRSS game. $\mathcal{C}$ chooses a random bit $b^\star \xleftarrow{\$} \{0,1\}$ and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{b^\star}, \widetilde{\omega}, P \wedge \{l(F)\}, \widetilde{t}, mpk^0_{\text{ABE}})$. $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and generates itself the second ciphertext by encrypting m' as $c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \widetilde{\omega}, \overline{P} \wedge \{l(F)\}, \widetilde{t}, mpk^1_{\text{ABE}})$. Finally, $\mathcal{A}_{\text{ABE}}$ forms the challenge problem instance $\sigma^\star = (c, c')$. $\mathcal{A}_{\text{ABE}}$ selects a bit $s \xleftarrow{\$} \{0,1\}$ and forms the verification key as $vk^\star = (g(m_s), g(m'), L_{\text{Reg}})$. Note that s intuitively corresponds to $\mathcal{A}_{\text{ABE}}$'s guess for $b^\star$.

7. $\mathcal{A}_{\text{HPVC}}$ receives the resulting parameters from $\mathsf{ProbGen}$ and is provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds in the following way:

   - $\mathsf{HPVC.FnInit}(\cdot, mk, pp)$ and $\mathsf{HPVC.Register}(\cdot, mk, pp)$ are executed as specified in Algorithms 2 and 3.
   - Queries of the form $\mathsf{HPVC.Certify}(\text{RPVC-AC}, S_i, (F, \psi), \{l(F)\}, \mathcal{F}_i, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the $\mathsf{Certify}$ oracle as specified in Figure 5.4. In case the queried set of subjective attributes ψ satisfy the challenge authorisation policy then $\mathcal{A}_{\text{ABE}}$ returns $\perp$. Otherwise, $\mathcal{A}_{\text{ABE}}$ executes Algorithm 4 as detailed with the exception in lines 6 and 7, as these rely on the master secret key msk^0_{ABE} held by the challenger. In order to simulate line 6, $\mathcal{A}_{\text{ABE}}$ requires to make a KeyGen oracle query of the form $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (F, A_\psi \cup \bigcup_{l_k \in L_i} l_k), msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. The challenger responds by running the KeyGen oracle as detailed in Figure 5.5 which returns a valid key if and only if $\widetilde{\omega} \notin \mathbb{O}$ or $A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \widetilde{\mathbb{S}}$ or $S_i \in \overline{R}$. However, for the query to have been made to KeyGen, $\mathcal{A}_{\text{ABE}}$ must not have returned $\perp$ in the $\mathsf{Certify}$ oracle request in Figure 5.4 and therefore $\psi \notin P$, and hence $\psi \notin \widetilde{\mathbb{S}}$. Therefore, the challenger can always return a valid decryption key sk^0_{ABE}.

     In order to simulate line 7 in the $\mathsf{Certify}$ algorithm, $\mathcal{A}_{\text{ABE}}$ makes a query to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. Here the challenger responds as detailed in Figure 5.5 which returns a valid update key if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Since $\overline{R}$ was initially chosen to be empty and thus $\overline{R} \subseteq Q_{\text{Rev}}$ for any Q_{Rev}. Therefore, the challenger can create a valid update key.

- Queries of the form $\mathsf{HPVC.Revoke}(\tau_{\theta_{F(X^\star)}}, mk, pp)$ are handled by $\mathcal{A}_{\mathrm{ABE}}$ executing Algorithm 8 with the exception of line 6. Here $\mathcal{A}_{\mathrm{ABE}}$ is required to make $\mathsf{KeyUpdate}$ oracle calls to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\mathrm{Rev}}, t, msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$. The challenger returns a valid update key through a $\mathsf{KeyUpdate}$ oracle call (Figure 5.5) if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\mathrm{Rev}}$. Since $\overline{R}$ was initially chosen to be empty and thus $\overline{R} \subseteq R$ for any R and in particular L_{Rev}. Therefore, the challenger can always create a valid update key.

8. Eventually $\mathcal{A}_{\mathrm{HPVC}}$ finishes its oracle query phase and outputs its guess $\theta^\star$ which corresponds to the result of $F(X^\star)$ protected by an authorisation policy P. Note that $\mathcal{A}_{\mathrm{HPVC}}$ never received a key for a set of authorisation attributes $s \in P$.

9. As $\theta^\star$ should appear valid, by construction it should contain a non-$\perp$ plaintext which we denote by Y. If $g(Y) = g(m_s)$, $\mathcal{A}_{\mathrm{ABE}}$ guesses $b' = s$. Else, $\mathcal{A}_{\mathrm{ABE}}$ guesses $b' = 1 - s$.

   If $g(Y) = g(m')$, $\mathcal{A}_{\mathrm{ABE}}$ makes a random guess $b' = \tilde{b} \xleftarrow{\$} \{0,1\}$ since $\mathcal{A}_{\mathrm{HPVC}}$ did not forge a result for either m_0 or m_1 and therefore is of no use for $\mathcal{A}_{\mathrm{ABE}}$ in order to break the IND-sHRSS game.

Now we consider the advantage of $\mathcal{A}_{\mathrm{ABE}}$ playing the IND-sHRSS game. By assumption, $\mathcal{A}_{\mathrm{HPVC}}$ has a non-negligible advantage δ against the selective authorised computation game. It follows

$$
\begin{aligned}
\Pr[b' = b^\star] &= \Pr[b' = b^\star | s = b^\star] \Pr[s = b^\star] + \Pr[b' = b^\star | 1 - s = b^\star] \Pr[1 - s = b^\star] \\
&\quad + \Pr[b' = b^\star | \tilde{b} = b^\star] \Pr[\tilde{b} = b^\star] \\
&= \Pr[g(Y) = g(m_s) | s = b^\star] \Pr[s = b^\star] \\
&\quad + \Pr[g(Y) \neq g(m_s) | 1 - s = b^\star] \Pr[1 - s = b^\star] \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \Pr[\tilde{b} = b^\star] \\
&= \frac{1}{2} \Pr[g(Y) = g(m_s) | s = b^\star] + \frac{1}{2} \Pr[g(Y) \neq g(m_s) | 1 - s = b^\star] \\
&\quad + \frac{1}{2} \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \\
&= \frac{1}{2} \Big(\Pr[g(Y) = g(m_s) | s = b^\star] + (1 - \Pr[g(Y) = g(m_s) | 1 - s = b^\star]) \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] \Big) \\
&= \frac{1}{2} \Big(\Pr[g(Y) = g(m_s) | s = b^\star] - \Pr[g(Y) = g(m_s) | 1 - s = b^\star] \\
&\quad + \Pr[g(Y) = g(m') | \tilde{b} = b^\star] + 1 \Big) \\
&= \frac{1}{2}(\delta + 1).
\end{aligned}
$$

Hence,

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}_{\mathrm{ABE}}} &\geqslant \left|\Pr[b' = b^\star] - \frac{1}{2}\right| \\ &\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\ &= \frac{\delta}{2}.\end{aligned}$$

Since δ is non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\mathrm{HPVC}}$ has advantage δ at breaking the selective authorised computation game then $\mathcal{A}_{\mathrm{ABE}}$ can win the IND-sHRSS game with non-negligible probability. However, since the indirectly revocable key DP-ABE scheme was assumed to be IND-sHRSS secure, such an adversary $\mathcal{A}_{\mathrm{HPVC}}$ cannot exist. Therefore, we conclude that if the revocable key DP-ABE scheme is IND-sHRSS secure then $\mathcal{HPVC}$ as instantiated by Algorithms 1–8 is secure in the sense of selective authorised computations. □

5.7 Conclusion

In this chapter, we have introduced an umbrella notion for PVC called hybrid publicly verifiable outsourced computation. HPVC supports three different modes of computation (RPVC, VDC, and RPVC-AC) that we have introduced throughout this thesis so far, and thus meets diverse user requirements of a large multi-user system. In other words, our model enables entities to request computations from other users, restrict which entities can perform computations on their behalf, perform computations for other users, and make data available for queries from other users, all in a verifiable manner. We provide an instantiation of HPVC built from a novel use of DP-ABE. DP-ABE has previously attracted relatively little attention in the literature, which we believe to be primarily due to its applications being less obvious than for the single-policy ABE schemes. Whilst KP- and CP-ABE are generally considered in the context of cryptographic access control, it is unclear that the policies enforced by DP-ABE are natural choices for access control. Thus an interesting side-effect of this chapter is to show that additional applications for DP-ABE do exist.

In future work, one may further investigate our revocable DP-ABE scheme to compare the efficiency of revoking the key- and ciphertext-policies. Furthermore, it would be beneficial to investigate techniques in order to enable the stored data at the server to be updated.

Chapter 6

Extended Functionality in Verifiable Searchable Encryption

Contents

In this chapter we use verifiable outsourced computation techniques from the previous chapters to enable a wider family of queries in verifiable searchable encryption. We introduce a scheme based upon ciphertext-policy attribute-based encryption that permits a user to verify that search results are correct and complete. Our scheme also permits verifiable computational queries over keywords and specific data values, that go beyond the standard keyword matching queries, to allow functions such as averaging or counting operations. The results of this chapter appear in [7].

6.1 Introduction

With the emergence of cloud computing, it is now common practice for data owners to outsource their data to public servers providing storage on a pay-as-you-go basis. This can reduce the costs of data storage compared with that of running a private data centre (e.g. hardware, construction, air conditioning and security costs), making this a cost effective solution. If the server is not fully trusted and the data is of a sensitive nature, the data owner may wish to encrypt the data to ensure confidentiality. Unfortunately, this prevents the efficient retrieval of specific portions of the data as the server is unable to identify the relevant information.

The above setting has been studied intensively in recent years. For example, *searchable encryption* as introduced in Section 2.4 addresses this issue by indexing the encrypted data in such a way as to allow a server to execute a search query (formed by the data

owner or an authorised data user) over the encrypted data and return the identifiers of any file that satisfies the search query. In this chapter, we study whether techniques from the area of publicly verifiable outsourced computation lend themselves in any meaningful way to the setting of searchable encryption. We answer in the affirmative and show that techniques from publicly verifiable delegable computation (VDC) as developed in Chapter 4 can be used to build a verifiable searchable encryption (VSE) scheme that is able to handle a wider class of queries compared to current VSE approaches. Concretely, we use CP-ABE to build an *extended verifiable searchable encryption (EVSE) scheme* and embed the static data set (documents) in a server's secret key whilst the queries can only be requested by authorised users being in possession of a secret user key. In more detail, the server holds an outsourced database from the data owner which is made available for being searched over by authorised users. It is crucial to have a verification mechanism in this context since the users querying the database never possessed the data themselves and wish to obtain an assurance that the server has not cheated by returning malformed search results. Note that by employing techniques from Chapter 4, we achieve the scheme being publicly verifiable enabling anyone to verify the correctness and completeness of the query results but the scheme is not publicly delegable.
By adapting techniques from VDC to VSE, our scheme is able to perform a wider family of queries including some types of computations. We can form the following queries.

- More expressive queries: our scheme supports queries such as boolean formulas involving conjunctions, disjunctions and negations, threshold operations, polynomials, arbitrary CNF and DNF formulas, and fuzzy search.[1]
- Evaluation of computations: our scheme supports the evaluation of some computations over the encrypted data, such as averaging and counting operations. As well as assigning keywords to label data, we propose to also assign keywords representing certain data values that may be computed over (either in the form of single keywords or as a string of keywords encoding binary data).

Related Work

The scheme by Zheng et al. [148] also uses ABE primitives in their construction. This scheme is able to achieve multi-level access, where users can be restricted to searching only certain parts of the database. Keywords are grouped with respect to their access control policies, and the search time is linear in the number of groups. However, their scheme is restricted to only achieve a single keyword equality search. Sun et al. [136] introduced a dynamic SE scheme that can support conjunctive queries and is based upon ABE in order to create the indexes. Each user in the system has a separate

[1] Depending on the choice of underlying ABE scheme; see Section 6.4.1.

public key to create their indexes which can be added to the server at anytime. This scheme uses a combination of bloom filters and signatures to achieve verifiability of search results. Curtmola et al. [57] extended the SE system model to allow multiple users to query the data, using broadcast encryption to manage user access privileges. We will make use of broadcast encryption in a similar way to authorise users to query data stored at the server. Abdalla et al. [1] raised the issue of building a searchable encryption scheme that enables a user to form more advanced queries, such as Boolean formulas for keywords. The schemes presented in [37, 84, 116] also provide solutions towards this issue by enabling users to form queries for conjunctive combinations of keywords, range queries, and subset queries. We believe that our work also contributes to achieving richer query functionality on encrypted data.

Organisation of Chapter

In Section 6.2, we describe and provide a formal definition of our verifiable searchable encryption model with extended functionalities and discuss the possible types of queries that are captured by our scheme. Next, in Section 6.3, we define three notions of security that are relevant in our context, and in Section 6.4 we provide an example construction of EVSE based on ciphertext-policy attribute-based encryption. In Section 6.5, we show that our given construction is secure according to the introduced security models. We conclude this chapter in Section 6.6.

6.2 Extended Verifiable Searchable Encryption

In this section, we introduce a model for *verifiable searchable encryption* using verifiable computation techniques from the previous chapters in order to enable a wider family of queries, and some types of computations, to be performed over outsourced encrypted data with (publicly) verifiable query results.

6.2.1 Informal Overview

Our system model comprises the same entity population as previously considered in the literature, i.e. it comprises a *data owner*, a remote *storage server*, and a set of authorised *data users*. The data owner wishes to outsource documents and controls which additional users are able to query its encrypted data. Following the accepted format used in searchable encryption, queries may be formulated over *keywords* which, for example, may identify documents that are associated with a given set of keywords. However, in our setting, we also allow *computational queries* of functions in the class NC^1, which consists of Boolean functions computable by circuits of depth $\mathcal{O}(\log n)$ where each gate has a fan-in of two, over encoded data values.

As a motivating practical scenario for our work, consider workgroups within an organisation. The manager or system administrator acts as the data owner for the organisation and outsources a shared encrypted database to a remote server. Authorisation is granted by issuing a secret key to each user within the organisation which is required when creating a query token qt_Q (or trapdoor) for a particular query Q. The token is sent to the server that evaluates the query on the encoded index to generate a search result θ_Q. We allow *any* entity to verify the correctness and completeness of the result and thus achieve *public verifiability*. That is, we also permit the server to verify correctness to avoid the rejection problem (cf. Section 2.7), where a server may learn some useful information by observing if results are accepted.

Throughout this chapter, we assume a strict separation between *queriers* (the data owner and users) and the remote server storing the data. We do not enable the server to issue queries itself since it would trivially be able to learn the encoding of the index and queries. In contrast, legitimate users know this encoding and are able to obtain meaningful results.

6.2.2 Formal Definition

An EVSE scheme for a family of queries $\mathcal{F}$ begins with the data owner initialising the scheme by running Setup to create the public parameters and the master secret key. The data owner wishes to outsource data D which is considered to be a collection of n documents. Prior to outsourcing, the data owner specifies a *pre-index* for D, denoted by $\delta(D)$, which assigns a set of descriptive labels to each document, for example keywords contained in the document or specific data values that may be computed upon. The encoded form of the data, including the descriptive labels, is referred to as the *index* of D and generated by running the algorithm BuildIndex. The algorithm outputs the index $\mathcal{I}_D$ and the data owner stores it at the server.

In order to be able to form a valid query, a data owner first authorises a user by issuing her with a secret key. An authorised user executes the Query algorithm to request a search over the encrypted data from the server. The user specifies a query Q and uses the authorisation key, and the algorithm outputs the query token qt_Q for Q as well as a verification key vk_Q that enables *anyone* to verify the search result. In the Search algorithm the server S uses its index $\mathcal{I}_D$ and the encoded query token to output an encoded search result θ_Q corresponding to the actual underlying result.

Any entity is able to verify the correctness of θ_Q using vk_Q. Verification outputs the result $r = Q(\delta(D))$ indicating that the search was performed correctly, or else $r = \perp$ showing that the search result is malformed.

Given that the data owner enrolled multiple users to query its outsourced database, it may well be desirable that misbehaving users, or in terms of the above example where users may leave an organisation, are prevented from further querying the database and as such being revoked from the system. In order to revoke a user the data owner could simply initialise a new system and provide each authorised user within this system with a new secret key. However, this is an expensive solution given that the data owner needs to invest a large amount of resources to process the data and keys yet again. Thus, in our scheme revocation is based on provided states between authorised users and the server, i.e. the states serve as a proof that each entity within the system is authorised to form queries. In case a user is revoked, the data owner simply updates and re-distributes the respective states to all non-revoked entities.

Note that we slightly change algorithm names for our scheme compared to "classical" searchable encryption schemes to accommodate the blend between searchable encryption and publicly verifiable outsourced computation. Our multi-user (single-server) EVSE scheme is more formally captured in the following definition.

Definition 6.1. *An* extended verifiable searchable encryption (EVSE) scheme *comprises the following algorithms:*

1. $(mk, pp) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm is run by the data owner to initialise the system. It takes as input the security parameter λ and a universe of attributes $\mathcal{U}$ representing keywords and data points. The algorithm outputs the data owner's master secret key mk that is used for further administrative tasks and public parameters pp, both of which are provided to the remaining algorithms where required;*

2. $(\mathcal{I}_D, st_\mathrm{s}, st_\mathrm{o}) \xleftarrow{\$} \mathsf{BuildIndex}(\delta(D), l(\delta(D)), G, mk, pp)$: *this randomised algorithm is run by the data owner to output a searchable index $\mathcal{I}_D$ for the data D, as well as a server state st_s and data owner state st_o.*[2] *The algorithm uses as input the master secret key and public parameters, the pre-index of the data $\delta(D)$ and a unique label $l(\delta(D))$ representing the pre-index, as well as the set G of authorised users;*

3. $sk_\mathrm{id} \xleftarrow{\$} \mathsf{AddUser}(\mathrm{id}, G, mk, pp)$: *this randomised algorithm is run by the data owner to authorise a user* id *to enable them to form valid queries for querying the data. The algorithm authorises a user* id *by issuing them a secret key sk_id. The inputs are a user* id*, the current set of authorised users, as well as the public parameters and master secret key;*

4. $(qt_Q, vk_Q) \xleftarrow{\$} \mathsf{Query}(Q, st_\mathrm{s}, st_\mathrm{o}, sk_\mathrm{id}, pp)$: *this randomised algorithm is run by an authorised user using its secret key, public parameters and both states to generate*

[2] The data owner shares its state with each authorised user and thus $st_\mathrm{o} = st_\mathrm{u}$. For simplicity we will only use the notation st_o.

a query token qt_Q for a query Q she wishes to search for over the data. It further also outputs a public verification key vk_Q;

5. $\theta_Q \xleftarrow{\$} \mathsf{Search}(\mathcal{I}_D, qt_Q, st_\mathrm{s}, sk_\mathrm{S}, pp)$: *this randomised algorithm is run by the server to execute a query specified by the query token qt_Q on the index $\mathcal{I}_D$. It generates an encoded result θ_Q which can be returned to the querying user or published;*

6. $r \leftarrow \mathsf{Verify}(\theta_Q, vk_Q, pp)$: *this algorithm can be run by any entity. The inputs are the encoded output θ_Q produced by the server, the verification key vk_Q and the public parameters pp. The algorithm produces an output $r = Q(\delta(D))$ if the search was performed correctly, or else $r = \perp$ indicating that the search was performed incorrectly;*

7. $(st'_\mathrm{s}, st'_\mathrm{o}) \xleftarrow{\$} \mathsf{RevokeUser}(\mathrm{id}, G, mk, pp)$: *this randomised algorithm is run by the data owner using its master secret key to revoke a user's authorisation to form further queries. It does so by generating new server and data owner states and distributes them accordingly.*

Although not explicitly mentioned, the data owner may update the public parameters pp during any algorithm in order to reflect any changes in the entity population as new users may have been added and granted the ability to perform searches over the outsourced database. The algorithms AddUser and RevokeUser can be run at any point in the scheme after the system was initialised and the index was created. Note that we assume throughout this chapter that the server does not collude with revoked users.

An EVSE scheme is *correct* if there is a negligible probability that verification does not succeed when all algorithms are run honestly. More formally this can be represented as follows.

Definition 6.2. *An extended verifiable searchable encryption scheme is* correct *for a family of queries $\mathcal{F}$ if for all queries $Q \in \mathcal{F}$ and all non-revoked entities, the following holds:*

$$
\begin{aligned}
&\Pr[(mk, pp) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U}),\\
&\quad (\mathcal{I}_D, st_\mathrm{s}, st_\mathrm{o}) \xleftarrow{\$} \mathsf{BuildIndex}(\delta(D), l(\delta(D)), G, mk, pp),\\
&\quad sk_\mathrm{id} \xleftarrow{\$} \mathsf{AddUser}(\mathrm{id}, G, mk, pp),\\
&\quad (qt_Q, vk_Q) \xleftarrow{\$} \mathsf{Query}(Q, st_\mathrm{s}, st_\mathrm{o}, sk_\mathrm{id}, pp),\\
&\quad \theta_Q \xleftarrow{\$} \mathsf{Search}(\mathcal{I}_D, qt_Q, st_\mathrm{s}, sk_\mathrm{S}, pp),\\
&\quad r \leftarrow \mathsf{Verify}(\theta_Q, vk_Q, pp)\\
&\quad (st'_\mathrm{s}, st'_\mathrm{o}) \xleftarrow{\$} \mathsf{RevokeUser}(\mathrm{id}, G, mk, pp)]\\
&\quad = 1 - \mathrm{negl}(\lambda).
\end{aligned}
$$

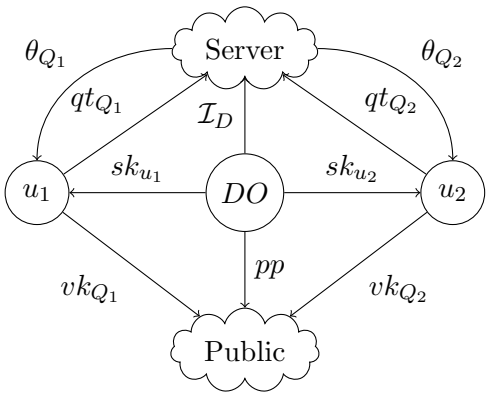

Figure 6.1: Operation of an EVSE scheme

We consider the following two main types of queries in this chapter:

- **Keyword matching queries**: Queries of this type have formed the basis of most prior work in SE. Suppose there exists a universe (dictionary) of keywords. Each encrypted data item is associated with an *index* of one or more keywords to describe the contents. Queries are formed over the same universe of keywords. In this work, we permit Boolean formulas over keywords (e.g. $((\mathtt{a} \wedge \mathtt{b}) \vee \mathtt{c})$ where $\mathtt{a}, \mathtt{b}, \mathtt{c}$ are keywords). We return an identifier for each data item whose associated keywords in the index satisfy this formula. Thus we can perform very expressive search queries over keywords.

- **Computational queries**: Queries of this type are similar to the operations commonly discussed in the context of outsourced computation. We allow statistical queries over keywords, e.g. counting the number of data items that satisfy a keyword matching query, as well as operations over selected data values that have been encoded using additional portions of the keyword universe. It is possible to encode the entire database in such a way to enable computations over all data fields, but it would usually be more efficient to select a (small) subset of fields that are most useful or most frequently queried. Clearly, keyword matching queries can be seen as a special case of computational queries where the function operator is equality testing.

We can also combine both the functionalities of the aforementioned query types and provide mixed queries. For example, such a query could be formulated as finding the average of data values contained in all documents associated with a particular keyword. All presented types of queries are performed in a verifiable manner to ensure that results are correct and complete. Furthermore, each type of query can be formed either by a single, authorised user or by a set of authorised users that may each contribute different clauses to the search query. We discuss this type of query more detailed in Section 6.4.4. In Figure 6.1, we illustrate the entity population and respective interaction within our EVSE model.

6.3 Security Models

In the context of EVSE we consider three different notions of security, namely public verifiability, index privacy and query privacy. As usual, we represent the security definitions in terms of a game-based notion. Throughout the following, the notation $\mathcal{A}^{\mathcal{O}}$ denotes the adversary being provided with oracle access to the following algorithms: $\mathsf{BuildIndex}(\cdot,\cdot,\cdot,mk,pp)$, $\mathsf{AddUser}(\cdot,\cdot,mk,pp)$, $\mathsf{Query}(\cdot,\cdot,\cdot,\cdot,pp)$ and $\mathsf{Search}(\cdot,\cdot,\cdot,\cdot,pp)$. We assume that oracle queries are performed in a logical order such that all required information is generated from previous queries.

6.3.1 Public Verifiability

In Figure 6.2, we present the notion of public verifiability. This notion ensures that a server cannot cheat by returning an incorrect result without being detected. The game begins with the challenger $\mathcal{C}$ initialising the system and providing the resulting public parameters pp to the adversary $\mathcal{A}$. The adversary is provided with oracle access as detailed above. The adversary $\mathcal{A}$ selects the challenge inputs consisting of the challenge

$\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{EVSE}, 1^{\lambda}\right]$

1 : $(pp, mk) \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, \mathcal{U})$
2 : $(Q, \delta(D^{\star}), l(\delta(D^{\star})), st_{\text{s}}, st_{\text{o}}) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}}(pp)$
3 : $G \subseteq \mathsf{Users}$
4 : $\text{id} \leftarrow_{\$} \mathsf{Users}$
5 : $sk_{\text{id}} \leftarrow_{\$} \mathsf{AddUser}(\text{id}, G, mk, pp)$
6 : $(qt_Q, vk_Q) \leftarrow_{\$} \mathsf{Query}(Q, st_{\text{s}}, st_{\text{o}}, sk_{\text{id}}, pp)$
7 : $\theta^{\star} \leftarrow_{\$} \mathcal{A}^{\mathcal{O}}(qt_Q, vk_Q, pp)$
8 : $r \leftarrow \mathsf{Verify}(\theta^{\star}, vk_Q, pp)$
9 : **if** $(r \neq \perp)$ **and** $(r \neq Q(\delta(D^{\star})))$ **then**
10 : **return** 1
11 : **else return** 0

Figure 6.2: The public verifiability experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{EVSE}, 1^{\lambda}\right]$

query Q, the challenge pre-index $\delta(D^{\star})$, respective pre-index label $l(\delta(D^{\star}))$ as well as server and data owner states. In the next step, the challenger runs $\mathsf{AddUser}$ on a randomly chosen id from the user space enrolling a querier into the system by providing them with a secret key. The challenger $\mathcal{C}$ then outputs a challenge by executing Query on the challenge query. The adversary receives the resulting parameters from the challenger and is again provided with oracle access as above. $\mathcal{A}$ wins if it produces an encoded output that verifies correctly but does not correspond to the actual result

$Q(\delta(D^\star))$.

Definition 6.3. *The* advantage *of a PPT adversary in the* PubVerif *game for an EVSE construction, $\mathcal{EVSE}$, is defined as:*

$$\mathbf{Adv}^{\text{PubVerif}}_{\mathcal{A},\mathcal{EVSE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\text{PubVerif}}_{\mathcal{A}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right].$$

An EVSE scheme, $\mathcal{EVSE}$, is secure *in the sense of public verifiability if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\text{PubVerif}}_{\mathcal{A},\mathcal{EVSE}}(1^\lambda) \leq \text{negl}(\lambda).$$

6.3.2 Selective Index Privacy

In Figure 6.3, we present the notion of index indistinguishability against a selective chosen keyword attack, which ensures no information regarding the attributes is leaked from the index. The game begins with the adversary outputting two sets of documents D_0 and D_1 that it wishes to be challenged on, with the restriction that $|D_0| = |D_1|$. The challenger runs Setup initialising the system and providing the resulting public parameters pp to the adversary $\mathcal{A}$, and initialises a set G of authorised users. The challenger selects a bit b uniformly at random selecting which set of documents to encode into the index. Before the index is created, the challenger needs to create the pre-index from the

$\mathbf{Exp}^{\text{sIndPriv}}_{\mathcal{A}}\left[\mathcal{EVSE}, 1^\lambda\right]$

1 : $(D_0, D_1) \leftarrow_\$ \mathcal{A}(1^\lambda, \mathcal{U})$
2 : **if** $(|D_0| \neq |D_1|)$ **then return** 0
3 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mathcal{U})$
4 : $b \leftarrow_\$ \{0, 1\}$
5 : $G \subseteq \mathsf{Users}$
6 : $(\delta(D_b), l(\delta(D_b))) \leftarrow \mathsf{Encode}(D_b)$
7 : $(\mathcal{I}_{D_b}, st_\text{s}, st_\text{o}) \leftarrow_\$ \mathsf{BuildIndex}(\delta(D_b), l(\delta(D_b)), G, mk, pp)$
8 : $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}}(\mathcal{I}_{D_b}, st_\text{s}, pp)$
9 : **if** $b' = b$ **then**
10 : **return** 1
11 : **else return** 0

Figure 6.3: The selective index privacy experiment $\mathbf{Exp}^{\text{sIndPriv}}_{\mathcal{A}}\left[\mathcal{EVSE}, 1^\lambda\right]$

set of documents D_b. This is done using an Encode mechanism that takes the elements of

D_b as input and outputs the pre-index $\delta(D_b)$ with respective label $l(\delta(D_b))$.[3] The challenger then runs $\mathsf{BuildIndex}$ using $\delta(D_b)$ and $l(\delta(D_b))$ to produce the index $\mathcal{I}_{D_b}$, which is given to $\mathcal{A}$. The adversary is then given oracle access to the following algorithms: $\mathsf{BuildIndex}(\cdot,\cdot,\cdot,mk,pp)$, $\mathsf{AddUser}(\cdot,\cdot,mk,pp)$, $\mathsf{Query}(\cdot,\cdot,\cdot,\cdot,pp)$ and $\mathsf{Search}(\cdot,\cdot,\cdot,\cdot,pp)$ which we denote by $\mathcal{O}$. However, we require the restriction that the query results are identical for each index $\mathcal{I}_{D_0},\mathcal{I}_{D_1}$, i.e. if $\theta_{Q_0} \leftarrow \mathsf{Search}(\mathcal{I}_{D_0},qt_Q,st_\mathrm{s},sk_\mathrm{S},pp)$ and $\theta_{Q_1} \leftarrow \mathsf{Search}(\mathcal{I}_{D_1},qt_Q,st_\mathrm{s},sk_\mathrm{S},pp)$ then we need $\theta_{Q_0}=\theta_{Q_1}$. After this query phase, $\mathcal{A}$ outputs a guess b' and wins the game if its guess corresponds to the randomly chosen bit b. In other words, $\mathcal{A}$ wins the game if it can identify which document set (D_0 or D_1) was encoded into the index $\mathcal{I}_{D_b}$.

Definition 6.4. *The* advantage *of a PPT adversary in the* sIndPriv *game for an EVSE construction, $\mathcal{EVSE}$, is defined as:*

$$\mathbf{Adv}^{\mathrm{sIndPriv}}_{\mathcal{A},\mathcal{EVSE}}(1^\lambda) = \Pr\left[\mathbf{Exp}^{\mathrm{sIndPriv}}_{\mathcal{A}}\left[\mathcal{EVSE},1^\lambda\right]\rightarrow 1\right]-\frac{1}{2}.$$

An EVSE scheme, $\mathcal{EVSE}$, is secure *in the sense of selective index privacy if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}^{\mathrm{sIndPriv}}_{\mathcal{A},\mathcal{EVSE}}(1^\lambda) \leq \mathrm{negl}(\lambda).$$

6.3.3 Selective Query Privacy

In Figure 6.4, we present the notion of selective query privacy that captures that the queries themselves should not leak any information about the queries. Note the queries only reveal the logical make-up (gates). This notion is formalised in a similar fashion to the notion of index privacy (Figure 6.3).

The game begins with the adversary outputting two queries Q_0 and Q_1 that use the same gates. We denote the gate structure of a query Q by $\mathcal{G}_Q$ and require that $\mathcal{G}_{Q_0}=\mathcal{G}_{Q_1}$, otherwise the challenger aborts the game. The challenger runs Setup, initialising the system and providing the resulting public parameters pp to the adversary $\mathcal{A}$. Additionally the challenger selects a bit b uniformly at random and initialises a set G of authorised users. In the next step, the challenger runs $\mathsf{BuildIndex}$ using $\delta(D)$ and $l(\delta(D))$ to produce the index $\mathcal{I}_D$ as well as the server and data owner state. The challenger provides the index and server state to $\mathcal{A}_{\mathrm{EVSE}}$. In the next step, the challenger runs $\mathsf{AddUser}$ on a randomly chosen id from the user space providing the entity with a secret key. The challenger uses an Encode mechanism in order to prepare the query Q_b that will be used as input to Query, and runs the algorithm outputting an encoded query token qt_{Q_b} and verification key vk_{Q_b} that is provided to the adversary. The adversary is then given oracle access to the following algorithms: $\mathsf{BuildIndex}(\cdot,\cdot,\cdot,mk,pp)$,

[3] Encode is not required in our instantiation as the pre-indexes can be chosen directly from $\widetilde{\mathcal{U}}$ as the user knows the permutation Π and the mapping from $\mathcal{U}'$ to $\widetilde{\mathcal{U}}$; the adversary however does not.

$\mathbf{Exp}_{\mathcal{A}}^{\text{sQueryPriv}}\left[\mathcal{EVSE}, 1^{\lambda}\right]$

1 : $(Q_0, Q_1) \leftarrow\!\!\$\ \mathcal{A}(1^{\lambda}, \mathcal{U})$
2 : **if** $(\mathcal{G}_{Q_0} \neq \mathcal{G}_{Q_1})$ **then return** 0
3 : $(pp, mk) \leftarrow\!\!\$\ \mathsf{Setup}(1^{\lambda}, \mathcal{U})$
4 : $b \leftarrow\!\!\$\ \{0,1\}$
5 : $G \subseteq \mathsf{Users}$
6 : $(\mathcal{I}_D, st_{\text{s}}, st_{\text{o}}) \leftarrow\!\!\$\ \mathsf{BuildIndex}(\delta(D), l(\delta(D)), G, mk, pp)$
7 : $\text{id} \leftarrow\!\!\$\ \mathsf{Users}$
8 : $sk_{\text{id}} \leftarrow\!\!\$\ \mathsf{AddUser}(\text{id}, G, mk, pp)$
9 : $\widetilde{Q}_b \leftarrow \mathsf{Encode}(Q_b)$
10 : $(qt_{\widetilde{Q}_b}, vk_{\widetilde{Q}_b}) \leftarrow\!\!\$\ \mathsf{Query}(\widetilde{Q}_b, st_{\text{s}}, st_{\text{o}}, sk_{\text{id}}, pp)$
11 : $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(qt_{\widetilde{Q}_b}, vk_{\widetilde{Q}_b}, \mathcal{I}_D, st_{\text{s}}, pp)$
12 : **if** $b' = b$ **then**
13 : **return** 1
14 : **else return** 0

Figure 6.4: The selective query privacy experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{sQueryPriv}}\left[\mathcal{EVSE}, 1^{\lambda}\right]$

$\mathsf{AddUser}(\cdot,\cdot,mk,pp)$, $\mathsf{Query}(\cdot,\cdot,\cdot,\cdot,pp)$ and $\mathsf{Search}(\cdot,\cdot,\cdot,\cdot,pp)$ which we denote by $\mathcal{O}$. Eventually, after the query phase it outputs its guess b' for b, and wins the game if the guess was correct.

Definition 6.5. *The* advantage *of a PPT adversary in the* sQueryPriv *game for an EVSE construction, $\mathcal{EVSE}$, is defined as:*

$$\mathbf{Adv}_{\mathcal{A},\mathcal{EVSE}}^{\text{sQueryPriv}}(1^{\lambda}) = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\text{sQueryPriv}}\left[\mathcal{EVSE}, 1^{\lambda}\right] \rightarrow 1\right] - \frac{1}{2}.$$

An EVSE scheme, $\mathcal{EVSE}$, is secure *in the sense of selective query privacy if for all PPT adversaries $\mathcal{A}$, it holds that*

$$\mathbf{Adv}_{\mathcal{A},\mathcal{EVSE}}^{\text{sQueryPriv}}(1^{\lambda}) \leq \text{negl}(\lambda).$$

6.4 Construction

6.4.1 Overview

In this section, we provide a construction of an EVSE scheme and we base its instantiation on a ciphertext-policy attribute-based encryption scheme. As shown in Chapter 4, CP-ABE can be used to verifiably request computations to be performed on data held

by a server, and we referred to this mode of computation as VDC. In VDC, a trusted key distribution centre (KDC) initialises the system and issues a CP-ABE decryption key to the server pertaining to the data it holds. Here in the setting of EVSE, we use a similar technique but have the data owner act as the KDC. Thus, we do not require the data to be revealed to an external KDC and our underlying entity population corresponds to that in the searchable encryption literature. In our EVSE setting, the data owner issues a CP-ABE decryption key to the server that corresponds to the index of the data. The particular encoding method is described in Section 6.4.2.

We consider the family $\mathcal{F}$ of Boolean functions closed under complement, i.e. if $F \in \mathcal{F}$ then $\overline{F}(X) = F(X) \oplus 1$ is also in $\mathcal{F}$. A query Q is represented as a Boolean function of keywords and computational data points. If a monotonic CP-ABE scheme is used then queries can be comprised of AND and OR gates (and negation can inefficiently be handled by including both a positively and negatively labelled attribute in the universe and requiring the presence of exactly one of them in the query). A non-monotonic CP-ABE scheme enables queries formed from AND, OR and NOT gates, which is a universal set of gates, and fuzzy CP-ABE enables fuzzy keyword search. We can achieve all functions in the class NC^1, which includes common arithmetic and comparison operators useful in queries. An n-bit result can be formed by performing n Boolean queries, each of which returns the ith bit of the output.

The query token qt_Q for a query $Q \in \mathcal{F}$ comprises two CP-ABE ciphertexts for access structures representing Q and $\overline{Q} \in \mathcal{F}$ respectively. To perform the search, the server attempts to decrypt each ciphertext under the secret decryption key and outputs the result. Each decryption succeeds if and only if the query evaluates correctly on the pre-index. Any entity within the system may perform the verification operation using the public verification key to determine the search result and whether the search was performed correctly.

6.4.2 Data Encoding

Defining the Index

Searchable encryption schemes usually operate on the level of documents with lists of keywords. In this chapter, we have moved more towards a practice-oriented setting where we operate on databases. That is, we suppose the database D to be outsourced comprises n documents, i.e. $D = D_1, \ldots, D_n$. In the following we discuss how to form a *pre-index* $\delta(D)$, which represents the keywords and data fields that may be queried over.

Let us denote by $\mathcal{D}$ a dictionary of keywords that describes the documents within the data. $\mathcal{D}$ alone suffices for keyword matching queries but for computational queries,

we also need to be able to encode data values such that they can be input to queries represented as access structures encoding Boolean functions. Thus, for each data field x that may be input to a computational query, let the maximum size of the data value be m_x bits. We define m_x additional attributes $A_{x,1}, A_{x,2}, \ldots, A_{x,m_x}$, and define the universe $\mathcal{C} = \bigcup_{x \in D} \cup_{i=1}^{m_x} A_{x,i}$ to be the union of these attributes over all data fields. Let y be a value stored in the data field x and let the binary representation of y be $y_1, \ldots, y_{m_x}$. We view y as a *characteristic tuple* (cf. Section 2.7.3) of an attribute set $A_y \subseteq \mathcal{C}$, where $A_y = \{A_{x,i} : y_i = 1\}$. In other words, we include an attribute for position i in the set if and only if the ith bit of y equals 1.

Finally, in order to enable the index for all n documents to be encoded within a single CP-ABE key (and hence for computations to be performed simultaneously on all documents), and to ensure that the correct index data is used for each query, we must encode a labelling of the document that each attribute pertains to. We define our attribute universe $\mathcal{U}$ for the CP-ABE scheme to be $\mathcal{U} = \{\mathcal{D} \cup \mathcal{C}\} \times [n]$, i.e. we take n copies of $\mathcal{D}$ and $\mathcal{C}$ and assume that all documents have the same fields. Each element of $\{\mathcal{D} \cup \mathcal{C}\}$ describes a particular keyword or data value, and each copy relates to a different document in D. We index each copy of an attribute $w \in \{\mathcal{D} \cup \mathcal{C}\}$ as $\{w_i\}_{i=1}^{n}$ and then w_i denotes the presence of w in document D_i. In practice, it may be desirable to use a 'large universe' CP-ABE scheme, wherein arbitrary textual strings are mapped to attributes (group elements), e.g. using a hash function H. Thus, for a keyword or data value w in the ith document, the attribute could be defined as $\mathsf{H}(w||i)$. Finally, the pre-index of the data D is a set of attributes $\delta(D) \subseteq \mathcal{U}$. The index that is outsourced will be a CP-ABE key generated over this attribute set.

Let us consider the following example that shows how we define the pre-index within our system model. Suppose we have three documents $D = D_1, D_2, D_3$ with the following characteristics:

- **Document 1:** *Keywords:* Male, Vaccinated. *Data:* Age $= 7 = 111_2$.
- **Document 2:** *Keywords:* Female. *Data:* Age $= 4 = 100_2$.
- **Document 3:** *Keywords:* Male, Vaccinated. *Data:* -

Then, we can define the dictionary of keywords that describes those documents as

$$\mathcal{D} = \{\texttt{Male}, \texttt{Female}, \texttt{Vaccinated}\}.$$

To enable computational queries on the documents, we need to define three additional attributes that provide us with

$$\mathcal{C} = \{\mathtt{A}_{\mathtt{Age},1}, \mathtt{A}_{\mathtt{Age},2}, \mathtt{A}_{\mathtt{Age},3}\}.$$

Next we need to form our attribute universe as

$$\begin{aligned}\mathcal{U} =&\{\mathcal{D} \cup \mathcal{C}\} \times [n]\\ =&\{\texttt{Male}_{\texttt{Doc1}}, \texttt{Male}_{\texttt{Doc2}}, \texttt{Male}_{\texttt{Doc3}},\\ &\texttt{Female}_{\texttt{Doc1}}, \texttt{Female}_{\texttt{Doc2}}, \texttt{Female}_{\texttt{Doc3}},\\ &\texttt{Vaccinated}_{\texttt{Doc1}}, \texttt{Vaccinated}_{\texttt{Doc2}}, \texttt{Vaccinated}_{\texttt{Doc3}},\\ &\texttt{A}_{(\texttt{Age},1),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc3}},\\ &\texttt{A}_{(\texttt{Age},2),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},2),\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},2),\texttt{Doc3}},\\ &\texttt{A}_{(\texttt{Age},3),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc3}}\}.\end{aligned}$$

We can now form the pre-index $\delta(D) \subseteq \mathcal{U}$ of the data D that represents the available keywords and data fields. Thus, the pre-index corresponds to

$$\begin{aligned}\delta(D) =&\{\texttt{Male}_{\texttt{Doc1}}, \texttt{Vaccinated}_{\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},2),\texttt{Doc1}}, \texttt{A}_{(\texttt{Age},3),\texttt{Doc1}},\\ &\texttt{Female}_{\texttt{Doc2}}, \texttt{A}_{(\texttt{Age},1),\texttt{Doc2}},\\ &\texttt{Male}_{\texttt{Doc3}}, \texttt{Vaccinated}_{\texttt{Doc3}}\},\end{aligned}$$

and the index given to the server is a CP-ABE decryption key over this attribute set.

Hiding the Index

In general, CP-ABE schemes do not hide the attributes within the decryption key. This is usually expected behaviour since CP-ABE is often used to cryptographically enforce access control policies [67, 93] and it is natural to assume that an entity is aware of their access rights. However, in this setting we are using CP-ABE not to protect objects from unauthorised access, but instead to prove the outcome of a function evaluation as in previous chapters. The decryption keys in our setting are formed over attributes encoding the index of outsourced data, as opposed to encoding access rights. Since the server should not learn any information about the data, *including* the index, we must implement a mechanism by which the decryption key hides the associated attributes.

In many CP-ABE schemes, the public parameters comprise an ordered set of group elements [142], each associated with an attribute from the universe. In more detail, that is, for all $i \in \mathcal{U}$, choose $t_i \xleftarrow{\$} \mathbb{Z}_p$, then form the encoded attribute set $\{g^{t_i}\}_{i\in\mathcal{U}}$. Thus, given a key (or ciphertext) that comprises g^{t_i}, it is possible, based on the ordering of this set, to determine the attribute $i \in \mathcal{U}$ it relates to. In addition, the attributes may be listed in the clear, and attached to keys and ciphertexts to indicate which group elements should be applied at each point. Clearly, this is unsuitable for our requirement to hide the index.

To this end, we first apply a random permutation to the universe of attributes $\mathcal{U}$ such that the position of the group elements within the ordered set does not reveal the attribute string (unless the permutation is known). We then use a symmetric encryption scheme to encrypt each attribute $x \in \mathcal{U}$ under a key k, and instantiate the CP-ABE scheme on this universe of *encrypted* attributes. Thus, without knowledge of the key k, the server should be unable to determine the attribute x that a given group element corresponds to. We assume that only the keywords or data items being searched and computed over are considered sensitive, and not the logical make-up of the Boolean function (in terms of gates).

6.4.3 Formal Details

The data owner initialises the system and encodes the data as an index which is pushed to the server. Each authorised user will be issued with a personalised secret key enabling them to form queries. Note that forming a query is similar to preparing a computational request in VDC (cf. Section 4.4). In order to form a query Q, a user chooses a message uniformly at random from the message space $\mathcal{M}$ to act as a verification token, and encrypt this using the CP-ABE scheme under the access structure encoding Q. The server attempts to decrypt the ciphertext and recovers the chosen message if and only if $Q(\delta(D)) = 1$. Note that this decryption procedure proceeds similarly to the basic PVC principles as introduced in Section 2.7.3. By the indistinguishability security of the CP-ABE scheme, the server learns nothing about the message if $Q(\delta(D)) = 0$ since this corresponds to an access structure not being satisfied. Thus, if a server returns the correct message, the user is assured that the query evaluated to 1 on the data. If, however, $Q(\delta(D)) = 0$, then decryption will return $\perp$. This is insufficient for verification purposes since the server can return $\perp$ to convince a user of a false negative search result. Thus, the user must produce two CP-ABE ciphertexts to overcome this one-sided error. As in the previous chapters, one ciphertext corresponds to the query Q, whilst the other ciphertext corresponds to the complement query $\overline{Q}$ (which always outputs the opposite result to Q as the query is represented as a Boolean function). Thus, if $Q(\delta(D)) = 0$ then, necessarily, $\overline{Q}(\delta(D)) = 1$. Hence, the server's key will decrypt *exactly one* ciphertext and the returned message will distinguish whether Q or $\overline{Q}$ was satisfied, and therefore the value of $Q(\delta(D))$. A well-formed response (d_0, d_1) from a server satisfies the following:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } Q(\delta(D)) = 1; \\ (\perp, m_1), & \text{if } Q(\delta(D)) = 0. \end{cases} \tag{6.1}$$

If the returned plaintext does not match one of the randomly chosen messages then the server has returned an incorrect result. This is also the case if both returned results are $\perp$ but a rational malicious server would never return this.

Public verifiability is achieved by publishing a token comprising a one-way function g applied to each plaintext. Any entity can apply g to the server's response and compare with this token to check correctness.
Our adversarial model allows the adversary (and hence servers in our system) to hold more than one index (key) as the data owner may have outsourced multiple data sets, and therefore we must ensure that a key cannot produce a valid looking response to a query on a different index. Similarly to the previous chapters, we achieve this by labelling each pre-index with a label $l(\delta(D))$ and define an attribute for each label. Then, for a pre-index $\delta(D)$, the decryption key is formed over the attribute set $(A_{\delta(D)} \cup l(\delta(D)))$.[4] Recall that encoded data stored on the server's side is a collection of n documents, which we label $D_1, \ldots, D_n$. When making a query $Q(\delta(D))$, a sub-query Q_i may be formed for each document (e.g. to check if a given keyword is contained in each document). In this case, during Query the CP-ABE encryption algorithm uses the access structure encoding of the conjunction $(Q_i \wedge l(\delta(D)))$ for $i \in [n]$. A valid result can only be formed by applying the sub-query to the specified document, which is also labelled by $D_i \in \mathcal{D}$. Thus, the CP-ABE decryption algorithm succeeds if and only if the query is satisfied *and* the label $l(\delta(D))$ is matched in the key and ciphertext. Note that a key for a different pre-index will *not* include the correct label and thus cannot be used to compute search results for other data sets. Inputs to the Query algorithm are assumed to be of this form.

6.4.4 Instantiation Details

Let $\mathcal{CP}$-$\mathcal{ABE}$ = (ABE.Setup, ABE.KeyGen, ABE.Encrypt, ABE.Decrypt) define a CP-ABE encryption scheme over the universe $\mathcal{U}$ for a class of queries $\mathcal{F}$ closed under complement. Let $\mathcal{SE}$= (SE.KeyGen, SE.Encrypt, SE.Decrypt) be a symmetric encryption scheme secure in the sense of IND-CPA. Let $\mathcal{BE}$ = (BE.KeyGen, BE.Encrypt, BE.Add, BE.Decrypt) be a broadcast encryption scheme that retains IND-CPA security against a coalition of revoked users. Finally, let g be a one-way function, and let Π and ϕ be pseudo-random permutations (PRPs). Then Algorithms 1–7 define an EVSE scheme for a class of queries $\mathcal{F}$.

1. Setup, presented in Algorithm 1, aims to initialise the scheme. It starts with running BE.KeyGen to create a broadcast encryption key mk_{BE} as well as SE.KeyGen to output a symmetric encryption key k_{SE}. It also chooses a random key κ serving as the key to the PRP Π. Since we aim to hide the index, we apply a PRP Π to the universe of attributes $\mathcal{U}$ such that the position of elements within the ordered set does not reveal the attribute string and thus we obtain $\mathcal{U}'$. We then

[4] Note that $A_{\delta(D)} = \delta(D)$. We chose to use this notation to emphasise that the key is formed over the attribute set as well as to keep consistency throughout this thesis with the used notation in the previous chapters.

use a symmetric encryption scheme to encrypt each attribute in $\mathcal{U}'$ under a key k_{SE} obtaining an encrypted universe $\widetilde{\mathcal{U}}$. The algorithm then calls the $\mathsf{ABE.Setup}$ algorithm in order to initialise the CP-ABE scheme on this universe of *encrypted* attributes $\widetilde{\mathcal{U}}$. The output of the algorithm consists of the public parameters pp and the master secret mk for the EVSE system. The public parameters consist of the master public key mpk_{ABE} of the CP-ABE scheme and $\widetilde{\mathcal{U}}$. The master secret comprises of the master secret key msk_{ABE} of the CP-ABE scheme, the broadcast encryption key mk_{BE}, the symmetric encryption key k_{SE} as well as the PRP Π and its key κ.

Algorithm 1 $(mk, pp) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$

$mk_{\mathrm{BE}} \leftarrow_{\$} \mathsf{BE.KeyGen}(1^\lambda)$
$k_{\mathrm{SE}} \leftarrow_{\$} \mathsf{SE.KeyGen}(1^\lambda)$
$\kappa \leftarrow_{\$} \{0,1\}^\lambda$
$\mathcal{U}' \leftarrow \Pi_\kappa(\mathcal{U})$
for $i \in \mathcal{U}'$ **do**
  $u_i \leftarrow_{\$} \mathsf{SE.Encrypt}(i, k_{\mathrm{SE}})$
endfor
$\widetilde{\mathcal{U}} \leftarrow \{u_i\}_{i \in \mathcal{U}'}$
$(msk_{\mathrm{ABE}}, mpk_{\mathrm{ABE}}) \leftarrow_{\$} \mathsf{ABE.Setup}(1^\lambda, \widetilde{\mathcal{U}})$
$pp \leftarrow (mpk_{\mathrm{ABE}}, \widetilde{\mathcal{U}})$
$mk \leftarrow (msk_{\mathrm{ABE}}, mk_{\mathrm{BE}}, k_{\mathrm{SE}}, \kappa, \Pi)$

2. $\mathsf{BuildIndex}$, presented in Algorithm 2, aims to generate a searchable index in form of a CP-ABE decryption key. This key is formed over the attribute set $(A_{\delta(D)} \cup l(\delta(D)))$ where $A_{\delta(D)}$ expresses the pre-index $\delta(D)$ of the outsourced data D which corresponds to a set of attributes in the universe $\widetilde{\mathcal{U}}$ that represent the available keywords and data fields. The algorithm outputs the index which is then given to the server. Next the algorithm samples a key j from $\{0,1\}^\lambda$ that is used to generate the server state st_{s}. The server state simply consists of a broadcast encryption of j and the set of authorised users that includes the server. Finally the algorithm outputs an owner state st_{o} which simply gets the key j assigned. Note that this state is also provided to all authorised users within G.

Algorithm 2 $(\mathcal{I}_D, st_\mathrm{s}, st_\mathrm{o}) \xleftarrow{\$} \mathsf{BuildIndex}(\delta(D), l(\delta(D)), G, mk, pp)$

1 : $\mathcal{I}_D \leftarrow_\$ \mathsf{ABE.KeyGen}((A_{\delta(D)} \cup l(\delta(D))), msk_\mathrm{ABE}, mpk_\mathrm{ABE})$
2 : $j \leftarrow_\$ \{0,1\}^\lambda$
3 : $st_\mathrm{s} \leftarrow_\$ \mathsf{BE.Encrypt}(G, j, mk_\mathrm{BE})$
4 : $st_\mathrm{o} \leftarrow j$

3. AddUser, presented in Algorithm 3, aims to enrol entities within the system. To add an entity, the algorithm generates an entity key uk_id for the broadcast encryption scheme running BE.Add. In case the entity is a user, then the user's secret key consists of the 5-tuple $(uk_\mathrm{u}, k_\mathrm{SE}, \kappa, \Pi, st_\mathrm{o})$ that enables her to form queries as well as being able to link back search results to the outsourced data. In case the entity is a server, it simply gets the entity key uk_s assigned as its secret key.

Algorithm 3 $sk_\mathrm{id} \xleftarrow{\$} \mathsf{AddUser}(\mathrm{id}, G, mk, pp)$

1 : $uk_\mathrm{id} \leftarrow_\$ \mathsf{BE.Add}(\mathrm{id}, \mathrm{mk}_\mathrm{BE})$
2 : **if** id is a user **then**
3 : $\quad sk_\mathrm{u} \leftarrow (uk_\mathrm{u}, k_\mathrm{SE}, \kappa, \Pi, st_\mathrm{o})$
4 : **else**
5 : $\quad sk_\mathrm{s} \leftarrow uk_\mathrm{s}$
6 : **endif**

4. Query, presented in Algorithm 4, aims to prepare the user's search request which is sent to the server. Before forming the search query token, an authorised user first retrieves the latest server state st_s and uses her user key uk_u to recover $\widetilde{j}$. If $\widetilde{j} \neq st_\mathrm{o}$, then the algorithm aborts. Otherwise, a successful check indicates that the user is in the correct state and thus authorised to form queries to search over the outsourced database. When making a query Q, we may form sub-queries Q_i for each document of the database. Thus, for each sub-query Q_i the algorithm chooses two equal length messages $m_{0,i}$ and $m_{1,i}$ uniformly at random from the message space. To form a search request (a query token qt_{Q_i}) the algorithm needs to form two CP-ABE ciphertexts $c_{0,i}$ and $c_{1,i}$ that encrypt the chosen messages under the access structures Q_i and $\overline{Q_i}$ respectively. Those two ciphertexts form the encoded query token qt_{Q_i}, and the algorithm then encrypts them using a PRP ϕ under key j outputting γ_i. Note that the user makes use of the PRP in order to *prove* to the server that she possesses the valid current state and thus is indeed authorised to send queries. The verification key vk_{Q_i} is created

by applying a one-way function g to each message and g allows the key to be published. Finally the algorithm outputs the set of γ_i as the query token qt_Q, and the set of verification keys as vk_Q.

Algorithm 4 $(qt_Q, vk_Q) \xleftarrow{\$} \mathsf{Query}(Q, st_\mathrm{s}, st_\mathrm{o}, sk_\mathrm{u}, pp)$

$\widetilde{j} \leftarrow \mathsf{BE.Decrypt}(st_\mathrm{s}, uk_\mathrm{u})$
if $\widetilde{j} \neq st_\mathrm{o}$ **then return** 0
for $i = 1$ **to** $|Q|$ **do**
  $(m_{0,i}, m_{1,i}) \leftarrow_\$ \mathcal{M} \times \mathcal{M}$
  $c_{0,i} \leftarrow_\$ \mathsf{ABE.Encrypt}(m_{0,i}, Q_i, mpk_\mathrm{ABE})$
  $c_{1,i} \leftarrow_\$ \mathsf{ABE.Encrypt}(m_{1,i}, \overline{Q_i}, mpk_\mathrm{ABE})$
  $qt_{Q_i} \leftarrow (c_{0,i}, c_{1,i})$
  $\gamma_i \leftarrow \phi_j(qt_{Q_i})$
  $vk_{Q_i} \leftarrow (g(m_{0,i}), g(m_{1,i}))$
endfor
$qt_Q \leftarrow \{\gamma_i\}_{i=1}^{|Q|}$
$vk_Q \leftarrow \{vk_{Q_i}\}_{i=1}^{|Q|}$

5. Search, presented in Algorithm 5, evaluates the search on the database returning a search result. The algorithm first decrypts the current server state st_s with its server key uk_s to recover the key j. The algorithm uses the key j to recover the query tokens by computing $\phi_j^{-1}(\gamma_i)$. Note that the key j currently used for ϕ is only known to the data owner and the set of currently authorised entities (including the server), and thus this PRP evaluation determines whether the request was formed by an authorised user and whether the server is authorised to evaluate this request. Next the algorithm attempts to decrypt both ciphertexts using the index $\mathcal{I}_D$. Decryption succeeds only if the query evaluates to 1 on the input database represented in terms of the pre-index, i.e. the access structure is satisfied. Since Q_i and $\overline{Q_i}$ output opposite results on the pre-index, this ensures that exactly one plaintext will correspond to a failure symbol $\perp$. Finally the algorithm outputs the set of all encoded search results denoted by θ_Q.

Algorithm 5 $\theta_Q \xleftarrow{\$} \mathsf{Search}(\mathcal{I}_D, qt_Q, st_\mathrm{s}, sk_\mathrm{S}, pp)$

$j \leftarrow \mathsf{BE.Decrypt}(st_\mathrm{s}, uk_\mathrm{S})$
for $i = 1$ **to** $|Q|$ **do**
  $(qt_{Q_i}) \leftarrow \phi_j^{-1}(\gamma_i)$
  $qt_{Q_i} = (c_{0,i}, c_{1,i})$
  $d_{0,i} \leftarrow \mathsf{ABE.Decrypt}(c_{0,i}, \mathcal{I}_D, mpk_\mathrm{ABE})$
  $d_{1,i} \leftarrow \mathsf{ABE.Decrypt}(c_{1,i}, \mathcal{I}_D, mpk_\mathrm{ABE})$
  $\theta_{Q_i} \leftarrow (d_{0,i}, d_{1,i})$
endfor
$\theta_Q \leftarrow \{\theta_{Q_i}\}_{i=1}^{|Q|}$

6. Verify, presented in Algorithm 6, first parses each encoded output θ_{Q_i} as $(d_{0,i}, d_{1,i})$ and each verification key vk_{Q_i} as $(g(m_{0,i}), g(m_{1,i}))$. The algorithm applies the one-way function g to each plaintext in θ_{Q_i} and compares the results to the components in the verification key. If either comparison is successful, this indicates that the server has indeed recovered a message. Otherwise, it shows that the server provided a malformed response. Note, if $m_{0,i}$ was returned then $Q_i(\delta(D)) = 1$, and otherwise if $m_{1,i}$ was returned then $Q_i(\delta(D)) = 0$ following equation (6.1).

Algorithm 6 $r \leftarrow \mathsf{Verify}(\theta_Q, vk_Q, pp)$

for $i = 1$ **to** $|Q|$ **do**
  if $g(m_{0,i}) = g(d_{0,i})$ **return** $r_i \leftarrow 1$
  elseif $g(m_{1,i}) = g(d_{1,i})$ **return** $r_i \leftarrow 0$
  else $r \leftarrow \perp$
  endif
endfor
$r \leftarrow \{r_i\}_{i=1}^{|Q|}$

7. RevokeUser, presented in Algorithm 7, aims to revoke a user in case the data owner wishes to cancel her authorisation. The algorithm samples a new key j' and creates a new updated server state. The updated server state st'_s simply consists of a broadcast encryption of j' and the set of authorised users (including the server) excluding the user to be revoked. The algorithm also outputs an updated owner state st'_o which simply gets the key j' assigned. The data owner sends st'_s to the server and to all non-revoked users st'_o.

Algorithm 7 $(st'_{\mathrm{s}}, st'_{\mathrm{o}}) \xleftarrow{\$} \mathsf{RevokeUser}(\mathrm{id}, G, mk, pp)$

1 : $j' \leftarrow_{\$} \{0,1\}^{\lambda}$

2 : $st'_{\mathrm{s}} \leftarrow_{\$} \mathsf{BE.Encrypt}(G \setminus \mathrm{id}, \mathrm{j}', \mathrm{mk}_{\mathrm{BE}})$

3 : $st'_{\mathrm{o}} \leftarrow j'$

Theorem 6.6. *Given an* IND-CPA *secure CP-ABE scheme, an authenticated symmetric encryption scheme and a broadcast encryption scheme, both secure in the sense of* IND-CPA*, pseudorandom permutations* Π *and* ϕ*, and a one-way function* g*. Let* $\mathcal{EVSE}$ *be the extended verifiable searchable encryption scheme defined in Algorithms 1–7. Then* $\mathcal{EVSE}$ *is secure in the sense of public verifiability (Figure 6.2), index privacy (Figure 6.3) and query privacy (Figure 6.4).*

The formal security proofs can be found in Section 6.5.

Note that in the algorithm RevokeUser the new state st'_{s} is given to the server that uses it to replace the old state. Thus, for all subsequent queries, the server uses the new key j' when inverting the PRP ϕ. Since revoked users (and unauthorised users) are not able to recover j', with overwhelming probability, their queries will not yield valid query tokens after the server applies $\phi_{j'}^{-1}(\gamma_i)$. In other words, users receive their keys for the broadcast encryption scheme only when they are authorised to search by the data owner. However, a user who has not joined the system yet could retrieve (by sending a request) the current server state st_{s} from the server, i.e. the broadcast encryption of j', but since the user does not possess a correct authorisation key she will not be able to recover j'. Thus, the user is not able to form valid query tokens. Similarly, this works for the case that a user is revoked as she cannot recover j' because she is not part of the authorised set of users. Note that even if a re-authorised user may be able to recover out-dated values of j that were used throughout searches while she was was revoked this does not provide the user with any advantage against the server since the values are no longer of interest. Furthermore, note that upon receiving the query token, the server only needs to evaluate the PRP in order to determine whether a user is revoked. In case we wish to implement additional contextual access control for authorisation, we can follow the discussion in Section 5.2.3 and replace the PRP ϕ with a key assignment scheme as introduced in [5].

In terms of the number of rounds of communication required per search, our EVSE scheme requires only one round of communication. The search time and size of the search results in our scheme is linear in n, i.e. it is linear in the amount of data items stored on the server. To this end, EVSE may be more suited to smaller databases to prevent these features from being prohibitively expensive. Our scheme hides the access

Scheme	Data type	Query type	Publicly Verifiable	Leakage	Computations
[140]	Static	Ranked equality	No	AP,SP	No
[104]	Dynamic	Equality	No	AP	No
[135]	Static	Conjunctive, Disjunctive	No	AP	No
[136]	Dynamic	Conjunctive	No	AP	No
[133]	Dynamic	Equality	No	AP, SP	No
[148]	Static	Equality	No	AP	No
[141]	Static	Fuzzy	No	AP, SP	No
[70]	Static	Semantic	No	AP, SP	No
[48]	Static	Equality	No	AP, SP	No
[51]	Static	Conjunctive	Yes	AP, SP	No
Our scheme	Static	Conjunctive, Disjunctive, Arbitrary CNF/DNF formulae, NC^1	Yes	$\mathcal{G}_Q$	Yes

Table 6.1: Comparison of functionalities in searchable encryption schemes

pattern as all search results are of the same form, regardless of what type of query was submitted.

As pointed out in Section 6.2.2, in some settings it might be desirable for multiple users to form a joint query. For example, it could be that a query should be formed from search terms arising from the expertise of multiple departments within an organisation. Thus, each department could contribute additional terms to narrow the filter of search results. Let us briefly consider three possible solutions to provide this functionality.
Firstly, one could use multi-input functional encryption [79] which extends CP-ABE to permit multiple users to create partial ciphertexts and for a single key to decrypt the union of these. Clearly, this provides a natural extension to our CP-ABE based instantiation, but current constructions of multi-input FE require complex and expensive primitives such as indistinguishability obfuscation. Instead, it could be possible to assume all data users are provided with access to a shared common reference string upon joining the system. This could simply be a random integer x along with a shared hash key for a hash function $\mathsf{H} : \mathbb{Z}_p \to \mathbb{Z}_p$. Then, when forming a ciphertext, all users compute $x = \mathsf{H}(x)$ and use the new value of x as the secret encryption exponent for a CP-ABE scheme. Thus, all users are essentially using the same random coins in a distributed version of the CP-ABE encryption algorithm.
Another solution is based on the use of oblivious transfer to allow one distinguished user to collect CP-ABE ciphertext fragments from all other users relating to their input. One implementation of this requires an ABE scheme with the *local encoding property* as described by Gordon et al. [88].

In Table 6.1 we provide a brief comparison between our scheme and those in the literature as discussed throughout this chapter and in the background Section 2.4. The abbreviation AP stands for *access pattern*, SP stands for *search pattern* and $\mathcal{G}_Q$ denotes the gate structure of a query Q.

6.5 Proofs of Security

In this section we present the full proof of Theorem 6.6 by providing proofs of security for the notions of public verifiability, index privacy and query privacy.

6.5.1 Public Verifiability

Lemma 6.7. *$\mathcal{EVSE}$ as defined in Algorithms 1–7 is secure in the sense of public verifiability (Figure 6.2) under the same assumptions as in Theorem 6.6.*

Proof. This proof is similar to the public verifiability proof presented in Section 4.5. Suppose $\mathcal{A}_{\text{EVSE}}$ is an adversary with non-negligible advantage against the public verifiability experiment (Figure 6.2) when instantiated with Algorithms 1–7. We begin by defining the following three games:

- **Game 0.** This is the public verifiability game as defined in Figure 6.2.
- **Game 1.** This is the same as **Game 0** with the modification that in Query, we no longer return an encryption of m_0 and m_1. Instead, we choose another equal length random message $m' \neq m_0, m_1$ and, if $Q(\delta(D^\star)) = 1$, we replace c_1 by $\mathsf{ABE.Encrypt}(m', \overline{Q} \wedge l(\delta(D^\star)), mpk_{\text{ABE}})$. Otherwise, we replace c_0 by $\mathsf{ABE.Encrypt}(m', Q \wedge l(\delta(D^\star)), mpk_{\text{ABE}})$. In other words, we replace the ciphertext associated with the unsatisfied query with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.
- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game (Figure 2.8).

We show that an adversary with non-negligible advantage against the public verifiability game can be used to construct an adversary that may invert the one-way function g.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{\text{EVSE}}$ can distinguish the two games with non-negligible advantage δ. We then construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{EVSE}}$ as a sub-routine to break the IND-CPA security of the CP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-CPA game (Figure 2.4)

with attribute universe $\widetilde{\mathcal{U}}$ with $\mathcal{A}_{\text{ABE}}$, who in turn acts as a challenger in the public verifiability game for $\mathcal{A}_{\text{EVSE}}$:

1. The challenger starts with initialising $\mathbb{A}^\star$ to be $\{\emptyset\}$. $\mathcal{C}$ continues by running the $\mathsf{ABE.Setup}$ algorithm on the security parameter and attribute universe $\widetilde{\mathcal{U}}$ to generate mpk_{ABE} and msk_{ABE}. The challenger gives mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.

2. $\mathcal{A}_{\text{ABE}}$ now simulates running $\mathsf{EVSE.Setup}$ such that the outcome is consistent with mpk_{ABE}. The master key mk is implicitly set to msk_{ABE}.

3. $\mathcal{A}_{\text{EVSE}}$ is provided with the public parameters and oracles access to the following functionalities, which are handled by $\mathcal{A}_{\text{ABE}}$.

   - $\mathsf{EVSE.BuildIndex}(\cdot, \cdot, \cdot, mk, pp)$ can be run to generate the index for a queried database D. To do so, $\mathcal{A}_{\text{ABE}}$ makes use of the KeyGen oracle $\mathcal{O}^{\mathsf{KeyGen}}$ in the CP-ABE game. $\mathcal{A}_{\text{ABE}}$ then sets the attribute data set $\widetilde{D}$ to be $(A_{\delta(D)} \cup l(\delta(D)))$ and makes an oracle query to the challenger $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyGen}}(\widetilde{D}, mk, pp)$. The challenger generates a CP-ABE decryption key $sk_{\text{ABE},\widetilde{D}}$ if and only if $\widetilde{D} \notin \mathbb{A}^\star$. $\mathcal{C}$ may generate the index $\mathcal{I}_D$ as, up to this point, $\mathbb{A}^\star$ is still $\{\emptyset\}$, and provide $\mathcal{A}_{\text{ABE}}$ with the key.
   - All other oracles run according to their respective algorithms.

4. $\mathcal{A}_{\text{EVSE}}$ outputs the query Q, its choice of pre-index $\delta(D^\star)$, the respective unique label $l(\delta(D^\star))$ as well as the server and data owner states as its challenge parameters.

5. $\mathcal{A}_{\text{ABE}}$ chooses the set of authorised users G.

6. $\mathcal{A}_{\text{ABE}}$ chooses a random id from the user space Users which will be used as input for the algorithm $\mathsf{AddUser}$. It simulates running $\mathsf{EVSE.AddUser}$ for the chosen id creating a valid key for the Query algorithm.

7. To generate the challenge input, $\mathcal{A}_{\text{ABE}}$ begins by choosing three random equal length messages m_0, m_1 and m' from the message space.

   Now $\mathcal{A}_{\text{ABE}}$ needs to choose its challenge access structure $\mathbb{A}^\star$ for the CP-ABE IND-CPA game. First, it computes $r = Q(\delta(D^\star))$. If $r = 0$, $\mathcal{A}_{\text{ABE}}$ sets $\mathbb{A}^\star = (Q \wedge l(\delta(D^\star)))$. Else it sets $\mathbb{A}^\star = (\overline{Q} \wedge l(\delta(D^\star)))$. Next it sends $\mathbb{A}^\star$ and the messages m_0 and m_1 to $\mathcal{C}$ as its challenge parameters for the CP-ABE game. We note that $\mathbb{A}^\star$ is a valid challenge access structure as the only queries made to the KeyGen oracle $\mathcal{O}^{\mathsf{KeyGen}}$ of the CP-ABE IND-CPA game were initiated by queries to the $\mathsf{BuildIndex}$ oracle $\mathcal{O}^{\mathsf{BuildIndex}}$ handled by $\mathcal{A}_{\text{ABE}}$. Note that due to the uniqueness of the labels present in the access structure, no requests to the $\mathsf{BuildIndex}$ oracle for input documents $D' \subset D^\star$ would result in a KeyGen query

for attributes that satisfy $\mathbb{A}^\star$. If the oracle is queried for $D' \supseteq D^\star$ then we can observe that $\mathbb{A}^\star$ was chosen specifically such that it is unsatisfied on this input. Thus, KeyGen is never queried for an attribute set that satisfies $\mathbb{A}^\star$, and therefore the challenge is valid.

$\mathcal{C}$ chooses a random bit b and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_b, \mathbb{A}^\star, mpk_{\mathrm{ABE}})$. Upon receiving $ct^\star$, $\mathcal{A}_{\mathrm{ABE}}$ chooses a random bit t which corresponds to $\mathcal{A}_{\mathrm{ABE}}$'s guess for the challenger's choice of b.

- If $r = 1$, $\mathcal{A}_{\mathrm{ABE}}$ generates

$$c \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', Q \wedge l(\delta(D^\star)), mpk_{\mathrm{ABE}})$$

  and sets $c' = ct^\star$ (formed over $\mathbb{A}^\star$ by $\mathcal{C}$). It also sets $vk = g(m')$ and $vk' = g(m_t)$.
- Else $r = 0$, and $\mathcal{A}_{\mathrm{ABE}}$ sets $c = ct^\star$ and computes

$$c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \overline{Q} \wedge l(\delta(D^\star)), mpk_{\mathrm{ABE}}).$$

  It sets $vk = g(m_t)$ and $vk' = g(m')$.

Finally, $\mathcal{A}_{\mathrm{ABE}}$ sets $qt_Q = (c, c')$ and $vk_Q = (vk, vk')$.

8. $\mathcal{A}_{\mathrm{ABE}}$ sends the output from Query along with the public information to $\mathcal{A}_{\mathrm{EVSE}}$, which is also given oracle access to which $\mathcal{A}_{\mathrm{ABE}}$ responds as follows.
   - $\mathsf{EVSE.BuildIndex}(\cdot, \cdot, \cdot, mk, pp)$: To generate the index for the queried database D, $\mathcal{A}_{\mathrm{ABE}}$ follows the same procedure as specified in step 3. However, we note that by the definition of the access structure $\mathbb{A}^\star$, $\widetilde{D}$ satisfies $\mathbb{A}^\star$ only if the unique labels in $\widetilde{D}$ and $\mathbb{A}^\star$ match. Furthermore, it follows that $\delta(D^\star)$ must satisfy either Q or $\overline{Q}$ as chosen in $\mathbb{A}^\star$. However, this was chosen specifically such that $\delta(D^\star)$ does not satisfy the query, and therefore $\widetilde{D} \notin \mathbb{A}^\star$ and $\mathcal{C}$ may generate the key which corresponds to the index.
   - All other oracles run according to their respective algorithms.
9. Eventually, $\mathcal{A}_{\mathrm{EVSE}}$ outputs $\theta^\star$ which it believes is a valid forgery (i.e. that the output will be accepted yet does not correspond to the correct value of $Q(\delta(D^\star))$).
10. $\mathcal{A}_{\mathrm{ABE}}$ parses $\theta^\star$ as (d, d'). One of d and d' will be $\perp$ (by construction) and we denote the other value by Y. Observe that, since $\mathcal{A}_{\mathrm{EVSE}}$ is assumed to be a successful adversary against public verifiability, the non-$\perp$ value, Y, that it will return will be the plaintext m_t since the challenge access structure was always set to be unsatisfied on the challenge input. Thus, if $g(Y) = g(m_t)$, $\mathcal{A}_{\mathrm{ABE}}$ outputs a guess $b' = t$ and otherwise guesses $b' = (1 - t)$.

If $t = b$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game 0** (since the verification key comprises $g(m')$ where m' is the message a legitimate server could recover, and $g(m_b)$ where m_b is the other plaintext).
Alternatively, $t = 1 - b$ and the distribution of the above experiment is identical to **Game 1** (since the verification key comprises the legitimate message and a random message m_{1-b} that is unrelated to the ciphertext).

Now, we consider the advantage of this constructed adversary $\mathcal{A}_{\text{ABE}}$ playing the IND-CPA game for CP-ABE. Recall that by assumption, $\mathcal{A}_{\text{EVSE}}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1**, that is

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right]\right| \geqslant \delta$$

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{\text{VDC}}}\left[\mathcal{EVSE}, 1^\lambda\right]$ denotes the output of running $\mathcal{A}_{\text{EVSE}}$ in **Game i**.
The probability of $\mathcal{A}_{\text{ABE}}$ guessing b correctly is:

$$\begin{aligned}
\Pr[b' = b] &= \Pr[t = b]\Pr[b' = b | t = b] + \Pr[t \neq b]\Pr[b' = b | t \neq b] \\
&= \frac{1}{2}\Pr[g(Y) = g(m_t) | t = b] + \frac{1}{2}\Pr[g(Y) \neq g(m_t) | t \neq b] \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right] + \frac{1}{2}\left(1 - \Pr[g(Y) = g(m_t) | t \neq b]\right) \\
&= \frac{1}{2}\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right] \\
&\quad + \frac{1}{2}\left(1 - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right]\right. \\
&\quad \left. - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\text{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right] + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1).
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\text{ABE}}} &\geqslant \left|\Pr[b = b'] - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&= \frac{\delta}{2}.
\end{aligned}$$

Therefore, if $\mathcal{A}_{\text{EVSE}}$ has advantage δ at distinguishing these games then $\mathcal{A}_{\text{ABE}}$ can win the IND-CPA game for CP-ABE with non-negligible probability. Thus since we assumed the CP-ABE scheme to be secure, we conclude that $\mathcal{A}_{\text{EVSE}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is simply to set the value of m' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game. We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{\text{EVSE}}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{\text{ABE}}$).

Final Proof. We now show that using $\mathcal{A}_{\text{EVSE}}$ in **Game 2**, $\mathcal{A}_{\text{ABE}}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ we can recover w. Specifically, during Query, we choose the messages as follows:

- if $Q(\delta(D^\star)) = 1$, we implicitly set m_1 to be w and set the verification key component $vk' = z$. We choose m_0 randomly from the message space and compute the remainder of the verification key as usual.
- if $Q(\delta(D^\star)) = 0$, we implicitly set m_0 to be w and set the verification key component $vk = z$. We choose m_1 randomly from the message space and compute the remainder of the verification key as usual.

Now, since $\mathcal{A}_{\text{EVSE}}$ is assumed to be successful, it will output a forgery comprising the plaintext encrypted under the unsatisfied query (Q or $\overline{Q}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{\text{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\text{EVSE}}$ has against the public verifiability game.

We conclude that if the CP-ABE scheme is IND-CPA secure and the one-way function is hard-to-invert, then $\mathcal{EVSE}$ as defined by Algorithms 1–7 is secure in the sense of public verifiability. □

6.5.2 Index Privacy

Lemma 6.8. *$\mathcal{EVSE}$ as defined in Algorithms 1–7 is secure in the sense of index privacy (Figure 6.3) under the same assumptions as in Theorem 6.6.*

Proof. Suppose $\mathcal{A}_{\text{EVSE}}$ is an adversary against the selective index privacy game (Figure 6.3) when instantiated with Algorithms 1-7. We begin by defining the following two games:

- **Game 0.** This is the selective index privacy game as defined in Figure 6.3.
- **Game 1.** This is the same as **Game 0** with the modification that we use a random permutation in Algorithm 1 to construct $\widetilde{\mathcal{U}}$.

As a first step of the proof, we show that there is a negligible function negl such that

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right]\right| \leq \mathrm{negl}(\lambda).$$

We use $\mathcal{A}_{\mathrm{EVSE}}$ to construct a distinguisher $\mathcal{D}$ for the permutation Π, and its goal is to determine whether this permutation is "pseudorandom" or "random" by observing whether $\mathcal{A}_{\mathrm{EVSE}}$ succeeds. The distinguisher is provided with oracle access $\mathcal{O}$ to some function which either outputs a pseudorandom permutation or random permutation during any invocation. If the adversary $\mathcal{A}_{\mathrm{EVSE}}$ succeeds then $\mathcal{D}$ guesses that its oracle must be a pseudorandom permutation, whereas if $\mathcal{A}_{\mathrm{EVSE}}$ does not succeed then $\mathcal{D}$ guesses that its oracle must be a random permutation.

The distinguisher $\mathcal{D}$ starts by simulating the selective index privacy game for $\mathcal{A}_{\mathrm{EVSE}}$, and queries the oracle $\mathcal{O}$ for any invocation in line 4 of the EVSE.Setup algorithm. The game proceeds as specified and eventually $\mathcal{A}_{\mathrm{EVSE}}$ outputs a bit b', and the distinguisher outputs 1 if $b' = b$ indicating that the adversary wins and 0 otherwise.
Now if $\mathcal{D}$'s oracle is a pseudorandom permutation, then the adversary's view when run as a sub-routine by the distinguisher is distributed identically to the adversary's view in **Game 0**, and hence

$$\Pr_{\kappa \leftarrow\$ \{0,1\}^\lambda}\left[\mathcal{D}^{\Pi_\kappa(\cdot)}(1^\lambda) \to 1\right] = \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right].$$

Similarly, it follows that if $\mathcal{D}$'s oracle is a random permutation, then the adversary's view when run as a sub-routine by the distinguisher is distributed identically to the adversary's view in **Game 1**, and hence

$$\Pr_{\widetilde{\Pi} \leftarrow\$ \mathrm{Perm}}\left[\mathcal{D}^{\widetilde{\Pi}(\cdot)}(1^\lambda) \to 1\right] = \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right].$$

By the assumption that Π is a pseudorandom permutation, it holds that

$$\left|\Pr\left[\mathcal{D}^{\Pi_\kappa(\cdot)}(1^\lambda) \to 1\right] - \Pr\left[\mathcal{D}^{\widetilde{\Pi}(\cdot)}(1^\lambda) \to 1\right]\right| \leq \mathrm{negl}(\lambda),$$

and hence it follows that the adversary's advantage in distinguishing which game it plays is negligible.

Reduction to IND-CPA. Now let $\mathcal{A}_{\mathrm{EVSE}}$ be an adversary with non-negligible advantage δ against **Game 1**. We then construct an adversary $\mathcal{A}_{\mathrm{SE}}$ that uses $\mathcal{A}_{\mathrm{EVSE}}$ as a sub-routine to break the IND-CPA security of the symmetric encryption scheme $\mathcal{SE}$. We consider a challenger $\mathcal{C}$ playing the IND-CPA game (Figure 2.1) with $\mathcal{A}_{\mathrm{SE}}$, that in turn acts as a challenger in **Game 1** for $\mathcal{A}_{\mathrm{EVSE}}$:

1. $\mathcal{A}_{\mathrm{EVSE}}$ chooses its two challenge data sets, namely $D_0 = (d_{0,1}, d_{0,2}, ..., d_{0,q})$ and

$D_1 = (d_{1,1}, d_{1,2}, ..., d_{1,q}) \subseteq \mathcal{U}$ such that both contain the same number of elements ($|D_0| = |D_1|$). Note that a rational adversary will always choose the two sets in such a way that they differ by at least one element. $\mathcal{A}_{\text{EVSE}}$ submits these challenge sets to $\mathcal{A}_{\text{SE}}$.

2. $\mathcal{A}_{\text{SE}}$ aborts the game in case the challenge data sets do not contain the same number of elements and $\mathcal{A}_{\text{EVSE}}$ loses immediately.

3. $\mathcal{A}_{\text{SE}}$ simulates running $\mathsf{EVSE.Setup}$ and is provided with oracle calls to the LoR encryption oracle by $\mathcal{C}$. The adversary $\mathcal{A}_{\text{SE}}$ passes both sets D_0 and D_1 to the challenger as its challenge input parameters for the IND-CPA game. In more detail, the challenger handles the LoR oracle call in the following way.

   - $\mathcal{C}$ samples a random bit $\widetilde{b}$ which it uses for the LoR encryption oracle.
   - The challenger inputs every pair $(d_{0,i}, d_{1,i}) \in D_0 \times D_1$ to the LoR oracle $\mathcal{O}^{\mathsf{LoR}}(d_{0,i}, d_{1,i}, k_{\text{SE}}, \widetilde{b})$ outputting challenge ciphertexts $ct_{\widetilde{b},i}$. All returned ciphertexts then form the set $D^\star$.

4. $\mathcal{A}_{\text{SE}}$ simulates the $\mathsf{ABE.Setup}$ algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{EVSE}}$. It retains msk_{ABE} and sets the public parameters to $pp \leftarrow (mpk_{\text{ABE}}, \widetilde{\mathcal{U}})$.

5. $\mathcal{A}_{\text{SE}}$ chooses a random bit $b \xleftarrow{\$} \{0,1\}$.

6. $\mathcal{A}_{\text{SE}}$ initialises the set of authorised users G from the user space.

7. $\mathcal{A}_{\text{SE}}$ needs to create the attribute representation of the data set $D^\star$, i.e. it requires to create a pre-index $\delta(D^\star)$ to encode it into the challenge index for $\mathcal{A}_{\text{EVSE}}$. This is done using the algorithm Encode, which takes as input the elements from the challenge set $D^\star$ and maps them to elements in $\widetilde{\mathcal{U}}$. It further also outputs an attribute label for the pre-index $l(\delta(D^\star))$.

8. $\mathcal{A}_{\text{SE}}$ runs $\mathsf{BuildIndex}$ using the pre-index created in the previous step to create the challenge index $\mathcal{I}_{D^\star}$ for $\mathcal{A}_{\text{EVSE}}$. In more detail, $\mathcal{A}_{\text{SE}}$ executes $\mathcal{I}_{D^\star} \leftarrow_{\$} \mathsf{ABE.KeyGen}((A_{\delta(D^\star)} \cup l(\delta(D^\star))), msk_{\text{ABE}}, mpk_{\text{ABE}})$ and provides $\mathcal{A}_{\text{EVSE}}$ with the challenge index. $\mathsf{BuildIndex}$ continues as specified and generates st_{s} and st_{o}. st_{s} is shared with $\mathcal{A}_{\text{EVSE}}$ and st_{o} is retained by $\mathcal{A}_{\text{SE}}$.

9. $\mathcal{A}_{\text{EVSE}}$ receives all relevant outputs from above, and then is provided with oracle access to which $\mathcal{A}_{\text{SE}}$ responds. All oracles are executed as specified in their respective algorithm. The only relevant restriction is that $\mathcal{A}_{\text{EVSE}}$ cannot request an index via a $\mathsf{BuildIndex}$ oracle query for the initial choice of data sets D_0 and D_1.

10. Eventually $\mathcal{A}_{\text{EVSE}}$ outputs its guess b' for b.

11. If $b' = b$, then $\mathcal{A}_{\text{SE}}$ guesses $\bar{b} = b$ as its guess for the IND-CPA game.

    Else $b' \neq b$, then $\mathcal{A}_{\text{SE}}$ makes a random guess $\bar{b} = \hat{b} \xleftarrow{\$} \{0,1\}$ since $\mathcal{A}_{\text{EVSE}}$ is of no use for $\mathcal{A}_{\text{SE}}$ to break the IND-CPA game.

Now we consider the advantage of $\mathcal{A}_{\text{SE}}$ playing the IND-CPA game. By assumption, $\mathcal{A}_{\text{EVSE}}$ has a non-negligible advantage against the selective index privacy game, i.e. $\Pr[b' = b] \geq \delta + \frac{1}{2}$. Therefore it follows:

$$\begin{aligned}
\Pr[\bar{b} = \widetilde{b}] &= \Pr[\bar{b} = \widetilde{b} | b' = b] \Pr[b' = b] + \Pr[\bar{b} = \widetilde{b} | b' \neq b] \Pr[b' \neq b] \\
&= \Pr[b = \widetilde{b}] \Pr[b' = b] + \Pr[\hat{b} = \widetilde{b}] \Pr[b' \neq b] \\
&= \frac{1}{2} \Pr[b' = b] + \frac{1}{2} \Pr[b' \neq b] \\
&= \frac{1}{2} \big(\Pr[b' = b] + \Pr[b' \neq b] \big) \\
&\geq \frac{1}{2} \Big(\delta + \frac{1}{2} + \frac{1}{2} \Big) = \frac{1}{2}(\delta + 1).
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\text{SE}}} &\geqslant \left| \Pr[\bar{b} = \widetilde{b}] - \frac{1}{2} \right| \\
&\geqslant \left| \frac{1}{2}(\delta + 1) - \frac{1}{2} \right| \\
&= \frac{\delta}{2}.
\end{aligned}$$

Since δ is non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\text{EVSE}}$ has advantage δ at breaking the selective index privacy game then $\mathcal{A}_{\text{SE}}$ can win the IND-CPA game with non-negligible probability. However, since the SE scheme was assumed IND-CPA secure, such an $\mathcal{A}_{\text{EVSE}}$ cannot exist. Therefore, we conclude that if the SE scheme is IND-CPA secure then $\mathcal{EVSE}$ as instantiated by Algorithms 1–7 is secure in the sense of selective index privacy. □

6.5.3 Query Privacy

Lemma 6.9. *$\mathcal{EVSE}$ as defined in Algorithms 1–7 is secure in the sense of query privacy (Figure 6.4) under the same assumptions as in Theorem 6.6.*

Proof. This proof is very similar to the proof of Lemma 6.8. We begin by assuming that $\mathcal{A}_{\text{EVSE}}$ is an adversary with non-negligible advantage against the query privacy game (Figure 6.4) when instantiated with Algorithms 1–7. We begin similarly to the previous proof by defining the following two games:

- **Game 0.** This is the selective query privacy game as defined in Figure 6.4.

- **Game 1.** This is the same as **Game 0** with the modification that we use a random permutation in Algorithm 1 to construct $\widetilde{\mathcal{U}}$.

As a first step of the proof, we show that there is a negligible function negl such that

$$\left|\Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right] - \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right]\right| \leq \mathrm{negl}(\lambda).$$

We use $\mathcal{A}_{\mathrm{EVSE}}$ to construct a distinguisher $\mathcal{D}$ for the permutation Π, and its goal is to determine whether this permutation is "pseudorandom" or "random" by observing whether $\mathcal{A}_{\mathrm{EVSE}}$ succeeds. The distinguisher is provided with oracle access $\mathcal{O}$ to some function which either outputs a pseudorandom permutation or random permutation during any invocation. If the adversary $\mathcal{A}_{\mathrm{EVSE}}$ succeeds then $\mathcal{D}$ guesses that its oracle must be a pseudorandom permutation, whereas if $\mathcal{A}_{\mathrm{EVSE}}$ does not succeed then $\mathcal{D}$ guesses that its oracle must be a random permutation.

The distinguisher $\mathcal{D}$ starts by simulating the selective index privacy game for $\mathcal{A}_{\mathrm{EVSE}}$, and queries the oracle $\mathcal{O}$ for any invocation in line 4 of the $\mathsf{EVSE.Setup}$ algorithm. The game proceeds as specified and eventually $\mathcal{A}_{\mathrm{EVSE}}$ outputs a bit b', and the distinguisher outputs 1 if $b' = b$ indicating that the adversary wins and 0 otherwise.
Now if $\mathcal{D}$'s oracle is a pseudorandom permutation, then the adversary's view when run as a sub-routine by the distinguisher is distributed identically to the adversary's view in **Game 0**, and hence

$$\Pr_{\kappa \leftarrow\$ \{0,1\}^\lambda}\left[\mathcal{D}^{\Pi_\kappa(\cdot)}(1^\lambda) \to 1\right] = \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right].$$

Similarly, it follows that if $\mathcal{D}$'s oracle is a random permutation, then the adversary's view when run as a sub-routine by the distinguisher is distributed identically to the adversary's view in **Game 1**, and hence

$$\Pr_{\widetilde{\Pi} \leftarrow\$ \mathrm{Perm}}\left[\mathcal{D}^{\widetilde{\Pi}(\cdot)}(1^\lambda) \to 1\right] = \Pr\left[\mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{\mathrm{EVSE}}}\left[\mathcal{EVSE}, 1^\lambda\right] \to 1\right].$$

By the assumption that Π is a pseudorandom permutation, it holds that

$$\left|\Pr\left[\mathcal{D}^{\Pi_\kappa(\cdot)}(1^\lambda) \to 1\right] - \Pr\left[\mathcal{D}^{\widetilde{\Pi}(\cdot)}(1^\lambda) \to 1\right]\right| \leq \mathrm{negl}(\lambda),$$

and hence it follows that the adversary's advantage in distinguishing which game it plays is negligible.

Reduction to IND-CPA. Now let $\mathcal{A}_{\mathrm{EVSE}}$ be an adversary with non-negligible advantage δ against **Game 1**. We then construct an adversary $\mathcal{A}_{\mathrm{SE}}$ that uses $\mathcal{A}_{\mathrm{EVSE}}$ as a sub-routine to break the IND-CPA security of the symmetric encryption scheme $\mathcal{SE}$. We consider a challenger $\mathcal{C}$ playing the IND-CPA game (Figure 2.1) with $\mathcal{A}_{\mathrm{SE}}$, that

in turn acts as a challenger in **Game 1** for $\mathcal{A}_{\text{EVSE}}$:

1. $\mathcal{A}_{\text{EVSE}}$ chooses its two challenge queries Q_0 and Q_1 with the restriction that the gate structures match in both queries, i.e. $\mathcal{G}_{Q_0} = \mathcal{G}_{Q_1}$. Note that in case $\mathcal{G}_{Q_0} \neq \mathcal{G}_{Q_1}$ then $\mathcal{A}_{\text{SE}}$ aborts the game and $\mathcal{A}_{\text{EVSE}}$ loses immediately. The queries itself are formed over attributes and a rational adversary will choose those queries in such a way that they differ in at least one position. We denote the used attributes for the challenge queries as $q_0 = (q_{0,1}, q_{0,2}, ..., q_{0,q})$ and $q_1 = (q_{1,1}, q_{1,2}, ..., q_{1,q}) \subseteq \mathcal{U}$, respectively. $\mathcal{A}_{\text{EVSE}}$ submits these challenge queries to $\mathcal{A}_{\text{SE}}$.

2. $\mathcal{A}_{\text{SE}}$ simulates running $\mathsf{EVSE.Setup}$ and is provided with oracle calls to the LoR encryption oracle by $\mathcal{C}$. The adversary $\mathcal{A}_{\text{SE}}$ passes both queries Q_0 and Q_1 to the challenger and the underlying attributes (of the queries) are used as its challenge input parameters for the IND-CPA game. In more detail, the challenger handles the LoR oracle call in the following way.

   - $\mathcal{C}$ samples a random bit $\widetilde{b}$ which it uses for the LoR encryption oracle.
   - The challenger inputs every pair $(q_{0,i}, q_{1,i}) \in q_0 \times q_1$ to the LoR oracle $\mathcal{O}^{\mathsf{LoR}}(q_{0,i}, q_{1,i}, k_{\text{SE}}, \widetilde{b})$ outputting challenge ciphertexts $ct_{\widetilde{b},i}$. All returned ciphertexts then form the attribute set $q^\star$ of the query $Q^\star$.

3. $\mathcal{A}_{\text{SE}}$ simulates the $\mathsf{ABE.Setup}$ algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{EVSE}}$. It retains msk_{ABE} and sets the public parameters to $pp \leftarrow (mpk_{\text{ABE}}, \widetilde{\mathcal{U}})$.

4. $\mathcal{A}_{\text{SE}}$ chooses a random bit $b \xleftarrow{\$} \{0, 1\}$.

5. $\mathcal{A}_{\text{SE}}$ initialises the set of authorised users G from the user space.

6. $\mathcal{A}_{\text{SE}}$ runs $\mathsf{BuildIndex}$ to create the challenge index $\mathcal{I}_D$ for $\mathcal{A}_{\text{EVSE}}$. In more detail, $\mathcal{A}_{\text{SE}}$ executes $\mathcal{I}_D \leftarrow_{\$} \mathsf{ABE.KeyGen}((A_{\delta(D)} \cup l(\delta(D))), msk_{\text{ABE}}, mpk_{\text{ABE}})$ and provides $\mathcal{A}_{\text{EVSE}}$ with the index. $\mathsf{BuildIndex}$ continues as specified and generates st_{s} and st_{o}. st_{s} is shared with $\mathcal{A}_{\text{EVSE}}$ and st_{o} is retained by $\mathcal{A}_{\text{SE}}$.

7. $\mathcal{A}_{\text{SE}}$ selects randomly a new identity id from the user space that it wishes to add into the system. It runs $\mathsf{AddUser}$ to produce sk_{id}.

8. $\mathcal{A}_{\text{SE}}$ needs to prepare the query $Q^\star$ such that it can be used as an input in the query algorithm. This is done using the algorithm Encode, which takes as input the elements and maps them to elements in $\widetilde{\mathcal{U}}$ such that it forms a valid query $\widetilde{Q}$.

9. $\mathcal{A}_{\text{SE}}$ inputs $\widetilde{Q}$ to the query algorithm Query and runs it as specified. The algorithm outputs $qt_{\widetilde{Q}}$ and $vk_{\widetilde{Q}}$ which are both given to $\mathcal{A}_{\text{EVSE}}$.

10. $\mathcal{A}_{\text{EVSE}}$ receives all relevant outputs from above, and then is provided with oracle access to which $\mathcal{A}_{\text{SE}}$ responds. All oracles are executed as specified in their respective algorithms. The only relevant restriction is that $\mathcal{A}_{\text{EVSE}}$ cannot request query tokens via a Query oracle query for the initial choice of challenge queries Q_0 and Q_1.

11. Eventually $\mathcal{A}_{\text{EVSE}}$ outputs its guess b' for b.

12. If $b' = b$, then $\mathcal{A}_{\text{SE}}$ guesses $\bar{b} = b$ as its guess for the IND-CPA game.

    Else $b' \neq b$, then $\mathcal{A}_{\text{SE}}$ makes a random guess $\bar{b} = \hat{b} \xleftarrow{\$} \{0,1\}$ since $\mathcal{A}_{\text{EVSE}}$ is of no use for $\mathcal{A}_{\text{SE}}$ to break the IND-CPA game.

Now we consider the advantage of $\mathcal{A}_{\text{SE}}$ playing the IND-CPA game. By assumption, $\mathcal{A}_{\text{EVSE}}$ has a non-negligible advantage against the selective query privacy game, i.e. $\Pr[b' = b] \geq \delta + \frac{1}{2}$. Therefore it follows:

$$\begin{aligned}
\Pr[\bar{b} = \widetilde{b}] &= \Pr[\bar{b} = \widetilde{b} | b' = b] \Pr[b' = b] + \Pr[\bar{b} = \widetilde{b} | b' \neq b] \Pr[b' \neq b] \\
&= \Pr[b = \widetilde{b}] \Pr[b' = b] + \Pr[\hat{b} = \widetilde{b}] \Pr[b' \neq b] \\
&= \frac{1}{2} \Pr[b' = b] + \frac{1}{2} \Pr[b' \neq b] \\
&= \frac{1}{2} \big(\Pr[b' = b] + \Pr[b' \neq b] \big) \\
&\geq \frac{1}{2} \Big(\delta + \frac{1}{2} + \frac{1}{2} \Big) = \frac{1}{2}(\delta + 1).
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_{\text{SE}}} &\geqslant \left| \Pr[\bar{b} = \widetilde{b}] - \frac{1}{2} \right| \\
&\geqslant \left| \frac{1}{2}(\delta + 1) - \frac{1}{2} \right| \\
&= \frac{\delta}{2}.
\end{aligned}$$

Since δ is non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{\text{EVSE}}$ has advantage δ at breaking the selective query privacy game then $\mathcal{A}_{\text{SE}}$ can win the IND-CPA game with non-negligible probability. However, since the SE scheme was assumed IND-CPA secure, such an $\mathcal{A}_{\text{EVSE}}$ cannot exist. Therefore, we conclude that if the SE scheme is IND-CPA secure then $\mathcal{EVSE}$ as instantiated by Algorithms 1–7 is secure in the sense of selective query privacy. □

6.6 Conclusion

In this chapter, we have introduced a model of multi-user verifiable searchable encryption with extended functionality enabling a querier to form a wider family of queries

including some type of computations, i.e. we extended the expressiveness of queries that can be achieved in VSE. With this work we have begun to consider the application of VC techniques from Chapter 4 in the setting of searchable encryption. On the searchable encryption side, this enables additional functionality in the form of computational queries (e.g. computing the average of outsourced data fields that are linked to a specific set of keywords), whilst on the VC side, this introduces additional privacy concerns regarding the outsourced data and computations. No other VSE schemes to our knowledge are able to perform the range of search queries or include negation of keywords in their search queries. Additionally our scheme leaks neither the access nor the search pattern to the server whilst executing a search. The choice of using VC techniques based on ABE stems from the natural correspondence between attributes and keywords in an index. We provided an instantiation of EVSE based on CP-ABE and presented relevant security models in this context.

In future work, we will investigate whether other forms of VC enable us to achieve different classes of functionality and especially improve efficiency. We would also like to consider a model whereby multiple data owners can store data on a server without each having to initialise their own scheme. In practice, this could result in employing the KDC from the previous chapters setting up the system and publishing public parameters that any data owner can use, but enabling each data owner to generate their own CP-ABE decryption keys for the data they hold to authorise users to query their data without the need to involve the KDC. Furthermore, our scheme currently only supports static data and in future work we wish to investigate extending our scheme to support a dynamic data set as in the schemes [104, 133, 136].

Chapter 7

Cloud Storage Proofs of Retrievability

Contents

This chapter deals with the setting of providing provable data storage guarantees. One possible concept achieving such guarantees is known as proofs of retrievability (PoR). We propose extensions and improvements to the current PoR proposal in order to accommodate a more practical and applicable framework. Thus, we introduce the notion of a storage container enabling a client to store multiple different-sized files and propose a provably secure instantiation checking retrievability of all files simultaneously. We discuss different strategies a client can use in order to obtain a statement about the retrievability from a server and finally evaluate the performance of our scheme. The results of this chapter are currently under submission [97].

7.1 Introduction

Cloud service providers (CSPs) are gaining continuous importance over the last few years as they offer various services in numerous application domains such as storage services, computing services and key management services. Well-known examples of providers are Amazon S3, Google Cloud Platform and Windows Azure. The huge success of the cloud model is based on offering various benefits such as flexible scalability and accessibility offered to companies and individuals to employ cloud services in a cost effective manner. However, to fulfil the client's expectation, the providers need to build applications that are highly available. This can be a daunting process in practice. Therefore, the underlying key assumption for designing such applications is to

assume that every component of the system will fail at some point in time. It is obvious that based on this assumption, there is a need to create highly available services and amongst other things we also require data consistency strategies for the cloud model.

Proofs of retrievability(PoR) [98, 126] offer such a strategy for the cloud's storage domain as they offer mechanisms in order to provide provable outsourced storage guarantees. We have introduced the concept of PoR in Section 2.8 and it has been subject to a vast amount of research over the past decade. In PoR the guarantees are formulated in terms of a checkable statement whether the client's outsourced data is *authentic* and *retrievable*. The first property expresses that the client wishes to verify that the received data is correct whereas the latter property reflects the need to assure that no data loss has occurred on the server. Unfortunately, CSPs do not offer those guarantees provably yet. Facing current trends where clients and companies produce and outsource a vast amount of data every day [60] forces the provider to invest in a large storage infrastructure as it seems to be common practice to produce triplicas (triple replication) [8, 76] to compensate against data loss. To reduce the storage overhead incurred by replicas, providers may transition to erasure codes [94, 102]. Therefore, there is an actual need to employ PoR in the realm of cloud storage and increasing their practicability will play a crucial role in the near future of cloud security.

CSPs offer cheap storage and computing solutions for clients. Thus, it is tempting for a client to outsource all her (different-sized) files to a provider as it is easy to access the files through various different (computational and storage restricted) devices. However, by outsourcing the data, a client somehow loses the "control" over it and thus would naturally wish to receive a provable assurance from a provider that the data is intact and retrievable at any time. To provide such an assurance, a cloud service provider could easily adapt the notion of a PoR into its architecture. Current PoR systems can indeed provide such an assurance however, they are limited to check a single file at a time. Thus, in this chapter, we introduce a new notion called *cloud storage proofs of retrievability* (CSPoR) that is designed to efficiently provide an assurance that all files are simultaneously retained by a server.

First, let us recall that current PoR schemes only provide a probabilistic assurance to a client that the data is retained by arguing that deleted file blocks can be found with overwhelming probability, i.e. a cheating server will be caught with very high probability. In contrast, we believe that such an assurance does not suffice in practice where a client wishes to receive a concrete *retrievability assurance*, i.e. an assurance that enough portions of data are available such that reconstruction of the underlying (original) file is possible. This assurance cannot usually be obtained with a single check but rather requires several checks until the client checked sufficiently many data portions. Second,

in case a verifier would reject a PoR instance, it currently seems that a client engages in an immediate download of her file in order to protect the data from further harm, reflecting a loss of trust in the provider. This "strategy" of downloading, however, is impractical as already a detected single bit erasure may lead to a `reject` decision within a PoR scheme and a client loses all benefits of outsourcing the data in the first place. This propagates to an even bigger problem in our cloud storage scenario where a client stores many different-sized files: already a small erasure in any one file would then lead to download all files as a client is not aware which file(s) is corrupted, and this behaviour renders the PoR functionality to be impractical for providing real-life provable storage guarantee. However, we believe that both arguments are linked, and show throughout this chapter that a CSPoR system is able to provide a retrievability assurance about the files. We also discuss in detail different strategies that, on one hand, provide the client with a flexible way to form valid PoR requests, and on the other hand, provide a strategy to handle invalid instances differently than downloading all files.

Our main contribution in this chapter is to introduce the new notion of *Cloud Storage PoR* (CSPoR). Our scheme has the following features:

- our scheme captures the client's need to store *many* (different-sized) files and enables her to efficiently check whether *all* files (or any subsets) are still retained by the server. To store several files we introduce the notion of a *storage container* which represents the underlying storage structure of cloud storage systems;
- in order to obtain a retrievability assurance we need to perform several audit steps. Therefore, we investigate the natural relation between the cost of computation (choosing challenges) and communication (sending and receiving challenges and replies respectively). We provide different strategies enabling the client to assess the involved costs. Furthermore, we discuss a strategy, other than immediate downloading of all files, that a client can employ in case a PoR verification has failed;
- our solution is realisable in terms of an abstract API that encompasses cloud APIs (e.g. from Amazon, Google and Windows) as special cases. Therefore, CSPoR can be translated into the cloud storage framework without huge amendments for the CSP;
- we evaluate the performance of CSPoR showing two examples where we outsourced per example 480 different-sized files. In the first example, the files are of size at most 32 MiB with an overall outsourced file size of 7.5 GiB and our scheme requires at most 7 seconds to obtain a retrievability assurance about those files. In the second example, the outsourced files are of size at most 512 GiB where

the overall outsourced storage volume is 65 TiB. Unfortunately, for large files the performance suffers and it takes hours to check retrievability.

We provide a rigorous definitional framework for CSPoR that we believe more accurately reflects real cloud environments than previously considered in the realm of PoR.

Related Work

In a recent and concurrent work, Paterson et al. [120] introduce a multiple server PoR system capturing new security models as the underlying entity population is different compared to classical PoR frameworks. The authors use the same idea of employing a statistical hypothesis test in order to evaluate whether the responses of the prover are sufficient to permit successful extraction. Since Paterson et al. consider a multiple server PoR system where the success probability between different provers for the same event may differ, the total number of successes follow a Poisson-binomial distribution and thus they use different statistical techniques compared to our approach.

Organisation of Chapter

In Section 7.2 we introduce our new system model framework for our cloud storage PoR scheme. This new model leads to an updated and extended security model which we discuss in detail in Section 7.3. This is followed, in Section 7.4, by a concrete instantiation of our CSPoR scheme and the respective security proof. In Section 7.5 we discuss different audit strategies a client could use in order to check the files. In Section 7.6 we provide a performance evaluation for our scheme showing that the scheme behaves reasonably. We conclude the chapter in Section 7.7.

7.2 Cloud Storage Proofs of Retrievability

The essential components of a *cloud storage proofs of retrievability* system are natural generalisations of "classical" PoR systems. We aim to enhance the current PoR models to enable a client to store f multiple (different-sized) files $F^{(1)}, \ldots, F^{(f)}$ with a provider. However, trivial extensions of known solutions do not support the multiple files case well. In more detail, one may simply perform a separate PoR for each file which is infeasible due to the increased workload which scales in the number of files over all procedures and requires the client to check each file individually. Another simple approach could be to concatenate all files into one file $\widehat{F} = F^{(1)} \| F^{(2)} \| \ldots \| F^{(f)}$ and execute a PoR scheme for the composed file $\widehat{F}$. Unfortunately, the type of ECC encoding required in this context becomes the bottleneck rendering this approach to be infeasible. One type of encoding over all files at once results in a *processed* file of the form $\mathcal{F} = F^{(1)} \| F^{(2)} \| \ldots \| F^{(f)} \| P^{(1,\ldots,f)}$ where $P^{(1,\ldots,f)}$ denotes the added re-

dundancy generated over the concatenation of the files. In this particular case it is possible to use existing PoR notions, however they suffer other drawbacks such that one should avoid using them. For example, in case one wishes to update a single file $F^{(i)}$, $i \in [f] := \{1, \ldots, f\}$, then she is required to download the whole processed file $\mathcal{F}$ since the redundancy was generated over all files present in the concatenation and thus makes updating files an expensive endeavour. Furthermore, in case a PoR check fails it is unclear in which file(s) the error has occurred and thus forces the client to download the whole processed file and therefore she looses her initial advantage of outsourcing the files in the first place. Another type of ECC encoding results in obtaining a processed file of the form $\mathcal{F} = F^{(1)} \| P^{(1)} \| F^{(2)} \| P^{(2)} \| \ldots \| F^{(f)} \| P^{(f)}$ where each original file $F^{(i)}$, $i \in [f]$, is initially processed before all files are concatenated and thus all parity parts $P^{(i)}$ are independent from each other. In this context, updating a file $F^{(i)}$ is easier since we can solely download the required one while simultaneously this approach suffers the *small file problem*. In more detail, if some file $F^{(i)}$ of the processed file $\mathcal{F}$ is small (e.g. the file solely consists of a password) then there is a non-negligible probability that the employed random checking mechanism does not examine this file at all and hence even a successful *PoR* check does not provide sufficient assurance about the retrievability of all files. Yet another drawback of this approach is that in case any file and respective parity blocks are deleted then this specific file is completely irrecoverable.

We overcome the above problem using our new PoR notion called *cloud storage proofs of retrievability* (CSPoR) scheme. In more detail, our scheme first enables a client to store multiple different-sized files in a set of storage containers located at the cloud service provider and the client may form an aggregated challenge query checking all files simultaneously. The server evaluates the challenge query and returns verification values which enable the client to form a PoR answer.

Following previous works on PoR, we also utilise an erasure-correction code (ECC) in our system requirements. ECC is a process that adds redundant data to the original data in such a way that a receiver may recover the processed data even when a number of erasures were introduced, either during the transmission of the file, or while storing the file. Furthermore, in the context of PoR, ECC boosts the probability to detect a misbehaving server since the server is required to delete more data than the original file initially consisted of in order to make the file irrecoverable. We require the ECC to be a *systematic* code to maximise data output whilst also being a *maximum distance separable* (MDS) code which yields maximum reliability with a minimum amount of storage overhead. We denote the erasure-correction code rate by $0 < \rho \leq 1$ and note that incorrect data can be recovered by the decoding procedure if no more than $(1 - \rho)$ of the data have been deleted. [1] Reed-Solomon codes meet those properties for any

[1] One can think of $(1 - \rho)$ as an abbreviation for $(1 - \rho) \cdot 100$ percent where $0 < \rho \leq 1$.

size of code and data words. A notable fast variant like Cauchy-Reed-Solomon codes [102] may be used to enhance the involved encoding time of processing the data.

We consider a rational malicious server that may try to delete bits, blocks, or rearrange files in order to make storage space available and to obtain financial benefits. We call this an adversarial erasure strategy in contrast to a purely random erasure strategy which may lead to make the file completely irretrievable for a client. To prevent successful adversarial erasure a client may encrypt and permute the parity part of the file, for example cf. [14].

In the remainder of this section we introduce the notion of *storage container* which is the underlying storage unit for our scheme. We also provide a generic definition of CSPoR.

7.2.1 Storage Container

Let us first introduce the notion of a *storage container* which acts as a storage unit being able to store multiple files within one location, also providing a client with a file system structure. This notion is motivated upon real-world cloud storage practices where common CSPs use a similar notion of storage containers called *Buckets* [8, 86] or *Blobs* [109].

Let a *storage container* be denoted by $\mathfrak{S}$ storing multiple arbitrary files $F \in \{0,1\}^*$. Throughout this chapter, we assume that a client may possess several storage containers $\mathfrak{S}^{(c)}$ hosted at a cloud service provider. This describes the client's potential need to handle different types of data in different storage containers, e.g. a client wishes to store important documents separately from her picture library. This also captures the common cloud storage practice where each storage container's storage space (size) is upper bounded by $\mathfrak{S}_{\max}^{(c)}$ which corresponds to the maximum number of storable files within a container, and thus we need the possibility to create multiple containers. We denote the total number of different storage containers by $\Gamma \in \mathbb{N}$. The set of all storage containers is denoted by $\widehat{\mathfrak{S}}$ and is called a *cloud storage*.

7.2.2 Formal Definition of CSPoR

We now present a formal definition of CSPoR. This scheme enables a client to check whether *all* files within a cloud storage are retained and retrievable from a cloud service provider.

Definition 7.1. *A* cloud storage proofs of retrievability (CSPoR) scheme *comprises the following procedures:*

- $(pk, sk, \widehat{\mathfrak{S}}, \widehat{\mathfrak{S}_{\mathrm{id}}}, \widehat{\gamma}) \xleftarrow{\$} \mathsf{CSPoRSetup}(1^\lambda)$*: this randomised algorithm generates a*

public-private key pair and takes as input the security parameter λ. Additionally, it creates a cloud storage (i.e. a set of Γ storage containers) $\widehat{\mathfrak{S}} := \{\mathfrak{S}^{(c)} \mid c \in [\Gamma]\}$ *and their set of respective associated unique identifiers* $\widehat{\mathfrak{S}_{\mathrm{id}}} := \{\mathfrak{S}_{\mathrm{id}}^{(c)} \mid c \in [\Gamma]\}$. *Furthermore, some metadata* $\widehat{\gamma} := \{\gamma^{(c)} \mid c \in [\Gamma]\}$ *is created for each storage container;*

- $(\widehat{\mathcal{F}}, \widehat{\tau}, \widehat{\mathfrak{S}}, \widehat{\gamma}) \xleftarrow{\$} \mathsf{CSPoRStore}(sk, \widehat{F}, \widehat{\mathfrak{S}_{\mathrm{id}}})$*: this randomised data storing algorithm takes as input a secret key sk,* $\widehat{\mathfrak{S}_{\mathrm{id}}}$ *and the set of all files* $\widehat{F} := \{\widehat{F}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \mid c \in [\Gamma]\}$ *a client wishes to store at the cloud service provider. Each* $\widehat{F}_{\mathfrak{S}_{\mathrm{id}}^{(c)}}$ *consists of* $K_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \in \mathbb{N}$ *files that will be stored within a particular* $\mathfrak{S}^{(c)}$ *where* $\widehat{F}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} := \{F^{(k)} \mid F^{(k)} \in \{0,1\}^*, k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]\}$ *for* $c \in [\Gamma]$. *Each file within each storage container gets processed yielding the set of all processed files for this storage container* $\widehat{\mathcal{F}}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} := \{\mathcal{F}^{(k)} \mid k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]\}$ *and a respective set of file tags is generated* $\widehat{\tau}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} := \{\tau^{(k)} \mid k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]\}$ *where each tag contains additional information (e.g. metadata) about the processed file. Finally the algorithm outputs the set of all such processed files* $\widehat{\mathcal{F}} := \{\widehat{\mathcal{F}}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \mid c \in [\Gamma]\}$*, tags* $\widehat{\tau} := \{\widehat{\tau}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \mid c \in [\Gamma]\}$ *and the updated cloud storage* $\widehat{\mathfrak{S}}$. *The metadata* $\widehat{\gamma}$ *is also updated;*

- $\delta \xleftarrow{\$} \left[\mathsf{CSPoRVerify}(pk, sk, \widehat{\tau}', \widehat{\mathfrak{S}_{\mathrm{id}}}) \rightleftharpoons \mathsf{CSPoRProve}(pk, \widehat{\mathcal{F}}', \widehat{\tau}', \widehat{\mathfrak{S}})\right]$*: this challenge-response protocol defines a protocol for proving cloud storage retrievability. The prover algorithm takes as input the public key pk, the file tag set* $\widehat{\tau}' := \{\widehat{\tau}'_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \mid \widehat{\tau}'_{\mathfrak{S}_{\mathrm{id}}^{(c)}} = \{\tau^{(k)} \mid k \in K'_{\mathfrak{S}_{\mathrm{id}}^{(c)}}\}, c \in \Gamma'\}$ *and the set of the processed files* $\widehat{\mathcal{F}}' := \{\widehat{\mathcal{F}}'_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \mid \widehat{\mathcal{F}}'_{\mathfrak{S}_{\mathrm{id}}^{(c)}} := \{\mathcal{F}^{(k)} \mid k \in K'_{\mathfrak{S}_{\mathrm{id}}^{(c)}}\}, c \in \Gamma'\}$*, where* $K'_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \subseteq [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]$ *and* $\Gamma' \subseteq [\Gamma]$. *The verification algorithm uses as input the key pair* (pk, sk)*, the file tag set* $\widehat{\tau}'$ *and identifiers* $\widehat{\mathfrak{S}_{\mathrm{id}}}$. *Algorithm* CSPoRVerify *outputs at the end of the protocol execution a binary value* δ *which equals* `accept` *if the verification succeeds, indicating the files in* $\widehat{\mathcal{F}}'$ *are being stored and retrievable from S, and* `reject` *otherwise.*

At the beginning of the CSPoRSetup procedure, the involved parties agree on the storage containers in the set $\widehat{\mathfrak{S}}$. Similarly, they agree on the files in the set $\widehat{\mathcal{F}}$ at the beginning of the CSPoRStore procedure. Note this does not require the files being given in clear within the agreement. An agreement could also consist of hashes of these files.
Note that the cloud storage $\widehat{\mathfrak{S}}$ may already contain data from previously performed CSPoRStore procedures. Furthermore, observe that $\widehat{\mathcal{F}}$ may not be exactly equal to $\widehat{F}$ but it must be guaranteed that $\widehat{F}$ can be recovered from $\widehat{\mathcal{F}}$.
We wish to remark that the involved file tag set $\widehat{\tau}'$ in the challenge-response protocol can correspond to either the full set of file tags $\widehat{\tau}$ or any subset of file tags that enable a CSPoR scheme to flexibly check any subset of files by specifying the appropriate tags.

Definition 7.2. *We denote the challenge-response procedure*

$$\left[\mathsf{CSPoRVerify}(pk, sk, \widehat{\tau}', \widehat{\mathfrak{S}_{\mathrm{id}}}) \rightleftharpoons \mathsf{CSPoRProve}(pk, \widehat{\mathcal{F}}', \widehat{\tau}', \widehat{\mathfrak{S}})\right]$$

by CSPoRP*. A single challenge-response step of a* CSPoRP *is called an* audit*.*

Our CSPoR scheme captures the same probabilistic assurance about the retrievability of the data as classical PoR schemes. However, as mentioned in the introduction, such a probabilistic assurance may not suffice in practice. Therefore, our scheme aims to provide a retrievability assurance about the files, i.e. informally we need to sample a certain minimum amount of file block information (depending on the file size and the ECC rate) per file in order to be assured about the retrievability of the files. Basically, to receive such an assurance we begin to execute CSPoRP several times on different challenge inputs and we denote the output as the ith audit by $\delta_i \in \{\texttt{accept}, \texttt{reject}\}$. Note that if the ith audit fails, i.e. $\delta_i = \texttt{reject}$, then the CSPoRP procedure is altogether not valid and outputs $\delta = \texttt{reject}$. However, even if some audit fails, this may not reflect whether the data is actually deleted or if simply not enough valid file blocks have been checked. Thus, we provide different *audit strategies* capturing, on the one hand, approaches to form flexible verification requests depending on a client's current resources in order to evaluate whether the responses of the prover are sufficient to permit successful extraction, and on the other hand, we discuss approaches a client can employ to detect which file(s) have been corrupted in case an audit has failed. Those approaches can be found in Section 7.5.

Informally, a CSPoR protocol is *correct* if all processed files $\mathcal{F}$ outputted by the store procedure will be accepted by the verification algorithm when interacting with a valid prover. More formally this is captured as follows.

Definition 7.3. *A CSPoR protocol is* correct *if there exists a negligible function* negl *such that for every security parameter λ, every key pair (pk, sk) and set of storage containers $\widehat{\mathfrak{S}}$ with respective identifiers $\widehat{\mathfrak{S}_{\mathrm{id}}}$ and metadata $\widehat{\gamma}$ generated by* CSPoRSetup*, for all sets of files $\widehat{F}$ (containing files $F^{(k)} \in \{0,1\}^*$), and for all $(\widehat{\mathcal{F}}, \widehat{\tau}, \widehat{\mathfrak{S}}, \widehat{\gamma})$ generated by* CSPoRStore*, it holds that*

$$\Pr\left[\left(\mathsf{CSPoRVerify}(pk, sk, \widehat{\tau}', \widehat{\mathfrak{S}_{\mathrm{id}}}) \rightleftharpoons \mathsf{CSPoRProve}(pk, \widehat{\mathcal{F}}', \widehat{\tau}', \widehat{\mathfrak{S}})\right) \nrightarrow \texttt{accept}\right] = \mathrm{negl}(\lambda).$$

In Figure 7.1, we illustrate the execution of a CSPoR scheme between a client and a server as well as represent a model of a storage container $\mathfrak{S}^{(c)}$. A client C is able to access the storage container held by a CSP S via a secure channel. The storage container can contain an arbitrary set of files $\widehat{\mathcal{F}}$ which were uploaded by a client during a CSPoRStore procedure, and a CSP may hold an arbitrary amount Γ of storage contain-

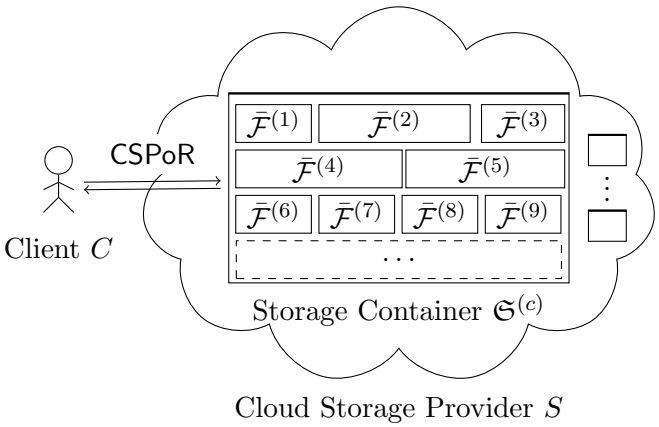

Figure 7.1: Model of a CSPoR scheme with a detailed representation of a generic storage container

ers. In more detail, $\bar{\mathcal{F}}^{(k)} := (\mathcal{F}^{(k)}||\tau^{(k)})$ for $k \in [K_{\mathfrak{S}_{\text{id}}^{(c)}}]$ denotes a processed file with respective file tag stored in a storage container $\mathfrak{S}^{(c)}$ located at a cloud storage provider. Note that in practice the processed files and file tags may be stored separately.

7.3 Security Model

In this section we discuss security within our CSPoR model. We do not explicitly consider *confidentiality* of a file F, but assume that a client may encrypt the files before the initiation of the CSPoR protocol. The adversary aims to convince a client with overwhelming probability that the outsourced files are still fully intact and retrievable. In the following we define the security notion of *extractability* for our CSPoR scheme following existing security notions for PoR models as introduced in Section 2.8.

Extractability

Intuitively, we wish to formalise and say that a CSPoR protocol is secure if any cheating prover that convinces the verification algorithm to accept is indeed storing all files in $\widehat{\mathcal{F}}$ with a sufficient level of probability. In other words, we wish to guarantee that, whenever a malicious prover is in a position of successfully passing a CSPoRP instance, it must *know* the entire file content of all files. We now follow the PoR security formalisation from Section 2.8 and extend it slightly to adjust the notion for our CSPoR system. As in Section 2.8, we require an extractor algorithm $\mathcal{E}(pk, sk, \widehat{\tau}, \mathcal{P}')$ taking as input the generated key pair, the set of file tag $\widehat{\tau}$ as well as a description of the machine implementing the prover's role in the CSPoR protocol. The extractor's output is the set of files $\widehat{F}$. As noted above, the extractor is given (non black-box) access to $\mathcal{P}'$ and in particular can rewind it. Furthermore, we require that the algorithm is efficient, i.e. $\mathcal{E}$'s running time needs to be polynomial in the security parameter.

Consider the following *extractability* game $\mathbf{Exp}^{\text{EXTRACT}}_{\mathcal{A},\epsilon}\left[\mathcal{CSPOR}, 1^\lambda\right]$ between a malicious adversary $\mathcal{A}$, an extractor $\mathcal{E}$, and a challenger $\mathcal{C}$.

1. The challenger initialises the system by running CSPoRSetup to generate the public and private key pairs. The public keys are provided to $\mathcal{A}$. It also generates a set of storage containers $\widehat{\mathfrak{S}}$, a set of respective storage container identifiers $\widehat{\mathfrak{S}_{\text{id}}}$ as well as metadata for each storage container $\widehat{\gamma}$ which are all given to the adversary $\mathcal{A}$.

2. The adversary $\mathcal{A}$ is now able to interact with the challenger that takes the role of an honest client. $\mathcal{A}$ is allowed to request executions to a CSPoRStore oracle by providing, for each query, a set of files $\widehat{F} = \{\widehat{F}_{\mathfrak{S}^{(c)}_{\text{id}}} \mid c \in [\Gamma]\}$ and the storage container identifiers $\widehat{\mathfrak{S}_{\text{id}}}$ basically indicate in which container the files should be stored.

3. Likewise, $\mathcal{A}$ can request executions of the CSPoRP procedure for any set of files on which it previously made a CSPoRStore query by specifying the corresponding tags $\widehat{\tau}$. In the procedures, the challenger will play the role of the honest verifier $\mathcal{V}$ and the adversary the role of the corrupted prover, i.e. $\mathcal{V}(pk, sk, \widehat{\tau}, \widehat{\mathfrak{S}_{\text{id}}}) \rightleftharpoons \mathcal{A}$. In the end of the execution the adversary is provided with the output of the verifier. Furthermore, the CSPoRStore oracle queries and the executions of CSPoRP can be interleaved arbitrarily.

4. Finally, the adversary outputs a set of challenge tags $\widehat{\tau}'$ returned from some CSPoRStore query and the description of a prover $\mathcal{P}'$.

5. Run the extractor algorithm $\widehat{F}' \leftarrow \mathcal{E}(pk, sk, \widehat{\tau}', \mathcal{P}')$ inputting the challenge tags $\widehat{\tau}'$ and description $\mathcal{P}'$ where $\mathcal{E}$ gets black-box rewinding access to $\mathcal{P}'$, and attempts to extract the file content of all files as $\widehat{F}'$.

6. If $\Pr\left[\left(\mathcal{V}(pk, sk, \widehat{\tau}, \widehat{\mathfrak{S}_{\text{id}}}) \rightleftharpoons \mathcal{P}'\right) \rightarrow \texttt{accept}\right] \geq \epsilon$ and $\widehat{F}' \neq \widehat{F}$ then output 1, else 0.

Note that we say a malicious prover $\mathcal{P}'$ is ϵ-*admissible* if the probability that it convincingly answers verification challenges is at least ϵ, i.e. if $\Pr\left[\left(\mathcal{V}(pk, sk, \widehat{\tau}, \widehat{\mathfrak{S}_{\text{id}}}) \rightleftharpoons \mathcal{P}'\right) \rightarrow \texttt{accept}\right] \geq \epsilon$. Here the probability is over the coins of the verifier and prover.

Definition 7.4. *We say that a $\mathcal{CSPOR}$ scheme is* ϵ-extractable *(or* secure*) if there exists an efficient extraction algorithm $\mathcal{E}$ such that, for all PPT adversaries $\mathcal{A}$ it holds that* $\Pr\left[\mathbf{Exp}^{\text{EXTRACT}}_{\mathcal{A},\epsilon}\left[\mathcal{CSPOR}, 1^\lambda\right] \rightarrow 1\right]$ *is negligible in the security parameter.*

7.4 Construction

In this section we provide a concrete instantiation of CSPoR. We start by outlining our main building blocks and then provide details about our instantiation. We choose to build our construction upon the private-key PoR scheme of Shacham and Waters [126] mainly due to its ability to handle an unbounded number of verification queries. In case a better communication complexity is required, one may build upon the scheme presented in [40]. However, in our particular case we believe that an unbounded number of verification queries is more beneficial since processing the files already involves using the client's valuable resources and it is not a task the client wishes to repeat regularly. Our CSPoR instantiation overcomes the identified limitations as discussed in Section 7.2, when employing existing schemes straightforwardly to prove retrievability for multiple different-sized files simultaneously.

Building Blocks

Unless otherwise specified all operations are performed in the finite field $\mathbb{F} = \mathbb{Z}_p$ where p is a λ-bit prime with λ being the security parameter. As we instantiate a private-key CSPoR system it suffices to make use of a symmetric encryption scheme and we set the public key $pk = \perp$. We make use of a pseudo-random function $g\colon \{0,1\}^* \times \{0,1\}^{\phi_{\text{prf}}} \to \mathbb{F}$, where ϕ_{prf} is the key length of the PRF[2], and a MAC scheme. Furthermore, we make use of storage containers $\mathfrak{S}^{(c)}$, $c \in [\Gamma]$, as introduced in Section 7.2.1.

Specification of the CSPoRSetup Procedure

In the CSPoRSetup procedure the client derives its keys. First the client chooses a random symmetric key $\kappa_{\text{enc}} \xleftarrow{\$} \mathcal{K}_{\text{enc}}$ and a random MAC key $\kappa_{\text{mac}} \xleftarrow{\$} \mathcal{K}_{\text{mac}}$. The client keeps the secret key $sk = (\kappa_{\text{enc}}, \kappa_{\text{mac}})$ and requests creating a cloud storage $\widehat{\mathfrak{S}}$, i.e. a set of storage containers located at the server S.

Specification of the CSPoRStore Procedure

The CSPoRStore procedure is initiated by a client holding files $\widehat{F} = \{\widehat{F}_{\mathfrak{S}_{\text{id}}^{(c)}} \mid c \in [\Gamma]\}$ that she wishes to store in $\widehat{\mathfrak{S}}$. The following steps are carried out for each file $F^{(k)}$, $k \in [K_{\mathfrak{S}_{\text{id}}^{(c)}}]_{c\in[\Gamma]}$, in parallel:

- First we apply an information dispersal algorithm (i.e. an erasure code) with code rate ρ over the file $F^{(k)}$. The resulting processed file is denoted by $\mathcal{F}^{(k)}$;
- Next we process a processed file $\mathcal{F}^{(k)}$ into $\tilde{n} \in \mathbb{N}$ blocks being s symbols long. That is $\mathcal{F}^{(k)} = \{f_{ij}^{(k)}\}$, where $1 \leq i \leq \tilde{n}$, $1 \leq j \leq s$ and $k \in [K_{\mathfrak{S}_{\text{id}}^{(c)}}]_{c\in[\Gamma]}$. Note that s is *constant* for all files while the number of blocks $\tilde{n}$ varies depending on

[2]Note that we use the shorthand $g_{\kappa_{\text{prf}}^{(k)}}(i) := g(\kappa_{\text{prf}}^{(k)}, i)$.

the respective underlying original file size. Each symbol $f_{ij}^{(k)}$ is encoded in terms of elements in $\mathbb{F}$;

- We sample uniformly at random a PRF key $\kappa_{\text{prf}}^{(k)} \xleftarrow{\$} \{0,1\}^{\phi_{\text{prf}}}$. We also sample s random elements from the finite field $\mathbb{F}$ which are kept private by the client, that is $\alpha_1^{(k)}, \ldots, \alpha_s^{(k)} \xleftarrow{\$} \mathbb{F}$;

- Next we compute for each file block of an outsourced file $i \in [\tilde{n}]$ an authentication tag σ_i as follows

$$\sigma_i^{(k)} \leftarrow g_{\kappa_{\text{prf}}^{(k)}}(i) + \sum_{j=1}^{s} \alpha_j^{(k)} f_{ij}^{(k)} \qquad \in \mathbb{F};$$

- Now we wish to compute a file tag $\tau^{(k)} = \tau_0^{(k)} \| \mathsf{MAC}_{\kappa_{\text{mac}}}(\tau_0^{(k)})$ where $\tau_0^{(k)}$ has the form $\tilde{n} \| \mathsf{Encrypt}_{\kappa_{\text{enc}}} \left(\kappa_{\text{prf}}^{(k)} \| \alpha_1^{(k)} \| \ldots \| \alpha_s^{(k)} \right)$. Note that the cloud service provider will also receive each file tag $\tau^{(k)}$. However, this does not enable the server to make use of the random elements as they are encrypted and it is not in possession of the respective secret key but it only learns the size of the outsourced file.

Finally, the client combines all file tags into the set of tags $\widehat{\tau}$ and all processed files are aggregated as the set $\widehat{\mathcal{F}}$ and will be uploaded to the respective storage container in $\widehat{\mathfrak{S}}$ located at the cloud service provider. Each processed file itself is of the form $\mathcal{F}^{(k)} := \left(\{f_{ij}^{(k)}\}, \{\sigma_i^{(k)}\} \right)_{1 \leq i \leq \tilde{n}, 1 \leq j \leq s}$.

Specification of the CSPoRP Procedure

The CSPoRP procedure consists of executing several audit steps to obtain an assurance about the retrievability of the files. In Section 7.5, we discuss various strategies a client can follow in order to execute the audit steps. In the following we describe the technical details of a single audit step (providing a reply δ_i) which will be repeated A times in order to obtain a final answer δ from the the CSPoRP procedure capturing our desired assurance:

- The client first verifies the MAC on each $\tau^{(k)}$ within $\widehat{\tau}$. If the MAC is invalid the client aborts the protocol and outputs 0. Otherwise, she parses all $\tau^{(k)}$ from $\widehat{\tau}$ and uses κ_{enc} in order to recover $\tilde{n}^{(k)}$, $\kappa_{\text{prf}}^{(k)}$ and $\alpha_1^{(k)}, \ldots, \alpha_s^{(k)}$ for all $k \in [K_{\mathfrak{S}_{\text{id}}^{(c)}}]_{c \in [\Gamma]}$;

- Next the client C generates for each file a challenge by picking a random subset $I^{(k)} \subseteq_{\$} [\tilde{n}^{(k)}]$ of size $\ell^{(k)}$;

- Next C chooses for each $i \in I^{(k)}$ a random element from the finite field $\nu_i^{(k)} \xleftarrow{\$} \mathbb{F}$ and aggregates this sampling per file to a set $Q^{(k)} = \{(i, \nu_i^{(k)})_{i \in I^{(k)}}\}$ of size $\ell^{(k)}$. All sets $Q^{(k)}$ per storage container can be aggregated as $\mathcal{Q}_{\mathfrak{S}_{\text{id}}^{(c)}} := \{Q^{(k)} \mid k \in$

$[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}], c \in [\Gamma]\}$. Finally, C sends as her challenge the combined set over all containers $\widehat{\mathcal{Q}} := \{\mathcal{Q}_{\mathfrak{S}_{\mathrm{id}}^{(c)}} \mid c \in [\Gamma]\}$ to the server.

The cloud service provider now parses all files from $\widehat{\mathcal{F}}$ as $\{f_{ij}^{(k)}\}$ and $\{\sigma_i^{(k)}\}$, and the corresponding challenges $Q^{(k)}$ from $\widehat{\mathcal{Q}}$. Then for $1 \leq j \leq s$, the provider computes

$$\mu_j^{(k)} \leftarrow \sum_{(i,\nu_i^{(k)}) \in Q^{(k)}} \nu_i^{(k)} f_{ij}^{(k)} \qquad \text{and} \qquad \sigma^{(k)} \leftarrow \sum_{(i,\nu_i^{(k)}) \in Q^{(k)}} \nu_i^{(k)} \sigma_i^{(k)}.$$

This execution is repeated for all files in all storage containers contained in $\widehat{\mathcal{Q}}$. Then S accumulates all responses and authentication tags as

$$\left(\widetilde{\mu}_1 := \sum_{c \in [\Gamma]} \sum_{k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \mu_1^{(k)}, \ \ldots, \ \widetilde{\mu}_s := \sum_{c \in [\Gamma]} \sum_{k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \mu_s^{(k)}, \quad \widetilde{\sigma} := \sum_{c \in [\Gamma]} \sum_{k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \sigma^{(k)} \right).$$

Finally, the client parses the provider's accumulated response and checks

$$\widetilde{\sigma} \stackrel{?}{=} \sum_{c \in [\Gamma]} \sum_{k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \left(\sum_{(i,\nu_i^{(k)}) \in Q^{(k)}} \nu_i^{(k)} g_{\kappa_{\mathrm{prf}}^{(k)}}(i) + \sum_{j=1}^{s} \alpha_j^{(k)} \widetilde{\mu}_j \right). \tag{7.1}$$

If this equality check is successful, the verifier outputs $\delta_i = \texttt{accept}$, and otherwise she outputs `reject`.

If a check fails ($\delta_i = \texttt{reject}$), a client assumes that the cloud service provider is malicious and takes actions in either immediately downloading all files or follows another strategy called *reduced CSPoR* as introduced in Section 7.5. However, we want to emphasise that downloading is very impractical as a client loses the advantage of outsourcing the files in the first place. Furthermore, we want to stress that if CSPoRP is successful then we have obtained a retrievability assurance that *all* files in $\widehat{\mathfrak{S}}$ are retained by the cloud service provider.

Correctness of the Instantiation

Now we present that our above scheme is correct. Let the PRF key be $\kappa_{\mathrm{prf}}^{(k)}$ and $\alpha_1^{(k)}, \ldots, \alpha_s^{(k)} \stackrel{\$}{\leftarrow} \mathbb{F}$ be the secret coefficients for all $k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]_{c \in [\Gamma]}$. Let the file symbols be denoted by $\{f_{ij}^{(k)}\}$, and the block authenticators are expressed as $g_{\kappa_{\mathrm{prf}}^{(k)}}(i) + \sum_{j=1}^{s} \alpha_j^{(k)} f_{ij}^{(k)}$. For a prover that responds honestly to queries from $\widehat{\mathcal{Q}}$ such that

$\widetilde{\mu}_j = \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \mu_j^{(k)}$ and $\widetilde{\sigma} = \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \sigma^{(k)}$ then we have

$$\begin{aligned}
\widetilde{\sigma} &= \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \sigma^{(k)} = \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \left(\sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} \sigma_i^{(k)} \right) \\
&= \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \left(\sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} \left(g_{\kappa_{\mathrm{prf}}^{(k)}}(i) + \sum_{j=1}^{s} \alpha_j^{(k)} f_{ij}^{(k)} \right) \right) \\
&= \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \left(\sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} g_{\kappa_{\mathrm{prf}}^{(k)}}(i) + \sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} \sum_{j=1}^{s} \alpha_j^{(k)} f_{ij}^{(k)} \right) \\
&= \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \left(\sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} g_{\kappa_{\mathrm{prf}}^{(k)}}(i) + \sum_{j=1}^{s} \alpha_j^{(k)} \sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} f_{ij}^{(k)} \right) \\
&= \sum_{c\in[\Gamma]} \sum_{k\in[K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]} \left(\sum_{(i,\nu_i^{(k)})\in Q^{(k)}} \nu_i^{(k)} g_{\kappa_{\mathrm{prf}}^{(k)}}(i) + \sum_{j=1}^{s} \alpha_j^{(k)} \mu_j^{(k)} \right),
\end{aligned}$$

which shows that verification is satisfied.

Now we formulate our main theorem capturing the security of our CSPoR scheme.

Theorem 7.5. *If the MAC scheme is unforgeable, the symmetric encryption scheme is semantically secure, and the PRF is secure, then no adversary (except with negligible probability) against the extractability game* Extract *of our CSPoR scheme ever causes the verifier to accept a cloud storage proofs of retrievability protocol instance, except by responding with correctly computed responses* $\widetilde{\mu}_j$ $(1 \leq j \leq s)$ *and authentication tag* $\widetilde{\sigma}$.

Proof. The security of our proposed scheme follows immediately from the security of the private-key SW scheme [126] as we base our construction on their scheme and thus inherit the security property. The only main difference in the proof is that we have to ensure the correctness of the aggregated set of responses $\{\widetilde{\mu}_j\}$.

First, we need to argue that the verification algorithm will reject answers except if the answers $\{\widetilde{\mu}_j\}$ were computed correctly by the prover. This can be shown via a sequence of games for which we argue that the adversary's distinguishing advantage between two consecutive games is negligible. The analysis follows via the same game hops as in [126] while adapting the notion of having aggregated answers $\{\widetilde{\mu}_j\}$ from all outsourced files.

Secondly, we need to argue that the extraction procedure can efficiently reconstruct a ρ fraction of file blocks when interacting with a prover that provides correctly computed

responses for a non-negligible fraction of the query space. The same arguments as in [126] apply to our scheme. Here we only need to slightly change the proof details when showing that a well-behaved ϵ-admissible cheating prover $\mathcal{P}'$ as the output of the extractability game (cf. Section 7.3) can be turned into an ϵ-polite adversary $\mathcal{B}$ (implemented as a probabilistic polynomial-time Turing machine). Note that we say an adversary is ϵ-polite if it responds with probability ϵ to given queries Q covering an ϵ fraction of the query- and randomness-tape space. $\mathcal{P}'$ can be used to construct the adversary $\mathcal{B}$. For a query Q, $\mathcal{P}'$ interacts with the verifier according to $\mathcal{V}(pk, sk, \widehat{\tau}, \widehat{\mathfrak{S}_{\mathrm{id}}}) \rightleftharpoons \mathcal{P}'$. If the interaction is successful the responses $(\widetilde{\mu}_1, \ldots, \widetilde{\mu}_s)$ will be written to its output tape while a wrong interaction leads to writing $\perp$ on the tape. Note that after k interactions we can represent all responses as a $(k \times s)$ matrix. Each time the adversary $\mathcal{B}$ runs the prover $\mathcal{P}'$ it is able to effectively rewind the prover. Since $\mathcal{P}'$ is well-behaved a successful interaction computes valid $(\widetilde{\mu}_1, \ldots, \widetilde{\mu}_s)$ and given that $\mathcal{P}'$ is ϵ-admissible we know that an ϵ fraction of the answers are computed correctly. Having this we can further follow SW and can represent the extractor's knowledge by a row in the matrix for each audit step, i.e. as $(\widetilde{\mu}_1^{(t)}, \ldots, \widetilde{\mu}_s^{(t)})$ where $t \in [A]$, and thus have sampled enough information to permit successful extraction.

Lastly, we argue that following the ECC reconstruction property it suffices to positively check any n blocks of a processed file consisting of $\tilde{n}$ blocks to recover the original file $F^{(k)}$ with all but negligible probability. This is trivially fulfilled by employing Reed-Solomon codes of rate ρ, since any ρ fraction of encoded file blocks suffice in order to reconstruct the underlying file. Note, however, that this does not protect a client from revealing correlations between the plaintext blocks and redundant blocks through the access pattern. This can be avoided if a client encrypts and permutes the parity part of the file following Ateniese et al. [14]. □

Armknecht et al. [11] introduced the notion of an *Outsourced PoR*. In their model, an external entity called an *auditor* joins the system and offers to audit the server on behalf of a client whereas a client is able to efficiently check whether the auditor performs a PoRP execution correctly. Their proposed scheme is also based upon the private-key PoR scheme of [126]. Hence, by replacing the single-file scheme with a CSPoR scheme leads to an outsourced CSPoR scheme.

7.5 Practicability of CSPoR

In this section, we discuss different audit strategies a client could use throughout a CSPoRP procedure.

7.5.1 Strategies for CSPoRP

As discussed throughout this chapter, we wish to specify concrete strategies a client may use within a CSPoR system. Recall that a "standard" PoR provides the client with a probabilistic assurance whether the data is retained or not. The usage of erasure-correcting codes in the realm of PoR was introduced to boost the probability to detect a malicious server as well as a mechanism to recover the original file in case some file blocks are deleted. However, in practice a probabilistic assurance about the retained data is not sufficient as it is not clear whether enough data is available to reconstruct the file and thus we provide strategies that enable the client to obtain such an assurance. We receive this assurance in case the client is able to determine that a sufficient part of the file is retained by the server. Recall that a processed file $\mathcal{F}^{(k)}$ consists of the original file plus some redundancy specified via the ECC, i.e. $\mathcal{F}^{(k)}$ consists of $\tilde{n} = n/\rho$. Due to the ECC it suffices to sample any n from $\tilde{n}$ blocks to recover the file using the decoding algorithm. Thus, this property assures that a file is available if a client is able to check successfully any n from $\tilde{n}$ valid file blocks. Note that this holds as long as the adversary does not delete more than $(1-\rho)\cdot 100$ percent of $\tilde{n}$ outsourced data blocks. However, we also wish to stress that in case a client detects that data has been deleted repeatedly she would lose all trust in the cloud service provider and would decide to change to a new provider. Since this would incur a financial penalty, we assume that this is usually a rare event.

7.5.2 Audit Strategies for CSPoRP

In this section, we discuss different strategies that enable a client to obtain a retrievability assurance about the original files through CSPoRP.

7.5.2.1 Workload Partition

In order to receive a retrievability-assurance, a client needs to execute several audit steps (within CSPoRP) in order to obtain a valid proof. Therefore, we believe that it would be beneficial for a client to be able to determine *a priori* the average number of required audits to prove retrievability, i.e. we wish to derive a formula computing the number of audits A based on a fixed challenge size ℓ. We show that this task is related to a new variant of the well-known *Coupon Collector's Problem* [124] to which we provide a detailed solution and finally argue that this approach is a possible solution for the above task. Furthermore, we also provide a solution for the reversed task, i.e. we derive a formula that computes the appropriate challenge size given a pre-defined number of audits A. We also show that a statistical hypothesis test may be used to argue whether the received responses suffice to obtain the required assurance.

Recall that our scheme checks ℓ different file blocks per audit and generally a CSPoRP

execution depends on n, $\tilde{n}$, ρ, ℓ, and A. Since ρ is given as a system parameter, we can derive the size of the processed file using the code rate combined with the size of the original file, and thus it remains to follow the above goal and compute the values for ℓ and A. Recall, for each audit step the values for the challenge of size ℓ are chosen randomly without replacement from the set of processed file blocks $[\tilde{n}]$.[3] We note that both values A and ℓ depend on each other. For example, if a client increases the challenge size per audit then the required number of audits decreases as a client checks more file blocks per request. Thus, this provides a client with a flexible trade-off between communication and computation depending on her current resources.

Computing the Number of Audits

In this part, we aim to derive a formula that computes the (average) number of required audits based on a fixed challenge size, the block identifiers for the original and processed file as well as the erasure-correction code rate. Obtaining such a formula enables a client to gain a priori knowledge about the necessary communication cost in order to perform CSPoRP such that it provides an assurance about the data retrievability. In other words, a client wishes to perform A audits of size ℓ in order to determine if any n of $\tilde{n}$ blocks are intact to be convinced that the original file is retrievable.

Computing the average required number of audits A can be expressed as a new variant of the *Coupon Collector's Problem* (CCP) [124].

Coupon Collector's Problem

The classical CCP describes a solution to compute the required (average) waiting time to obtain all coupons of a collection while drawing one coupon at a time whereas all coupons appear equally likely. In our context, the waiting time corresponds to the required number of audits, and the challenge size corresponds to the number of drawn coupons at a time while the collection size corresponds to the total number of blocks of a processed file, that is $\tilde{n}$. In order to derive our required (average) number of audits from the CCP, we need to reformulate the above problem as follows. We wish to compute the average waiting time A to obtain any partial collection of n coupons from the full collection of size $\tilde{n}$ while drawing ℓ coupons at a time with the restriction that $\ell \leq n$. Note that all coupons are drawn from the full collection, i.e. from the set $[\tilde{n}]$, and all coupons appear equally likely. In other words, we need to evaluate the expected number of groups of coupons to obtain at least n different coupons. Ferrante and Frigo [65] present a solution of yet another variant of the above problem where they compute the waiting time to obtain *all* coupons of the collection while drawing ℓ coupons at a time. We follow their approach using kth order statistics to derive a

[3] This is mainly done to prevent an adversarial cloud service provider deleting queried blocks since the probability of choosing the same block again is equally likely to sampling a new block.

solution for our problem. A good overview about the Coupon Collector's Problem can be found in [66].

We begin by considering that coupons are drawn in groups of constant size ℓ, where $1 \leq \ell \leq n$, with the types of items in any group of coupons being independent random variables.[4] In our context, we require that each group does not contain more than one coupon of any type. Thus, the total number of groups will be $\binom{\tilde{n}}{\ell}$ and each group G can be identified with a vector $(g_1, \ldots, g_\ell) \in \{1, \ldots, \tilde{n}\}$ with $g_i < g_{i+1}$ for $i \in \{1, \ldots, \ell - 1\}$.

Definition 7.6. *We denote by q_k the probability of drawing (at any given time) the kth group of coupons in lexicographical order, where $i \in \{1, \ldots, \binom{\tilde{n}}{\ell}\}$.*

We consider the case of uniform probabilities and thus we have that the probability corresponds for any group k to $q_k = \frac{1}{\binom{\tilde{n}}{\ell}}$. We set V_i to be a random variable which equals the number of groups to be drawn in order to obtain the first coupon of type i. Following Ferrante and Frigo [65], these random variables follow a geometric law with parameter $1 - \frac{\binom{\tilde{n}-1}{\ell}}{\binom{\tilde{n}}{\ell}}$. The random variables $\min(V_i, V_j)$ follow a geometric law with parameter $1 - \frac{\binom{\tilde{n}-2}{\ell}}{\binom{\tilde{n}}{\ell}}$ and this continues the same way up to the random variables $\min(V_{i_1}, \ldots, V_{i_{\tilde{n}-\ell}})$ having a geometric law with parameter $1 - \frac{1}{\binom{\tilde{n}}{\ell}}$. Note that the minimum of more random variables $\min(V_{i_1}, \ldots, V_{i_k})$ for $k > \tilde{n} - \ell + 1$ equals to the constant random variable 1. Applying the Maximum-Minimum principle provides us with the expected number of groups of coupons to complete the collection as

$$\begin{aligned}\sum_{1 \leq i_1 < i_2 < \ldots < i_\phi \leq \tilde{n}} \mathbb{E}[\min(V_{i_1}, V_{i_2}, \ldots, V_{i_\phi})] &= \binom{\tilde{n}}{\phi} \frac{1}{1 - \frac{\binom{\tilde{n}-\phi}{\ell}}{\binom{\tilde{n}}{\ell}}} \\ &= \binom{\tilde{n}}{\phi} \frac{\binom{\tilde{n}}{\ell}}{\binom{\tilde{n}}{\ell} - \binom{\tilde{n}-\phi}{\ell}}. \end{aligned} \tag{7.2}$$

We denote by $V_{(\tilde{n})}$ the $\tilde{n}$th order statistic and following the usual convention we have $V_{(\tilde{n})} = \max(V_1, \ldots V_{\tilde{n}})$ and $V_{(1)} = \min(V_{i_1}, \ldots, V_{i_{\tilde{n}}}) = \min(V_1, \ldots, V_{\tilde{n}})$. Then we can derive the 2nd order statistic as

$$V_{(2)} = \sum_{1 \leq i_1 < \ldots < i_{\tilde{n}-1} \leq \tilde{n}} \min(V_{i_1}, \ldots, V_{i_{\tilde{n}-1}}) - (\tilde{n} - 1) \min(V_1, \ldots, V_{\tilde{n}}),$$

[4] Similarly to above, those groups of coupons are sampled randomly without replacement.

and the 3rd order statistic can be obtained as

$$\begin{aligned}
V_{(3)} &= \sum_{1\le i_1<\ldots<i_{\tilde{n}-2}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-2}}) - (\tilde{n}-2)V_{(2)} - \binom{\tilde{n}-1}{\tilde{n}-3}V_{(1)} \\
&= \sum_{1\le i_1<\ldots<i_{\tilde{n}-2}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-2}}) \\
&\quad - (\tilde{n}-2)\sum_{1\le i_1<\ldots<i_{\tilde{n}-1}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-1}}) \\
&\quad + (\tilde{n}-2)(\tilde{n}-1)\min(V_1,\ldots,V_{\tilde{n}}) \\
&\quad - \frac{(\tilde{n}-1)(\tilde{n}-2)}{2}\min(V_{i_1},\ldots,V_{i_{\tilde{n}}}) \\
&= \sum_{1\le i_1<\ldots<i_{\tilde{n}-2}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-2}}) \\
&\quad - (\tilde{n}-2)\sum_{1\le i_1<\ldots<i_{\tilde{n}-1}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-1}}) \\
&\quad + \frac{(\tilde{n}-1)(\tilde{n}-2)}{2}\min(V_1,\ldots,V_{\tilde{n}}).
\end{aligned}$$

The general formula can be derived for any $1 \le n \le \tilde{n}$ as

$$\begin{aligned}
V_{(n)} &= \sum_{1\le i_1<\ldots<i_{\tilde{n}-(n-1)}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-(n-1)}}) \\
&\quad - (\tilde{n}-(n-1))\sum_{1\le i_1<\ldots<i_{\tilde{n}-(n-2)}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-(n-2)}}) \\
&\quad + \frac{(\tilde{n}-(n-1))(\tilde{n}-(n-2))}{2!}\sum_{1\le i_1<\ldots<i_{\tilde{n}-(n-3)}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-(n-3)}}) + \ldots \\
&\quad + (-1)^{n+1}\frac{(\tilde{n}-(n-1))(\tilde{n}-(n-2))\ldots(\tilde{n}-(n-(n-1)))}{(n-1)!}\min(V_1,\ldots,V_{\tilde{n}}) \\
&= \sum_{j=1}^{n}\left[(-1)^{j+1}\binom{\tilde{n}-n+j-1}{j-1}\sum_{1\le i_1<\ldots<i_{\tilde{n}-n+j}\le\tilde{n}} \min(V_{i_1},\ldots,V_{i_{\tilde{n}-n+j}})\right].
\end{aligned}$$

We can finally compute the expectation as follows

$$\begin{aligned}
\mathbb{E}\left[X_n^\ell(\tilde{n})\right] &= \mathbb{E}\left[V_{(n)}\right] \\
&= \mathbb{E}\left[\sum_{j=1}^{n}\left[(-1)^{j+1}\binom{\tilde{n}-n+j-1}{j-1}\sum_{1\le i_1<\ldots<i_{\tilde{n}-n+j}\le\tilde{n}}\min(V_{i_1},\ldots,V_{i_{\tilde{n}-n+j}})\right]\right] \\
&= \sum_{j=1}^{n}\left[(-1)^{j+1}\binom{\tilde{n}-n+j-1}{j-1}\sum_{1\le i_1<\ldots<i_{\tilde{n}-n+j}\le\tilde{n}}\mathbb{E}\left[\min(V_{i_1},\ldots,V_{i_{\tilde{n}-n+j}})\right]\right] \\
&= \sum_{j=1}^{n}\left[(-1)^{j+1}\binom{\tilde{n}-n+j-1}{j-1}\binom{\tilde{n}}{\tilde{n}-n+j}\frac{\binom{\tilde{n}}{\ell}}{\binom{\tilde{n}}{\ell}-\binom{\tilde{n}-(\tilde{n}-n+j)}{\ell}}\right] \\
&= \binom{\tilde{n}}{\ell}\sum_{j=1}^{n}\left[(-1)^{j+1}\binom{\tilde{n}-n+j-1}{j-1}\binom{\tilde{n}}{\tilde{n}-n+j}\frac{1}{\binom{\tilde{n}}{\ell}-\binom{n-j}{\ell}}\right]. \qquad (7.3)
\end{aligned}$$

This expected value corresponds to the number of audits we initially aimed to compute, i.e. $A = \mathbb{E}\left[X_n^\ell(\tilde{n})\right]$. However, the precise computation of this value involves a huge amount of computation and thus renders a quick computation for a computationally weak client to be infeasible. Thus, we wish to obtain an approximation for A. Starting from a classical version of the CCP, we argue that each draw of a new set of coupons (consisting of elements in the size of the challenge ℓ) increases the overall amount of drawn sets and is also likely to provide us with new coupons we have not seen before. For example, in the first draw all coupons are new and have not been seen before. However, in the next draws this changes since the drawn sets of coupons may contain coupons that we have already seen before. We stop drawing new groups of coupons after a total of n different coupons have been seen. The (constant) size of the drawn sets of coupons plays a crucial role and thus results in a modulo operation in the formula. Note that this approach does not take into account all possible overlaps of the groups but results in an expected value which is very easy to compute and may suffice as a good approximation. Since the expected value corresponds to the number of block identifiers which have been sampled, we divide the value by the challenge size ℓ to obtain the required number of groups and thus the number of audits. Hence, the derived approximation for A can be expressed as

$$A \approx A' := \frac{1}{\ell}\sum_{i=0}^{n-1}\frac{\tilde{n}-(i \bmod \ell)}{\tilde{n}-i}. \qquad (7.4)$$

For practical usage, one may use $\lceil A' \rceil$ for the number of audits.

Before providing an example to obtain a better understanding of the concept of audits, we wish to emphasise that in order to form a PoR request the client needs to sample the block identifiers uniformly at random, as opposed to only asking blocks which have not

been queried before. This prevents the adversary from deleting blocks after they have been queried, and the client ensures that all blocks are still equally likely to be checked to prevent the adversary from cheating. We illustrate the concept of audits in form of an example in Table 7.1. The number of rows of a CSPoRP represent the number of performed audits A while the number of bullets per row represents the challenge size ℓ.

CSPoRP No.	Audit No.	Sampled Block Identifiers of $\mathcal{F}$												Overall Distinct Block Ids Requested B
		1	2	3	4	5	6	7	8	9	10	11	12	
1	1	•	•								•			3
	2	·	•					•			•			4
	3	·	·		•			·	•	•	·			$7 \geq 6 = n$
2	1		•					•					•	3
	2	•	·				•	·			•		·	$6 \geq 6 = n$
	3	•	·		•		·	·		•	·		·	$8 \geq 6 = n$
3	1	•		•				•						3
	2	•		·				•					•	4
	3	•	•	·		•		·					·	$6 \geq 6 = n$
4	1		•	•				•						3
	2		•	·				•					•	4
	3		•	·		•		·					•	$5 < 6 = n$

Table 7.1: An example of four randomised CSPoRP procedures each consisting of three audits using the same parameters

For the above example, we choose the original file F to consist of $n = 6$ blocks and the ECC code rate to be $\rho = 1/2$. Thus, the processed file $\mathcal{F}$ consists of $\tilde{n} = 12$ blocks and we choose the challenge size as $\ell = 3$. Using equation (7.4), we compute the average required number of audits in order to obtain a retrievability assurance about the original file. We obtain $A' = 2.3789$ which becomes for a practical use $\lceil A' \rceil = 3$. As we chose $\ell = 3$, we sample three different block identifiers per audit uniformly at random from the set of processed block identifier. Those block identifier are illustrated as a bullet • while in the following audit steps we use a small dot · to illustrate that this block identifier has been already sampled. As soon as we sample enough available blocks, i.e. $B \geq n$, CSPoRVerify should return `accept` indicating that enough block information are obtained and F can be reconstructed, cf. Section 2.8. In the above example, we present four randomised CSPoRP procedures for the same file showing that the randomised sampling has an impact on the number of sampled block identifier B. The first three CSPoRP procedures sample overall enough block identifier to ensure that the file is retained by the cloud service provider. However, the last CSPoRP procedure does not sample enough block identifier and CSPoRVerify returns `reject`. Thus, either the file is corrupted and the client may not be able to reconstruct the file even using the ECC decoding algorithm or the file is retained by the provider but the

client was "unlucky" with the random sampling as too many same block identifier were checked. Therefore, as a strategy, the client may run a new CSPoRP procedure and increase the number of audits to ensure that it is more likely to query enough different block identifier.

Computing the Challenge Size

In this part, we derive a formula that computes the challenge size ℓ for the query set given a fixed number of audits A, the block identifiers of the original file n and of the processed file $\tilde{n}$.

Let us now derive an approximation for the challenge size ℓ starting from equation (7.4). We can write $\tilde{n} = m\ell + r$ where $r < \ell, m \in \mathbb{N}_0$, and thus it follows

$$\begin{aligned} A \overset{\{1\}}{<} & \frac{1}{\ell}\sum_{i=1}^{n}\frac{\tilde{n}-\overline{\tilde{n}-i}}{i} \overset{\{2\}}{\leq} \frac{1}{\ell}\sum_{i=1}^{n}\frac{\tilde{n}-\overline{\tilde{n}}+\overline{i}}{i} \\ = & \frac{1}{\ell}\sum_{i=1}^{n}\frac{m\ell+r-r+\overline{i}}{i} = \frac{1}{\ell}\sum_{i=1}^{n}\frac{m\ell}{i} + \frac{1}{\ell}\sum_{i=1}^{n}\frac{\overline{i}}{i} \\ = & \underbrace{m\sum_{i=1}^{n}\frac{1}{i}}_{=mH_n} + \underbrace{\frac{1}{\ell}\sum_{i=1}^{n}\frac{\overline{i}}{i}}_{<H_n} < (m+1)H_n, \end{aligned}$$

where H_n denotes the nth harmonic number and the term $\overline{i}$ is an abbreviation for $(i \bmod \ell)$.

Let us remark that the first inequality $\{1\}$ may evaluate to equality in case $\rho = 1$. This means that the processed file corresponds to the original file and thus no redundancy was encoded into the file. Note that the second inequality $\{2\}$ evaluates to equality if either $\ell = n = \tilde{n}$ or $\ell = 1$. Furthermore, in case we restrict ourselves to the case $n = \tilde{n}$, then in the above equation we would have $\overline{\tilde{n}-i} \leq \overline{\tilde{n}} + \overline{i} + 1$. The "+1" term propagates to an additional term $y := \frac{1}{\ell}\sum_{i=1}^{n}\frac{1}{i}$ in the above equation which results later into a slightly different approximation of the upper bound. However, we do not restrict ourselves to this case and can therefore proceed with the above and conclude that

$$mH_n \leq A < (m+1)H_n. \tag{7.5}$$

Rearranging the first inequality of equation (7.5) yields a lower bound for ℓ as follows. Recall that $\tilde{n} = m\ell + r$ and therefore we obtain

$$mH_n \leq A \Leftrightarrow \frac{\tilde{n}}{\ell} - \frac{r}{\ell} \leq \frac{A}{H_n} \Leftrightarrow \frac{\tilde{n}}{\ell} < \frac{A}{H_n} + 1$$

since $\frac{r}{\ell} < 1$. This yields the lower bound to be

$$\frac{\tilde{n}H_n}{A + H_n} < \ell.$$

Next we rearrange the right inequality of (7.5) to obtain an upper bound. We get

$$A < (m+1)H_n \Leftrightarrow \frac{A}{H_n} - 1 < \frac{\tilde{n}}{\ell} - \frac{r}{\ell} \leq \frac{\tilde{n}}{\ell}.$$

This yields the upper bound to be

$$\ell < \frac{\tilde{n}H_n}{A - H_n}.$$

Note that, as mentioned above, in case $n = \tilde{n}$, the additional "+1" term results into a slightly different upper bound with $A - 2H_n$ in the denominator. However, altogether it follows that the bounds for ℓ can be approximated as

$$\frac{\tilde{n}H_n}{A + H_n} < \ell < \frac{\tilde{n}H_n}{A - H_n}.$$

Finally, a good choice in practice would be to choose the arithmetic mean of the bounds, that is

$$\ell = \left\lceil \frac{1}{2} \left(\frac{\tilde{n}H_n}{A + H_n} + \frac{\tilde{n}H_n}{A - H_n} \right) \right\rceil. \tag{7.6}$$

7.5.2.2 Statistical Hypothesis Testing

Another auditing strategy can be achieved by determining whether the success probability of a prover is sufficiently high. This can be done using a statistical hypothesis test. In more detail, we wish to determine whether the average success probability ξ of a prover $\mathcal{P}$ is at least μ where the success probability is defined as $\xi(\mathcal{P}) = \Pr[\mathcal{P}(\widehat{\mathcal{Q}}) \rightarrow \delta_i]$ and $\widehat{\mathcal{Q}}$ represents the combined challenges for all files. Thus, we formulate the null hypothesis as

$$H_0 \colon \xi(\mathcal{P}) < \mu$$

and the alternative hypothesis is formulated as

$$H_1 \colon \xi(\mathcal{P}) \geq \mu.$$

The goal is now to distinguish both hypotheses from each other. Suppose the client sends k randomly chosen challenges to the server. [5] If the server now has a success probability $\xi(\mathcal{P})$ then the number of correct challenge responses received from the

[5] Note that we can combine the statistical hypothesis test with the previous strategy "workload partition", i.e. the number of challenges k can be identical to the number of audits A.

server follows the binomial distribution $\mathsf{Binom}(k, \xi(\mathcal{P}))$. Further let us assume that the server's failure rate is minimal and we note that if the average success probability is high enough then extraction will be successful. Since the success probability $\xi(\mathcal{P})$ is unknown, we can estimate it using the k randomly chosen challenges to the server, i.e.

$$\widehat{\xi(\mathcal{P})} = \frac{\delta_{\texttt{accept}}}{k}$$

where $\delta_{\texttt{accept}} = |\{\delta_i \mid \delta_i = \texttt{accept}, i \in [k]\}|$. Following standard literature in statistical sciences [46], we know that the probability of the estimation corresponding exactly to the success probability is 0, that is more formally $\Pr[\widehat{\xi(\mathcal{P})} = \xi(\mathcal{P})] = 0$. Therefore, we approach the problem by estimating a confidence interval CI for the success probability. It follows

$$\mathrm{CI} = \left\{ \xi(\mathcal{P}) : \frac{\widehat{\xi(\mathcal{P})} - \xi(\mathcal{P})}{\sqrt{\xi(\mathcal{P})(1-\xi(\mathcal{P}))/k}} \leq z_{1-\alpha} \right\} \tag{7.7}$$

where $z_{1-\alpha}$ being the $1-\alpha$ quantile of a $\mathsf{Normal}(0,1)$ distribution and α is referred to as the error level. The probability that the true success probability ξ is covered by CI is approximately $\gamma = 1-\alpha$ which is usually denoted as the confidence parameter. Hence, we can reject the null hypothesis if

$$\beta := \frac{\widehat{\xi(\mathcal{P})} - \mu}{\sqrt{\mu(1-\mu)/k}} > z_{1-\alpha}. \tag{7.8}$$

Note that if we are able to reject the null hypothesis, this enables us to argue that our average success probability is significantly larger than μ and therefore extraction of the files should be successful.

Let us consider the following examples. First we fix $\mu = 0.95$ and the client sends $k = 1000$ challenges to the server. We use the typical error level $\alpha = 0.05$ and thus $z_{0.95} = 1.6449$.

	$\delta_{\texttt{accept}}$	$\widehat{\xi(\mathcal{P})}$	CI	δ
Example 1	600	3/5	[0.57428, 1)	`reject`
Example 2	970	97/100	[0.95978, 1)	`accept`

In Example 1, the client received 600 valid replies from a server and thus $\widehat{\xi(\mathcal{P})} = 3/5$. Using equation (7.7), we obtain a confidence interval in which the success probability $\xi(\mathcal{P})$ will be contained with a probability of 95%. Now we can use inequality (7.8) to check whether the null hypothesis may be rejected. It follows that $\beta = -50.7833$. Thus $\beta \not> z_{0.95}$ and we cannot reject the null hypotheses as the success probability is low. This means that extraction of the files will not be successful and CSPoRP will output `reject`.

Similarly, in Example 2, we have that the client receives 970 valid replies from a server and thus $\widehat{\xi(\mathcal{P})} = 97/100$. The client can compute the respective confidence interval and derives $\beta = 2.9019$. Here $\beta > z_{0.95}$ and thus the null hypothesis can be rejected and thus the success probability is significantly higher than μ. Therefore, extraction of the files will be successful and CSPoRP outputs `accept`.

Note that other approaches can also be considered to calculate the confidence interval. A good overview of them can be found in [2]. For example, a well-known approach was introduced by Clopper and Pearson [56] which uses a relationship between the cumulative binomial distribution and the beta distribution in order to state an easy formula to calculate the confidence interval.

7.5.2.3 Certificates

In this strategy the client wishes to receive a certificate from the server proving the retrievability of her files. Note that this certificate is not meant to be a cryptographic certificate. It is meant as a certificate that attests the validity and quality of the server storing data appropriately. A certificate basically consists of a *single* audit step where C chooses the challenge to be of size n, i.e. $\ell^{(k)} = n^{(k)}$ for all $k \in [K_{\mathfrak{S}_{\mathrm{id}}^{(c)}}]$. If CSPoRP is successful the server may create a certificate indicating that the request was executed correctly and thus leading to a valid proof stating the retrievability of the outsourced data. Additionally, the server may include a time stamp to state the validity of the certificate. Note that this certificate may be valid for a longer time interval than a usual audit request (e.g. one month) and makes the server liable for any data loss within this time interval.

7.5.2.4 Scheduled CSPoR

A different strategy could be implemented by a scheduled CSPoR. Here a client may wish to perform CSPoRP within a specific time frame (e.g. five hours) or may additionally label the files with tags indicating their classification levels. For example, a client may tag all her bank and insurance files with the classification level *secret* while her music library is tagged with *unrestricted*. Since our CSPoR system also enables to check any specified subset of the cloud storage, the client is able audit files with a classification tag *secret* more frequently than files tagged with *unrestricted* to ensure that data loss in valuable files is detected immediately.

7.5.3 Handling Erasure Detection Using CSPoR

In this section, we develop methods a client can use in case a CSPoRP execution has failed and thus the outsourced data may be damaged but not yet irrecoverable. In this context, we say that files are irrecoverable if more than $(1 - \rho)$ of the data have been

deleted, and thus the ECC decoding procedure cannot recover the underlying file.

7.5.3.1 Immediate Download

The approach of downloading all files in case an error is detected (that is, at least one of the audit steps returned `reject`) is the most cautious one a client could follow. It seems that this approach is the only strategy other PoR works have considered so far. However, this approach has enormous drawbacks for a client. Since C does not know which file contains an error she would be required to download all files within our cloud storage model. This is impractical for a client and she loses her initial benefits of outsourcing the files in the first place. In order to avoid downloading all files, we present in the following section a different strategy called *reduced CSPoR*.

7.5.3.2 Reduced CSPoR

In case a CSPoRP procedure fails (that is, at least one of the audits returned `reject`) we engage in a *reduced CSPoR* as we do not know in which file(s) an error occurred. Here, we present the strategy *reduced CSPoR* where a client runs a *b-ary search* to detect in which branch the CSPoRP failed. However, note that always all branches of a node need to be tested, since multiple files may be damaged. In case the corrupted files have been found, a download or repair can be initiated. Note that errors only occur rarely compared to successful audits, and thus files are usually retrievable. Otherwise, the client may subscribe to a new CSP. Note that simultaneously executed reduced CSPoRP are slower than a single CSPoRP since we are required to increase the amount of communication with the server in the number of simultaneously executed CSPoRP. This is acceptable since we assume that errors only occur very rarely. In Figure 7.2, we provide two examples where each node represents a CSPoRP over all its children and the leaves represent the respective files. A completely black node illustrates a corrupted file whereas a thick black circled node represents a failed CSPoRP as an error occurred. The left figure represents a scheme with no errors, thus a single CSPoRP is sufficient to prove that all files are retrievable. The right figure contains errors in the files $\mathcal{F}^{(1)}$ and $\mathcal{F}^{(6)}$. Thus, the initial CSPoRP over all files will fail. To find the corrupted files, we need to execute three CSPoRP at the same time, each over three different file sets. The first and second check will fail and hence we know that an error must be in at least two files. In order to locate the corrupted files, we execute six CSPoRP at the same time over each leaf, i.e. an individual check over each file $\mathcal{F}^{(1)}$ to $\mathcal{F}^{(6)}$. This finally identifies the corrupted files and the client can proceed to recover the damaged files. Note that the client only needs to recover the corrupted files and can leave all intact files with the server. Thus, with CSPoR the client still possesses the benefit of initially outsourcing the files and in case an error is detected only a small number of files need to be fixed. Another strategy could be to initially split the received responses from CSPoRProve into subsets and compare the smaller replies to determine immediately which files may

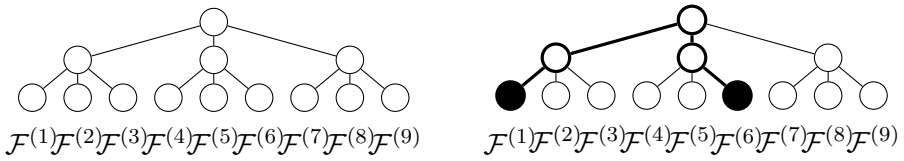

Figure 7.2: Illustration of the strategy reduced CSPoR

be corrupted. We believe that this may be an interesting strategy to investigate in future work.

7.5.4 Communication Model

The above introduced CSPoR system can be translated straightforwardly into present cloud architectures. This can be achieved by introducing procedures that capture the communication steps between a client and a cloud service provider.
The expression

$$\Pi\colon\ [C\colon in_C;\ S\colon in_S] \longrightarrow [C\colon out_C;\ S\colon out_S]$$

denotes the event that a client C and a provider S run an interactive protocol Π where $in_{\mathcal{X}}$ and $out_{\mathcal{X}}$ denote the input and output of entity $\mathcal{X}$ (either C or S), respectively.

In the following we describe the execution of the required procedures. Following the order of our algorithms in Definition 7.1, we first need to run the procedure Create between C and S in order to create a storage container $\mathfrak{S}^{(j)}$ located at the server in which the client stores her files. Note that a storage container is upper-bounded by $\mathfrak{S}^{(c)}_{\max}$, i.e. C and S need to engage in another Create procedure to create a new storage container as soon as the storage capacity is reached or a client wishes to store different types of data in different storage containers. In more detail, the procedure

$$\mathsf{Create}\colon\ [C\colon pk;\ S\colon pk] \longrightarrow [C\colon \mathfrak{S}^{(j)}_{\mathrm{id}}, \gamma_C;\ S\colon \mathfrak{S}^{(j)}, \gamma_S]$$

takes no other inputs than the public keys of both parties and outputs the identifier $\mathfrak{S}^{(j)}_{\mathrm{id}}$ to identify the storage container $\mathfrak{S}^{(j)}$ and a tag γ_C which contains metadata related to $\mathfrak{S}^{(j)}$ for the client. The CSP initialises the storage container $\mathfrak{S}^{(j)}$ on its infrastructure and obtains a tag γ_S as its output.

After the successful generation of a storage container a client wishes to store her files by executing a Store procedure as follows

$$\mathsf{Store}\colon\ [C\colon \widehat{F}, \mathfrak{S}^{(j)}_{\mathrm{id}};\ S\colon \mathfrak{S}^{(j)}] \longrightarrow [C\colon \widehat{\kappa}, \widehat{\tau};\ S\colon \widehat{\mathcal{F}}, \mathfrak{S}^{(j)}, \widehat{\tau}].$$

A client needs to provide as input her set of files $\widehat{F}$ and the respective storage container identifier $\mathfrak{S}_{\mathrm{id}}^{(j)}$ to store the files in $\mathfrak{S}^{(j)}$. The procedure outputs the set of processed files $\widehat{\mathcal{F}}$ and the updated storage container $\mathfrak{S}^{(j)}$ for the server. Furthermore, the client receives a set $\widehat{\kappa}$ which contains keys for file dependent functions (e.g. MACs or PRFs) and a set of verification tags $\widehat{\tau}$ which are computed on the client side. Those tags are also provided to S and used to check consistency of the file sizes.

Finally, a client C and cloud service provider S engage in a CSPoRP procedure.

$$\mathsf{CSPoRP}\colon\ [C\colon \widehat{\tau}, \widehat{\mathfrak{S}_{\mathrm{id}}};\ S\colon \widehat{\tau}, \widehat{\mathfrak{S}}] \longrightarrow [C\colon \delta;\ S\colon \perp].$$

In this procedure, a client provides her file tags $\widehat{\tau}$ and her respective set of storage container identifier $\widehat{\mathfrak{S}_{\mathrm{id}}}$. Note that in general our CSPoR scheme enables a client to check whether all outsourced files in $\widehat{\mathfrak{S}}$ are intact and retrievable. However, it is also possible for a client to check only a subset of her outsourced files by simply choosing a subset of tags $\widehat{\tau}'$ from $\widehat{\tau}$. The server inputs $\widehat{\mathfrak{S}}$ and the file tags $\widehat{\tau}$. The protocol run is accepted by the verifier if $\delta = \texttt{accept}$, or rejected otherwise. More precisely, the CSPoRP procedure uses additional locally computed values in order to be executed. Following our instantiation in Section 7.4, a client prepares her challenge set $\widehat{\mathcal{Q}}$ according to the file tags $\widehat{\tau}$ and sends the challenge set to S.

$$\mathsf{SendChallenge}\colon\ [C\colon \widehat{\mathcal{Q}};\ S\colon \perp] \longrightarrow [C\colon \perp;\ S\colon \widehat{\mathcal{Q}}].$$

The server uses the challenge set and computes the *authentication tags* σ and *responses* μ as its replies which are returned to C.

$$\mathsf{Response}\colon\ [C\colon \perp;\ S\colon \widehat{\mathfrak{S}}, \widehat{\tau}, I] \longrightarrow [C\colon \sigma, \mu;\ S\colon \perp].$$

Finally, a client uses the authentication tags and response values to verify the CSPoRP procedure as in Section 7.4. The client outputs a binary decision value δ indicating whether she accepts or rejects the CSPoRP procedure.

Realisation of Procedures

The introduced procedures are easily translated into current cloud architectures. We assume that a CSP exposes to its client a standard interface offering a handful of commands in order to execute some basic operations such as storing a file, downloading a file, as well as other commands. To implement such an interface for our CSPoR system, we can use currently employed APIs from Amazon [8], Google [87] or Microsoft [109]. Following those APIs, it suffices to use only two commands to implement the above procedures for a CSPoR system in current cloud architectures, namely `POST` and `GET`. These commands can achieve different functionalities by simply specifying different

parameters as detailed in their respective APIs.

7.5.5 Dynamic Updates

Our CSPoR scheme can achieve dynamic updates in its simplest form which seems to be inherent in current cloud storage implementations [8, 87, 109]. In more detail, current architectures do not support updating specific blocks of an outsourced file stored at a cloud service provider. Thus, a local file update on the provider's side is not possible, and a client needs to first retrieve the whole file, perform an update locally and upload the updated file back to the provider. Following the notion of procedures from Section 7.5.4, this can formally be described as

$$\mathsf{Download}\colon\ [C\colon \tau^{(k)}, \widehat{\mathfrak{S}_{\mathrm{id}}};\ S\colon \widehat{\mathfrak{S}}] \longrightarrow [C\colon F^{(k)};\ S\colon \perp].$$

Here, a client obtains the processed file $\mathcal{F}^{(k)}$ from the cloud service provider. Now she first needs to use the ECC decoding algorithm to obtain the original file $F^{(k)}$. After the file is updated locally by the client, she encodes $F^{(k)}$ again with the ECC encoding algorithm and uses the Store procedure to upload and store the file with the provider.

Note that most CSPs [8, 87, 109] also support versioning, i.e. the CSP still holds all old version of the files even after the files have been updated, and a client is able to retrieve any *old* version of the file by specifying a pointer in the command to the old file.

7.6 Evaluation

In this section we evaluate the trade-off between computation and communication. While the execution of each strategy can be different depending on each client's individual resources, we provide an example execution of the above strategy "workload partition". We denote each specific execution of a strategy as an *audit plan*.

We evaluate the audit plan "Fixed Audits". Another notable audit plan may be to leverage a table with a column for each file and the number of audits for the largest file is the number of rows. Then, instead of checking all files at the beginning, this strategy may distribute the workload of checking the files more evenly.

Let us fix the following main parameters for this evaluation. Each file block consists of $s = 512$ bytes (4096 bits) and each sector is a symbol in $\mathbb{F}$. Hence, the size of two blocks is one kibibyte (1 KiB, 1024 bytes). For simplicity, we store the files in only one storage container ($\Gamma = 1$) and choose the ECC rate $\rho = 1/2$.

Audit Plan: Fixed Audits

In this audit plan we wish to execute a CSPoRP procedure within a (previously) fixed number of audits and wish to evaluate the respective communication and computation cost for the client and server. For simplicity, we assume the following file structure. Each file $F^{(k)}$ consists of $n^{(k)} = 2^k$ blocks, i.e. with increasing k we obtain a larger file. In Table 7.2 and Table 7.3, we provide two example evaluations over 480 files with different-sized files for various different fixed number of audits ranging from $A = 1$ to $A = 10000$. Furthermore, the tables provide information about the sum of required challenge sizes of all files and provide details about the respective overall communication cost (payload) from client to server ($\mathfrak{P}[C \to S]$), and vice versa ($\mathfrak{P}[S \to C]$), for executing a CSPoRP. We also detail the portion of transmitted data in relation to the overall outsourced data which we denote as Datasize% and finally we compute the required time $\mathfrak{T}$ to execute a CSPoRP. Note, for simplicity, we omit disk read times and possible latencies for a client and cloud service provider.

In Table 7.2, we provide our first example evaluation and describe the procedure of obtaining the respective values. The table describes an evaluation of 480 outsourced

Data	A	L	Audit size	$\mathfrak{P}[C \to S]$	Datasize%	$\mathfrak{P}[S \to C]$	$\mathfrak{T}$
480 files: 7.5 GiB	1	7202070	54.9 MiB	54.9 MiB	0.72	2 KiB	0.5 sec
480 files: 7.5 GiB	5	5392050	41.1 MiB	206 MiB	2.68	10 KiB	2.6 sec
480 files: 7.5 GiB	10	4105920	31.3 MiB	313 MiB	4.08	20 KiB	3.1 sec
480 files: 7.5 GiB	30	2103150	16 MiB	481 MiB	6.26	60 KiB	4.8 sec
480 files: 7.5 GiB	60	1214970	9.27 MiB	556 MiB	7.24	120 KiB	5.6 sec
480 files: 7.5 GiB	120	658830	5.03 MiB	603 MiB	7.85	240 KiB	6 sec
480 files: 7.5 GiB	180	452010	3.45 MiB	621 MiB	8.08	360 KiB	6.2 sec
480 files: 7.5 GiB	500	169200	1.29 MiB	645 MiB	8.4	0.98 MiB	6.5 sec
480 files: 7.5 GiB	1000	85680	669 KiB	654 MiB	8.51	1.96 MiB	6.6 sec
480 files: 7.5 GiB	2500	34710	271 KiB	662 MiB	8.62	4.89 MiB	6.7 sec
480 files: 7.5 GiB	6000	14730	115 KiB	674 MiB	8.78	11.7 MiB	6.8 sec
480 files: 7.5 GiB	10000	8940	69.8 KiB	682 MiB	8.88	19.6 MiB	7 sec

Table 7.2: All parameters for a CSPoRP execution with 480 files with a total size of 7.5 GiB

files consisting of sixteen different files with increasing file sizes ranging from 1 KiB to 32 MiB where we outsource 30 different files of each size, i.e. 30 times $\widehat{F} = \{F^{(k)} \mid F^{(k)} \in \{0,1\}^*, k \in [16]\}$. The overall number of file blocks can be computed as $n = 30\sum_k n^{(k)} = 30(2^{k+1} - 2)$ and corresponds here to $n = 3932100$ blocks. Next we apply the erasure-correcting code with rate $\rho = 1/2$ to the files and thus $\tilde{n} = 2n$. Furthermore, we need to generate authentication tags σ for each file block of all $\tilde{n}$ blocks. Thus, the overall number of blocks we outsource to a server consists of $4n$ blocks which corresponds to 7.5 GiB. We calculate the challenge sizes for the respective files using equation (7.6) for different fixed number of audits and calculate the sum of all challenge sizes of one audit as $L := 30\sum_{k=1}^{16} \ell^{(k)}$.

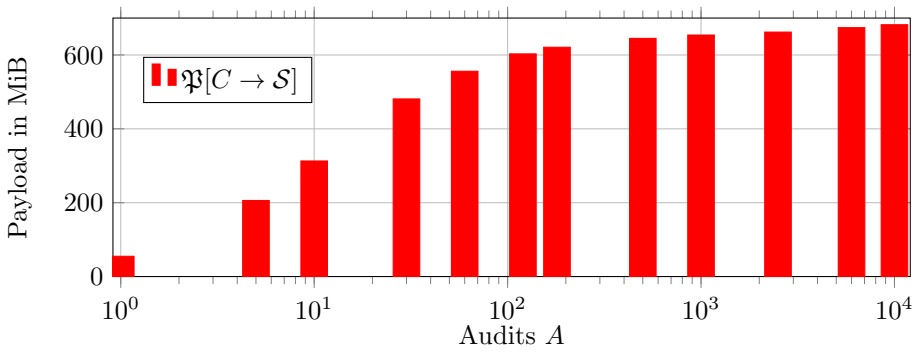

Figure 7.3: Payload $\mathfrak{P}$ from C to S for different number of audits for outsourced data of size 7.5 GiB

In Figure 7.3, we illustrate the respective payload from a client to a cloud service provider for different fixed number of audits in order to execute a CSPoRP procedure. The figure shows that the entire payload for a client increases the more audit steps she uses to execute a CSPoRP. This is the expected behaviour since we enable a client to choose the number of audits depending on her current resources. Figure 7.4 illustrates that the more audit steps are performed the smaller is the size of a single audit step itself. For example in Table 7.2 we show that if a client chooses to check all outsourced

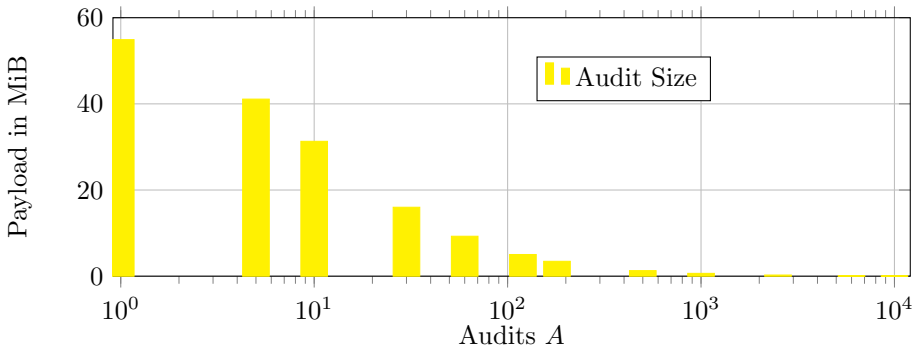

Figure 7.4: Single audit size for different number of audits for outsourced data of size 7.5 GiB

files within a single audit step $A = 1$ then the challenge size corresponds to 54.9 MiB whereas in case she wishes to check all outsourced files within ten thousand audit steps $A = 10000$ the respective challenge size per audit step is only 69.8 KiB.[6] Thus, CSPoR enables a client to flexibly choose the number of audits depending on her resources (this may also depend on the device she uses) determining whether she samples small or large challenges.

[6]The example with $A = 1$ corresponds to the discussed strategy of certificates as introduced in Section 7.5.2.3.

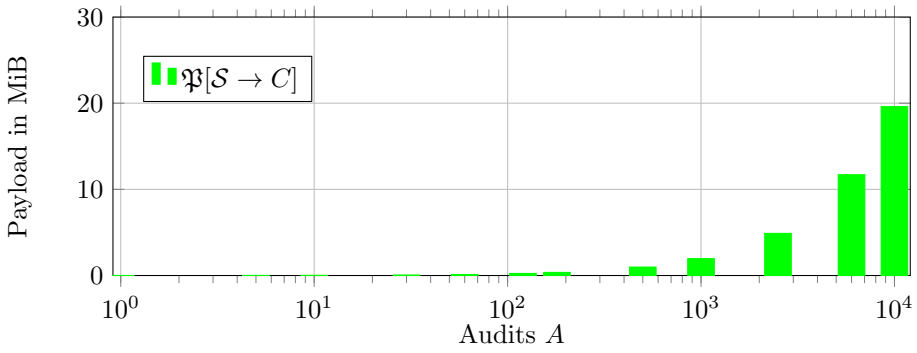

Figure 7.5: Payload $\mathfrak{P}$ from S to C for different number of audits for outsourced data of size 7.5 GiB

In Figure 7.5, we illustrate the respective payload from a cloud service provider to a client. The provider's payload is very small compared to the client's payload and ranges from 2 KiB to 19.6 MiB as it only returns $s + 1$ aggregated values per file.

In Figure 7.6, we illustrate the overall required time for executing a CSPoRP procedure over 480 files of a total size of 7.5 GiB assuming a communication throughput of 100 MiB/s. Depending on the number of audits the required time to obtain a retrievability assurance ranges between 0.5 seconds and 7 seconds.

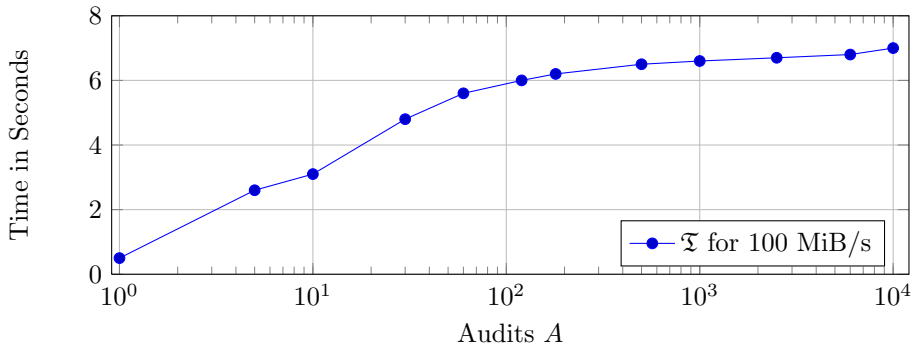

Figure 7.6: Required time $\mathfrak{T}$ for a full CSPoRP execution for different number of audits for outsourced data of size 7.5 GiB

In Table 7.3, we provide our second example evaluation. Similarly to the previous example, the table describes an evaluation of 480 outsourced files consisting of thirty different files of increasing file size ranging from 1 KiB to 512 GiB, and we outsource 16 different files of each size, i.e. 16 times $\widehat{F} = \{F^{(k)} \mid F^{(k)} \in \{0,1\}^*, k \in [30]\}$. The overall number of file blocks can be computed as $n = 16 \sum_k n^{(k)} = 16(2^{k+1} - 2)$ and corresponds here to $n = 34359738336$ blocks. Next we apply the error correcting code with rate $\rho = 1/2$ to the files and thus $\tilde{n} = 2n$ and we need to generate authentication

Data	A	L	Audit size	$\mathfrak{P}[C \to S]$	Datasize%	$\mathfrak{P}[S \to C]$	$\mathfrak{T}$
480 files: 65 TiB	1	65542267808	488.33 GiB	488.33 GiB	0.75	2 KiB	5000.47 sec
480 files: 65 TiB	5	55317309232	412.15 GiB	2.012 TiB	3.14	10 KiB	21101.88 sec
480 files: 65 TiB	10	46294897296	344.92 GiB	3.368 TiB	5.26	20 KiB	35320.2 sec
480 files: 65 TiB	30	28023896624	208.79 GiB	6.12 TiB	9.56	60 KiB	64141.61 sec
480 files: 65 TiB	60	17605552992	131.17 GiB	7.69 TiB	12.01	120 KiB	80591 sec
480 files: 65 TiB	120	10098261536	75.24 GiB	8.82 TiB	13.78	240 KiB	92452.35 sec
480 files: 65 TiB	180	7079565040	52.75 GiB	9.27 TiB	14.49	360 KiB	97223.03 sec
480 files: 65 TiB	500	2728924896	20.33 GiB	9.93 TiB	15.51	0.98 MiB	104100.22 sec
480 files: 65 TiB	1000	1392163536	10.37 GiB	10.13 TiB	15.83	1.96 MiB	106213.65 sec
480 files: 65 TiB	2500	563732864	4.2 GiB	10.25 TiB	16.02	4.89 MiB	107523.51 sec
480 files: 65 TiB	6000	236020816	1.76 GiB	10.30 TiB	16.1	11.7 MiB	108041.87 sec
480 files: 65 TiB	10000	141807840	1.06 GiB	10.31 TiB	16.12	19.6 MiB	108190.99 sec

Table 7.3: All parameters for a CSPoRP execution with 480 files with a total size of 65 TiB

tags σ for each of $\tilde{n}$ blocks. Thus, the overall number of blocks we outsource to a server consists of $4n$ blocks which corresponds to 65 TiB. We calculate the challenge sizes for the respective files using equation (7.6) for different fixed number of audits and calculate the sum of all challenge sizes of one audit as $L := 16 \sum_{k=1}^{30} \ell^{(k)}$.

As in the previous example, in Figure 7.7 we evaluate the payload from a client to a cloud service provider for different fixed number of audits in order to execute a CSPoRP procedure. Figure 7.7 looks similar to Figure 7.3 while the main difference is that the client needs to sample more blocks per audit step since she outsourced larger files and thus requires to check more blocks in order to obtain an assurance about the retrievability of the files. Thus, the client's payload ranges from approximately 0.5 TiB to 10.31 TiB whereas before the maximum payload was 682 MiB.

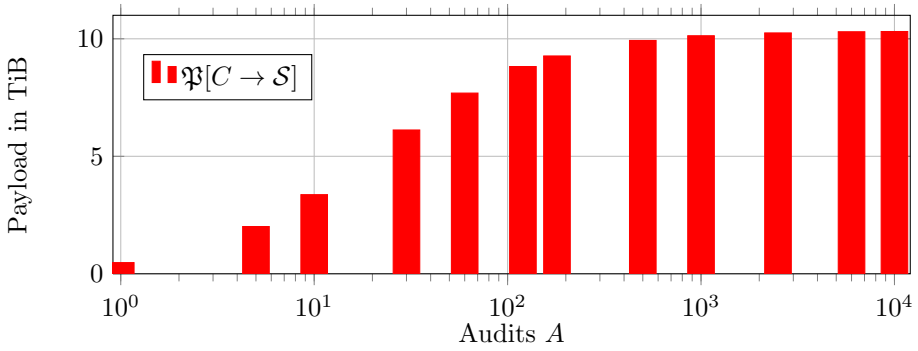

Figure 7.7: Payload $\mathfrak{P}$ from C to S for different number of audits for outsourced data of size 65 TiB

In Figure 7.8, we illustrate as in the previous example that the more audit steps are performed for a CSPoRP procedure the smaller is the challenge size of a single audit itself.

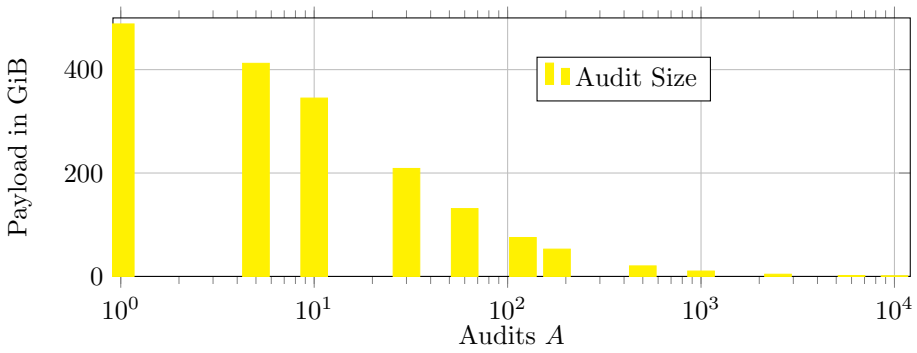

Figure 7.8: Single audit size for different number of audits for outsourced data of size 65 TiB

We illustrate in Figure 7.9 the payload from the cloud service provider to a client. This figure is exactly the same as Figure 7.5 from the above example and thus the provider's payload does not change at all even if a client outsources more data. The reason is that the provider needs always to compute $s + 1$ aggregated values per file which are independent of the file size.

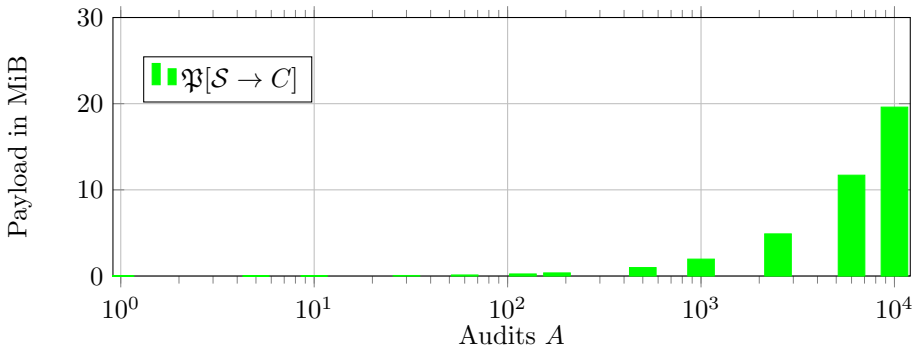

Figure 7.9: Payload $\mathfrak{P}$ from S to C for different number of audits for outsourced data of size 65 TiB

Lastly, in Figure 7.10, we evaluate the overall required time for executing a CSPoRP procedure over 480 files of a total size of 65 TiB assuming a communication throughput of 100 MiB/s. The performance of CSPoRP is in magnitudes slower than in the above example due to the large outsourced files. Here the required time to obtain an information-theoretical assurance ranges between 5000 seconds and 108190 seconds, or in other words it takes between 1.38 hours and 30.05 hours. Thus, we conclude that the performance of CSPoR is very good as long as a lot of small files are checked. In future work, we wish to enhance the performance of the scheme for large files.

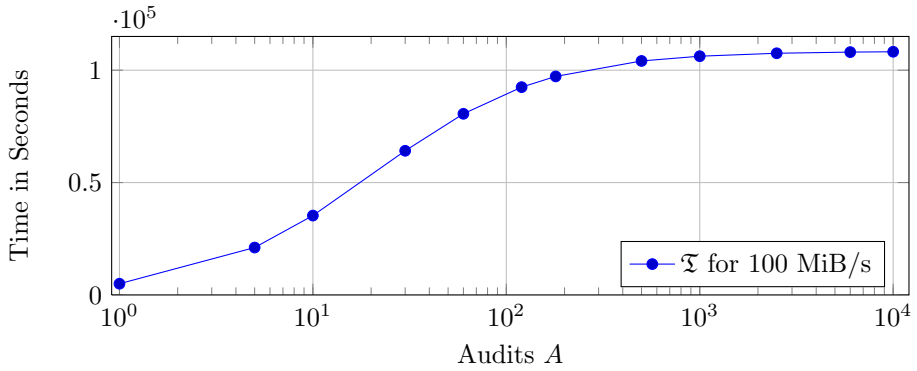

Figure 7.10: Required time $\mathfrak{T}$ for a full CSPoRP execution for different number of audits for outsourced data of size 65 TiB

7.7 Conclusion

We introduced in this chapter an extension to the traditional PoR concept which we call a cloud storage proofs of retrievability scheme. A CSPoR enables a client to check whether *all* outsourced files at a cloud storage provider are still intact and retrievable. Our model introduces and uses the notion of a storage container modelling the underlying storage structure which plays a crucial practical role in today's cloud service provider's infrastructure. We proposed an efficient instantiation of CSPoR based on well-known constructions and argue in detail about possible strategies in order to obtain a retrievability assurance about all files. Our proposed solution overcomes shortcomings in current models when a client wishes to audit *all* files simultaneously.

We evaluated the performance of the scheme and showed that, for example, CSPoR can check the retrievability of 480 different-sized files (of size at most 32 MiB) in between 0.5 seconds and 7 seconds depending on the number of performed audit steps. Unfortunately, the performance of the scheme gets of magnitudes slower as soon as we outsource large files of size 512 GiB. We provide a discussion about the required API to lift CSPoR into a cloud system (e.g. Amazon's S3).

In future work we believe it would be interesting and beneficial to consider other entity populations such as a multi-server scheme similarly to Bowers et al. [39] who considered this case for PDP. This scenario has been recently considered by Paterson et al. [120] in the realm of PoR, although limited to the single file case. We also envisage that the case of multiple clients outsourcing a vast amount of files to a cloud service provider being interesting to investigate as other concepts such as storage deduplication may play a crucial role. Since the performance of CSPoR was of magnitudes slower as soon as a client outsourced large files, it would be interesting to investigate ways to enhance the performance. One possible solution may be to split large files in smaller ones.

Chapter 8

Conclusion

In this thesis we have considered systems and solutions regarding the verification of computations and data retrievability.

The main part of this thesis is devoted to the setting of publicly verifiable outsourced computation which allows computationally weak devices to delegate computations to a more powerful but yet untrusted server, and to verify the correctness of returned results based on public information. It has been shown by Parno et al. [118] that KP-ABE can be used as a verifiable proof mechanism for the satisfaction of Boolean functions in PVC.

We began in Chapter 3 to consider the notion of PVC in a more practical framework in which we accommodated multiple servers offering a computational service for multiple functions simultaneously. Servers offering such a service may be rewarded per computation, and as such have an incentive to cheat by returning malformed responses rather than devoting resources and time to compute a valid result. Thus, we implemented a revocation mechanism into the PVC setting to remove misbehaving servers from the system. This can be seen as a mechanism to save other delegators resources since they do not need to waste their valuable resources by outsourcing to a misbehaving server in the first place.

In Chapter 4 we considered a setting in which an untrusted server holds a data set in such a way that any client can ask the server to compute a function on any input portion of the data set. We showed that ciphertext-policy ABE can be used as a building block in order to instantiate this mode of computation called publicly verifiable delegable computation (VDC). This model can be seen as a reversed system architecture in comparison to the proposal in Chapter 3. VDC is more akin to the traditional client-server model and we showed that the model has some interesting possible applications to verifiable queries to remote databases and verifiable processing of large data sets.

In Chapter 5 we brought together the previous notions and unified them into a single publicly verifiable outsourced computation system called hybrid PVC. This model only requires a single setup stage in order to provide a flexible outsourced computation solution capturing both previous systems. This was achieved by a novel use of

dual-policy ABE which combines KP-ABE and CP-ABE. We briefly introduced yet another mode of publicly verifiable computation in this chapter that enables us to enforce (graph-based) access control policies over the delegators, servers and verifiers. This mode was motivated on the observation that even in a publicly verifiable setting it is unlikely that all entities will have unrestricted access to all functionalities of the system and thus we may enforce access control policies to restrict the access accordingly.

In Chapter 6 we investigated the possibility of using techniques developed mainly for PVC in the realm of verifiable searchable encryption. We introduced an extended verifiable searchable encryption scheme based upon CP-ABE and techniques from Chapter 4 that permits a user to verify that search results are correct and complete. Our scheme enables the client to perform a wider class of queries, i.e. it permits verifiable computational queries over keywords and specific data values, that go beyond the standard keyword matching queries to allow functions such as averaging or counting operations.

In Chapter 7 we turned our attention to the setting of providing provable outsourced data storage guarantees. We introduced a new proofs of retrievability (PoR) model called cloud-storage PoR which enables a client to outsource multiple different-sized files to a cloud service provider and we use homomorphic properties to efficiently check whether all outsourced files are still retained and intact. This proposed extension aims to provide a more practical approach to this problem and overcomes limitations in the literature where only a single file can be checked. We discussed different strategies a client may use in order to obtain a statement about the retrievability and also evaluated the performance of our scheme.

Throughout this thesis we have emphasised in each chapter on possible future research problems and we believe that both areas will continue being very active in the next coming years.

Bibliography

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[2] A. Agresti and B. A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, May 1998.

[3] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015.

[4] J. Alderman, C. Janson, C. Cid, and J. Crampton. Revocation in publicly verifiable outsourced computation. In D. Lin, M. Yung, and J. Zhou, editors, *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, volume 8957 of *Lecture Notes in Computer Science*, pages 51–71. Springer, 2014.

[5] J. Alderman, C. Janson, C. Cid, and J. Crampton. Access control in publicly verifiable outsourced computation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 657–662, New York, NY, USA, 2015. ACM.

[6] J. Alderman, C. Janson, C. Cid, and J. Crampton. Hybrid publicly verifiable computation. In K. Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2016.

[7] J. Alderman, C. Janson, K. M. Martin, and S. L. Renwick. Extended functionality in verifiable searchable encryption. In E. Pasalic and L. R. Knudsen, editors, *Cryptography and Information Security in the Balkans - Second International Conference, BalkanCryptSec 2015, Koper, Slovenia, September 3-4, 2015, Revised Selected Papers*, volume 9540 of *Lecture Notes in Computer Science*, pages 187–205. Springer, 2015.

[8] Amazon. Amazon S3 API, 2015. `http://docs.aws.amazon.com/AmazonS3/latest/API/s3-api.pdf`.

[9] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. Verifiable oblivious storage. In H. Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2014.

[10] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. Spirakis, editors, *Automata, Languages and Programming*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer Berlin Heidelberg, 2010.

[11] F. Armknecht, J. Bohli, G. O. Karame, Z. Liu, and C. A. Reuter. Outsourced proofs of retrievability. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 831–843. ACM, 2014.

[12] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[13] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, Jan. 1998.

[14] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. N. J. Peterson, and D. Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1):12, 2011.

[15] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 598–609. ACM, 2007.

[16] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In A. Levi, P. Liu, and R. Molva, editors, *4th International ICST Conference on Security and Privacy in Communication Networks, SECURECOMM 2008, Istanbul, Turkey, September 22-25, 2008*, page 9. ACM, 2008.

[17] N. Attrapadung and H. Imai. Attribute-based encryption supporting direct/indirect revocation modes. In M. G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 278–300. Springer, 2009.

[18] N. Attrapadung and H. Imai. Dual-policy attribute based encryption. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 168–185, 2009.

[19] N. Attrapadung and H. Imai. Dual-policy attribute based encryption: Simultaneous access control with ciphertext and key policies. *IEICE Transactions*, 93-A(1):116–125, 2010.

[20] L. Babai. Trading group theory for randomness. In R. Sedgewick, editor, *STOC*, pages 421–429. ACM, 1985.

[21] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 271–286. IEEE Computer Society, 2015.

[22] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 863–874. ACM, 2013.

[23] M. Barbosa and P. Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. In O. Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 296–312. Springer Berlin Heidelberg, 2012.

[24] A. Beimel. Secure schemes for secret sharing and key distribution. PhD thesis, Technion-Israel Institute of technology, Faculty of computer science, 1996.

[25] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. Incentivizing outsourced computation. In *Proceedings of the 3rd International Workshop on Economics of Networked Systems*, NetEcon '08, pages 85–90, New York, NY, USA, 2008. ACM.

[26] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[27] M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1992.

[28] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 294–304, New York, NY, USA, 1993. ACM.

[29] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In R. D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 401–414. ACM, 2013.

[30] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer Berlin Heidelberg, 2011.

[31] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[32] M. A. Bishop. *Computer Security. Art and Science*. Addison-Wesley Professional, 2002.

[33] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012.

[34] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.

[35] A. Boldyreva, V. Goyal, and V. Kumar. Identity-based encryption with efficient revocation. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM Conference on Computer and Communications Security*, pages 417–426. ACM, 2008.

[36] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[37] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In S. P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007.

[38] C. Bösch, P. H. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, 2014.

[39] K. D. Bowers, A. Juels, and A. Oprea. HAIL: a high-availability and integrity layer for cloud storage. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, pages 187–198. ACM, 2009.

[40] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: theory and implementation. In R. Sion and D. Song, editors, *Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, November 13, 2009*, pages 43–54. ACM, 2009.

[41] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 501–514. ACM, 2011.

[42] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.

[43] R. Canetti, B. Riva, and G. Rothblum. Verifiable computation with two or more clouds. In *Workshop on Cryptography and Security in Clouds*, 2011.

[44] C. Carlet. Boolean functions for cryptography and error correcting codes. *Boolean models and methods in mathematics, computer science, and engineering*, 2:257, 2010.

[45] H. Carter, C. Lever, and P. Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 266–275, New York, NY, USA, 2014. ACM.

[46] G. Casella and R. Berger. *Statistical Inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002.

[47] D. Cash, A. Küpçü, and D. Wichs. Dynamic Proofs of Retrievability via Oblivious RAM. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 279–295. Springer, 2013.

[48] Q. Chai and G. Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Proceedings of IEEE International Conference on Communications, ICC 2012*, pages 917–922. IEEE, 2012.

[49] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005.

[50] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

[51] R. Cheng, J. Yan, C. Guan, F. Zhang, and K. Ren. Verifiable searchable symmetric encryption from indistinguishability obfuscation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 621–626, New York, NY, USA, 2015. ACM.

[52] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In A. Sahai, editor, *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 499–518. Springer, 2013.

[53] K. Chung, Y. T. Kalai, F. Liu, and R. Raz. Memory delegation. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011.

[54] K. Chung, Y. T. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010.

[55] M. Clear and C. McGoldrick. Policy-based non-interactive outsourcing of computation using multikey FHE and CP-ABE. In P. Samarati, editor, *SECRYPT*, pages 444–452. SciTePress, 2013.

[56] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):pp. 404–413, 1934.

[57] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *3th ACM Conference on Computer and Communications Security*, pages 79–88. ACM, 2006.

[58] R. Curtmola, O. Khan, R. C. Burns, and G. Ateniese. MR-PDP: Multiple-Replica Provable Data Possession. In *ICDCS*, pages 411–420. IEEE Computer Society, 2008.

[59] T. W. Cusick and P. Stanica. *Cryptographic Boolean functions and applications*. Academic Press, 2009.

[60] S. Daily. Big Data, for better or worse: 90last two years, 2013. http://www.sciencedaily.com/releases/2013/05/130522085217.htm.

[61] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[62] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of Retrievability via Hardness Amplification. In O. Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2009.

[63] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 213–222. ACM, 2009.

[64] M. Etemad and A. Küpçü. Database outsourcing with hierarchical authenticated data structures. *IACR Cryptology ePrint Archive*, 2015:351, 2015.

[65] M. Ferrante and N. Frigo. A note on the coupon - collector's problem with multiple arrivals and the random sampling, 2012.

[66] M. Ferrante and M. Saltalamacchia. The coupon collectors problem. *MATerials MATemtics*, Volum 2014(treball no. 2):35, 2014.

[67] A. L. Ferrara, G. Fuchsbauer, and B. Warinschi. Cryptographically enforced RBAC. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26-28, 2013*, pages 115–129. IEEE, 2013.

[68] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 501–512. ACM, 2012.

[69] D. Fiore, R. Gennaro, and V. Pastro. Efficiently verifiable computation on encrypted data. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 844–855. ACM, 2014.

[70] Z. Fu, J. Shu, X. Sun, and N. Linge. Smart cloud search services: verifiable keyword-based semantic search over encrypted cloud data. *Consumer Electronics, IEEE Transactions on*, 60(4):762–770, 2014.

[71] S. Gajek. Dynamic symmetric searchable encryption from constrained functional encryption. In K. Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA,*

February 29 - March 4, 2016, Proceedings, volume 9610 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2016.

[72] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[73] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[74] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[75] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In L. Fortnow and S. P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 99–108. ACM, 2011.

[76] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In M. L. Scott and L. L. Peterson, editors, *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, pages 29–43. ACM, 2003.

[77] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/`.

[78] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the Association for Computing Machinery*, 43:431–473, 1996.

[79] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

[80] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *STOC*, pages 555–564. ACM, 2013.

[81] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In C. Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122. ACM, 2008.

[82] S. Goldwasser, H. Lin, and A. Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[83] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA, 1985. ACM.

[84] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004.

[85] Google. Google Compute Engine – Cloud Computing & IaaS – Google Cloud Platform. `http://cloud.google.com/compute/`, 2014. [Online; accessed 23-October-2014].

[86] Google. Google Cloud Platform: Concepts and Techniques, 2015. `http://cloud.google.com/storage/docs/concepts-techniques/`.

[87] Google. Google Storage API Reference, 2015. `https://cloud.google.com/storage/docs/json_api/v1/`.

[88] S. D. Gordon, J. Katz, F. Liu, E. Shi, and H. Zhou. Multi-client verifiable computation with stronger security guarantees. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 144–168. Springer, 2015.

[89] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[90] M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of ABE ciphertexts. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

[91] C. Gritti, W. Susilo, and T. Plantard. Efficient dynamic provable data possession with public verifiability and data privacy. In E. Foo and D. Stebila, editors, *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*, volume 9144 of *Lecture Notes in Computer Science*, pages 395–412. Springer, 2015.

[92] C. Guan, K. Ren, F. Zhang, F. Kerschbaum, and J. Yu. Symmetric-key based proofs of retrievability supporting public verification. In G. Pernul, P. Y. A. Ryan, and E. R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, volume 9326 of *Lecture Notes in Computer Science*, pages 203–223. Springer, 2015.

[93] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800:162, 2014.

[94] W. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2010.

[95] International Organization for Standardization (ISO). ISO/IEC 11889-1:2015: Information technology - Trusted Platform Module Library - Part 1: Architecture, 2008.

[96] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In B. Preneel, editor, *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*, volume 152 of *IFIP Conference Proceedings*, pages 258–272. Kluwer, 1999.

[97] C. Janson, C. A. Reuter, F. Armknecht, and C. Cid. Cloud storage proofs of retrievability, 2016. In Submission.

[98] A. Juels and B. S. K. Jr. PORs: Proofs Of Retrievability for Large Files. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007.

[99] S. Kamara, C. Papamonthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Conference on Computer and Communications Security*, pages 965–976. ACM, 2012.

[100] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[101] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.

[102] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In W. J. Bolosky and J. Flinn, editors, *Proceedings of the 10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012*, page 20. USENIX Association, 2012.

[103] V. Kher and Y. Kim. Securing distributed storage: challenges, techniques, and systems. In V. Atluri, P. Samarati, W. Yurcik, L. Brumbaugh, and Y. Zhou, editors, *Proceedings of the 2005 ACM Workshop On Storage Security And Survivability, StorageSS 2005, Fairfax, VA, USA, November 11, 2005*, pages 9–25. ACM, 2005.

[104] K. Kurosawa and Y. Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *Cryptology and Network Security - 12th International Conference, CANS 2013*, volume 8257 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2013.

[105] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Fuzzy keyword search over encrypted data in cloud computing. In *INFOCOM 2010. 29th IEEE International Conference on Computer Commu- nications, Joint Conference of the IEEE Computer and Communications Societies*, pages 441–445. IEEE, 2010.

[106] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. A Cooperative Internet Backup Scheme. In *USENIX Annual Technical Conference, General Track*, pages 29–41. USENIX, 2003.

[107] P. Liu, J. Wang, H. Ma, and H. Nie. Efficient verifiable public key encryption with keyword search based on KP-ABE. In *Ninth International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2014*, pages 584–589. IEEE, 2014.

[108] S. Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.

[109] Microsoft. Microsoft Azure: How to use Blob storage from .NET, 2015. `https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/`.

[110] F. Monrose, P. Wyckoff, and A. D. Rubin. Distributed execution with remote audit. In *NDSS*, volume 99, pages 3–5, 1999.

[111] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.

[112] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 195–203, New York, NY, USA, 2007. ACM.

[113] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In A. Sahai, editor, *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 222–242. Springer, 2013.

[114] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 91–110. Springer, 2011.

[115] D. J. Park, K. Kim, and P. J. Lee. Public key encryption with conjunctive field keyword search. In *Information Security Applications, 5th International Workshop*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004.

[116] D. J. Park, K. Kim, and P. J. Lee. Public key encryption with conjunctive field keyword search. In C. H. Lim and M. Yung, editors, *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004.

[117] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (S&P), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[118] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[119] M. B. Paterson, D. R. Stinson, and J. Upadhyay. A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. *J. Mathematical Cryptology*, 7(3):183–216, 2013.

[120] M. B. Paterson, D. R. Stinson, and J. Upadhyay. Multi-prover proof-of-retrievability. Cryptology ePrint Archive, Report 2016/265, 2016. `http://eprint.iacr.org/`.

[121] V. Pham, M. H. R. Khouzani, and C. Cid. Optimal contracts for outsourced computation. In R. Poovendran and W. Saad, editors, *Decision and Game Theory for Security - 5th International Conference, GameSec 2014, Los Angeles, CA, USA, November 6-7, 2014. Proceedings*, volume 8840 of *Lecture Notes in Computer Science*, pages 79–98. Springer, 2014.

[122] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[123] Y. Ren, J. Xu, J. Wang, and J.-U. Kim. Designated-verifier provable data possession in public cloud storage. *International Journal of Security and Its Applications*, 7(6):11–20, 2013.

[124] B. Rosen. On the coupon collector's waiting time. *Ann. Math. Statist.*, 41(6):1952–1969, 12 1970.

[125] R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.

[126] H. Shacham and B. Waters. Compact proofs of retrievability. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2008.

[127] S.-T. Shen and W.-G. Tzeng. Delegable provable data possession for remote data in the clouds. In S. Qing, W. Susilo, G. Wang, and D. Liu, editors, *ICICS*, volume 7043 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2011.

[128] E. Shi, E. Stefanov, and C. Papamanthou. Practical dynamic proofs of retrievability. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM Conference on Computer and Communications Security*, pages 325–336. ACM, 2013.

[129] J. Shi, J. Lai, Y. Li, R. H. Deng, and J. Weng. Authorized keyword search on encrypted data. In M. Kutylowski and J. Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, volume 8712 of *Lecture Notes in Computer Science*, pages 419–435. Springer, 2014.

[130] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[131] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In P. Q. Nguyen and D. Pointcheval, editors,

Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[132] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy, Berkeley, California, USA*, pages 44–55. IEEE, 2000.

[133] E. Stefanov, C. Papamonthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society, 2014.

[134] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea. Iris: a scalable cloud file system with efficient integrity checks. In R. H. Zakon, editor, *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, 3-7 December 2012*, pages 229–238. ACM, 2012.

[135] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel Distributed Systems*, 25(11):3025–3035, 2014.

[136] W. Sun, S. Yu, W. Lou, T. Hou, and H. Li. Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. *Parallel and Distributed Systems, IEEE Transactions on*, (99), 2013.

[137] J. van den Hooff, M. F. Kaashoek, and N. Zeldovich. Versum: Verifiable computations over large public logs. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1304–1316. ACM, 2014.

[138] V. Vu, S. T. V. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 223–237. IEEE Computer Society, 2013.

[139] C. Wang, N. Cao, J. Li, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *International Conference on Distributed Computing Systems, ICDCS 2010*, pages 253–262. IEEE Computer Society, 2010.

[140] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions Parallel Distributed Systems*, 23(8):1467–1479, 2012.

[141] J. Wang, H. Ma, J. Li, H. Zhu, S. Ma, and X. Chen. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer Science Information Systems*, 10(2):667–684, 2013.

[142] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

[143] L. Xu and S. Tang. Verifiable computation with access control in cloud computing. *The Journal of Supercomputing*, 69(2):528–546, 2014.

[144] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.

[145] J. Yuan and S. Yu. Proofs of retrievability with public verifiability and constant communication cost in cloud. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, Cloud Computing '13, pages 19–26, New York, NY, USA, 2013. ACM.

[146] M. Yung. Zero-knowledge proofs of computational power (extended summary). In J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, volume 434 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 1989.

[147] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. *IACR Cryptology ePrint Archive*, 2016:172, 2016.

[148] Q. Zheng, S. Xu, and G. Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pages 522–530. IEEE, 2014.