On the use of Attribute-based Encryption in Publicly Verifiable Outsourced Computation

James Alderman

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 Jason Crampton.

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

James Alderman

February, 2016

Abstract

Publicly verifiable outsourced computation (PVC) allows devices with restricted resources to delegate computations to external servers, and to verify the correctness of results. Servers may be rewarded per computation, and so have an incentive to cheat rather than devote resources to a computation. Also, within an organisation, it is likely that individual user permissions will vary and so outsourced actions should be restricted accordingly. This gives rise to two interesting problems in the PVC setting addressed in this thesis: finding a method to revoke and punish cheating servers; and enforcing access control policies that restrict the computations each entity may outsource, compute or read the results of.

In this thesis, we use primitives traditionally used to cryptographically enforce access control policies to construct secure PVC systems that meet these requirements. We first extend prior PVC schemes based on *key-policy attribute-based encryption* (ABE) to accommodate a broader system model where servers may compute multiple functions and be prevented from performing further computations if found cheating. We then show how a *key assignment scheme* can provide flexible access control over entities. Finally, we consider an alternative scenario in which input data is held by the server rather than the client, and construct a provably secure instantiation based on *ciphertext-policy ABE*. We conclude by showing that *dual-policy ABE* can accommodate both models of outsourced computation and provide a level of access control within a single system.

Contents

List of Figures

List of Tables

List of Security Games

Acknowledgements

Firstly, I would like to thank my supervisor Jason Crampton for his invaluable support and guidance over the course of my PhD, and for his always detailed feedback, even when I pulled him away from his area of expertise.

I am grateful to BAE Systems Advanced Technology Centre and the EPSRC for their financial support.

I have been very fortunate to make some great friends within the department. Thank you especially to Susan for always surpassing my expectations, to Dale for the rants, massages and gifts, to Shahram for the maple syrup, and to Gordon for the pictures. Thanks to Christian for being a wonderful co-author and for teaching me German, to Dan and Rachel for the Tuesday outings and for their constant lookout for llamas and to Naomi, for being there.

Finally, thank you to my family for their love and support, during my PhD and always.

Chapter 1

Introduction

Contents

This chapter provides the motivation for, and structure of, the thesis.

1.1 Motivation

It is increasingly common for mobile devices to be used as general purpose computing devices and for low-power sensors to be deployed to collect and analyse data. There is also a trend towards cloud computing and performing computations over enormous volumes of data ("big data"), which means that many computations of practical interest may require considerable computing resources. In short, there is a growing discrepancy between the computing resources of end-user devices and the resources required to perform complex computations on large datasets. This discrepancy, in part, has led to the increasing use of software-as-a-service, where computing applications are provided by an external, potentially untrusted, cloud service provider.

Consider, for example, a company that operates a "bring your own device" policy, or where employees must be able to perform work duties when not in the office; thus, employees may use devices such as smartphones and tablets for work purposes. Due to resource limitations, it may not be possible for these devices to perform complex computations locally, as perhaps would have been the case on a company-issued desktop device. Instead, the company subscribes to a cloud service provider offering software-as-a-service and allows employees to outsource their computations over some network to the more powerful cloud

servers. Computational results are returned to the client device. More precisely, given a function F to be computed by a server S, the client sends an input x to S, which should return $y = F(x)$ to the client. However, there may be an incentive for the server (or an imposter) to cheat and to return an invalid result $y' \neq F(x)$ to the client. The server may wish to convince a client of incorrect information, or the server may be too busy or may not wish to devote resources to perform the computation, particularly if servers are rewarded per computation performed. Thus, it is important that the client gains some assurance that the result y returned by the server is, in fact, $F(x)$.

Another example arises in the context of battlefield communications [57] where a setup operation is performed by a trusted server within a military base and then each member of a squadron of soldiers is deployed into the field with a reasonably light-weight computing device. The soldiers gather data from their surroundings and send it to regional servers for analysis before receiving tactical commands based on results. Those servers may not be fully trusted, e.g. the servers may be hosted in enemy territory or the soldiers may be part of a (weak) coalition. Thus, soldiers must have an assurance that the command has been computed correctly before making tactical decisions. A final example is that of sensor networks where lightweight sensors transmit readings to a more powerful base station that computes statistics that can be verified by an experimenter.

Solutions to the above problems are known as *verifiable computation* (VC) schemes. VC schemes enable the outsourcing of computation requests to a more powerful server and verification of the results of such computations. In particular, the outsourcing and verification stages (both run by the client) must be more efficient (require less computational resources) than computing the function itself. Otherwise there is little to gain from outsourcing such computations; indeed, it is assumed that client devices lack these resources. These efficient outsourcing and verification operations may follow an expensive, one-time setup phase to initialise the system, the aim being to amortise this cost over many outsourced computations. For example, the setup phase could be performed on an employee's desktop device so that a lightweight, mobile device can be used outside the office environment, or the setup could be performed by a trusted server within a military base [57].

Recent work in this area has led to the notion of *publicly verifiable computation* (PVC) where, once a system is initialised, any entity may outsource computations or verify computational results using only public information. PVC distributes the heavy setup cost

by allowing many users to reap the benefits of one client's effort, rather than every single user having to exert the effort themselves; thus PVC provides a more practical and cost-effective alternative to VC. PVC schemes in particular lend themselves to practical environments such as those discussed in the examples above wherein multiple users (e.g. employees, soldiers or sensors within a network) may wish to outsource computations and do not wish to each perform the expensive setup operation. Indeed, it may well be the case that it is difficult to amortise the cost of setup over the computations outsourced by a single client device but, when distributed over multiple clients, the joint cost is easier to justify. Additionally, particularly in a multi-user environment, many users may be interested in the same results and may all wish to verify correctness before using the results in further work; PVC allows a single computation to be performed, the results of which can be used by all interested clients. In this thesis, we focus on PVC as a desirable alternative to VC and aim to enhance the existing solutions in terms of practicality and functionality.

1.2 Thesis Structure

In this thesis, we aim to explore PVC in more detail. In Chapter 2, we introduce relevant background material and notation that is required throughout the remainder of the thesis. We then, in Chapter 3, take a closer look at the system architecture considered in current PVC schemes and see that there exists a distinguished client that must perform the system setup operations. This client, in effect, becomes an authority on computational servers that may be used by other clients. Current schemes consider only a single server, whereas we believe it to be a realistic model that multiple, competing, cloud service providers may be employed by a large set of clients, e.g. a company may wish to change providers without re-initialising the entire system, or may prefer certain servers for particular computational tasks (e.g. based on cost, latency to the current location of a mobile device, or the security classification of the computation in question). Thus, we extend the current schemes to allow multiple servers to be certified for multiple functions, and allow the distinguished client, as the authority on servers, to add new servers and to revoke misbehaving servers, preventing them from performing further computations if they are known not to be trustworthy. This chapter is based on a paper by Alderman et al. [5].

In Chapter 4, we consider the need for access control in PVC systems, building upon our revised system model from Chapter 2, where we consider multiple clients outsourcing computations to multiple servers, each of which can be certified to perform multiple functions. As a multi-user, distributed system operating over potentially sensitive data and results, we believe the cryptographic enforcement of access control policies is both a natural and necessary requirement. We discuss policies that limit:

- the operation of clients such that they can only outsource computations that they are authorised to perform locally (given the necessary resources) or to require clients to hold a valid subscription for the computation (based on the required resources to perform the computation);
- the set of computations a server may perform;
- the set of computational results a verifier may read (as such results may reveal sensitive information).

We discuss how these requirements can be expressed in terms of *graph-based information flow policies* and provide appropriate security models for their cryptographic enforcement. We also provide an example instantiation which uses a symmetric *key assignment scheme* (KAS) as the enforcement mechanism. As the distinguished setup entity is an authority on servers within the system, we again extend its duties to be authoritative on *all* entities in the system and to issue keys corresponding to appropriate access rights. Since the proofs of security for this scheme are quite similar in their line of argument, we include one exemplar proof in full in the main body of this thesis, and then include the remaining proofs in Appendix A to avoid repetition. The work presented in this chapter is based on a paper by Alderman et al. [3].

In Chapter 5, we extend the PVC construction to consider a reversed model of outsourced computation. Previously, we considered client devices that were data owners but lacked the resources to perform expensive computations on their data and so outsourced the relevant input data, along with a request for a particular computation, to an external server. In contrast, in Chapter 5, we consider servers that own some data and make it available for specific sets of public queries. We discuss this reversed system architecture and relevant security notions, and see that it has natural applications to verifiable queries on remote databases and verifiable parallel processing using the MapReduce framework.

This notion is similar in some regards to notions such as memory delegation [42]; in this thesis, we observe that the techniques used in the constructions of PVC can be adapted to this setting too.

Finally, in Chapter 6, we unify the work in prior chapters using the new notion of *hybrid PVC* (HPVC). An HPVC scheme requires only a single setup stage but provides a flexible outsourced computing solution. The notions of security for HPVC generally combine notions from the previous chapters with updated notation; as such, to avoid repetition, we defer these and their proofs to Appendix B.

We also show that HPVC provides a natural application for *dual-policy attribute-based encryption* (DP-ABE). In fact, we will require a new primitive which we call *revocable key dual-policy attribute-based encryption scheme* which combines DP-ABE with a revocation mechanism to disable decryption keys. We introduce and define this primitive in Section 6.3.1 but defer the construction and proof of security to Appendix B due to the length and the fact that the construction of HPVC can be followed only with knowledge of the definition. Chapters 5 and 6 are based on a paper by Alderman et al. [4].

To date, DP-ABE has not attracted much attention in the literature, which we believe to be primarily due to its applications being less obvious than for other forms of attribute-based encryption (particularly in regards to the cryptographic enforcement of access control policies). Thus, one outcome of this thesis is to show that DP-ABE can be a useful tool in other settings (namely in outsourced computation protocols as well as in cryptographically protected file systems).

Indeed, a theme throughout this work is to look at expanding the applications of primitives (both symmetric and asymmetric) primarily designed for cryptographic access control. Such primitives are generally only considered in a static, file-system environment where files are encrypted and users granted decryption keys that allow them to read only those files for which they are authorised. In this work, we show how such primitives are also of practical use in interactive (dynamic) protocols to protect messages sent between entities. Furthermore, we use these primitives not only to prove authorisation to perform specific operations within the system (in effect, using read access control to enable a form of execution access control), but also as a proof mechanism for the outcome of an outsourced computation of a Boolean function.

1.3 Author Contributions

The work presented in this thesis is derived from three published papers [3–5] which I co-authored with three others; Crampton and Cid acted in a supervisory role to Christian Janson and I, as PhD students. The contents are therefore derived (although in many cases edited) from collaborative work with Christian Janson, arising from joint discussions, proof-reading and iterative editing by both. This section is intended to indicate instances where I was the main author or had a significant contribution.

The idea, first arising in Chapter 3, to use labels to enable servers to perform multiple functions (without allowing them to use an evaluation key for an alternate function to cheat) was my own, as was the novel use of dual-policy attribute-based encryption to both enable computation and enforce access control in Chapter 6.

In Chapter 3, the security model and proof for public verifiability and vindictive managers were primarily developed by me, whilst the definition and construction of revocable PVC was developed during joint meetings. I was responsible for the formulation of access control policies in Chapter 4 with guidance from Jason Crampton. I also created the security model and proof for the notions of authorised outsourcing and weak input privacy.

The notion of verifiable delegable computation, and the resulting definition, in Chapter 5 arose from discussions with Christian Janson about whether ciphertext-policy attribute-based encryption could be used in a similar fashion to the key-policy variant used in previous chapters. I was the main author of the instantiation of VDC and the discussion of the system model in Section 5.3 (with input from Jason Crampton).

Finally, as mentioned, I had the idea to combine the notions of the prior chapters and to instantiate Hybrid PVC using dual-policy attribute-based encryption (DP-ABE). Thus, the definitions and instantiation are mainly my own work. The instantiation relies on a new primitive known as revocable DP-ABE; this was mainly developed by Christian Janson (and hence many of the details can be found in Appendix B.1 rather than the main thesis body), however I was heavily involved in discussions to formulate the definition and basic ideas, as well as in completing the details of the security proof.

Chapter 2

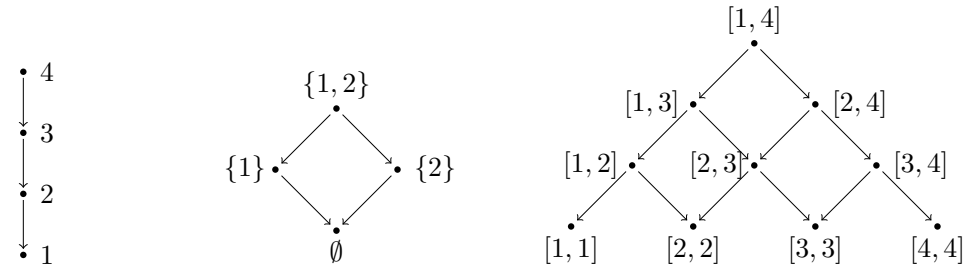
Background Material

Contents

This chapter introduces the notation, cryptographic primitives and security notions that will be used throughout the remainder of this thesis. We defer discussion of the security notions for each primitive to Section 2.8 where we introduce the format of the notions that shall be used throughout the thesis.

2.1 Notation

First we introduce some notation to be used in the remainder of this thesis. The set of integers is denoted $\mathbb{Z}$, the set of non-zero integers is denoted $\mathbb{Z}^\star = \mathbb{Z} \setminus \{0\}$, the set of reals is denoted $\mathbb{R}$, and we let the set of natural numbers be the set of non-negative integers, denoted $\mathbb{N} = \{z \in \mathbb{Z} : z \geq 0\}$. We write $[i, j]$, for integers $i \leqslant j$, to denote the set of consecutive integers $\{i, \ldots, j\}$, $[n]$ to denote the set $\{1, \ldots, n\}$ for $n \geqslant 1$, and $\emptyset$ to denote the empty set. When X is a set, we denote the *powerset* of X by 2^X which is the set of all subsets of X. A *partially ordered set*, or *poset*, is a set L equipped with a binary



(a) C_4 (b) $2^{[2]}$ (c) T_4

Figure 2.1: Example Hasse diagrams

relation $\leqslant$ such that for all $x, y, z \in L$ the following conditions hold: $x \leqslant x$ (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 $(L, \lessdot)$ wherein vertices are labelled by the elements of L and an edge connects vertex v to w if and only if $w \lessdot v$. We write C_n to denote the chain (total order) on n elements and T_n to denote the *temporal* poset $(\{[i, j] : 1 \leqslant i \leqslant j \leqslant n\}, \subseteq)$. Figure 2.1 shows Hasse diagrams for C_4, $2^{[2]}$ and T_4.

For the purposes of this thesis, a binary Boolean operator $O : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$ takes two bits as input and outputs a single bit result. We use the notation 1 and `true` (and similarly 0 and `false`) interchangeably to denote the outcome of such operations. We denote by $\vee$ the binary OR operator which returns 1 if at least one input is 1, and denote by $\wedge$ the binary AND operator which returns 1 if and only if *both* inputs is 1. A Boolean function $F : \{0, 1\}^n \to \{0, 1\}$ of *arity* n takes n bits as input and outputs a single bit result. Every Boolean function can be written as a *propositional formula* made from binary Boolean operators over n variables where the truth value of the result is determined by assigning truth values to each variable and evaluating each Boolean operator. Two such formulae are equivalent if and only if they express the same Boolean function (i.e. agree on the output of F for all possible inputs). If F evaluates to `true` on a set of input variables I, we say that I is a *satisfying input set*. F can also be completely described as a set of satisfying sets, which we shall term an *access structure*. We say a Boolean function is *monotone* if, for any satisfying input sets, changing any input values from `false` to `true` does not change the truth of the function. That is, adding additional `true` variables to a satisfying input set, or increasing the *Hamming weight* of the input, does not cause the set to become unsatisfying. We denote by $\mathcal{F}_n$ the family of n-ary monotone Boolean functions closed under complement — that is, if $F : \{0, 1\}^n \to \{0, 1\}$ belongs to $\mathcal{F}_n$ then

$\overline{F}$, where $\overline{F}(x) = F(x) \oplus 1$, also belongs to $\mathcal{F}_n$. When n is clear from the context, we shall refer to $\mathcal{F}_n$ simply as $\mathcal{F}$. A more formal mathematical treatment of Boolean functions can be found in [47]. We denote the domain of a function F by $\mathsf{Dom}(F)$ and the range by $\mathsf{Ran}(F)$. A function, $f : \mathbb{N} \to \mathbb{R}$, is *negligible* on its input if, for every non-constant real-valued polynomial $p(\cdot)$, there exists an N such that for all integers $n > N$, $f(n) < \frac{1}{p(n)}$. Throughout this thesis, we shall denote an arbitrary negligible function by $\mathrm{negl}(\cdot)$.

When specifying algorithms, we write $y \leftarrow x$ to denote assigning the value x to the variable y, and write $y \xleftarrow{\$} S$ to denote choosing an element from a finite set S uniformly at random and assigning it to the variable y. If A is a deterministic algorithm, we write $y \leftarrow A(x_1, \ldots\ x_n)$ for the action of running A on the inputs x_1 to x_n and assigning the result to an output y. Similarly, if A is a probabilistic algorithm, we write $y \xleftarrow{\$} A(\cdot)$ to denote the output of A being assigned to the variable y. This can be read as sampling an output y from the range of possible outputs of A according to a distribution defined by the input arguments — that is, for a particular input x, A may produce one of a set of outputs S_x, and we sample $y \xleftarrow{\$} S_x$. Equivalently, one can consider A taking an additional input R of random coins which determine the outcomes of probabilistic choices during the algorithm execution — in this case, we would write $y \leftarrow A(\ \cdot\ ; R)$ since the explicit choice of R renders A deterministic.

For a particular cryptographic scheme we denote the message space by $\mathcal{M}$, the keyspace by $\mathcal{K}$, the security parameter by $\ell \in \mathbb{Z}^\star$ and its unary representation as 1^ℓ (that is, a bitstring of length ℓ where all the bits are 1). Often the keyspace will be all bitstrings of length ℓ and ℓ can be increased to asymptotically strengthen the cryptographic primitive. Informally, a larger security parameter defines a larger keyspace and hence increases the cost of performing an exhaustive search of possible keys. We let ϵ denote the empty string (or an empty list where relevant), $\perp \notin \mathcal{M}$ denote a distinguished failure symbol output by an algorithm, and PPT denotes probabilistic polynomial-time.

2.1.1 Terminology for Binary Trees

A *tree* is a connected acyclic graph (i.e. a unique path exists between all pairs of vertices in the graph). We may produce a directed tree by orienting each edge to flow from a *parent* to a *child* vertex, and then consider the *in-degree* (resp. *out-degree*) of a vertex v

as the number of edges beginning (resp. ending) at v. A binary tree is a directed, rooted tree (i.e. a unique vertex exists with in-degree 0) where each vertex has at most 2 children (i.e. each vertex has out-degree at most 2) such that there exists a unique path from the root to each vertex. A leaf is a vertex with out-degree 0.

Let $\mathcal{L} = \{1, \ldots, n\}$ be the set of leaves of a tree. Let $\mathcal{X}$ be a set of labels labelling each vertex in the tree according to some ordering, and we can identity a vertex with its unique label.. For a leaf $i \in \mathcal{L}$, let $\mathsf{Path}(i) \subset \mathcal{X}$ be the set of labels for vertices on the (unique) path from the root to i (including i and the root).

For $R \subseteq \mathcal{L}$, let $\mathsf{Cover}(R) \subset \mathcal{X}$ be defined as follows [16]. First mark all all the vertices in $\mathsf{Path}(i)$ if $i \in R$. Then $\mathsf{Cover}(R)$ is the set of all unmarked children of marked vertices. It can be shown that $\mathsf{Cover}(R)$ is the minimal set that contains no vertex in $\mathsf{Path}(i)$ if $i \in R$ but contains at least one vertex in $\mathsf{Path}(i)$ if $i \notin R$.

2.2 Verifiable Outsourced Computation

Verifiable computation (VC) may be seen as a protocol between two polynomial-time parties, a *client*, C, and a *server*, S. A successful run of the protocol results in C being provided with a (correct) proof of the computation of $F(x)$ by the server for an input x supplied by the client. As mentioned in Chapter 1, the need for such protocols is motivated by resource constrained devices that need to perform computationally intensive tasks, and so employ a more powerful, yet potentially untrusted, computational server. In many settings, the client must be assured of the correctness of the computation before results can be used in further computations to ensure accidental or malicious errors have not been introduced. A malicious sever may wish to convince a client of incorrect results to affect future client behaviour or, if it believes incorrect results will not be detected, the server may try to avoid expending computational resources to perform the computation and instead return a random result.

Some solutions to this problem rely on *auditing* [21,79] (that is, the client will either employ multiple servers or recompute itself some portion of the computation) to verify correctness. However, such solutions may be infeasible for the client (which is assumed to have limited resources), require detailed knowledge of the server hardware [88] (which is impractical

in many cloud environments), or require redundantly employing multiple servers (thus increasing the cost) which are required not to collude. Other solutions [39] rely on the computational servers holding trusted platform modules [70] or other secure computing environments, which clearly restricts the cloud servers that may be employed and adds to the cost of employing them. Finally, some VC solutions use interactive proofs [61, 63, 75] or the more efficient probabilistic checkable proofs (PCPs) [11, 23] (where the verifier is able to check a proof in only a small number of places).

Non-interactive verifiable computation was introduced by Gennaro et al. [57]. Non-interactivity requires that there be 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.

Definition 2.1. *A* non-interactive VC scheme *to compute a single function F comprises four algorithms as follows:*

- $(EK_F, SK_F) \xleftarrow{\$} \mathsf{Setup}(1^\ell, F)$*: takes as input the unary representation of the security parameter and the function F to be computed, and outputs a public evaluation key EK_F which the server will use to compute F and a key SK_F which is kept secret by the client;*
- $(\sigma_{F,x}, VK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_F)$*: takes as input an input value x and the secret key SK_F and generates a public encoded input $\sigma_{F,x}$ and a verification key $VK_{F,x}$ which is kept secret by the client;*
- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, EK_F)$*: takes the encoded input $\sigma_{F,x}$ and the evaluation key for F and computes an output $\theta_{F(x)}$ encoding the result $F(x)$;*
- $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x}, SK_F)$*: takes the encoded output $\theta_{F(x)}$, the verification key $VK_{F,x}$ and the secret key SK_F as input and generates an output y which is either $F(x)$ if $\theta_{F(x)}$ is valid, or else a distinguished failure symbol $\perp$ if the result is incorrect.*

The operation of a VC scheme is illustrated in Figure 2.2. Note that Setup is performed once whilst the remaining algorithms may be performed many times. Setup is run by the client and may be computationally expensive but the remaining operations should be efficient for the client. In other words the cost of the setup phase (to the client)

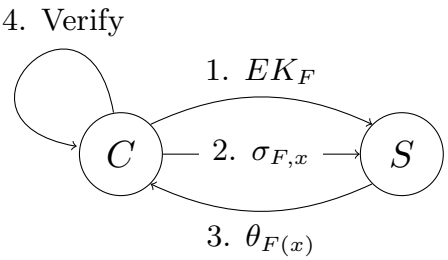


Figure 2.2: The operation of a verifiable computation scheme

is amortised over multiple computations of F. The cost of outsourcing a computation must, in particular, be cheaper than performing it locally. Different VC schemes are able to evaluate different families of functions, denoted $\mathcal{F}$ (e.g. Boolean circuits, arithmetic circuits etc.).

A VC scheme is *correct* if the client will always accept and recover the value $F(x)$ if all algorithms are run honestly. A more formal correctness definition follows.

Definition 2.2. *A verifiable computation scheme for a family of functions $\mathcal{F}$ is* correct *if for all functions $F \in \mathcal{F}$, for all inputs $x \in \mathsf{Dom}(F)$,*

$$\begin{aligned}\Pr[&(EK_F, SK_F) \xleftarrow{\$} \mathsf{Setup}(1^\ell, F),\\ &(\sigma_{F,x}, VK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_F),\\ &\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, EK_F),\\ &F(x) \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x}, SK_F)] = 1.\end{aligned}$$

A VC scheme is *secure* if a malicious server cannot convince a client of an incorrect result. This is defined more formally in Section 2.8.1.

A trivial solution to verifiable computation is to redundantly outsource the same computation to multiple servers and compare the results, taking the majority answer to be correct. Most VC schemes [9,18,19,26] attempt to improve on this solution to remove the redundancy, improve collusion resilience and to use only a single server per computation. Gennaro et al. [57] introduced a construction based on *Yao's garbled circuits* [97] which provides a "one-time" verifiable computation scheme allowing a client to outsource the evaluation of a function on a single input. It is, however, insecure if the circuit is reused on a different input; thus the cost of Setup cannot be amortised. Moreover, the cost

of generating a new garbled circuit is approximately equal to the cost of evaluating the function itself. The authors suggest using a fully homomorphic encryption scheme [59] to re-randomise the garbled circuit for multiple executions on different inputs, but this too is currently impractical.

In independent and parallel work to that described in Chapter 3, Carter et al. [34] introduced a more powerful third party to generate garbled circuits for such schemes. However, they required this entity to be online throughout the computation and modelled the system as a secure multi-party computation between the client, server and third-party. We do not believe this solution is practical in general since trusted third parties may not always be available to take an active part in computations. For example, VC schemes are sometimes motivated in the context of battlefield communications where soldiers deployed with a reasonably light-weight computing device gather data from their surroundings and send it to regional servers for analysis [57]. In this setting, the trusted third party could be physically located within a high security base or governmental building and soldiers may receive relevant keys before being deployed. However, it may not be feasible, or desirable, for a remote soldier to contact the headquarters and maintain a communication link with them for the duration of the computation. It seems a reasonable assumption that in a VC system there could be many available servers but only a single (or small number of) trusted third parties. The third party, then, could easily become a bottleneck in the system and limit the number of computations that can take place at any one time.

Some authors have considered the multi-client case in which the input data sent to the server is jointly formed by multiple clients, where notions such as input privacy become more important. Choi et al. [40] extended the garbled circuit approach [57] using a proxy oblivious-transfer primitive to achieve input privacy in a non-interactive scheme. Recent works [60, 60] allow multiple clients to provide input to a functional encryption algorithm.

Parno et al. [84] introduced another notion of VC termed *multi-function verifiable computation* to allow servers to apply multiple functions to a single input, as opposed to the above notions defined for a single function F. Thus, a client can encode an input once, yet request multiple functions to be evaluated upon it. This is clearly useful if the client data is likely to remain static and be used multiple times for different purposes.

Definition 2.3. *A* non-interactive, multi-function verifiable outsourced computation (MF-VC) scheme *consists of five algorithms as follows:*

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell)$*: takes as input the unary representation of the security parameter, and outputs some public parameters* PP *and a secret key* MK *which are independent of any function to be computed;*
- $(EK_F, SK_F) \xleftarrow{\$} \mathsf{KeyGen}(F, \mathrm{MK}, \mathrm{PP})$*: takes a function F to be computed, the master secret key* MK *and the public parameters* PP *and outputs an evaluation key EK_F and a secret verification key SK_F for F;*
- $(\sigma_x, VK_x) \xleftarrow{\$} \mathsf{ProbGen}(x, \mathrm{MK}, \mathrm{PP})$*: takes as input an input value x, the master secret key and the public parameters, and generates an encoded input σ_x and a secret verification key VK_x. Note that the inputs to this algorithm are independent of F;*
- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_x, EK_F, \mathrm{PP})$*: takes an encoded input σ_x for input x and both public values (the evaluation key for a function F and the public parameters), and computes an output $\theta_{F(x)}$ encoding the result $F(x)$;*
- $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_x, SK_F)$*: takes the encoded output $\theta_{F(x)}$, the secret, input specific verification key VK_x and the secret, function specific key SK_F as input, and generates an output y which is either $F(x)$ if $\theta_{F(x)}$ is valid, or else a distinguished failure symbol $\perp$ if the result is incorrect.*

Definition 2.4. *A non-interactive, multi-function verifiable outsourced computation (MF-VC) scheme for a family of functions $\mathcal{F}$ is* correct *if for all functions $F \in \mathcal{F}$, for all inputs $x \in \mathsf{Dom}(F)$,*

$$\begin{aligned}\Pr[&(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell),\\ &(EK_F, SK_F) \xleftarrow{\$} \mathsf{KeyGen}(F, \mathrm{MK}, \mathrm{PP}),\\ &(\sigma_x, VK_x) \xleftarrow{\$} \mathsf{ProbGen}(x, \mathrm{MK}, \mathrm{PP}),\\ &\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_x, EK_F, \mathrm{PP}),\\ &F(x) \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_x, SK_F)] = 1.\end{aligned}$$

Parno et al. [84] introduced *publicly verifiable computation* (PVC) which we will extend in this thesis. This notion extends the prior VC notion to include multiple clients. A single client C_1 computes EK_F, as well as publishing information PK_F that enables *any* other client to encode inputs, meaning that only one client has to run the expensive pre-processing stage. Each time a client submits an input x to the server, it may publish $VK_{F,x}$, which enables *any* other client to verify that the output is correct. Thus, in PVC,

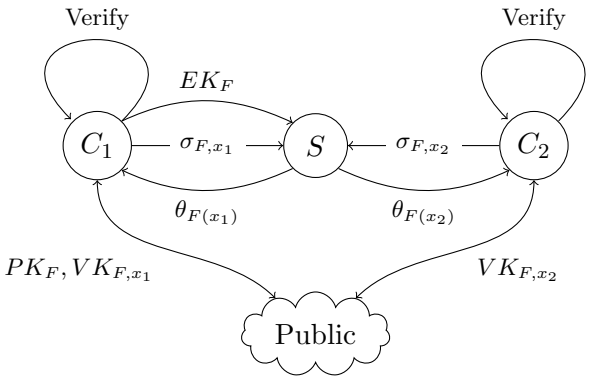


Figure 2.3: The operation of a publicly verifiable computation scheme

outsourcing and verifying computations rely only on public information, and Setup need only be run once per function rather than once per function per client; we say that PVC achieves *public delegation* and *public verification*.

Definition 2.5. *A* non-interactive publicly verifiable outsourced computation (PVC) scheme *to compute a single function F comprises four algorithms as follows:*

- $(EK_F, PK_F) \xleftarrow{\$} \mathsf{Setup}(1^\ell, F)$*: takes as input the unary representation of the security parameter and the function F to be computed, and outputs a public evaluation key EK_F which can be used to compute F and a public key PK_F which allows delegation of computations of F;*
- $(\sigma_{F,x}, VK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, PK_F)$*: takes as input an input value x and the public delegation key PK_F and generates a public encoded input σ_x and a public verification key VK_x;*
- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, EK_F)$*: takes the encoded input $\sigma_{F,x}$ and the evaluation key for F and computes an output $\theta_{F(x)}$ encoding the result $F(x)$;*
- $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x})$*: takes the encoded output $\theta_{F(x)}$ and the verification key $VK_{F,x}$ as input and generates an output y which is either $F(x)$ if $\theta_{F(x)}$ is valid, or else a distinguished failure symbol $\perp$ if the result is incorrect.*

The operation of a publicly verifiable outsourced computation scheme is illustrated in Figure 2.3 and can be compared to the standard version of VC illustrated in Figure 2.2. Note that in both this definition and Definition 2.1 for 'standard' verifiable computation, the Setup operation is run by the client for a single function F and can be expensive

(approximately the cost of executing the function F itself). Given that the client is assumed to have restricted resources (to motivate the use of VC in the first place) and that a client is likely to be interested in computing multiple functions, this definition is somewhat restrictive. The primary differences between the two definitions are that the secret key SK_F generated in Setup for a VC scheme is published in a PVC scheme to allow any entity to outsource a computation of F, and that the verification key $VK_{F,x}$ is also published as a result of $\mathsf{ProbGen}$ to allow any entity to verify a result.

A PVC scheme is *correct* if, as with VC schemes, a honest run of the protocol will always return the correct value of $F(x)$.

Definition 2.6. *A publicly verifiable computation scheme for a family of functions $\mathcal{F}$ is* correct *if for all functions $F \in \mathcal{F}$, for all inputs $x \in \mathsf{Dom}(F)$,*

$$\begin{aligned}\Pr[&(EK_F, PK_F) \xleftarrow{\$} \mathsf{Setup}(1^\ell, F),\\ &(\sigma_{F,x}, VK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, PK_F),\\ &\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, EK_F),\\ &F(x) \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x})] = 1.\end{aligned}$$

Parno et al. motivate PVC in the setting of a scientific lab where the lead researcher determines the complex data analysis function that should be computed whilst a team of researchers decide the inputs for each trial and verify results. Results could also be verified by interested third parties, e.g. patients or funding bodies.

Public verification leads to a stronger notion of security since any entity, including the computational server, can perform verification. Thus, the server itself is able to tell whether a result will be accepted or rejected by a client. In some constructions of non-publicly verifiable VC (meeting Definition 2.1), this knowledge could lead to an attack (known as 'the rejection problem') — knowledge of whether previous proofs were accepted could reveal information about secret verification keys that would aid future forgeries [62].

Some prior work also allows public delegation [30, 61, 62] but are mostly in the random oracle model or rely on non-standard assumptions; public verifiability does not seem to have been considered prior to the work of Parno et al. (with the exception of a scheme limited to set operations [82]), but has been since [53, 83]. Parno et al. mentioned that

a VC scheme could be multi-function, publicly verifiable or both, but left it as an open problem to find a multi-function PVC scheme.

2.3 Access Control Policies

Throughout this thesis we will make extensive use of cryptographic primitives primarily designed to enforce access control policies. As mentioned in Chapter 1, one theme of this thesis is to consider the use of such primitives in new settings and to protect messages transmitted during an interactive protocol (rather than static documents in a file system). In this section, we briefly review current access control policies that are of practical interest which will inform both our choice of policy representation in Chapter 4 and the description of attribute-based encryption later in this background chapter. More detailed general introductions to access control can be found in [28, 87].

Let U be a set of users or entities within a system, O be a set of objects or resources that should be protected from unauthorised access, and A be a set of actions that may be performed by a user on an object. An *access control request* is a tuple of the form (u, o, a) representing the user $u \in U$ requesting to perform action $a \in A$ on the object $o \in O$. An *access control policy* dictates whether such a request should be considered *authorised* or *unauthorised*. A request is generally made to some form of *policy enforcement point* (PEP) that employs an access control mechanism to respond to requests. When the access control mechanism is some form of cryptographic primitive, it is usual to restrict A to be the `read` action, and then objects can be identified with performing the `read` action on the object, and requests become a pair (u, o).

Early access control policies were formulated directly on the members of the sets U and O. For example, *access control lists* [28, 87] are defined over objects in O and define for each $o \in O$, a set of authorised users $U_o \subseteq U$. Any user u within this authorised set may read o — that is, a request (u, o) is deemed authorised if and only if $u \in U_o$. Similarly, *capability lists* [28, 87] are defined over users in U. For each user $u \in U$, a capability list O_u is defined as a set of objects that u may read — $O_u \subseteq O$. A request (u, o) is authorised if and only if $o \in O_u$.

More recent forms of access control policies have attempted to be more flexible and fine-

grained by introducing levels of indirection between users and objects in the form of *attributes*. The set of attributes is called the *attribute universe*, $\mathcal{U}$.

2.3.1 Information Flow Policies

One example of such attribute-based policies are *information flow policies* [28] where the set of attributes is an ordered set of security labels, one of which is associated to every user and object. Let $\lambda : U \cup O \to \mathcal{U}$, where $\mathcal{U}$ is the set of labels, be a labelling function assigning attributes to users and objects. The access control policy then states that a request (u, o) is authorised if and only if $\lambda(u) \geqslant \lambda(o)$ according to the ordering relation on labels.

Example 2.1. *An example information flow policy is the Bell-LaPadula model [22] for multi-level security policies. Consider a totally ordered set (chain) of security labels* {`Unclassified` $<$ `Classified` $<$ `Secret` $<$ `Top Secret`}. *A user with label* `Secret` *is permitted to access (read) all objects associated with the labels* `Secret`, `Classified` *and* `Unclassified`, *but* not *any objects labelled as* `Top Secret`.

2.3.2 Role-based Access Control Policies

Another example of policies being defined over attributes is *role-based access control (RBAC) policies* where the set of attributes is now a set of *roles*, $\mathcal{U}$. Users are permitted to act in certain roles, and objects may be acted upon only by specified roles. In *core RBAC* (or RBAC_0) [6, 86], an access control policy is defined by a user-role assignment relation, $UA \subseteq U \times \mathcal{U}$, and an object-role assignment relation, $OA \subseteq O \times \mathcal{U}$. Note that RBAC policies generally define UA and PA, where $PA \subseteq P \times \mathcal{U}$ is a permission-role assignment relation, and a permission is defined to be an object-action pair. Since we consider only `read` actions, we can associate permissions with objects and define OA as above. A request (u, o) is authorised if and only if there exists a role $r \in \mathcal{U}$ such that $(u, r) \in UA$ and $(o, r) \in OA$.

Equivalently, define a labelling function $\lambda : U \cup O \to 2^{\mathcal{U}}$. For users $u \in U$ and objects $o \in O$, let $\lambda(u) = \{r_1, \ldots, r_n\} \in 2^{\mathcal{U}}$ and $\lambda(o) = \{r'_1, \ldots, r'_m\} \in 2^{\mathcal{U}}$. Then a request (u, o) is authorised if and only if there exists a role $r \in \mathcal{U}$ such that $r \in \lambda(u)$ and $r \in \lambda(o)$ i.e.

$\lambda(u) \cap \lambda(o) \neq \emptyset$.[1] Further, as with information flow policies, we may define a (partial) order relation over $\mathcal{U}$, resulting in *hierarchical RBAC* (or RBAC_1) [6,86]. A request (u, o) is authorised if and only if there exists roles $r, r' \in \mathcal{U}$ such that $(u, r) \in UA$, $r \geqslant r'$, and $(o, r') \in OA$.

Definition 2.7. *Let $(L, \leqslant)$ be a poset and $X \subseteq L$. The* downset *of X is $\downarrow X = \{l \in L : \exists x \in X \text{ such that } l \leqslant x\}$. Similarly, the* upset *of X is $\uparrow X = \{l \in L : \exists x \in X \text{ such that } x \leqslant l\}$. For singleton sets, we denote $\downarrow\{x\}$ as $\downarrow x$ and $\uparrow\{x\}$ as $\uparrow x$.*

Given these definitions, one can again write $\lambda : U \cup O \to 2^{\mathcal{U}}$ where $(2^{\mathcal{U}}, \leqslant)$ is a partial order, and say that a request (u, o) is authorised if and only if $\downarrow\lambda(u) \cap \uparrow\lambda(o) \neq \emptyset$.

2.3.3 Attribute-based Access Control Policies

A more general formulation of access control policies defined in terms of attributes is known as *attribute-based access control* (ABAC) [68], as used in the XACML 3.0 access control markup language [95] for example. This is motivated by the observation that in many large distributed systems, the user and object populations cannot feasibly be described explicitly in terms of individual identifiers (as in access control lists for example). Generally, ABAC schemes associate either the user population or the object population with policies and the other with attribute sets. When policies are associated with objects, we refer to them as *object-centric* and when policies are associated to users we term them *user-centric*. The attributes sets are also generally *unordered* (in contrast to information flow policies for example), and policies are usually satisfied based on subset inclusion — as we shall describe below, a policy can take the form of an *access structure* which comprises a collection of satisfying attribute sets; a set of attributes should be a subset of the access structure in order for access to be granted. As such, we usually consider only *monotonic* policies in this setting, where any superset of an authorised set of attributes is also authorised — granting additional attributes does not revoke access.

Example 2.2. *Consider a university system with attribute universe* $\mathcal{U} = \{$`Professor, Admin, Student, Mathematics, Computer Science, Information Security`$\}$. *The resource space, O, is all publications, lecture notes and assignments, while the user space, U, is all students and staff. Both of these spaces could be too large, or change too frequently,*

[1] Notice that users and objects can be assigned to multiple roles.

to formulate policies over individual identities. Thus, policies are defined as Boolean formulas over $\mathcal{U}$*, e.g. the policy* `Admin` $\wedge$ `(Mathematics` $\vee$ `Computer Science)` *defines authorised users to be administrative staff in the Mathematics or Computer Science groups.*

A policy can be represented in many forms (e.g. access trees, monotone span programs or Boolean formulae). In this thesis, we will primarily consider policies represented as monotone Boolean formulae $F : 2^{\mathcal{U}} \to \{0, 1\}$, which is a very natural way to describe authorised sets of users in terms of logical AND and OR gates.

Any policy (represented as a Boolean formula F defined over a subset of an attribute universe $\mathcal{U}$) can be written as a set of *satisfying sets* — that is, the satisfying sets $\mathbb{A} = \{S_i\}$ where $S_i \subseteq \mathcal{U}$ for all i where $F(S_i) = 1$ (i.e. the policy is satisfied by inputs S_i) for all $S_i \in \mathbb{A}$, and $F(S_j) = 0$ for all $S_j \notin \mathbb{A}$. This says that the formula F is satisfied by the inputs S_1 or S_2 or ... or $S_{|\mathbb{A}|}$ i.e. it has been written in disjunctive normal form. A non-empty collection of satisfying sets, $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \{\emptyset\}$, is called an *access structure* [20, Definition 3.5]. Note that, equivalently, $\mathbb{A}$ can be said to be an element of the set $2^{2^{\mathcal{U}}} \setminus \{\emptyset\}$.

$\mathbb{A}$ is monotone if $S' \in \mathbb{A}$ whenever $S \in \mathbb{A}$ and $S \subseteq S'$ i.e. all supersets of satisfying sets are also satisfying. Equivalently, one may say that a monotone access structure is an order filter in $(2^{\mathcal{U}} \setminus \{\emptyset\}, \subseteq)$ [49, Definition 1.27]. If $\mathcal{U}$ is finite then an access structure can be uniquely defined by the set of minimal elements (those S_i that are not supersets of any other $S_j \in \mathbb{A}$). These minimal elements may be viewed as an antichain in $(2^{\mathcal{U}} \setminus \{\emptyset\}, \subseteq)$.

Example 2.3. *In Example 2.2, the minimal elements of the access structure are*

$$\mathbb{A} = \{\{\texttt{Admin}, \texttt{Mathematics}\}, \{\texttt{Admin}, \texttt{ComputerScience}\}\}.$$

We may associate each user with a set of descriptive attributes. Each object (or resource to be protected) may be associated with a policy, effectively specifying the attribute sets (and hence the users assigned such attribute sets) that should be considered authorised to access the object. In other words, for such "object-centric" policies, we can define a labelling function $\lambda : U \cup O \to 2^{2^{\mathcal{U}}} \setminus \{\emptyset\}$. Then, each user u is assigned a label $\lambda(u) = \{A\}$ for some attribute set $A \subseteq \mathcal{U} \setminus \{\emptyset\}$. Each object o is assigned a policy, or access structure,

$\lambda(o) = \mathbb{A} = \{S_1, \ldots, S_n\} \subseteq 2^{2^{\mathcal{U}}} \setminus \{\emptyset\}$. An access control request (u, o) is authorised if and only if $\lambda(u) \in \lambda(o)$ (if $\mathbb{A}$ is the full access structure) or if and only if $A \cap S_i = S_i$ for some $1 \leqslant i \leqslant n$ (if $\mathbb{A}$ is represented by its minimal elements).

Similarly, we can define "user-centric" policies wherein objects are associated with a descriptive attribute set (describing their contents and classification levels) and users are associated with policies (describing which sets of attributes, and therefore associated objects, they should be allowed to access). Such policies can be formulated in terms of a labelling function as above with the roles of the user and object reversed.

2.3.4 General Representation of Access Control Policies.

We have seen that, by using an appropriate labelling function, RBAC policies can be written in the same format as information flow policies — that is, access is granted if and only if $\lambda(u) \cap \lambda(o) \neq \emptyset$. It is trivial to see that access control lists and capability lists fit this format. Crampton [44] has also shown that by redefining the poset order relation appropriately, ABAC policies can also be written in this form. Thus, assuming an appropriate order relation and labelling function, we have a simple way to describe many forms of flexible access control policy in terms of information flow policies. Additionally, efficient, symmetric cryptographic enforcement mechanisms for information flow policies exist in the form of key assignment schemes, which we introduce in Section 2.4. For this reason, when discussing access control policies in the setting of PVC in Chapter 4, we focus our attention on information flow policies only.

2.4 Key Assignment Schemes

Let $(L, \leqslant, U, O, \lambda)$ denote an information flow policy (see Section 2.3) on a set of users U and a set of objects O with a labelling function λ assigning labels from the poset $(L, \leqslant)$. The policy can be represented by the Hasse diagram $H(L, \leqslant)$. Henceforth we may refer to such policies as *graph-based access control policies*. The policy requires that information flow from objects to users preserves the partial ordering relation; a user $u \in U$ may read an object $o \in O$ if and only if $\lambda(u) \geqslant \lambda(o)$. Equivalently, there must exist a directed path from $\lambda(u)$ to $\lambda(o)$ in the Hasse diagram. Note that this statement is the *simple security property*

of the Bell-LaPadula security model [22]. The enforcement of a graph-based access control policy prevents an entity assigned clearance label x from accessing objects classified with label y if $x \not\geqslant y$. Posets of the form shown in Figure 2.1 have been used extensively as the basis for graph-based access control policies, notably in the Bell-LaPadula model [22] (Figure 2.1a) and in temporal access control [45] (Figure 2.1c).

A *key assignment scheme* provides a generic, cryptographic enforcement mechanism for graph-based access control policies in which a unique cryptographic key is associated to each node (representing a security label) in the graph $H(L, \leqslant)$. Akl and Taylor [1] introduced the idea of a KAS to manage the problem of key distribution by allowing a trusted setup authority to distribute a single cryptographic key, $\kappa(x)$, to each entity assigned label $x \in L$. The entity may then combine knowledge of this secret key with some publicly available information to derive all additional keys $\kappa(y)$ that he is authorised to hold. Henceforth, we write κ_x to represent the cryptographic key $\kappa(x)$.

Definition 2.8. *A* key assignment scheme (KAS) *for an information flow policy over the poset $(L, \leqslant)$ is defined by the following four algorithms [46]:*

- $\kappa_L \xleftarrow{\$} \mathsf{MakeKeys}(1^\ell, (L, \leqslant))$: *takes the security parameter and the label poset as input and returns a labelled set of cryptographic keys $\{\kappa_x : x \in L\}$, denoted κ_L;*
- $\omega_L \xleftarrow{\$} \mathsf{MakeSecrets}(1^\ell, (L, \leqslant))$: *takes as input the unary representation of the security parameter and the label poset and returns a labelled set of secret values $\{\omega_x : x \in L\}$, denoted ω_L;*
- $Pub_{(L,\leqslant)} \xleftarrow{\$} \mathsf{MakePublicData}(1^\ell, (L, \leqslant))$: *takes the security parameter and the label poset as input and returns a set of data $Pub_{(L,\leqslant)}$ that is published by the trusted setup authority to aid key derivation;*
- $K \leftarrow \mathsf{GetKey}(x, y, \omega_x, Pub_{(L,\leqslant)})$*: takes as input $x, y \in L$, the secret information ω_x for x, and the public information, $Pub_{(L,\leqslant)}$. The algorithm returns a value K which is either κ_y if $y \leqslant x$, or a distinguished failure symbol $\perp$ otherwise.*

Definition 2.9. *A key assignment scheme is* correct *if for all posets $(L, \leqslant)$, for all $x \in L$*

and for all $y \leqslant x$,

$$\begin{aligned}\Pr[\kappa_L &\xleftarrow{\$} \mathsf{MakeKeys}(1^\ell, (L, \leqslant)),\\ \omega_L &\xleftarrow{\$} \mathsf{MakeSecrets}(1^\ell, (L, \leqslant)),\\ Pub_{(L,\leqslant)} &\xleftarrow{\$} \mathsf{MakePublicData}(1^\ell, (L, \leqslant)),\\ \kappa_y &\leftarrow \mathsf{GetKey}(x, y, \omega_x, Pub_{(L,\leqslant)})] = 1.\end{aligned}$$

A well-known KAS construction, known as *iterative key encrypting* (IKE) KAS [46], publishes encrypted keys. In particular, for each directed edge (x, y) in the Hasse diagram, $\mathsf{Enc}(\kappa_y, \kappa_x)$ (i.e. the encryption of κ_y under κ_x) is published. Then for any $x > y$, there is a (directed) path in the Hasse diagram from x to y and the key associated with each node on that path can be derived (in an iterative fashion) by an entity that knows κ_x.

Specific choices of the poset of security labels give rise to the information flow policies made famous in the Bell-LaPadula model [22] and temporal access control policies, wherein an entity is permitted to derive cryptographic keys referring to specific intervals. A survey of existing generic schemes is given in [46] whilst further details of temporal access control and more general interval-based schemes can be found in [45] and [13]. Notions of security for KASs are discussed in Section 2.8.

2.5 Encryption Schemes

Encryption is a technique primarily for preserving the confidentiality of messages exchanged between a sender and receiver (or writer and reader respectively). As such, it is often a key component of cryptographic enforcement mechanisms for access control policies to prevent unauthorised entities from reading the contents of protected objects. In this section, we review several forms of encryption that we will make use of, or build from, later in the thesis.

2.5.1 Symmetric Encryption Schemes

Symmetric encryption [69] relies on a secret key k held by both the sender and recipient (or writer and reader) of a ciphertext.

Definition 2.10. *A* symmetric encryption scheme *comprises three algorithms:*

- $k \xleftarrow{\$} \mathsf{KeyGen}(1^\ell)$*: takes the security parameter as input and randomly selects a symmetric key k from $\mathcal{K}$;*
- $CT \xleftarrow{\$} \mathsf{Encrypt}(m, k)$*: takes the symmetric key k and a message m from $\mathcal{M}$ as input and outputs a ciphertext CT;*
- $m \leftarrow \mathsf{Decrypt}(CT, k)$*: takes the ciphertext CT and the symmetric key k as input and returns the message m encrypted in CT. Note that the decryption algorithm is deterministic.*

Correctness of a symmetric encryption scheme requires that for all security parameters and all messages, decryption of an honestly ciphertext using an honestly generated key will return the correct message. More formally,

Definition 2.11. *A symmetric encryption scheme is* correct *if for all security parameters ℓ and all $m \in \mathcal{M}$,*

$$\begin{aligned}\Pr[&k \xleftarrow{\$} \mathsf{KeyGen}(1^\ell),\\ &CT \xleftarrow{\$} \mathsf{Encrypt}(m, k),\\ &m \leftarrow \mathsf{Decrypt}(CT, k)] = 1.\end{aligned}$$

A *symmetric authenticated encryption scheme* is syntactically almost identical to Definition 2.10, but considers integrity as well as privacy of messages. That is, it is possible to detect if a ciphertext has been modified since encryption (e.g. by a malicious eavesdropper or through a faulty/noisy transmission medium). Authenticated encryption therefore assures the *integrity* of received messages, and also provides data origin authentication (i.e. assurance that a particular entity generated the ciphertext) since it should not be possible for anyone other than the alleged encryptor to create a valid ciphertext, else an

adversary could similarly create modified ciphertexts. The only change to Definition 2.10 is that Decrypt returns an element PT which is either the correct plaintext message m or a distinguished symbol $\perp$ if the ciphertext is deemed 'inauthentic'.

2.5.2 Asymmetric Encryption Schemes

Asymmetric, or *public key*, encryption (PKE) [71] removes the requirement (from symmetric encryption) for both the encryptor and decryptor to share the same key. Instead, each entity A is associated with two keys: a public key used by an encryptor to send a message to A; and a private key used by A to decrypt messages encrypted using A's public key. Thus, the sender is not required to know the decryption key. Additionally, the public key may be transmitted or published in the clear so there is no requirement for confidential channels before transmitting a message (although, distribution channels for the public key must still be authenticated).

Aside from easing key distribution, PKE also leads to increased functionality as it is possible to write objects (encrypt) without being authorised to read (decrypt) such objects. It also allows a recipient to receive messages from multiple senders whilst keeping only a single secret decryption key. However, in general, public key cryptosystems are significantly slower than symmetric key schemes and, as a result, hybrid encryption is widely adopted. In hybrid encryption, an object is (efficiently) encrypted using a symmetric encryption scheme (this is termed the *data encapsulation mechanism*) and the symmetric key used for object encryption is itself encrypted using a public key scheme (termed a *key encapsulation mechanism*). Thus, the slower public key scheme is only used to encrypt a reasonably short symmetric key, whilst the more efficient symmetric key scheme is used to encrypt the larger object.

Public key schemes are defined in a very similar way to symmetric schemes, but the key generation algorithm outputs two keys: a public key and a private key. Correctness is defined analogously and must hold with all but negligible probability.

2.6 Attribute-based Encryption

Attribute-based encryption (ABE) is a public key primitive that allows the decryption of a ciphertext if and only if the encrypted data and decrypting entity satisfy certain properties defined in terms of attribute sets. As such, it is well-suited to the cryptographic enforcement of attribute-based access control policies (see Section 2.3.3). We define a universe $\mathcal{U}$ of "attributes" which are labels that may describe data or entities. We then form a set of attributes $A \in 2^{\mathcal{U}}$ and a policy or access structure $\mathbb{A} \in 2^{2^{\mathcal{U}}} \setminus \{\{\emptyset\}\}$. Decryption succeeds if and only if $\{A\} \in \mathbb{A}$. Recall the example access structure in Examples 2.2 and 2.3.

There exist several variants of ABE that vary in the association of attribute sets and access structures to ciphertexts and decryption keys. In *key-policy ABE* (KP-ABE) [66], a policy is associated with the decryption key and a set of attributes is associated with each ciphertext. Conversely, in *ciphertext-policy ABE* (CP-ABE) [27], a policy is associated to a ciphertext and decryption keys are associated with sets of attributes. Finally, in *dual-policy ABE* (DP-ABE) [15], both ciphertexts and decryption keys are associated with both a policy and an attribute set, and the key attributes must satisfy the ciphertext policy and vice versa.

2.6.1 Key-policy Attribute-based Encryption

In KP-ABE [66], each private key is associated with a family of satisfying attribute sets $\mathbb{A} = \{A_1, \ldots, A_n\}$, while each ciphertext is computed using a single, system-wide public key and associated with a single subset of attributes A. Decryption succeeds if the access structure $\mathbb{A}$ associated with the private key includes the attribute set under which the message was encrypted: that is $A_i = A$ for some $i \in [n]$. Since the access structure describes objects that the key may be used to access, such policies are referred to as *objective*. In most schemes, the access structures considered are *monotonic*, meaning $A' \in \mathbb{A}$ whenever there exists $A \subset A'$ such that $A \in \mathbb{A}$. A notable non-monotonic scheme was given by Ostrovsky et al. [80].

Definition 2.12. *A* key-policy attribute-based encryption scheme *comprises the following algorithms:*

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell, \mathcal{U})$*: takes the unary representation of the security parameter* ℓ *and the attribute universe* $\mathcal{U}$ *as input and outputs public parameters* PP *and a master key* MK*;*
- $CT \xleftarrow{\$} \mathsf{ABE.Encrypt}(m, A, \mathrm{PP})$*: takes as input a message* m*, a set of attributes* A *and the public parameters* PP*, and outputs a ciphertext* CT*;*
- $SK_{\mathbb{A}} \xleftarrow{\$} \mathsf{ABE.KeyGen}(\mathbb{A}, \mathrm{MK}, \mathrm{PP})$*: takes as input an access structure* $\mathbb{A}$*, the master key* MK *and the public parameters* PP*, and outputs a private decryption key* $SK_{\mathbb{A}}$ *for this access structure;*
- $PT \leftarrow \mathsf{ABE.Decrypt}(CT, SK_{\mathbb{A}}, \mathrm{PP})$*: takes as input a ciphertext* CT *of a message* m *associated with a set of attributes* A*, a decryption key* $SK_{\mathbb{A}}$ *and the public parameters. It outputs a plaintext* PT *which is either the message* m *encrypted in* CT*, if* $A \in \mathbb{A}$ *(i.e. if* A *is a satisfying set of* $\mathbb{A}$*), or a distinguished failure symbol,* $\perp$*, otherwise (i.e.* $\perp$ *is returned if the policy is unsatisfied).*

We do not give the correctness or security properties in this background section as we will be interested in using a revocable extension of KP-ABE, which we introduce in Section 3.2.2. The reader is referred to the cited prior literature for more details.

2.6.2 Ciphertext-policy Attribute-based Encryption

In CP-ABE [27], attributes are used to describe users and each ciphertext is associated with an access structure. In this regard, CP-ABE may be more closely related than KP-ABE to traditional access control policies such as access control lists and RBAC that describe authorised entities for each protected object. The policies in CP-ABE are said to be *subjective*.

Definition 2.13. *A* ciphertext-policy attribute-based encryption scheme *comprises four algorithms as follows:*

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{U})$*: takes the security parameter and the attribute universe* $\mathcal{U}$ *as input and generates public parameters* PP *and a master secret key* MK*;*
- $CT \xleftarrow{\$} \mathsf{Encrypt}(m, \mathbb{A}, \mathrm{PP})$*: takes a plaintext* m*, an access structure* $\mathbb{A}$ *and the public parameters and outputs a ciphertext* CT*;*

- $SK_A \xleftarrow{\$} \mathsf{KeyGen}(A, \mathrm{MK}, \mathrm{PP})$*: takes as input an attribute set* A*, the master secret key and the public parameters, and outputs a secret decryption key* SK_A *for this attribute set;*

- $PT \leftarrow \mathsf{Decrypt}(CT, SK_A, \mathrm{PP})$*: takes a ciphertext for an access structure* $\mathbb{A}$*, a secret key for an attribute set* A *and the public parameters as input, and outputs a plaintext* PT *which is* m *if and only if the attribute set* A *satisfies the access structure* $\mathbb{A}$ *(that is,* $A \in \mathbb{A}$*), and the distinguished failure symbol* $\perp$ *otherwise.*

Definition 2.14. *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* $A \in \mathbb{A}$*,*

$$\begin{aligned}
&\Pr[(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell),\\
&\quad SK_A \xleftarrow{\$} \mathsf{ABE.KeyGen}(A, \mathbb{A}, \mathrm{MK}, \mathrm{PP}),\\
&\quad CT \xleftarrow{\$} \mathsf{ABE.Encrypt}(m, \mathbb{A}, \mathrm{PP}),\\
&\quad m \leftarrow \mathsf{ABE.Decrypt}(CT, SK_A, \mathrm{PP})]\\
&\quad = 1 - \mathrm{negl}(\ell).
\end{aligned}$$

2.6.3 Dual-Policy Attribute-Based Encryption

Recall that KP-ABE enforces objective policies whilst CP-ABE enforces subjective policies. *Dual-policy attribute-based encryption* (DP-ABE), introduced by Attrapadung and Imai [16], combines these approaches such that both the ciphertext and the decryption key comprise an attribute set *and* an access policy. Thus, the ciphertext is associated with both a subjective access policy (as per CP-ABE) detailing which entities may decrypt it *and* an objective attribute set describing the data. Decryption keys comprise an objective access policy (as per KP-ABE) and a subjective attribute set. Decryption succeeds if and only if *both* attribute sets satisfy their corresponding access policies.

Definition 2.15. *A* dual-policy attribute-based encryption scheme *comprises four algorithms as follows:*

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{U})$*: takes the security parameter and attribute universe as input and generates public parameters* PP *for the system, and a master secret key* MK *which is kept by the executor of this algorithm;*

- $CT_{\omega,\mathbb{S}} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), \mathrm{PP})$*: takes a message* m *to be encrypted, an objective attribute set* ω*, a subjective access policy* $\mathbb{S}$ *and the public parameters* PP*, as input. The algorithm outputs a ciphertext* $CT_{\omega,\mathbb{S}}$*;*

- $SK_{\mathbb{O},\psi} \xleftarrow{\$} \mathsf{KeyGen}((\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PP})$*: takes as input an objective access policy* $\mathbb{O}$ *and a subjective attribute set* ψ*, as well as the master secret key* MK *and the public parameters* PP*. The algorithm outputs a secret decryption key* $SK_{\mathbb{O},\psi}$*;*

- $PT \leftarrow \mathsf{Decrypt}(CT_{\omega,\mathbb{S}}, SK_{\mathbb{O},\psi}, \mathrm{PP})$*: takes as input a ciphertext* $CT_{\omega,\mathbb{S}}$ *and a secret key* $SK_{\mathbb{O},\psi}$ *as well as the public parameters* PP*. The algorithm outputs a plaintext* PT *which is the correct plaintext* m *if and only if the set of objective attributes* ω *satisfies the objective access structure* $\mathbb{O}$ and *the set of subjective attributes* ψ *satisfies the subjective policy* $\mathbb{S}$ *— that is,* $\omega \in \mathbb{O}$ *and* $\psi \in \mathbb{S}$*. If not,* PT *is defined to be a distinguished failure symbol* $\perp$*. We assume throughout that the policies and attributes are implicit from the relevant keys and ciphertexts (otherwise these can also be given as arguments to this function).*

Definition 2.16. *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* $\psi \in \mathbb{S}$*,*

$$\begin{aligned}
&\Pr[(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{U}),\\
&\quad SK_{\mathbb{O},\psi} \xleftarrow{\$} \mathsf{KeyGen}((\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PP}),\\
&\quad CT_{\omega,\mathbb{S}} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), \mathrm{PP}),\\
&\quad m \leftarrow \mathsf{Decrypt}(CT_{\omega,\mathbb{S}}, SK_{\mathbb{O},\psi}, \mathrm{PP})]\\
&\quad = 1 - \mathrm{negl}(\ell).
\end{aligned}$$

2.6.4 Instantiating Attribute-based Encryption Schemes

Many ABE schemes choose a secret value uniformly at random during the encryption or key generation algorithms and use a *linear secret sharing scheme* to split this secret over a set of attributes or clauses in a policy. They then use *Lagrange interpolation* to reconstruct the secret if and only if a satisfying set of attributes are provided to the decryption algorithm. Additionally, many schemes are built using bilinear pairings which give rise to hardness assumptions based on the Diffie-Hellman problem in bilinear groups.

In Appendix B.1 we shall construct a new revocable DP-ABE primitive. In this section, we introduce some additional preliminary notions that we shall require to do so. We first provide more details regarding secret sharing schemes and Lagrange interpolation, as well as bilinear maps and the associated hardness problems we shall use to prove security of our primitive. We end by discussing some terminology for binary trees which we shall use to determine the appropriate update material to issue such that revoked user keys are rendered useless.

2.6.4.1 Linear Secret Sharing Schemes

Secret sharing is a fundamental cryptographic tool that enables a secret value s to be divided amongst a set of entities in such a way that all *authorised* sets of entities may combine their individual shares to reconstruct the value of s, for example, any k out of the n entities may form an authorised set. Any set of entities that does not form an authorised set learns nothing more than their respective shares, and cannot reconstruct s. A secret sharing scheme is *linear* if the reconstruction operation is a linear function of the shares; almost all known secret sharing schemes are linear [20].

Recall from Section 2.3.3, the definition of an *access structure* as a collection of satisfying sets of a Boolean formula. Equivalently, an access structure can be generically defined as follows [20].

Definition 2.17. *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, if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An* access structure *(resp., monotonic access structure) is a collection (resp., 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 the* unauthorised sets*.*

A linear secret sharing scheme can then be defined as follows [92].

Definition 2.18. *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 a monotone 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:*

- $M\boldsymbol{v} \xleftarrow{\$} \mathsf{Share}(s, (M, \pi))$*: takes as input* $s \in \mathbb{Z}_p$ *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 the* i^{th} *row in* M *by* $\boldsymbol{M_i}$*;*

- $\{(i, \mu_i)\}_{i \in I} \leftarrow \mathsf{Recon}S, \{\lambda_{\pi(i)}\}_{\pi(i) \in S}, (M, \pi)$*: takes as input an authorised set* $S \in \mathbb{A}$*, the set of shares for this set, 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* $s = \sum_{i \in I} \mu_i \cdot \lambda_{\pi(i)}$*. The set* $\{\mu_i\}_{i \in I}$ *can be found in polynomial time in the size of* M *[20, 92].*

In Appendix B.1,we will require the following fact [92]:

Proposition 2.1. *Let* (M, ρ) *be an LSSS for an 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*, the target vector* $(1, 0, \ldots, 0)$ *is in the span of* $I = \{i : \pi(i) \in S\}$*. For all unauthorised set* $S \notin \mathbb{A}$*, 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* $\boldsymbol{w} \cdot \boldsymbol{M_i} = 0$*.*

In Appendix B.1, we will make use of a particular reconstruction algorithm for LSSSs known as Lagrange interpolation. This formed the basis for Shamir's well-known secret sharing scheme [89] for threshold policies (requiring k out of n shares to reconstruct the secret). The reconstruction procedure can be defined as follows [14].

Definition 2.19. *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 $|S| = d + 1$ then, 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 Appendix B.1, we will use Lagrange interpolation for a first degree polynomial. In particular, if $f(z)$ is a first degree polynomial, one can obtain $f(0)$ from two any points

$(i_1, f(i_1)), (i_2, f(i_2))$ where $i_1 \neq i_2$ by computing

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

2.6.4.2 Bilinear Maps and Hardness Assumptions

Most ABE schemes are built over groups with efficiently computable bilinear maps. These allow various parameters to be combined together in a secure fashion. In this section, we define bilinear maps and groups [14] , as well as a cryptographic hardness assumption [15, 33] upon which we shall base the security on our revocable DP-ABE scheme in Appendix B.1.

Definition 2.20. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative groups of prime order p, and let g be a generator of $\mathbb{G}$. A* bilinear map *(of type 1 [56]) 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 $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$*
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 the bilinear map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is efficiently computable.*

Definition 2.21. *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 $Z = e(g, h)^{a^{q+1}}$. For ease of notation, we write g_i to denote $g^{a^i} \in \mathbb{G}$. Let $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, G_{q+2}, g_{2q})$. An algorithm $\mathcal{A}$ that outputs $b \in \{0, 1\}$ has advantage ϵ in solving the Decisional q-BDHE problem in $\mathbb{G}$ if

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

where the probability is over the random choice of generators $g, h \in \mathbb{G}$, the random choice of $a \in \mathbb{Z}_p$, the random choice of $Z \in \mathbb{G}_T$, and the randomness of $\mathcal{A}$. We refer to the distribution on the left as $\mathcal{P}_{BDHE}$ and the one on the right hand side as $\mathcal{R}_{BDHE}$. We say

that the Decision q-BDHE assumption holds in $\mathbb{G}$ if no polynomial-time algorithm has a non-negligible advantage in solving the problem.

2.7 Digital Signatures

Digital signatures provide a proof of message integrity, as well as data origin authentication (since keys can be associated to particular users). A message is signed using a private signing key owned by a particular entity, and a public verification key can be used to verify that the signature was actually generated using the corresponding signing key and that the contents of the message have not changed since the signature was computed.

A digital signature scheme Sig comprises three polynomial-time algorithms Sig.KeyGen, Sig.Sign and Sig.Verify defined as follows [74]:

- $(SK, VK) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\ell)$: takes as input the security parameter and generates a signing key SK and a verification key VK;
- $\gamma \xleftarrow{\$} \mathsf{Sig.Sign}(m, SK)$: takes as input a message to be signed and the signing key, and outputs a signature γ of m;
- $D \leftarrow \mathsf{Sig.Verify}(m, \gamma, VK)$: takes as input a message and corresponding signature to be verified as well as the verification key, and outputs a decision D which is accept if γ is a valid signature on m and reject otherwise.

Definition 2.22. *A digital signature scheme is* correct *if for all security parameters ℓ and all $m \in \mathcal{M}$,*

$$\begin{aligned}\Pr[&(SK, VK) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\ell),\\ &\gamma \xleftarrow{\$} \mathsf{Sig.Sign}(m, SK),\\ &\mathsf{accept} \leftarrow \mathsf{Sig.Verify}(m, \gamma, VK)] = 1.\end{aligned}$$

2.8 Notions of Security

Thus far in this chapter, we have defined several cryptographic primitives and associated notions of *correctness* that will be required throughout the remainder of this thesis. As well as being *correct*, a cryptographic primitive must be *secure*. In this section, we define notions of security for a number of the previously discussed primitives (some primitives, such as KP-ABE are not discussed here as we will define and use variants of these in later chapters instead).

We will define notions of security for each cryptographic scheme as a *game* (or algorithm) written in pseudo-code. The game is run by a entity known as the *challenger*, $\mathcal{C}$. The execution of the game is designed to reflect realistic system evolution and threats. Each game takes as input the particular construction being proved secure and the unary representation of the security parameter.

An *adversary* against a particular scheme is modelled as a PPT algorithm $\mathcal{A}$ which is called at relevant points by $\mathcal{C}$. The inputs to the adversary are chosen to represent the knowledge a real attacker may learn by observing the execution of the system and by corrupting relevant parties. The adversary algorithm may be multi-stage (i.e. be called several times by the challenger, with different input parameters) and may maintain state between invocations. This represents the adversary performing tasks at different points during the execution of the system. For simplicity and generality, we do not explicitly provide the state as an input or output of the adversary or distinguish between single-stage and multi-stage adversaries; we also refer to both simply as $\mathcal{A}$, noting that a multistage adversary could be thought of as a single adversary $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ comprising sub-algorithms for each stage.

The challenger may provide *oracle access* to an adversary for a specified set of functions. This allows the adversary to query $\mathcal{C}$ for the results of these functions run over inputs chosen by the adversary as well as secret values only known to $\mathcal{C}$. Thus, the adversary is able to learn the outcome of some functions without holding the secret values needed to run them itself. This models the adversary monitoring the system to observe various messages and activity, as well as corrupting entities to learn their local secrets (e.g. in the case of querying a user key generation algorithm) or to cause particular events to occur. Providing the adversary with oracle access to a function F on inputs $x_1, \ldots, x_n$ held by the

challenger is denoted $\mathcal{A}^{\mathcal{O}^{\mathrm{F}}(\cdot,\ldots,\cdot,x_1,\ldots,x_n)}$; we use the notation $\cdot$ to denote parameters where the adversary has free choice of input value. Sometimes, when providing oracle access to multiple functions, we will use the notation $\mathcal{A}^{\mathcal{O}}$ and describe the specific functions in the accompanying text.

Many oracles simply run the relevant algorithm and return the result to the adversary. If the challenger is required to perform additional steps (e.g. to perform 'housekeeping' on its state to reflect changes arising from the query or to check whether the result of a query would lead to a trivial win for $\mathcal{A}$) then the oracle is given as an additional pseudo-code algorithm alongside the relevant games. No oracle query should allow the adversary to gain information that it would not usually be able to observe in practice or that would lead to a trivial win (e.g. an oracle should not issue the decryption key for a challenge ciphertext as the adversary would be able to win the game, even though this does not model a realistic attack as the adversary would not be an authorised decryptor in practice).

Each game ends with the challenger outputting either 1 or 0 based on the output of the adversary algorithm; the output 1 (synonymous with `true`) denotes that the adversary output was 'correct'. In some games, we may write **return** $(b' = b)$ to denote the output of the game; this denotes an equality comparison between the variables b' and b, and the game will return `true` if and only if these variables are identical.

Formally, we write $\mathbf{Exp}^{\mathrm{X}}_{\mathcal{A}}[\mathcal{Y}, 1^{\ell}, args]$ to refer to a game for a security notion X, interacting with an adversary $\mathcal{A}$, against a construction $\mathcal{Y}$ when run with security parameter ℓ and additional arguments $args$. We denote the advantage of such an adversary $\mathcal{A}$ as $Adv^{\mathrm{X}}_{\mathcal{A}}(\mathcal{Y}, 1^{\ell}, args')$. For each game, we define the *advantage* of $\mathcal{A}$ in terms of the probability that the security game results in `true` (i.e. the adversary was correct) which we denote $\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathrm{X}}_{\mathcal{A}}[\mathcal{Y}, 1^{\ell}, args]\right]$. Informally, the advantage is defined to be the difference between the adversary's success probability and the probability of success one would expect to achieve by making a simple guess. We usually define a scheme to be *secure* if all PPT adversaries have at most a *negligible* advantage in terms of the security parameter.

Game 2.1 $\mathbf{Exp}_{\mathcal{A}}^{\text{VERIF}}\left[\mathcal{VC}, 1^{\ell}, F\right]$

1: $(EK_F, SK_F) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, F)$
2: $x \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}}(\cdot, SK_F)}(EK_F)$
3: $(\sigma_{F,x}, VK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_F)$
4: $\theta_{F(x)} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}}(\cdot, SK_F)}(\sigma_{F,x}, EK_F)$
5: $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x}, SK_F)$
6: **if** $((y \neq \perp)$ **and** $(y \neq F(x)))$ **then return** 1
7: **else return** 0

Oracle 2.1 $\mathcal{O}^{\mathsf{ProbGen}}(z, SK_F)$

1: $(\sigma_{F,z}, VK_{F,z}) \xleftarrow{\$} \mathsf{ProbGen}(z, SK_F)$
2: **return** $\sigma_{F,z}$

2.8.1 Verifiable Outsourced Computation

As mentioned in Section 2.2, security notions for VC schemes have focussed on *verifiability* — that is, ensuring that a server cannot return an incorrect computational result and have it accepted by a client. This is shown for the VC setting in Game 2.1 and Oracle 2.1 for a function F in the family of admissible functions $\mathcal{F}$, where the ProbGen oracle simply runs the ProbGen algorithm and returns the $\sigma_{F,x}$ part of the result. The adversary is given the public parameters from the Setup algorithm and access to the ProbGen oracle (to simulate the adversary observing client behaviour in a real system). The adversary chooses an input x and the challenger honestly produces an encoded input and verification key, both of which are given to the adversary. We require the challenger to perform this step to ensure that it knows the correct verification key corresponding to the computation. The adversary wins if it can produce an encoded output which is accepted by the Verify algorithm but does not in fact encode the value $F(x)$.

Definition 2.23. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* VERIF *game for a VC construction $\mathcal{VC}$ and a function F is defined as:*

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

A VC scheme, $\mathcal{VC}$, is verifiable *for a function F if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{VERIF}}(\mathcal{VC}, 1^{\ell}, F) \leqslant \text{negl}(\ell).$$

Gennaro et al. [57] also consider notions of input and output privacy such that an adversary

Game 2.2 $\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{PVC}, 1^{\ell}, F\right]$

1: $(EK_F, PK_F) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, F)$
2: $x \xleftarrow{\$} \mathcal{A}(EK_F, PK_F)$
3: $(\sigma_{F,x}, VK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, PK_F)$
4: $\theta_{F(x)} \xleftarrow{\$} \mathcal{A}(EK_F, PK_F, \sigma_{F,x}, VK_{F,x})$
5: $y \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x})$
6: **if** $((y \neq \perp)$ and $(y \neq F(x)))$ **then return** 1
7: **else return** 0

cannot distinguish from $\sigma_{F,x}$ which input value x was encoded, and that $\theta_{F(x)}$ does not reveal the value $F(x)$.

Verifiability in the PVC setting is defined similarly in Game 2.2. Compared to Game 2.1, this game does not require a ProbGen oracle as the outsourcing of computations is based purely on public information and hence can be run by the adversary itself. Again, the adversary wins if it can produce an encoded output which is accepted by the Verify algorithm but is not in fact a valid encoding of $F(x)$.

Definition 2.24. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* PubVerif *game for a PVC construction $\mathcal{PVC}$ and a function F is defined as:*

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

A PVC scheme, $\mathcal{PVC}$, is publicly verifiable *for F if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{PubVerif}}(\mathcal{PVC}, 1^{\ell}, F) \leqslant \text{negl}(\ell).$$

2.8.2 Key Assignment Schemes

Atallah et al. [12] proposed two fundamental security properties for KASs — *key recovery* (KR) and *key indistinguishability* (KI). Key recovery is the weaker of the two notions and requires that the derivation of $\kappa_{v^\star}$ from a set of keys $\kappa_{v_1}, \ldots, \kappa_{v_n}$ should be possible if and only if there exists i such that $v_i \geqslant v^\star$. This property asserts that a set of users cannot recover a key that no one of them is not already authorised to derive. An interactive key encrypting KAS is known to be secure against key recovery provided that the encryption function is chosen appropriately [12].

Game 2.3 $\mathbf{Exp}^{\mathrm{KI}}_{\mathcal{A}}\left[\mathcal{KAS}, 1^{\ell}, (L, \leqslant)\right]$

$\kappa_L \xleftarrow{\$} \mathsf{MakeKeys}(1^{\ell}, (L, \leqslant))$
$\omega_L \xleftarrow{\$} \mathsf{MakeSecrets}(1^{\ell}, (L, \leqslant))$
$Pub_{(L,\leqslant)} \xleftarrow{\$} \mathsf{MakePublicData}(1^{\ell}, (L, \leqslant))$
$Q \leftarrow \epsilon$
$v^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Corrupt}}(\cdot,\perp)}(Pub_{(L,\leqslant)})$
for all $v_i \in Q$ **do**
  if $(v^{\star} \leqslant v_i)$ **then return** 0
$b \xleftarrow{\$} \{0,1\}$
if $(b = 0)$ **then** $\kappa^{\star} \leftarrow \kappa_{v^{\star}}$
else $\kappa^{\star} \xleftarrow{\$} \mathcal{K}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Corrupt}}(\cdot,v^{\star})}(\kappa^{\star}, Pub_{(L,\leqslant)})$
return $(b' = b)$

Oracle 2.2 $\mathcal{O}^{\mathsf{Corrupt}}(v_i, v^{\star})$

if $(v_i \geqslant v^{\star})$ **then return** $\perp$
$Q \leftarrow Q \cup v_i$
return $(\kappa_{v_i}, \omega_{v_i})$

Key indistinguishability requires that, given a set of keys $\kappa_{v_1}, \ldots, \kappa_{v_n}$, an adversary should not be able to distinguish between the key for a challenge node $v^{\star}$ (not a descendent of any v_i) and a randomly chosen key. This property is crucial if derived keys are to be used in other cryptographic protocols.

Both notions can be viewed in terms of *static* or *adaptive* adversaries. A static adversary will be provided with the graph $H(L, \leqslant)$ and must first select a node $v^{\star} \in L$ that will form the challenge *before* the challenger has initiated the KAS and hence before receiving the public and (appropriate) secret information. An adaptive (or dynamic) adversary, on the other hand, may make oracle queries to the challenger to request keys and secrets for nodes chosen in an adaptive fashion *before* selecting the challenge node $v^{\star}$ (subject to $v^{\star}$ not being a descendant of any queried node for which secret information was given, to avoid a trivial win). The adversary may similarly make such oracle queries after choosing the challenge node (again requiring that the returned values do not allow the trivial derivation of the challenge key $\kappa_{v^{\star}}$). Adaptive key indistinguishability is presented formally in Game 2.3, with the associated oracle being given in Oracle 2.2.

The game begins with the challenger running the setup algorithms for the KAS scheme and initialising an empty list Q of queried nodes. The adversary algorithm is then called with the KAS public information as input. The adversary algorithm may make oracle queries to the Corrupt algorithm, given in Oracle 2.2, for the adversary's choice of input v_i. For this query phase, the challenger fixes the second input parameter, $v^{\star}$, to be a

distinguished symbol $\perp$ (to identify that this is the first query phase, in which the second parameter is not required). The oracle first performs a check as to whether the query would lead to a trivial win (if $v_i \geqslant v^\star$). As $v^\star$ has been fixed to be $\perp$, these elements are incomparable and the if statement is not satisfied. The queried node is added to the list Q and the key and secret for the node is returned to $\mathcal{A}$.

After making a polynomial number of queries, $\mathcal{A}$ will output a choice of challenge node $v^\star$. The challenger then checks if this choice is a valid challenge — that is, it is not a descendent of any queried node, as this would trivially allow the adversary to derive the challenge key and win. If the choice is invalid, then the adversary loses the game and the game returns 0. Otherwise, the challenger chooses a bit b uniformly at random. If $b = 0$ then the challenge key $\kappa^\star$ is set to be the real key, $\kappa_{v^\star}$, for the challenge label. Otherwise, the challenge key is chosen uniformly at random from the keyspace. The adversary algorithm is then called again, this time with inputs $\kappa^\star$ and the public information again, and given access to the Corrupt oracle again. This time, the second input to Corrupt is set to be the challenge node $v^\star$ chosen by $\mathcal{A}$ such that the oracle can check whether responding to a query would lead to a trivial win (if the queried node is an ancestor of the challenge node and therefore would allow derivation of the key $\kappa_{v^\star}$).

Finally, the adversary returns a bit b' which is its guess of the bit b chosen by $\mathcal{C}$ — that is, whether the challenge key was a real key or a randomly chosen key. If this guess is correct then the challenger returns 1 as the output of the game, and otherwise $\mathcal{C}$ returns 0.

The advantage of an adversary is defined to be the difference between the probability that the adversary guesses whether the key is real or random correctly (that is, the probability that the game outputs 1) and the probability of an adversary randomly guessing the correct outcome (which occurs with probability $\frac{1}{2}$). For a KAS to be secure in the sense of key indistinguishability, we require this advantage to be negligible in the security parameter — the adversary should almost certainly not do better than randomly guessing.

Definition 2.25. *The* advantage *of a PPT adversary $\mathcal{A}$ against the* KI *game for a particular KAS construction $\mathcal{KAS}$ is defined as:*

$$Adv^{\mathrm{KI}}_{\mathcal{A}}(\mathcal{KAS}, 1^\ell, (L, \leqslant)) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathrm{KI}}_{\mathcal{A}}\left[\mathcal{KAS}, 1^\ell, (L, \leqslant)\right]\right] - \tfrac{1}{2}.$$

A KAS construction, $\mathcal{KAS}$, is secure in the sense of key indistinguishability against adap-

tive adversaries *if for all PPT adversaries* $\mathcal{A}$*,*

$$Adv^{\mathrm{KI}}_{\mathcal{A}}(\mathcal{KAS}, 1^{\ell}, (L, \leqslant)) \leqslant \mathrm{negl}(\ell).$$

As observed by Freire et al. [55], it is straightforward to see that the static and adaptive notions of key indistinguishability are polynomially equivalent. A dynamic adversary can clearly choose its oracle queries and challenge node to be consistent with that of a static adversary. A static adversary can also be constructed that, before the KAS is instantiated, makes a guess for the node $v^{\star}$ that a corresponding adaptive adversary would choose. Such a guess is correct with probability at least $\frac{1}{|L|}$ (where L must be polynomial in size for the KAS to be efficiently instantiable), and the static adversary aborts the game if the guess is revealed to be incorrect. Thus, the static adversary succeeds with probability at least $\frac{1}{|L|}$ times the success probability of the dynamic adversary, and hence a scheme secure against static adversaries is also secure against dynamic adversaries (with a polynomial loss in the advantage). As a result, many papers focus only on static adversaries. In this thesis, however, we will primarily consider adaptive adversaries as this is more straightforwardly applicable to the interactive protocol settings we consider.

Freire et al. [55] introduced the notions of *strong key recovery* (S-KR) and *strong key indistinguishability* (S-KI) to reflect realistic attacks where the keys for nodes $v_i > v^{\star}$ may leak (through key misuse or cryptanalysis, for example). Thus, the adversary is able to learn all keys κ_{v_i} for $v_i \neq v^{\star}$ and the secret information ω_{v_i} for all nodes where $v^{\star} \not\leqslant v_i$ and, respectively, must recover $\kappa_{v^{\star}}$ or distinguish between $\kappa_{v^{\star}}$ and a random key. Strong key indistinguishability is shown for an adaptive adversary in Game 2.4 and Oracle 2.3. The game proceeds as in Game 2.3 except for the Corrupt oracle which now returns (only) the key for nodes $v_i > v^{\star}$.

Definition 2.26. *The* advantage *of a PPT adversary* $\mathcal{A}$ *against the* S-KI *game for a particular KAS construction* $\mathcal{KAS}$ *is defined as:*

$$Adv^{\text{S-KI}}_{\mathcal{A}}(\mathcal{KAS}, 1^{\ell}, (L, \leqslant)) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\text{S-KI}}_{\mathcal{A}}\left[\mathcal{KAS}, 1^{\ell}, (L, \leqslant)\right]\right] - \tfrac{1}{2}.$$

A KAS construction, $\mathcal{KAS}$*, is* secure in the sense of strong key indistinguishability against

Game 2.4 $\mathbf{Exp}^{\text{S-KI}}_{\mathcal{A}}\left[\mathcal{KAS}, 1^\ell, (L, \leqslant)\right]$

$b \xleftarrow{\$} \{0,1\}$
$\kappa_L \xleftarrow{\$} \mathsf{MakeKeys}(1^\ell, (L, \leqslant))$
$\omega_L \xleftarrow{\$} \mathsf{MakeSecrets}(1^\ell, (L, \leqslant))$
$Pub_{(L,\leqslant)} \xleftarrow{\$} \mathsf{MakePublicData}(1^\ell, (L, \leqslant))$
$Q \leftarrow \epsilon$
$v^\star \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Corrupt}}(\cdot,\perp)}(Pub_{(L,\leqslant)})$
for all $v_i \in Q$ **do**
  if $(v^\star \leqslant v_i)$ **then return** 0
if $(b = 0)$ **then** $k^\star \leftarrow k_{v^\star}$
else $k^\star \xleftarrow{\$} \mathcal{K}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Corrupt}}(\cdot,v^\star)}(k^\star, Pub_{(L,\leqslant)})$
return $(b' = b)$

Oracle 2.3 $\mathcal{O}^{\mathsf{Corrupt}}(v_i, v^\star)$

if $(v^\star \not\leqslant v_i)$ **then**
  $Q \leftarrow Q \cup v_i$
  return $(\kappa_{v_i}, \omega_{v_i})$
else if $(v_i > v^\star)$ **then**
  return κ_{v_i}
else
  return $\perp$

adaptive adversaries (S-KI) *if for all PPT adversaries* $\mathcal{A}$*,*

$$Adv^{\text{S-KI}}_{\mathcal{A}}(\mathcal{KAS}, 1^\ell, (L, \leqslant)) \leqslant \text{negl}(\ell).$$

Although Freire et al. formally separated the notions of key recovery and strong key recovery, Castiglione et al. [35] showed polynomial equivalence between key indistinguishability and strong key indistinguishability. However, as we shall see in Chapter 4, the format of the adaptive S-KI game is well suited to some settings, particularly security reductions for interactive protocols where keys must be used before the challenge period, and so we shall continue to consider the strong key indistinguishability game.

2.8.3 Symmetric Encryption

There are many notions of security for symmetric encryption (see [11] for relations amongst four notions of security). Perhaps the most commonly discussed, and the one that we shall use later in this thesis, is *indistinguishability against chosen plaintext attacks* (IND-CPA). This notion requires that an adversary running in polynomial time, and with the ability to request the encryptions of arbitrary messages, can not distinguish which of two messages of his choice has been encrypted. Informally, the encryption scheme should hide

Game 2.5 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CPA}}\left[\mathcal{SE}, 1^{\ell}\right]$

1: $b \xleftarrow{\$} \{0,1\}$
2: $k^{\star} \xleftarrow{\$} \mathsf{KeyGen}(1^{\ell})$
3: $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\text{LoR}}(\cdot,\cdot,k^{\star})}(1^{\ell})$
4: **return** $(b' = b)$

Oracle 2.4 $\mathcal{O}^{\text{LoR}}(m_0, m_1, k^{\star})$

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

Game 2.6 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CCA}}\left[\mathcal{SE}, 1^{\ell}\right]$

1: $b \xleftarrow{\$} \{0,1\}$
2: $L \leftarrow \epsilon$
3: $k^{\star} \xleftarrow{\$} \mathsf{KeyGen}(1^{\ell})$
4: $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\text{LoR}}(\cdot,\cdot,k^{\star}),\mathsf{Decrypt}(\cdot,k^{\star})}(1^{\ell})$
5: **return** $(b' = b)$

Oracle 2.5 $\mathcal{O}^{\text{LoR}}(m_0, m_1, k^{\star})$

1: $CT \xleftarrow{\$} \mathsf{Encrypt}(m_b, k^{\star})$
2: **if** $(m_0 \neq m_1)$ **then** $L \leftarrow L \cup CT$
3: **return** CT

Oracle 2.6 $\mathcal{O}^{\mathsf{Decrypt}(CT,k^{\star})}$

1: **if** $(CT \in L)$ **then return** $\perp$
2: **return** $\mathsf{Decrypt}(CT, k^{\star})$

all information about the underlying plaintext so that a ciphertext reveals nothing about which message was encrypted.

The IND-CPA notion is formally defined in Game 2.5, Oracle 2.4 and Definition 2.27. The game begins with the challenger choosing a bit b uniformly at random and running the KeyGen algorithm to generate a challenge key $k^{\star}$. The adversary algorithm is then called with the security parameter as input, and it is given access to an LoR oracle, which implements the encryption functionality. This oracle takes two messages of the adversary's choice. It first checks that the messages are of the same length and then chooses one of the messages according to the bit b which is encrypted under the key $k^{\star}$ and the resulting ciphertext returned to the adversary. Eventually, the adversary returns a bit b' and the game outputs 1 if b' is the bit b chosen by the challenger. Note that the adversary may generate challenge ciphertexts using the LoR oracle by submitting two distinct messages of the same length, but may also use LoR as a simple encryption oracle by submitting the same message twice, $m_0 = m_1$.

Similarly, one can consider *Indistinguishability against Chosen Ciphertext Attacks* (IND-CCA) defined in Game 2.6 and Oracles 2.5 and 2.6. The game proceeds as in Game 2.5 but the challenger additionally maintains a list L of ciphertexts which is initially empty. As

shown in Oracle 2.5, the LoR oracle is modified to add generated ciphertexts to the list L if the input messages are distinct — that is, if the query forms a challenge ciphertext based on the value of b (which clearly is not the case if both message options are identical). The adversary is additionally given a decryption oracle (Oracle 2.6) to which it can submit ciphertexts to recover the corresponding plaintexts. However, to avoid trivial wins, a distinguished failure symbol $\perp$ is returned if the queried ciphertext has previously been generated by the LoR oracle as a result of a query for two distinct messages. Clearly, given the decryption of such a ciphertext, it would be possible to determine the value of b based on which message is returned.

For both notions, we define the advantage of an adversary to be the difference between the probabilities of the adversary guessing the bit b correctly (indicating which message was encrypted) and randomly guessing (which is correct with probability $\frac{1}{2}$).

Definition 2.27. *The* advantage *of a PPT adversary $\mathcal{A}$ against a notion*

$$\mathrm{X} \in \{\text{IND-CPA, IND-CCA}\}$$

for a symmetric encryption scheme $\mathcal{SE}$ is defined as:

$$Adv^{\mathrm{X}}_{\mathcal{A}}(\mathcal{SE}, 1^{\ell}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathrm{X}}_{\mathcal{A}}\left[\mathcal{SE}, 1^{\ell}\right]\right] - \tfrac{1}{2}.$$

A symmetric encryption scheme, $\mathcal{SE}$, is secure in the sense of notion X *if for all PPT adversaries $\mathcal{A}$,*

$$Adv^{\mathrm{X}}_{\mathcal{A}}(\mathcal{SE}, 1^{\ell}) \leqslant \mathrm{negl}(\ell).$$

The choice of security property for a symmetric encryption scheme will depend on the context in which the scheme is used and what information an adversary is likely to observe in practice. Security in the sense of IND-CPA and IND-CCA can also be defined for the public key setting where the adversary is given the challenge public key and can use this as an encryption oracle to select the challenge messages.

2.8.4 Symmetric Authenticated Encryption

Bellare and Namprempre [24] considered two notions of integrity for authenticated sym-

Game 2.7 $\mathbf{Exp}_{\mathcal{A}}^{\text{INT-PTXT}}\left[\mathcal{SE}, 1^{\ell}\right]$

1: $L \leftarrow \epsilon$
2: $k^{\star} \xleftarrow{\$} \mathsf{KeyGen}(1^{\ell})$
3: $CT^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Encrypt}(\cdot,k^{\star}),\mathrm{Ver}(\cdot)}}(1^{\ell})$
4: **return** $((\mathrm{Ver}(CT^{\star}) = 1)$ **and** $(\mathsf{Decrypt}(CT^{\star}, k^{\star}) \notin L))$

Oracle 2.7 $\mathcal{O}^{\mathsf{Encrypt}}(m, k^{\star})$:

1: $CT \xleftarrow{\$} \mathsf{Encrypt}(m, k^{\star})$
2: $L \leftarrow L \cup m$
3: **return** CT

Oracle 2.8 $\mathcal{O}^{\mathrm{Ver}(CT)}$:

1: **if** $(\perp \not\leftarrow \mathsf{Decrypt}(CT, k^{\star}))$ **then return** 1
2: **else return** 0

metric encryption schemes: *integrity of plaintexts* (INT-PTXT) and *integrity of ciphertexts* (INT-CTXT) under chosen message attacks. INT-PTXT requires it to be hard to create a ciphertext which decrypts to a message never encrypted by a legitimate sender, while INT-CTXT requires it to be hard to create a ciphertext that was not previously generated by a legitimate sender (regardless of the underlying plaintext). We will only require the first notion in this thesis, which is given in Game 2.7 and Oracles 2.7 and 2.8. The challenger initialises a list of queried messages L which is initially empty, and generates a symmetric key $k^{\star}$. The adversary is given the security parameter and access to both an encryption oracle and a verification oracle. The encryption oracle, Oracle 2.7, encrypts the adversary's choice of message m under $k^{\star}$ and adds m to the list of queried messages, L. The verification oracle, Oracle 2.8, attempts to decrypt a ciphertext provided by the adversary. It returns 1 if the queried ciphertext is deemed authentic (i.e. $\mathsf{Decrypt}$ does not return $\perp$) and 0 otherwise; thus the adversary is able to tell whether a given ciphertext would be accepted in the game or not, without being given the functionality of a decryption oracle (it cannot learn the plaintext for a given ciphertext using Oracle 2.8). The adversary wins if it can produce a ciphertext that is deemed authentic and that decrypts to a message not previously queried to the encryption oracle (i.e. not stored in L).

Definition 2.28. *The* advantage *of a PPT adversary $\mathcal{A}$ against the* INT-PTXT *game for an authenticated symmetric encryption scheme $\mathcal{SE}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{INT-PTXT}}(\mathcal{SE}, 1^{\ell}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{INT-PTXT}}\left[\mathcal{SE}, 1^{\ell}\right]\right].$$

An authenticated symmetric encryption scheme, $\mathcal{SE}$, is secure with respect to INT-PTXT

if for all PPT adversaries $\mathcal{A}$*,*

$$Adv_{\mathcal{A}}^{\text{INT-PTXT}}(\mathcal{SE}, 1^{\ell}) \leqslant \text{negl}(\ell).$$

In this thesis, we will require an authenticated symmetric encryption scheme that is secure in the sense of IND-CPA∧INT-PTXT. As noted by Bellare and Namprempre [24], such a scheme can be constructed from an IND-CPA symmetric encryption scheme and a weakly or strongly unforgeable message authentication code MAC [72] using the generic composition techniques of *MAC-then-Encrypt* or *Encrypt-then-MAC*. As this latter composition is shown to be secure for all security choices of the encryption and MAC schemes, we will adopt this construction in this paper. Thus, let $\mathcal{SE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a symmetric authenticated encryption scheme constructed from a symmetric encryption scheme $\mathcal{SE}' = (\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}')$ and a MAC $\mathcal{MAC} = (\mathsf{KeyGen}'', \mathsf{Tag}, \mathsf{Verify})$. KeyGen outputs two keys: SK_E corresponding to KeyGen' and SK_M relating to KeyGen''[2]. The key for the authenticated encryption scheme is defined to be the concatenation $SK = SK_E \| SK_M$. The encryption operation is defined as $\mathsf{Encrypt}(m, SK) = C \| \mathsf{Tag}(C, SK_M)$ for $C \xleftarrow{\$} \mathsf{Encrypt}'(m, SK_E)$.

2.8.5 Ciphertext-policy Attribute-based Encryption

As CP-ABE is an encryption mechanism, the security goals are very similar to those for symmetric (and in particular asymmetric) encryptions schemes — namely, the adversary should not be able to distinguish which of two messages was encrypted.

Recall that different CP-ABE keys grant access to different classes of documents; ciphertexts are no longer generated with a particular user in mind but rather a class of users. An important consideration in ABE schemes, therefore, is that users may not collude in order to decrypt a ciphertext which no one of them could decrypt alone. In the CP-ABE setting, consider a ciphertext encrypted with the policy $\texttt{Professor} \wedge \texttt{Maths}$; two users assigned attribute sets $\{\texttt{Professor}, \texttt{Physics}\}$ and $\{\texttt{Student}, \texttt{Maths}\}$ respectively should not be able to pool their `Professor` and `Maths` attributes to decrypt the ciphertext as neither of them actually satisfy the policy. To model collusion between users, we provide

[2]Note that when keys are derived from a KAS, the KAS could simply output strings that can be split into SK_E and SK_M.

Game 2.8 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CPA}}\left[\mathcal{CPABE}, 1^{\ell}, \mathcal{U}\right]$

$\mathbb{A}^{\star} \leftarrow \{\emptyset\}, Q \leftarrow \epsilon$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U})$
$(m_0, m_1, \mathbb{A}^{\star}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, \text{MK}, \text{PP})}(\text{PP})$
if $(|m_0| \neq |m_1|)$ **then return** 0
for all $A \in Q$ **do**
  if $(A \in \mathbb{A}^{\star})$ **then return** 0
$b \xleftarrow{\$} \{0, 1\}$
$CT^{\star} \xleftarrow{\$} \mathsf{Encrypt}(m_b, \mathbb{A}^{\star}, \text{PP})$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, \text{MK}, \text{PP})}(CT^{\star}, \text{PP})$
return $(b' = b)$

Oracle 2.9 $\mathcal{O}^{\mathsf{KeyGen}}(A, \text{MK}, \text{PP})$

if $A \notin \mathbb{A}^{\star}$ **then**
  $Q \leftarrow Q \cup A$
  return $\mathsf{KeyGen}(A, \text{MK}, \text{PP})$
else
  return $\perp$

Game 2.9 $\mathbf{Exp}_{\mathcal{A}}^{s\text{IND-CPA}}\left[\mathcal{CPABE}, 1^{\ell}, \mathcal{U}\right]$

$\mathbb{A}^{\star} \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{U})$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U})$
$(m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, \text{MK}, \text{PP})}(\text{PP})$
if $((|m_0| \neq |m_1|))$ **then return** 0
$b \xleftarrow{\$} \{0, 1\}$
$CT^{\star} \xleftarrow{\$} \mathsf{Encrypt}(m_b, \mathbb{A}^{\star}, \text{PP})$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot, \text{MK}, \text{PP})}(CT^{\star}, \text{PP})$
return $(b' = b)$

the adversary with an additional KeyGen oracle so that he can request multiple decryption keys for different attribute sets. To avoid trivial wins, we require that all queried attribute sets do not satisfy the challenge policy (else the adversary holds a valid key and can decrypt the challenge ciphertext himself).

Security for ABE schemes in general can be classed as *full* or *selective* security. Full security is the ideal notion that we would like to achieve, whilst selective security has traditionally been more readily achievable and gives more of a sense of heuristic security. The notions differ based on whether the adversary is given the public parameters before or after making a choice of challenge. The selective notion (where the challenge access structure in CP-ABE or challenge attribute set in KP-ABE) allows the challenger to *partition* the system into queries that it must be able to answer and those it will not (as some queries will be restricted to avoid giving the adversary a trivial win); thus, the challenger is able to embed secrets for a reductive proof, for example, a Diffie-Hellman challenge group element, such that it need not know the corresponding trapdoor.

Oracle 2.10 $\mathcal{O}^{\mathsf{KeyGen}}(A, \mathrm{MK}, \mathrm{PP})$

1: **if** $(A \notin \mathbb{A}^\star)$ **then return** $\mathsf{KeyGen}(A, \mathrm{MK}, \mathrm{PP})$
2: **else return** $\perp$

The full notion of *indistinguishability against chosen plaintext attacks* (IND-CPA) for a CP-ABE scheme is given in Game 2.8 and Oracle 2.9. The *selective* IND-CPA notion (sIND-CPA) given in Game 2.9 and Oracle 2.10. The full game begins with the challenger initialising the challenge access structure and an empty list Q of queried attribute sets. The selective game, on the other hand, begins with the adversary algorithm, given the security parameter and attribute universe, selecting an access structure $\mathbb{A}^\star$ that it will attempt to attack. Then, in both games, the challenger runs the Setup algorithm and calls the adversary with the generated public parameters as input. The adversary is also given access to a KeyGen oracle which returns a valid decryption key for the adversary's choice of attribute set A only if A does not satisfy the challenge access structure $\mathbb{A}^\star$. Otherwise, a distinguished failure symbol $\perp$ is returned to avoid allowing the adversary a trivial win.

After a polynomial number of queries, the adversary will return two messages of equal length (if the lengths differ then the adversary loses the game). In the full IND-CPA game, the adversary also chooses the challenge access structure $\mathbb{A}^\star$ at this point. If $\mathbb{A}^\star$ is satisfied by any attribute set queried to the KeyGen oracle (i.e. listed in Q), then the adversary loses the game as it has not found a valid attack target.

The challenger chooses a bit b uniformly at random and uses this to choose one of the two messages to be encrypted under the challenge access structure. The adversary is called again with the resulting ciphertext and again given access to the KeyGen oracle. Eventually, $\mathcal{A}$ must return a guess b' of the value b (i.e. which message was encrypted). The game returns 1 if this guess is correct, and 0 otherwise.

Definition 2.29. *The* advantage *against a notion* X $\in$ {IND-CPA, sIND-CPA} *of a PPT adversary* $\mathcal{A}$ *for a CP-ABE construction* $\mathcal{CP}$-$\mathcal{ABE}$ *is defined as:*

$$Adv^{\mathrm{X}}_{\mathcal{A}}(\mathcal{CP}\text{-}\mathcal{ABE}, 1^\ell, \mathcal{U}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{X}_{\mathcal{A}}\left[\mathcal{CP}\text{-}\mathcal{ABE}, 1^\ell, \mathcal{U}\right]\right] - \tfrac{1}{2}.$$

A CP-ABE scheme, $\mathcal{CP}$-$\mathcal{ABE}$*, is* secure with respect to X *if for all PPT adversaries* $\mathcal{A}$*,*

$$Adv^{\mathrm{X}}_{\mathcal{A}}(\mathcal{CP}\text{-}\mathcal{ABE}, 1^\ell, \mathcal{U}) \leqslant \mathrm{negl}(\ell).$$

Game 2.10 $\mathbf{Exp}_{\mathcal{A}}^{s\text{IND-CPA}}\left[\mathcal{DPABE}, 1^{\ell}, \mathcal{U}\right]$

1: $(\omega^{\star}, \mathbb{S}^{\star}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{U})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U})$
3: $(m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}((\cdot,\cdot),\text{MK},\text{PP})}(\text{PP})$
4: **if** $(|m_0| \neq |m_1|)$ **then return** 0
5: $b \xleftarrow{\$} \{0,1\}$
6: $CT^{\star} \xleftarrow{\$} \mathsf{Encrypt}(m_b, (\omega^{\star}, \mathbb{S}^{\star}), \text{PP})$
7: $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}((\cdot,\cdot),\text{MK},\text{PP})}(CT^{\star}, \text{PP})$
8: **return** $(b' = b)$

Oracle 2.11 $\mathcal{O}^{\mathsf{KeyGen}}((\mathbb{O}, \psi), \text{MK}, \text{PP})$

1: **if** $(\omega^{\star} \notin \mathbb{O}$ **or** $\psi \notin \mathbb{S}^{\star})$ **then return** $\mathsf{KeyGen}((\mathbb{O}, \psi), \text{MK}, \text{PP})$
2: **else return** $\perp$

2.8.6 Dual-policy Attribute-based Encryption

Security for DP-ABE is defined similarly to security for CP-ABE. Selective security for DP-ABE is defined in Game 2.10 and Oracle 2.11. As before, a full notion of security (or adaptive security) can be defined where the adversary receives the public parameters before selecting the challenge input. Note that notions of sIND-CPA for both KP-ABE and CP-ABE can be defined by ignoring the relevant attribute sets and access structures.

Definition 2.30. *The* advantage *of a PPT adversary $\mathcal{A}$ against the* sIND-CPA *game for a DP-ABE construction $\mathcal{DP}$-$\mathcal{ABE}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{sIND-CPA}}(\mathcal{DP}\text{-}\mathcal{ABE}, 1^{\ell}, \mathcal{U}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{sIND-CPA}}\left[\mathcal{DP}\text{-}\mathcal{ABE}, 1^{\ell}, \mathcal{U}\right]\right] - \tfrac{1}{2}.$$

A DP-ABE scheme, $\mathcal{DP}$-$\mathcal{ABE}$, is selectively secure in the sense of indistinguishability against chosen plaintext attack (sIND-CPA) *if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{sIND-CPA}}(\mathcal{DP}\text{-}\mathcal{ABE}, 1^{\ell}, \mathcal{U}) \leqslant \text{negl}(\ell).$$

2.8.7 Digital Signatures

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 that it saw a signature for. More formally, this is defined in Game 2.11 and Oracle 2.12.

Game 2.11 $\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}\left[\mathcal{SIG}, 1^{\ell}\right]$

1: $Q = \epsilon$
2: $(SK, VK) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^{\ell})$
3: $(m^{\star}, \gamma^{\star}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}(\cdot, SK)}(VK)$
4: **if** (accept $\leftarrow \mathsf{Sig.Verify}(m^{\star}, \gamma^{\star}, VK)$ **and** $m^{\star} \notin Q$) **then return** 1
5: **else return** 0

Oracle 2.12 $\mathcal{O}^{\mathsf{Sign}}(m, SK)$

1: $Q \leftarrow Q \cup m$
2: **return** $\mathsf{Sig.Sign}(m, SK)$

Game 2.12 $\mathbf{Exp}_{\mathcal{A}}^{Invert}\left[1^{\ell}, g\right]$

1: $w \xleftarrow{\$} \{0,1\}^{\ell}$
2: $z \leftarrow g(w)$
3: $w' \xleftarrow{\$} \mathcal{A}(1^{\ell}, g, z)$
4: **return** $(g(w') = z)$

Definition 2.31. *The* advantage *of a PPT adversary $\mathcal{A}$ against the* EUF-CMA *game for a particular digital signature construction $\mathcal{SIG}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{EUF-CMA}}(\mathcal{SIG}, 1^{\ell}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}\left[\mathcal{SIG}, 1^{\ell}\right]\right].$$

A digital signature construction, $\mathcal{SIG}$, is existentially secure against chosen message attacks (EUF-CMA) *if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{EUF-CMA}}(\mathcal{SIG}, 1^{\ell}) \leqslant \text{negl}(\ell).$$

2.8.8 One-way Functions

A *one-way function* g is characterised by 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 PPT algorithm to invert g (that is, to find a pre-image of a given value y) except with negligible probability. This requirement is captured in the *inverting experiment* (Game 2.12) where we consider the experiment for a function g. The challenger chooses an input uniformly at random from the domain of g, applies g to this input and gives the adversary both the result and a description of the function. It suffices for $\mathcal{A}$ to find any value of x' for which $g(x') = y = g(x)$.

Definition 2.32. *A function g is* one-way *if the following two conditions hold [74]:*

1. (Easy to compute) *there exists a polynomial-time algorithm M_g computing g; i.e. $M_g(w) = g(w)$ for all $w \in \mathsf{Dom}(g)$;*

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

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

Chapter 3

Revocation in Publicly Verifiable Outsourced Computation

Contents

This chapter looks at the setting of publicly verifiable outsourced computation (PVC) in which it has been shown that attribute-based encryption can be used, not as an access control enforcement primitive, but instead to prove correctness of a computation. We investigate the current proposal and propose improvements to achieve a more practical system model, including a simple method for servers to compute multiple functions and a method to revoke misbehaving servers.

3.1 Introduction

As discussed in Section 2.2, *verifiable outsourced computation* (VC), has attracted a lot of attention in the community recently. VC aims to allow a single client with limited resources to outsource computations to an external server and to verify whether returned results are correct. Publicly verifiable computation (PVC) [84] aims to provide a more practical VC solution wherein only one client must perform an expensive setup operation

and subsequently *any* other entity may use only public information to outsource computations and to verify results; thus PVC aims to be a multi-client system. However, in our opinion, the current PVC schemes do not support multiple servers computing multiple functions particularly well.

We believe that both of these are important requirements for many PVC systems that might be used in practice. It may be desirable for a set of clients to be able to choose from a set of servers on a per computation basis. For example, certain computations may require different computational resources (e.g. a certain amount of RAM or processor cores) that are only found in some cloud service providers, or clients may wish to outsource a computation to servers which are geographically nearby to minimise latency (if computation/retrieval time is important). If multiple servers are able to provide computational services within a PVC system, they may compete amongst themselves to reduce costs or may be able to bid on computations based on whether they currently have resources readily available; as a client is assumed to set up a PVC system and is therefore the system owner, it is in the client's interest to introduce multiple servers to enable this cost reduction.

We believe that it is unlikely that a client would be willing to expend the resources to initialise a PVC system (in terms of computational resources, and in terms of the monetary cost of contracting a cloud server provider) to outsource computations of only a single function. Indeed, we also believe it unlikely that an outsourced computation solution for a single function would provide the level of functionality required in practice — if company employees are to rely only on mobile or lightweight devices, then any provided PVC solution should enable them to perform *all* of their duties. In existing schemes, to outsource a second function, either an entirely new PVC system would need to be initialised or more complex primitives must be used to instantiate the scheme [84].

It is also conceivable that multiple sets of clients (e.g. multiple companies) will be largely interested in outsourcing similar sets of computations (e.g. common statistical computations), albeit on different, client-specific input data. In current PVC proposals, it is likely that each group of clients would have a distinguished client that performs the expensive Setup operation and issue delegation and evaluation keys for specific functions on behalf of their own group. Given that the functions of interest to these groups may overlap, it could be that much of this effort is redundantly replicated by multiple distinguished

clients. In addition, the introduction of multiple computational servers and multiple functions to PVC systems results in an increase workload for the distinguished clients (who must issue evaluation keys to servers for each function); the role of these clients becomes akin to an authority on entities within the system. We therefore suggest the introduction of a single trusted party which we call a *key distribution centre* (KDC); the KDC initialises the system and issues evaluation keys on behalf of *all* entities in the system.

Finally, given that we have enabled multiple untrusted servers to enrol in a PVC system, we may wish to *revoke* servers that are detected as misbehaving (either maliciously or through poor performance which introduces errors to computations) such that they are prevented from performing future computations. In the traditional, single-client setting of VC, the client itself would simply choose to no longer use the server, and in both VC and prior PVC schemes, a new system would need to be initialised. However, in the multi-client setting, it is important that all clients are informed that a server is known not to be trustworthy. In our new model of PVC, the system may include other servers that can still be used, so initialising a new system is not a desirable option. Note that if other clients were to outsource a computation to a misbehaving server, any errors would still be detected due to the verification property, but we wish to prevent clients wasting their (limited) resources delegating to a 'bad' server and to discourage servers from cheating in the first place, as they know they will be detected, revoked and therefore potentially incur a significant (financial) penalty from not receiving future work.

Our main contribution in this chapter, then, is to introduce the new notion of *revocable publicly verifiable computation* (RPVC). We allow multiple servers to enrol in a PVC system and allow the outsourced computation of multiple functions within a single PVC system. In some sense, enabling the evaluation of multiple functions can be seen as a shift from SIMD- to MIMD-style (that is, *single instruction, multiple data* to *multiple instructions, multiple data* [54]) PVC environments, where servers can compute multiple functions on multiple inputs provided by clients, albeit not necessarily in parallel. We give a rigorous definitional framework for RPVC that we believe more accurately reflects real environments than existing proposals. This new framework both removes redundancy and facilitates additional functionality, leading to several new security notions.

In Section 3.2, we briefly review the PVC construction of Parno et al. [84] and the revocable key-policy attribute-based encryption scheme of Attrapadung and Imai [14], both

of which will inform our construction of RPVC later in this chapter. In Section 3.3, we define our system model and framework for RPVC and in Section 3.4 we define relevant security models. In Section 3.5, we provide an overview, technical details and a concrete instantiation of our framework using attribute-based encryption and finally, in Section 3.6, we provide full security proofs for our construction.

3.2 Background Material

3.2.1 Construction of Publicly Verifiable Computation Schemes

Parno et al. [84] provide a PVC construction using key-policy attribute-based encryption (KP-ABE) [66], for the family of monotone Boolean functions.[1] Our construction of RPVC will be based on this construction, and therefore we discuss the basic principles here.

The idea of Parno et al. was to use the KP-ABE decryption functionality as a proof that a monotone Boolean function is satisfied (i.e. outputs 1) on a given input. Recall that in KP-ABE, decryption keys are associated with access structures and ciphertexts are associated with attribute sets. Decryption succeeds if and only if the attribute set in the ciphertext satisfies the access structure in the user's decryption key. Recall also that an access structure is a collection of satisfying attribute sets, and that any monotone Boolean formula F can also be written in a similar form (the set of all inputs x such that $F(x) = 1$). If inputs to Boolean functions can be written as attribute sets then we may identify monotone access structures and monotone Boolean formulas. In the PVC setting, we can view functions to be computed as access structures and issue computational servers with corresponding decryption keys. Note that there may be exponentially many x such that $F(x) = 1$ and, thus, the initial phase can indeed be computationally expensive for the client. Input data can be represented as an attribute set and associated with ciphertexts; we shall discuss this representation shortly.

To outsource a computation of $F(x)$, Parno et al. select a random message m_0 from the message space of the ABE scheme and encrypt it under the attribute set A_x, corresponding to x. A computational server is issued an evaluation key in the form of an ABE decryption

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

key for the access structure encoding F. If the server can correctly decrypt the ciphertext and return the correct message then the client can be assured (by the correctness and security of the ABE scheme) that $F(x) = 1$. If the message space is large enough (which it must be for the ABE scheme to be secure) then the server is unable to guess the correct message to return with significant probability.

However, a malicious server could return the distinguished symbol $\perp$ in the hope of convincing the client that $F(x) = 0$ — that is, the input attributes did not satisfy the access structure. Therefore, Parno et al. initialise a second ABE system and perform the same operations as above to encrypt a random message m_1 under the input set corresponding to x, and to issue the server a decryption key corresponding to 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 $\overline{F}(x)$ will output 1 and therefore exactly one of the messages will be output by the decryption algorithm. By observing which message is returned and which ABE system the message was encrypted with, the client can determine whether $F(x)$ or $\overline{F}(x)$ was satisfied and therefore whether $F(x) = 1$ or 0 respectively. A valid response from a server, therefore, comprises the outputs (d_0, d_1) from two decryption operations and is of the following form:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } F(x) = 1; \\ (\perp, m_1), & \text{if } F(x) = 0. \end{cases} \tag{3.1}$$

Note that because KP-ABE is a public-key encryption primitive, *any* entity can form ciphertexts and hence the construction achieves public *delegability*. Public *verifiability*, on the other hand, is achieved using a one-way function g (e.g. a pre-image resistant hash function). When outsourcing a computation, a client publishes a verification key comprising the result of applying g to each randomly chosen message m_0 and m_1. On receipt of a computational result from the server (i.e. exactly one 'decrypted' message), *any* entity may apply g to the returned message and compare this with the verification key to verify correctness. Note that a malicious server gains no advantage from the verification key as it cannot invert g to recover either message. The mapping between PVC and KP-ABE parameters is shown in Table 3.1.

As mentioned above, we must be able to define input data to outsourced computations as attribute sets. Parno et al. did not state how this should be done; in this thesis, we perform

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 3.1: Mapping between PVC and KP-ABE parameters.

the following encoding procedure. Define a universe $\mathcal{U}$ of n attributes and associate $V \subseteq \mathcal{U}$ with a binary n-tuple where the i^{th} bit is 1 if and only if the i^{th} attribute is in V. We call this the *characteristic tuple* of V. Thus, there is a natural one-to-one correspondence between n-tuples and attribute sets; we write A_x to denote the attribute set associated with a characteristic tuple x. An alternative way to view this is to let $\mathcal{U} = \{A_1, A_2, \ldots, A_n\}$. Then, a bit string $\bar{v} = v_1 \ldots v_n$ of length n is the characteristic tuple of the set $V \subseteq \mathcal{U}$ where $V = \{A_i : v_i = 1\}$. 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 $i \in [n]$. 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 this thesis, we shall mainly refer to monotonic Boolean functions, in line with the majority of the ABE literature which refer only to monotonic access structures. However, we note that the use of a non-monotonic KP-ABE scheme [80] could easily accommodate general Boolean functions, and hence our outsourcing schemes would be able to outsource the NC^1 class of functions. As the KP-ABE scheme is used in a black-box manner in both our construction and that of Parno et al. [84], this change should be largely transparent. A simple alternative to such schemes is to adjust our encoding scheme for input data, as 0 values in the input bitstring can now affect the outcome of the computation. Doing so is a straightforward extension where $\mathcal{U}$ is defined to have $2n$ 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 $V \subseteq \mathcal{U}$ where $V = \{A_i^j : v_i = j\}$. By applying De Morgan's laws to a non-monotonic Boolean function F, any negations within the function can be moved such that they apply only to the input variables, and hence by choosing the value of $j \in \{0,1\}$ for each attribute appropriately in the input attribute set, a non-monotonic function can be satisfied.

3.2.2 Revocable Key-Policy Attribute-based Encryption

Revocation is a key problem in cryptography, particularly in the attribute-based setting where many users hold keys for the same functionality (that is, keys which grant access to the same objects). Revocation mechanisms aim to disable this functionality for certain users, for example, if they are dishonest or if they leave the system. In the attribute-based setting, revocation can either target specific attributes (to disable certain policies within the system) or specific users (to account for a changing user population). In this thesis, we shall focus on the latter as we wish to prevent misbehaving servers (users) from participating in the PVC system at all; not just to revoke certain functionality. User revocation itself leads to two different modes [14]:

- *Direct revocation* allows users to specify a revocation list during the encryption algorithm which lists all currently revoked users. Hence periodic rekeying is not required, but encryptors must have knowledge of 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. The update material enables non-revoked users to update their key to be functional during the relevant time period. A revoked user will not be able to use the update material and thus their key will not decrypt ciphertexts associated with the current time period attribute. With indirect revocation, users need only 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 use the *indirectly revocable KP-ABE scheme* of Attrapadung and Imai [14], itself a more formal definition of a scheme due to Boldyreva et al. [31]. This choice will enable the revocation of misbehaving servers in a PVC scheme such that they cannot perform further computations. We choose indirect revocation to minimise the workload (in terms of maintaining synchronised, up-to-date revocation lists) of the weak client devices. Indirectly revocable KP-ABE schemes define the universe of attributes to be $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\text{ID}}$ where:

- $\mathcal{U}_{\text{attr}}$ is the normal attribute universe for describing ciphertexts and forming access control policies;

- $\mathcal{U}_{\text{time}}$ comprises attributes representing time periods;
- $\mathcal{U}_{\text{ID}}$ contains attributes encoding user identities.

Definition 3.1. *An* indirectly revocable key-policy attribute-based encryption scheme *comprises the following algorithms:*

- $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell, \mathcal{U})$*: takes the security parameter and the universe of attributes as input and outputs public parameters* PP *and master secret key* MK*;*
- $CT \xleftarrow{\$} \mathsf{ABE.Encrypt}(m, A, t, \text{PP})$*: takes a message* m*, an attribute set* $A \subseteq \mathcal{U}_{attr}$*, the current time period* $t \in \mathcal{U}_{time}$ *and the public parameters, and outputs a ciphertext that is valid for time* t*;*
- $SK_{\mathsf{id},\mathbb{A}} \xleftarrow{\$} \mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A}, \text{MK}, \text{PP})$*: takes as input an identity* $\mathsf{id} \in \mathcal{U}_{ID}$ *for a user, an access structure* $\mathbb{A}$ *encoding a policy, as well as the master secret key and public parameters. It outputs a decryption key for the user* id*;*
- $UK_{R,t} \xleftarrow{\$} \mathsf{ABE.KeyUpdate}(R, t, \text{MK}, \text{PP})$*: takes a revocation list* $R \subseteq \mathcal{U}_{ID}$ *containing the identities of currently revoked entities, the current time period* t*, as well as the master secret key and public parameters. It outputs updated key material* $UK_{R,t}$*;*
- $PT \leftarrow \mathsf{ABE.Decrypt}(CT, SK_{\mathsf{id},\mathbb{A}}, UK_{R,t}, \text{PP})$*: takes a ciphertext, a decryption key, an update key and the public parameters as input. It outputs a plaintext* PT *which is either* m *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* $\perp$ *otherwise to denote decryption failure.*

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

Definition 3.2. *An indirectly 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* $\mathbb{A} \subseteq 2^{\mathcal{U}_{attr}} \setminus \{\emptyset\}$*, all* $A \subseteq \mathcal{U}_{attr}$ *and all* $t \in \mathcal{U}_{time}$*, if* $A \in \mathbb{A}$ *and*

$\mathsf{id} \notin \mathsf{R}$, *then*

$$\begin{aligned}
&\Pr[(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell, \mathcal{U}),\\
&\quad CT \xleftarrow{\$} \mathsf{ABE.Encrypt}(m, A, t, \mathrm{PP}),\\
&\quad SK_{\mathsf{id},\mathbb{A}} \xleftarrow{\$} \mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A}, \mathrm{MK}, \mathrm{PP}),\\
&\quad UK_{R,t} \xleftarrow{\$} \mathsf{ABE.KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP}),\\
&\quad m \leftarrow \mathsf{ABE.Decrypt}(CT, SK_{\mathsf{id},\mathbb{A}}, UK_{R,t}, \mathrm{PP})]\\
&\quad = 1 - \mathrm{negl}(\ell).
\end{aligned}$$

The schemes [14, 31] mentioned above use the *complete-subtree* method to arrange users as the leaves of a binary tree such that the size of 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 inclusive. 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 material for just 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, network clock. In this case, rekeying 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 $\mathsf{ABE.KeyUpdate}$ algorithm is run. This would be more akin to a "push" system where entities should be notified by the key authority when newly updated key material is required. For generality, we assume a time source $\mathbb{T}$ from which the current time period t (be that a literal time value, counter or otherwise) may be efficiently sampled as $t \leftarrow \mathbb{T}$.

Attrapadung and Imai [14] defined several notions of security for revocable KP-ABE schemes. The security property we will require in this thesis is *indistinguishability against selective-target with semi-static query attack* (IND-sHRSS) [14], presented in Game 3.1 and Oracles 3.1 and 3.2. This is a *selective* notion where the adversary must declare at the beginning of the game the set of attributes $(t^\star, A^\star)$, including the time attribute $t^\star$, to be challenged upon. It is also possible to define a stronger, *full* notion of security whereby

the adversary may receive the public parameters and make oracle queries *before* selecting the challenge attributes (provided no query would lead to a trivial win). As we use the revocable KP-ABE scheme as a black box in our construction, it should be easy to change to a fully secure scheme if found; to the best of our knowledge, current primitives for indirect revocation in the KP-ABE setting only achieve selective security.

In Game 3.1, after the adversary chooses its challenge attributes $(t^\star, A^\star)$, the challenger runs Setup and gives the adversary 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 *semi-static* restriction requires that this revocation list must be chosen before the adversary is given oracle access to the $\mathsf{ABE.KeyGen}$ and $\mathsf{ABE.KeyUpdate}$ functions. Again, a stronger notion of an *adaptive* adversary may be defined where the adversary may choose the revocation list at the time of the challenge instead.

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 this restriction was not enforced, the adversary would hold a secret decryption key and would also receive update material for the challenge time period (as the identity is not revoked) and could therefore successfully decrypt the challenge ciphertext.

Similarly, for an update key 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, an update key could be issued which the adversary could combine 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 of equal length and the challenger chooses one of them at random to encrypt and passes the resulting ciphertext to the adversary. The adversary, again given oracle access as before, then guesses which message was encrypted. The advantage of the adversary is given in Definition 3.3.

Definition 3.3. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* IND-sHRSS *game for an indirectly revocable KP-ABE construction $\mathcal{KP}$-$\mathcal{ABE}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{IND-sHRSS}}(\mathcal{KP}\text{-}\mathcal{ABE}, 1^\ell, \mathcal{U}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{KP}\text{-}\mathcal{ABE}, 1^\ell, \mathcal{U}\right]\right] - \tfrac{1}{2}.$$

Game 3.1 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{ABE}, 1^{\ell}, \mathcal{U}\right]$

1: $(t^{\star}, A^{\star}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{U})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U})$
3: $\overline{R} \xleftarrow{\$} \mathcal{A}(\text{PP})$
4: $(m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,\text{MK},\text{PP}),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,\text{MK},\text{PP})}(\text{PP})$
5: **if** $(|m_0| \neq |m_1|)$ **then return** 0
6: $b \xleftarrow{\$} \{0,1\}$
7: $CT^{\star} \xleftarrow{\$} \mathsf{Encrypt}(m_b, A^{\star}, t^{\star}, \text{PP})$
8: $b^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,\text{MK},\text{PP}),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,\text{MK},\text{PP})}(CT^{\star}, \text{PP})$
9: **return** $(b' = b)$

Oracle 3.1 $\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{id}, \mathbb{A}, \text{MK}, \text{PP})$:

1: **if** $((A^{\star} \in \mathbb{A})$ **and** $(\mathsf{id} \notin \bar{\mathsf{R}}))$ **then return** $\perp$
2: **return** $\mathsf{KeyGen}(\mathsf{id}, \mathbb{A}, \text{MK}, \text{PP})$

Oracle 3.2 $\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, \text{MK}, \text{PP})$:

1: **if** $((t = t^{\star})$ **and** $(\overline{R} \not\subseteq R))$ **then return** $\perp$
2: **return** $\mathsf{KeyUpdate}(R, t, \text{MK}, \text{PP})$

An 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* $\mathcal{A}$*,*

$$Adv_{\mathcal{A}}^{\text{IND-sHRSS}}(\mathcal{KP}\text{-}\mathcal{ABE}, 1^{\ell}, \mathcal{U}) \leqslant \text{negl}(\ell).$$

3.3 Revocable Publicly Verifiable Computation

As previously mentioned, our main goal in this chapter is to enhance the existing PVC system model to reflect a more practical, multi-user, multi-server, multi-function setting. We allow multiple servers to compute multiple functions in a secure manner. In particular, a server cannot use an evaluation key for a function G to return a valid result for a computation of $F(x)$.

In the remainder of this section, we discuss in more detail the introduction and role of the KDC and the mechanism by which we allow servers to compute multiple functions. The introduction of a single trusted entity known as a key distribution centre (KDC) that is an authority on entities enrolled in the system aims to make entity management more straightforward. We also discuss two example system architectures that motivate our construction, which we call the *standard model* and the *manager model*.

3.3.1 Key Distribution Centre

Existing frameworks assume that a client or clients run the expensive phases of a VC scheme and that a single server performs all outsourced computations. We believe that this is undesirable 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. In other words, we may wish to enforce some kind of access control policy where an internal trusted entity will operate both as a facilitator of outsourced computation and as the policy enforcement point. We will examine the integration of RPVC and access control in Chapter 4.

The KDC that we introduce could, in fact, still be a distinguished client device (which has the additional resources required to perform the expensive setup operations), but here we consider it to be a separate entity to emphasise the distinction between the clients that request computations, and the KDC that is authoritative on the system and users. The KDC could be authoritative over many sets of clients (e.g. at an organisational level as opposed to a work group level), 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 organisational boundaries, performing all computations would negate the benefits gained from outsourcing computations to the cloud.

The basic idea of our scheme is to have the KDC perform the expensive setup operation. The KDC provides each server with a distinct set of keys that allow the computation of a set of functions. A client may request the computation of $F(x)$ from any server that is certified to compute F.

3.3.2 Handling Multiple Functions

Recall that in the PVC model of Parno et al. [84], the system is initialised for a single function, and that a new system must be initialised if a client wished to outsource the computation of a different function. Recall also, from Definition 2.3, that Parno et al. also introduced *multi-function VC* for the non-publicly verifiable setting; the authors left it as an open problem to devise a multi-function PVC scheme. In multi-function VC, the system is initialised independently of any functions, and evaluation keys and delegation information for different functions can be generated separately. Input data can be encoded once and used as input to multiple computations of different functions. To instantiate multi-function VC in the non-publicly verifiable setting, Parno et al. used a primitive known as *KP-ABE with outsourcing* where a trusted party can perform most of the decryption process on behalf of a user.

In this chapter, we move towards a solution for multi-function PVC. We take a different viewpoint than Parno et al. [84] did for multi-function VC and consider a 'middle-ground' model. In contrast to Parno et al., we require that clients encode their input per computation they outsource. We believe that, given an efficient outsourcing procedure, it is not unreasonable for clients to perform some work per outsourcing request. Indeed, in some cases, it could be that the data held by a client is updated as the result of each computation and so a persistent encoding would not be useful. As a result, we achieve a solution which uses a "plain" ABE scheme (of which there are many efficient, well-studied constructions) instead of the somewhat more unusual ABE with outsourced decryption [67] used by Parno et al. for the VC setting.

3.3.3 Standard Model

Our *standard model* extends the PVC architecture of Parno et al. [84] with the addition of a KDC. The entities comprise a set of clients, a set of servers and a trusted KDC. The KDC initialises the system and generates keys to enable verifiable computation. Keys to delegate computations are published for the clients, whilst keys to evaluate specific functions are given to individual servers. Clients submit computation requests, for a given input, to a particular server and publish some verification information. Servers receive encoded input values from clients and perform computations to generate an encoded result. Any

party can verify the correctness of a server's output. If the output is incorrect, the verifier may report the server to the the KDC for revocation, which will prevent the server from performing any further computations.

Figure 3.1 gives a table illustrating which entities are responsible for running each algorithm in normal verifiable outsourced computation (VC), publicly verifiable outsourced computation (PVC), the standard model of RPVC detailed in this section, and finally RPVC in the manager model which we will discuss next. The figure also includes an illustration of how the entities interact in the standard and manager models.

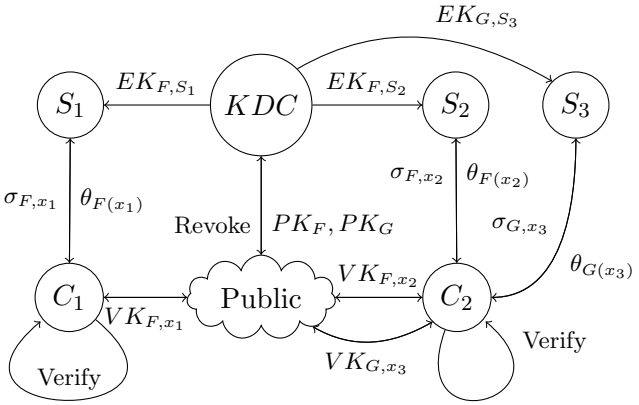
3.3.4 Manager Model

The *manager model*, in contrast, employs an additional entity known as the *manager* 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 if servers are to be rewarded per computation. A plausible scenario is that servers enrol with a manager to "sell" the use of spare resources, whilst clients subscribe to utilise these resources through the manager.

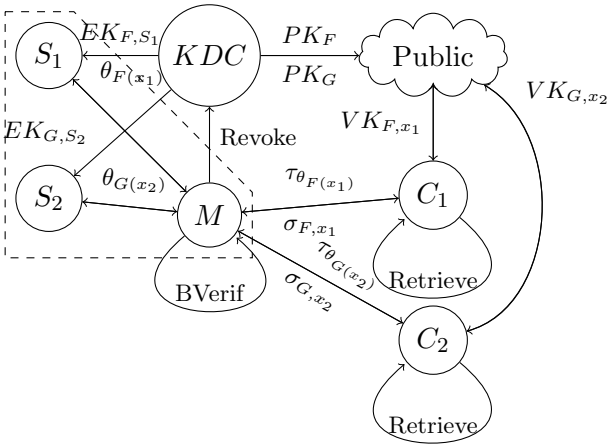
Results are returned to the manager who should be able to verify the server's work. The manager forwards correct results to the client; a misbehaving server may be reported to the KDC for revocation, and the job assigned to another server. Due to the public verifiability of current schemes, any party with access to the output and the verification token can also verify and learn the result. However, in many situations we may not desire external entities to learn sensitive computational results; yet, even so, there remain legitimate reasons for certain entities, such as the manager, to *just* verify correctness, without learning the result. Thus, we allow *blind verification* such that the manager (or other entity) may verify the validity of the computation without learning the output — that is, blind verification can determine whether a result should be accepted or rejected, but does not reveal the value $F(x)$. The delegating client generates an additional piece of information, known as a *retrieval key*, which can be shared with authorised entities and which enables the actual computational result to be learnt.

The interaction between entities in this model is illustrated in Figure 3.1b, where the

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	$C_1, C_2, \ldots$	$C_1, C_2, \ldots$	–
Blind Verify	—	—	—	M
Retrieve	—	—	—	$C_1, C_2, \ldots$



(a) Standard model



(b) Manager model

Figure 3.1: The operation of a revocable publicly verifiable outsourced computation scheme

manager is denoted by M. The manager and computational servers are shown within a dashed region to illustrate the boundaries of internal and external entities — that is, the entities not within the dashed region could all be within an organisation that wishes to utilise the external resources provided by the manager to outsource computational work. Notice that in Figure 3.1b the manager performs a blind verification operation (denoted BVerify) but only entities within the organisation may run the output retrieval algorithm to learn the actual result of the computation; there is a distinction between the capabilities

of entities external to the organisation (servers and the manager) and those internal entities (such as the clients).

3.3.5 Formal Definition

Our scheme comprises four types of entity. Firstly, a *key distribution centre* is responsible for setting up the system and being authoritative on other entities within the system. The KDC is trusted by the other entities to act honestly. Note that in the prior PVC scheme of Parno et al. [84], this role was played by a client device and would also be assumed to act honestly on behalf of all other clients. In this work, we have expanded the duties of the KDC to allow revocation of malicious servers and have referred to it as a distinct entity, rather than a client device, to make the separation of duties explicit, however we have not changed the underlying trust assumption regarding this entity. We assume that the KDC can maintain state between the execution of each algorithm but, to ease readability, do not explicitly show this as an input and output of each algorithm run by the KDC.

Secondly, the system includes a set of *clients* (which are assumed to possess limited computational resources). These clients wish to outsource computations that they lack the resources to compute locally. Each client is assumed to act honestly regarding a computation that they themselves outsource (as they wish to gain from a correct computation) but need not trust each other.

Thirdly, the system includes a set of computational *servers* that are willing to perform some set of computations on behalf of clients. These servers are not trusted by the clients to perform the computation correctly and so the servers are required to produce an efficiently verifiable proof of the correctness of their computation. We will assume that servers have unique identifiers in the form of natural numbers (note that arbitrary identifiers can also be efficiently mapped to unique natural numbers by a hash function or look up table).

The final type of entity within our system is *verifiers*. Verifiers can either be client devices or other entities; they trust the client that outsourced a particular computation but do not trust the server that actually performed the computation.

We now present a formal definition of the algorithms in a RPVC scheme.

Definition 3.4. A *revocable publicly verifiable outsourced computation scheme (RPVC)* comprises the following algorithms:

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$: run by the KDC to initialise the system and establish public parameters PP and a master secret key MK for a family of functions $\mathcal{F}$;
- $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate a public delegation key, PK_F, allowing clients to outsource computations of a function F;
- $SK_S \xleftarrow{\$} \mathsf{Register}(S, \mathrm{MK}, \mathrm{PP})$: run by the KDC to enrol a computation server S in the system and generate a signing key SK_S used to identify S;
- $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate an evaluation key $EK_{F,S}$ enabling the computation server S to perform computations of a function F;
- $(\sigma_{F,x}, VK_{F,x}, RK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, PK_F, \mathrm{PP})$: run by a client to delegate the computation of $F(x)$ to a server. The output values are: the encoded input, $\sigma_{F,x}$, of x for the function F; a verification key, $VK_{F,x}$, that is used (only) to verify correctness of the result; and a retrieval key $RK_{F,x}$ which will enable the output value $F(x)$ to be read by authorised parties;
- $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, EK_{F,S}, SK_S, \mathrm{PP})$: run by a server S holding an encoded input $\sigma_{F,x}$ of x for the function F, an evaluation key $EK_{F,S}$ for F and a signing key SK_S. The algorithm outputs an encoding, $\theta_{F(x)}$, of $F(x)$;
- $(y, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x}, RK_{F,x}, \mathrm{PP})$: verification comprises two sub-algorithms (which could be performed together in a single Verify operation as written here if the blind verification property is not required). The sub-algorithms are:
  - $(RT_{F(x)}, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{BVerif}(\theta_{F(x)}, VK_{F,x}, \mathrm{PP})$: run by any verifying party (in the standard model), or by the manager (in the manager model), in possession of an encoded output, $\theta_{F(x)}$ and a verification key $VK_{F,x}$. The output is a retrieval token, $RT_{F(x)}$, which encodes the actual output value (this can be thought of as a partial translation from $\theta_{F(x)}$ to $F(x)$). It also outputs a token $\tau_{\theta_{F(x)}}$ which is (accept, S) if the output is valid, or (reject, S) if S misbehaved;
  - $y \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x)}}, RT_{F(x)}, VK_{F,x}, RK_{F,x}, \mathrm{PP})$: run by an authorised verifier holding the retrieval key $RK_{F,x}$ to read the actual result y from the retrieval token $RT_{F(x)}$. The value of y is either $F(x)$ or the distinguished symbol $\perp$ (i.e. the algorithm fails if an invalid result is returned);

- $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(x)}}, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate update material UM if a misbehaving server is reported i.e. that the Verify algorithms returned a token $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$. If $\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ then UM is set to be a distinguished symbol $\perp$ as no user is required to be revoked. Otherwise, this algorithm revokes all evaluation keys $EK_{\cdot,S}$ of the server S, thereby preventing S from performing any further evaluations. The update material UM is a set of updated evaluation keys $EK_{\cdot,S'}$ which are issued to all servers.[2]

Although not explicitly stated, the KDC may update the public parameters PP during the execution of any algorithm. This reflects any changes that may be required to reflect changes in the user population (e.g. servers are added or removed from the system, or granted the ability to compute additional functions).

Note that if a server is not given the retrieval key $RK_{F,x}$ then it too cannot learn the output value $F(x)$ and we gain output privacy. In the above, we assume that $RK_{F,x}$ is distributed to all authorised readers of the resulting value $F(x)$ by the client that performs $\mathsf{ProbGen}$ to outsource the computation. As an example, this could be performed using a broadcast encryption scheme to the set of authorised users [52].

Intuitively, we say that a RPVC scheme is *correct* if, when all algorithms are run honestly in any arbitrary execution sequence and the result is computed by a non-revoked server, the verifying party always accepts the returned result *and* the result is correct. We can model this as a cryptographic game between a challenger and a PPT adversary; the adversary aims to find an (honestly generated) encoded output (from a non-revoked server) which either does not encode the correct result, or which does encode the correct result yet which will not be accepted by the verification algorithm.

The adversary is given access to a set of oracles; for each algorithm in Definition 3.4, the adversary is given a corresponding oracle. Each oracle executes the corresponding algorithm on arguments provided by the adversary, and returns the output of the algorithm to the adversary, as well as maintaining some internal lists, which we shall detail below. The adversary may call the Setup oracle only once (before making any other oracle queries), but can thereon call the remaining oracles any number of times and in any order.

[2] In some instantiations, it may not be necessary to issue entirely new evaluation keys to each entity. In Section 3.5, for example, we only need to issue a partially updated key.

The challenger maintains two lists, L_{Reg} and L_F; L_{Reg} is a list of tuples comprising server identities, S, and the resulting signing keys, SK_S, that have been queried to the Register oracle, whilst L_F comprises tuples of the form $(S, F, EK_{F,S})$ denoting that the server S has been queried to the Certify oracle for the function F and that $EK_{F,S}$ was generated — that is, S has been certified to compute F. When the adversary makes a Revoke query with a revocation token that identifies a server S to be revoked (that is, if $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ is given as input to the Revoke oracle), the challenger removes all entries of the form $(S, \cdot, \cdot)$ (i.e. all entries for S for any function) from L_F.

The challenger also creates and maintains a table T which records the parameters and values relating to each computation performed through the oracle queries. T is updated in the following oracles:

- ProbGen: the challenger creates a new row in T comprising 8 components, all of which are initialised to be empty; it then assigns x, F, the result $F(x)$ (computed by the challenger itself), $\sigma_{F,x}, VK_{F,x}$ and $RK_{F,x}$ to the first 6 components;
- Compute: the challenger first searches T for all rows that contain the queried $\sigma_{F,x}$ in the 4^{th} component and where the 7^{th} component is empty (i.e. those rows relating to computations on this encoded input that have not yet been performed). For each such row, r, the challenger takes the second component (the function identifier, $\tilde{F}$), and checks that there exists a server identity $\tilde{S}$ such that the tuple $(\tilde{S}, SK_{\tilde{S}}) \in L_{\text{Reg}}$ (where $SK_{\tilde{S}}$ is that given as input to the Compute oracle) *and* such that the tuple $(\tilde{S}, \tilde{F}, EK_{F,\tilde{S}}) \in L_F$ (where $EK_{F,\tilde{S}}$ is also that given as input to the Compute oracle). This check ensures that there is a currently un-revoked server (as the entries of L_F for $\tilde{S}$ have not been removed) that holds the signing key and evaluation key being used to perform the computation and which is certified for a function $\tilde{F}$ for which the encoded input $\sigma_{F,x}$ for this computation was generated[3].

  The challenger then performs the Compute algorithm on the queried $\sigma_{F,x}, EK_{F,S}$ and SK_S to produce an output $\theta_{F(x)}$. For each of the rows r of T found above, the challenger writes $\theta_{F(x)}$ and $\tilde{S}$ to the 7^{th} and 8^{th} components of r respectively. Thus, a row of T will only have a (non-empty) value in the 7^{th} component if there

[3]Note that there could be multiple such identities $\tilde{S}$ satisfying this check, if the parameters SK_S and $EK_{F,S}$ are both generated for multiple identities; in practice, good randomised algorithms should ensure that there is a negligible chance that this actually occurs. We are only interested in ensuring that there *exists* at least one non-revoked server with these parameters to perform the computation, and so if multiple such server identifiers are found then the challenger may choose $\tilde{S}$ from this set at random.

exists a non-revoked, certified server to perform the computation for which $\sigma_{F,x}$ was generated.

Thus, when complete, the entries of T will be of the form

$$(x, F, F(x), \sigma_{F,x}, VK_{F,x}, RK_{F,x}, \theta_{F(x)}, S).$$

After a polynomial number of queries, the adversary will return a value $\theta^{\star}_{F(x)}$ which he believes either encodes an incorrect computational result or which encodes a correct computational result yet which the Verify algorithm will reject (that is, an output for which the protocol execution will not be correct). The challenger first performs a look up in T for all entries containing $\theta^{\star}_{F(x)}$ in the 7^{th} position of the tuple, and stores any such entries as another table $\tilde{T}$. Note that this means that $\theta^{\star}_{F(x)}$ must have been honestly generated by the Compute oracle (else it would not be in T).

For each such row[4], the challenger uses the 5^{th} and 6^{th} components of the row (the verification key and retrieval key) to run Verify on $\theta^{\star}_{F(x)}$ to generate the outputs y and $\tau_{\theta_{F(x)}}$.

The challenger first checks whether y matches the 3^{rd} component of the row (that is, whether y is the correct computational result $F(x)$). If so, it then checks whether $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$, and if so it ends the game by returning 1 to indicate that the adversary has won the game (the adversary has found a valid encoding of a correct result, computed by a certified, non-revoked server, that the Verify algorithm is incorrectly rejecting).

On the other hand, if y did *not* match the correct value of $F(x)$, the challenger also ends the game by returning 1 to indicate that the adversary has won the game (the adversary in this case has found an incorrect result that was computed honestly by the algorithms).

If no row in $\tilde{T}$ allows the adversary to win, then the challenger outputs 0 to indicate that the adversary has lost. An RPVC scheme is *correct* if, for all PPT adversaries, the probability that the adversary wins the game described above is 0.

[4]Again, in practice it is unlikely that randomised algorithms will produce the same encoded output on different inputs but we allow it in the correctness definition.

3.4 Security Models

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:

- *Public verification.* Since multiple servers may be certified to compute multiple functions, it is important to ensure that servers cannot collude or otherwise cheat in order to convince a client of an incorrect output;
- *Revocation.* We must ensure that neither an uncertified nor a revoked server can convince a client to accept an output;
- *Vindictive servers.* We must ensure that a malicious server cannot convince a client 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 these security concerns, we define four notions of security for RPVC, each modelled as a cryptographic game. In the cases of public verifiability, revocation and vindictive managers, we also define weaker notions of security which we term *selective, semi-static* notions. This is due to the particular IND-sHRSS indirectly revocable key-policy attribute-based encryption scheme we use in our construction, which requires similar restrictions. In other words, given the current primitives we use in our construction, we cannot achieve full security for these notions, but can achieve the slightly weaker variants presented here. As our construction uses the KP-ABE scheme as a black-box, if stronger primitives are found it should be easy to swap to using these to achieve full security. In this section, we will first introduce the ideal notions of security we would like an RPVC scheme to achieve. We then, in Section 3.4.2, discuss the necessary modifications to the above notions to define the selective and semi-static notions that we can currently achieve. We also discuss their relation to the IND-sHRSS game, although it may be helpful to refer back to this discussion after the construction has been introduced in Section 3.5.

Game 3.2 $\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
2: $(F, x^{\star}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
4: $(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(x^{\star}, PK_F, \text{PP})$
5: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, \text{PP})$
6: $(y, \tau_{\theta^{\star}}) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, \text{PP})$
7: **if** $(((y, \tau_{\theta^{\star}}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ **and** $(y \neq F(x^{\star})))$ **then**
8: **return** 1
9: **else return** 0

3.4.1 Ideal Security Properties

In this section we discuss the ideal security notions we would like to achieve in the RPVC setting. Even though we cannot currently achieve all of these notions completely, we include them for completeness.

3.4.1.1 Public Verifiability

We extend the *public verifiability* game of Parno et al. [84] to formalise that servers should not be able to cheat or collude to gain an advantage in convincing any verifying party of an incorrect output (i.e. that Verify does not return accept on an encoded output $\theta_{F(x)}$ that does not in fact encode the correct output $F(x)$).

Recall that Parno et al. [84] considered the case where the adversary is limited to learning only one evaluation key (as the system is initialised for only 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 who may collude. Each server can 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 our game, we allow the adversary to learn multiple evaluation keys for different functions and associated to different servers (since evaluation keys are server-specific in our setting to enable per-server revocation). In our setting, it is likely that the set of servers will be performing computations on behalf of multiple clients simultaneously, and so the adversary is also able to learn multiple encoded inputs by performing the ProbGen algorithm.

The ideal notion of public verifiability for RPVC is presented in Game 3.2. The game

Game 3.3 $\mathbf{Exp}_{\mathcal{A}}^{\text{mPubVerif}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$;
2: $\{(F_i, x_i^{\star})\}_{i\in[n]} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$;
3: **for** $i = 1$ **to** n **do**
4: $\quad PK_{F_i} \xleftarrow{\$} \mathsf{FnInit}(F_i, \text{MK}, \text{PP})$;
5: $\quad (\sigma_{F_i,x_i^{\star}}, VK_{F_i,x_i^{\star}}, RK_{F_i,x_i^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(x_i^{\star}, PK_{F_i}, \text{PP})$;
6: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\{\sigma_{F_i,x_i^{\star}}, VK_{F_i,x_i^{\star}}, RK_{F_i,x_i^{\star}}, PK_{F_i}\}, \text{PP})$;
7: **if** $(\exists i \in [n]$ s.t. $(((y, \tau_{\theta^{\star}}) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F_i,x_i^{\star}}, RK_{F_i,x_i^{\star}}, \text{PP}))$ **and** $((y, \tau_{\theta^{\star}}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ **and** $(y \neq F_i(x_i^{\star}))))$ **then**
8: $\quad$ **return** 1
9: **else return** 0

begins with the challenger setting up the system. The adversary, $\mathcal{A}$, is given the resulting public parameters and given oracle access to $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, \text{MK}, \text{PP})$ and $\mathsf{Revoke}(\cdot, \text{MK}, \text{PP})$, denoted by $\mathcal{O}$. All oracles simply run the relevant algorithm. This models the adversary observing an existing RPVC system and corrupting various servers to learn their evaluation keys.

Eventually, the adversary will finish this query phase and outputs a choice of challenge function F with input $x^{\star}$ — the adversary will attempt to convince the challenger of an incorrect result for the computation of $F(x^{\star})$. The challenger will then run FnInit to initialise the challenge function F and generate a challenge by running $\mathsf{ProbGen}$ on this input, and give the resulting encoded input to $\mathcal{A}$. The adversary is again given oracle access and wins if it can produce an encoded output that verifies correctly but does not encode the value $F(x^{\star})$ — that is, the challenger accepts an incorrect result.

Definition 3.5. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* PubVerif *game for an RPVC construction, $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

An RPVC scheme, $\mathcal{RPVC}$, is secure with respect to public verifiability *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{PubVerif}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

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 to choose a polynomially sized set of n input values to be challenged

upon to model these simultaneous inputs, and for the adversary to win against any one of the inputs. This is shown for the case of public verifiability in Game 3.3. However, it is easy to see that this is polynomially equivalent to the case where the adversary chooses a single challenge input, as in Game 3.2.

Theorem 3.1. *Let $n \in \mathbb{N}$ be polynomial in the security parameter ℓ. Then public verifiability where the adversary may target an arbitrary set of n challenge inputs (Game 3.3) is polynomially equivalent to public verifiability where the adversary chooses a single challenge input (Game 3.2).*

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 an RPVC scheme secure when the adversary $\mathcal{A}_S$ makes a single choice of challenge input. In order to obtain a contradiction, suppose $\mathcal{A}_M$ is an adversary with non-negligible advantage δ against $\mathcal{RPVC}$ when it can make multiple challenge choices. We show that $\mathcal{A}_S$ could use $\mathcal{A}_M$ 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 Game 3.2 with $\mathcal{A}_S$ who in turn acts as the challenger for $\mathcal{A}_M$ in Game 3.3.

1. $\mathcal{C}$ runs Setup and sends the resulting public parameters to $\mathcal{A}_S$ who simply forwards them to $\mathcal{A}_M$.
2. $\mathcal{A}_M$ makes oracle queries which $\mathcal{A}_S$ passes to $\mathcal{C}$ and forwards the response to $\mathcal{A}_M$.
3. $\mathcal{A}_M$ will return a set of n challenge inputs $\{(F_i, x_i^\star)\}_{i\in[n]}$.
4. $\mathcal{A}_S$ chooses one of these challenges 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 $x^\star$ respectively and provides oracle access to $\mathcal{A}_S$.
6. $\mathcal{A}_S$ can query the FnInit oracle for all other PK_{F_i} and run $\mathsf{ProbGen}$ for the remaining inputs $\{x_i^\star\}_{i\in[n]} \setminus x^\star$ (as $\mathsf{ProbGen}$ relies only on public information), and returns

the whole set to $\mathcal{A}_M$. Since no other query was made between $\mathcal{C}$ generating the challenge and these ProbGen queries, the system parameters have not changed and all challenges are consistent.

7. $\mathcal{A}_M$ makes oracle queries which $\mathcal{A}_S$ again forwards to $\mathcal{C}$, and then outputs a challenge output $\theta^\star$.

8. Let $x_j^\star$ be the challenge input that $\theta^\star$ corresponds to — that is, since $\mathcal{A}_M$ is assumed to be successful, $\mathsf{Verify}(\theta^\star, VK_{F_j,x_j^\star}, RK_{F_j,x_j^\star}, \mathrm{PP})$ does not return $(\perp, (\mathsf{reject}, \cdot))$ 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 $\mathcal{A}_S$ forwards $\theta^\star$ to $\mathcal{C}$ as its result. Otherwise, $\mathcal{A}_S$ stops.

Hence,

$$Adv_{\mathcal{A}_S}^{\text{PubVerif}}(\mathcal{RPVC}, 1^\ell, \mathcal{F}) = \Pr[x^\star = x_j^\star] \cdot Adv_{\mathcal{A}_M}^{\text{MPubVerif}}(\mathcal{RPVC}, 1^\ell, \mathcal{F}) = \frac{\delta}{n}.$$

Now, since we assumed δ was non-negligible and n is polynomial, we conclude that $Adv_{\mathcal{A}_S}^{\text{PubVerif}}(\mathcal{RPVC}, 1^\ell, \mathcal{F})$ 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. □

A similar argument holds for the remaining games, and henceforth we only consider single challenges.

3.4.1.2 Revocation

The notion of *revocation* requires that, if a server is detected as misbehaving (i.e. a server S returns a result that causes the Verify algorithm to output $(\perp, (\mathsf{reject}, S))$), then any subsequent computations by S should be rejected (even if the result is correct). We aim to remove any incentive for an malicious server to attempt to provide an outsourcing service since it knows the result will not be accepted, and we may punish and further discourage malicious servers by removing their ability to perform work (and earn rewards). Finally, from a privacy perspective, we may not wish to supply input data to a server that is known to be untrustworthy.

Game 3.4 $\mathbf{Exp}^{\text{REV}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]$

$\texttt{chall} \leftarrow \texttt{false}$
$Q_{\text{Rev}} \leftarrow \epsilon$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$
$(F, x^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
$PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
$\texttt{chall} \leftarrow \texttt{true}$
$(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}) \xleftarrow{\$} \mathsf{ProbGen}(x^\star, PK_F, \text{PP})$
$\theta^\star \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{x^\star}, VK_{F,x^\star}, RK_{F,x^\star}, PK_F, \text{PP})$
if $((((y, (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\theta^\star, VK_{F,x^\star}, RK_{F,x^\star}, \text{PP}))$ **and** $(S \in Q_{\text{Rev}}))$ **then**
    return 1
else return 0

Oracle 3.3 $\mathcal{O}^{\mathsf{Certify}}(S, F', \text{MK}, \text{PP})$

if $(\texttt{chall} = \texttt{false})$ **then** $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S$
return $\mathsf{Certify}(S, F', \text{MK}, \text{PP})$

Oracle 3.4 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(x)}}, F', \text{MK}, \text{PP})$

$UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F'(x)}}, F', \text{MK}, \text{PP})$
if $(UM \neq \perp$ **and** $\texttt{chall} = \texttt{false})$ **then** $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \cup S$
return UM

The ideal notion of revocation, presented in Game 3.4, begins by declaring a Boolean flag `chall` which is initially set to `false` and an (empty) list Q_{Rev} which servers will be added to when 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 servers that are revoked at the challenge time and hence all servers that, if an adversary can output a result 'from' one of these servers and have it accepted, will count as a win for the adversary.

The game proceeds in a similar fashion to public verifiability with the challenger running Setup to initialise the system and providing the public parameters to the adversary. The adversary is also given oracle access to the functions $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, \text{MK}, \text{PP})$ and $\mathsf{Revoke}(\cdot, \text{MK}, \text{PP})$, denoted by $\mathcal{O}$. All oracles simply run the relevant algorithms with the exception of $\mathsf{Certify}$ and Revoke which additionally maintain the list of revoked entities as mentioned above and as specified in Oracles 3.3 and 3.4 respectively. After the adversary has finished this query phase, it outputs a choice of challenge function F and challenge input $x^\star$. The challenger runs FnInit for F 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, 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 outputs accept for a server that was revoked when

Game 3.5 $\mathbf{Exp}_{\mathcal{A}}^{\text{VINDS}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $Q_{\text{Reg}} \leftarrow \epsilon$
2: $Q_{\text{chall}} \leftarrow \epsilon$
3: $\tilde{S} \leftarrow \perp$
4: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
5: $(F, x^{\star}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
6: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
7: $(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(x^{\star}, PK_F, \text{PP})$
8: $\tilde{S} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, \text{PP})$
9: **if** $(\tilde{S} \in Q_{\text{chall}})$ **then return** $\perp$
10: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}, \mathcal{O}^{\mathsf{Compute}}}(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, \text{PP})$
11: $(y, \tau_{\theta^{\star}}) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, \text{PP})$
12: **if** $(((y, \tau_{\theta^{\star}}) = (\perp, (\mathsf{reject}, \tilde{S})))$ **and** $(\perp \nleftarrow \mathsf{Revoke}(\tau_{\theta^{\star}}, \text{MK}, \text{PP})))$ **then**
13: **return** 1
14: **else return** 0

the challenge was generated (even if the result is correct).

Definition 3.6. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* Rev *game for an RPVC construction, $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{REV}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{REV}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]\right].$$

An RPVC scheme, $\mathcal{RPVC}$, is secure with respect to revocation *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{REV}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

3.4.1.3 Vindictive Servers

This notion is particularly relevant in the context of the manager model. Recall that clients submit 'jobs' to a manager who distributes the work to a server selected from a pool of available computational servers. Thus, the client may not *a priori* know the identity of the selected server. 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 ID leading to the revocation (and punishment) of an honest server. The pool of available servers for future computations is therefore reduced in size, potentially to advantage of the malicious server.

In Game 3.5, the challenger maintains a list of currently registered entities Q_{Reg}, a list

Oracle 3.5 $\mathcal{O}^{\mathsf{Register}}(S, \mathrm{MK}, \mathrm{PP})$

1: **if** $(S = \tilde{S})$ **then return** $\perp$
2: **if** $(S, \cdot) \notin Q_{\mathrm{Reg}}$ **then**
3: $\quad SK_S \xleftarrow{\$} \mathsf{Register}(S, \mathrm{MK}, \mathrm{PP})$
4: $\quad Q_{\mathrm{Reg}} \leftarrow Q_{\mathrm{Reg}} \cup (S, SK_S)$
5: $\quad Q_{\mathrm{chall}} \leftarrow Q_{\mathrm{chall}} \cup S$
6: **return** SK_S

Oracle 3.6 $\mathcal{O}^{\mathsf{Register2}}(S, \mathrm{MK}, \mathrm{PP})$

1: **if** $(S, \cdot) \notin Q_{\mathrm{Reg}}$ **then**
2: $\quad SK_S \xleftarrow{\$} \mathsf{Register}(S, \mathrm{MK}, \mathrm{PP})$
3: $\quad Q_{\mathrm{Reg}} \leftarrow Q_{\mathrm{Reg}} \cup (S, SK_S)$
4: **return** $\perp$

Oracle 3.7 $\mathcal{O}^{\mathsf{Compute}}(\sigma_{F',x}, EK_{F',\tilde{S}}, SK_{\tilde{S}}, \mathrm{PP})$

1: **if** $((x = x^\star)$ **and** $(F' = F))$ **then return** $\perp$
2: **return** $\mathsf{Compute}(\sigma_{F',x}, EK_{F',\tilde{S}}, SK_{\tilde{S}}, \mathrm{PP})$

Q_{chall} of entities for which the adversary has learnt the signing key, and defines $\tilde{S}$, the target server identity, to be initially $\perp$ until the adversary chooses its target. The game proceeds like the previous ones except that, on lines 8 and 10, the adversary selects the ID for a target server, $\tilde{S}$, and then generates an encoded output that he hopes will result in the revocation of $\tilde{S}$. He is given oracle access to $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register2}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Revoke}(\cdot, \mathrm{MK}, \mathrm{PP})$, denoted by $\mathcal{O}$. These oracles, described below, must ensure that the adversary is never issued the signing key $SK_{\tilde{S}}$ as he would then trivially be able to act like $\tilde{S}$ and win the game. For the same reason, when choosing the target server $\tilde{S}$ on line 8, the adversary loses the game if he has previously learnt the signing key $SK_{\tilde{S}}$ (i.e. $\tilde{S}$ is listed on Q_{chall}). On line 10, the adversary is also given access to a $\mathsf{Compute}$ oracle $\mathcal{O}^{\mathsf{Compute}}(\cdot, EK_{\cdot,\tilde{S}}, SK_{\tilde{S}}, \mathrm{PP})$ which allows the adversary to view evaluation results generated (only) by the target server $\tilde{S}$; this models the adversary observing $\tilde{S}$ prior to attacking.

The $\mathsf{Register}$ oracle, presented in Oracle 3.5, returns a failure symbol $\perp$ if queried for the challenge identity $\tilde{S}$, and otherwise adds the queried server S to the list Q_{chall} (as it will issue the signing key SK_S). On line 8 the adversary is additionally given access to a modified $\mathsf{Register}$ oracle, defined in Oracle 3.6. This $\mathsf{Register2}$ oracle performs the $\mathsf{Register}$ algorithm but *does not* return the resulting key SK_S (it may, however, update the public parameters to reflect the additional registered entity). The adversary may query *any* identity to $\mathsf{Register2}$ (including $\tilde{S}$). The purpose of this oracle is to allow the adversary to enrol servers in the system without learning the corresponding server specific secrets; this models the adversary observing uncorrupted servers within the RPVC system which

he can target for revocation. Clearly, if the adversary corrupts a server and learns the signing key, then it can output an incorrect answer and trivially cause the server to be revoked; the goal in this game is to cause an *honest*, uncorrupted, server to be revoked.

Both Oracle 3.5 and 3.6 first check whether the queried server S has already been added to the list Q_{Reg}; if not, both algorithms run Register and add the server identity and signing key to Q_{Reg}. If the server was already listed in Q_{Reg} then the Register oracle returns the stored signing key for the server, whilst the Register2 oracle will return $\perp$. Thus, both oracles will, together, generate a single signing key per server.

Finally, to prevent a trivial win in which the adversary simply forwards a prior result actually generated by $\tilde{S}$, we restrict queries to the Compute oracle in Oracle 3.7. The adversary cannot ask for the evaluation of the challenge computation $F(x^\star)$ from $\tilde{S}$. Note that for all other servers, the adversary can run Compute itself using parameters learnt from other queries.

The adversary wins if the challenger believes $\tilde{S}$ generated y and revokes $\tilde{S}$.

Definition 3.7. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* VINDS *game for an RPVC construction, $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

An RPVC scheme, $\mathcal{RPVC}$, is secure with respect to vindictive servers *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{VINDS}}(\mathcal{RPVC}, 1^\ell, \mathcal{F}) \leqslant \text{negl}(\ell).$$

3.4.1.4 Vindictive Managers

The notion of *vindictive managers* is a natural extension of public verifiability to the manager model where a vindictive manager may attempt to provide a client with an incorrect answer. If clients subscribe to a pool of servers maintained by a manager, the manager may not wish to own up to incorrect results to avoid losing business, but also may not have the available resources within its system to recompute an incorrect computation.

Game 3.6 $\mathbf{Exp}_{\mathcal{A}}^{\text{VINDM}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
2: $(F, x^{\star}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
4: $S \xleftarrow{\$} \mathcal{U}_{\text{ID}}$
5: $SK_S \xleftarrow{\$} \mathsf{Register}(S, \text{MK}, \text{PP})$
6: $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \text{MK}, \text{PP})$
7: $(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(x^{\star}, PK_F, \text{PP})$
8: $\theta_{F(x^{\star})} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x^{\star}}, EK_{F,S}, SK_S, \text{PP})$
9: $(RT_{F(x^{\star})}, \tau_{\theta_{F(x^{\star})}}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^{\star}}, \theta_{F(x^{\star})}, VK_{F,x^{\star}}, PK_F, \text{PP})$
10: **if** $((y \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x^{\star})}}, RT_{F(x^{\star})}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, \text{PP}))$ **and** $(y \neq F(x^{\star}))$ **and** $(y \neq \perp))$ **then**
11: **return** 1
12: **else return** 0

Thus, occasionally, the manager may try to send an incorrect result to the client with supposed assurance that it is correct.

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

The ideal notion of security against vindictive managers, presented in Game 3.6, begins with the challenger initialising the system as usual. The adversary is given oracle access to the functions $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, \text{MK}, \text{PP})$ and $\mathsf{Revoke}(\cdot, \text{MK}, \text{PP})$, denoted by $\mathcal{O}$. Each oracle simply runs the relevant algorithm. After a polynomial number of queries, the adversary outputs a challenge function F and input $x^{\star}$. The challenger runs FnInit and selects a server identity uniformly at random from the space of all identities $\mathcal{U}_{\text{ID}}$; this identity will be used to generate the challenge. It runs Register and Certify for this server (if not already done during the oracle queries), creates a problem instance by running ProbGen on $x^{\star}$ and finally runs Compute on the generated encoded input. The adversary is then given the encoded input, verification key, the output from Compute and oracle access as above. The adversary must output a retrieval token $RT_{F(x^{\star})}$ and an acceptance token $\tau_{\theta_{F(x^{\star})}}$. The challenger runs Retrieve on $RT_{F(x)}$ to get an output value y, and the adversary wins if the challenger accepts this output and $y \neq F(x^{\star})$

(i.e. the retrieved result is incorrect).

Definition 3.8. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* VINDM *game for an RPVC construction, $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

An RPVC scheme, $\mathcal{RPVC}$, is secure with respect to vindictive managers *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{VINDM}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

3.4.2 Restricted Security Notions

As mentioned, using current primitives, we cannot achieve the ideal notions of public verifiability, revocation or vindictive managers. We therefore introduce slightly weakened versions of these security notions to reflect the similar restrictions placed on our construction by the IND-sHRSS indirectly revocable KP-ABE scheme we use (see Section 3.2.2). As we use this primitive as a black box, it should be easy to achieve the ideal security notions if a fully secure primitive with the same functionality is found; this will be the subject of future work.

These variants introduce two additional restrictions on the adversary. Firstly, a *selective* restriction requires the adversary to declare the set of input values to be used in the challenge stage before seeing the public parameters. This is in contrast to the full game where the inputs are chosen after the adversary has oracle access to the system. As mentioned in Section 2.8.5, this restriction has similarly been used in many ABE schemes to give a heuristic level of security when full security is difficult to achieve, as it allows the system to be initialised with a particular attack target in mind. A possible motivation for this restriction in practice is when there are high-value targets within the system which are most likely to be attacked.

Secondly, a *semi-static* restriction requires the adversary to declare a list $\overline{R}$ of servers that must be revoked when the challenge encoded inputs are generated from ProbGen. The adversary must do this *before* receiving oracle access. This restriction arises from

the revocation mechanism of the revocable KP-ABE scheme and means that oracles are able to refuse to respond to queries that would lead to a trivial win, e.g. that would issue functional keys to users that should be revoked for the challenge time period.

To remove the first (selective) restriction, we require a fully secure indirectly revocable KP-ABE scheme. To remove the second (semi-static) restriction, we require an adaptive notion of revocation. At present, instantiating such a primitive is an open problem.[5]

To implement the semi-static restriction, we must add some additional steps to each security notion. The challenger must now define two additional parameters: t and Q_{Rev}. The variable t models system time and is initialised to 1. It is incremented each time a revocation query is made to illustrate that keys generated at prior time periods may no longer function.

In the IND-sHRSS game [14], update keys are associated with a time period and queries can be made for update keys for arbitrary time periods. However, in our setting, we consider an interactive protocol; as such, time must increase monotonically. The time period is important in the consideration of the revocation functionality — a user should not have access to a secret decryption key *and* an update key for any time period which together would form a functional decryption key for the challenge ciphertext and would allow a trivial win. The adversary in the IND-sHRSS game selects a time period for the challenge as well as a challenge input. In our game, however, we parametrise the adversary on the number, q, of queries he may make to his oracles and define security over all choices of q. In particular, we restrict the adversary to make $q_t \leqslant q$ queries to the Revoke oracle in its first query phase (before the challenge is generated). Since t is incremented only when a Revoke query is made, the challenge will occur at time $t^\star = q_t$, and hence the challenger may select $t^\star$ as its challenge time in a reductive proof.

The other additional parameter, Q_{Rev}, is a list (initialised to be empty) comprising all servers that are revoked during the current time period. Servers are added to the list when the Revoke oracle is queried with a reject token, and are removed from the list if subsequently certified for a function. Thus, unless *one* server is added or removed as mentioned, the revocation list remains consistent over consecutive oracle queries to model realistic system evolution (whereas, in the IND-sHRSS game, the revocation list can be

[5] Attrapadung and Imai [14] defined a notion with adaptive queries but did not provide an instantiation.

Game 3.7 $\mathbf{Exp}^{\text{sPubVerif}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]$

1: $(F, x^\star) \xleftarrow{\$} \mathcal{A}(1^\ell, \mathcal{F})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
4: $(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}) \xleftarrow{\$} \mathsf{ProbGen}(x^\star, PK_F, \text{PP})$
5: $\theta^\star \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}, PK_F, \text{PP})$
6: $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, VK_{F,x^\star}, RK_{F,x^\star}, \text{PP})$
7: **if** $(((y, \tau_{\theta^\star}) \neq (\perp, (\mathsf{reject}, \cdot))))$ **and** $(y \neq F(x^\star)))$ **then**
8: **return** 1
9: **else return** 0

dynamically changed per query). By the semi-static restriction, the adversary must choose a revocation list $\overline{R}$ detailing all servers that should be revoked at the challenge time. If the actual list of revoked servers, Q_{Rev}, at the challenge time $t^\star$ is *not* a superset of this list (i.e. there exists a server that the adversary claimed would be revoked but actually is not) then the adversary has not requested a suitable sequence of oracle queries and loses the game to avoid a trivial win — the oracles that responded to queries based on $\overline{R}$ may well have issued key material that would allow the adversary to respond trivially to the challenge.

To avoid other trivial wins, we must restrict the oracle queries that the adversary may make such that he cannot obtain *both* a secret key and an update key (i.e. a full evaluation key in our terminology) for a server that is revoked at the challenge time. Otherwise, if the adversary could obtain a valid update key for the challenge time and a secret key, he can form a full, functional evaluation key which will evaluate the challenge encoded input and form a correct result; clearly, a revoked server would not have such an ability in practice.

Note that unlike the oracle queries in the IND-sHRSS game, *both* "KeyGen" (Certify) queries and "Update KeyGen" (Revoke) queries include a notion of identity and Revoke queries cannot be made for arbitrary time periods. Hence the oracle restrictions in these games differ slightly from those in the IND-sHRSS game but capture the same principle.

3.4.2.1 Selective Public Verifiability

We define a *selective notion of public verifiability* in Game 3.7. The only difference between this and the ideal notion in Game 3.2 is that, in the selective notion, the adversary chooses the challenge inputs F and $x^\star$ *before* Setup is run. Note that in this notion, we do not

Game 3.8 $\mathbf{Exp}_{\mathcal{A}}^{\text{sSS-Rev}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}, q_t\right]$

1: $(F, x^{\star}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{F}, q_t)$
2: $Q_{\text{Rev}} \leftarrow \epsilon$
3: $t \leftarrow 1$
4: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
5: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
6: $\overline{R} \xleftarrow{\$} \mathcal{A}(PK_F, \text{PP})$
7: $\mathcal{A}^{\mathcal{O}}(PK_F, \text{PP})$
8: **if** $(\overline{R} \not\subseteq Q_{\text{Rev}})$ **then return** 0
9: $(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(x^{\star}, PK_F, \text{PP})$
10: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, \text{PP})$
11: **if** $((((y, (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, \text{PP}))$ **and** $(S \in \overline{R}))$ **then**
12: **return** 1
13: **else return** 0

Oracle 3.8 $\mathcal{O}^{\mathsf{Certify}}(S, F', \text{MK}, \text{PP})$

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** $\mathsf{Certify}(S, F', \text{MK}, \text{PP})$

require the semi-static restriction since the revocation mechanism is not considered as part of the winning condition.

Definition 3.9. *The* advantage *of a PPT adversary* $\mathcal{A}$ *in the* sPubVerif *game for an RPVC construction,* $\mathcal{RPVC}$*, for a family of functions* $\mathcal{F}$ *is defined as:*

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

An RPVC scheme, $\mathcal{RPVC}$*, is* secure with respect to selective public verifiability *if, for all PPT adversaries* $\mathcal{A}$*,*

$$Adv_{\mathcal{A}}^{\text{sPubVerif}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

3.4.2.2 Selective, Semi-static Revocation

The *selective, semi-static notion of revocation* is given in Game 3.8 and Oracles 3.8 and 3.9. Recall that an adversary wins in this notion if it can output *any* result that is formed by a revoked entity yet is accepted by the challenger. Since revocation is an inherent requirement in the winning condition, we require both the selective and the semi-static restrictions to accommodate the IND-sHRSS game.

The adversary first selects an input value and function F to be outsourced. The challenger

Oracle 3.9 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(x)}}, \mathrm{MK}, \mathrm{PP})$

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

initialises an (empty) list of currently revoked entities Q_{Rev} and a time parameter t before running Setup and FnInit to create a public delegation key for the function F (lines 2 to 5). The adversary is given the generated public parameters and must output a list $\overline{R}$ of servers to be revoked when the challenge is created. It is then, on line 7, given oracle access to the functions $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Revoke}(\cdot, \mathrm{MK}, \mathrm{PP})$, denoted by $\mathcal{O}$.

The challenger responds to Certify and Revoke queries as detailed in Oracles 3.8 and 3.9 respectively, whilst all other oracles simply run the relevant algorithm and return the results to the adversary. $\mathcal{C}$ must ensure that Q_{Rev} is kept up-to-date by adding or removing the queried entity, and in the case of revocation must increment the time parameter. It also ensures that issued keys will not lead to a trivial win. In Oracle 3.8, for the Certify algorithm, this amounts to not issuing an evaluation key $EK_{F,S}$ for the challenge function F and for a server S that may not be revoked at the time that the challenge is generated — otherwise, an issued key may be valid and functional for the challenge and the adversary can trivially evaluate the challenge computation as a non-revoked server.

Recall that the adversary is parameterised to make exactly q_t revocation queries and that the time period is incremented only during the revocation algorithm; the challenge time period is therefore when $t = q_t$. An evaluation key for a server S should also not be issued by Oracle 3.8 if requested during the challenge time period q_t and if there exists a server (other than S as it is about to be certified and removed from Q_{Rev}) that should be revoked according to challenge revocation list $\overline{R}$ chosen by the adversary but has not actually been revoked (is not listed on Q_{Rev}). The intuition behind this restriction is that Certify issues an evaluation key which is functional for the current time period (it may only be disabled by revoking the server but this would increment the time period too). In particular, as in our construction, Certify may reveal update material (such as that generated by the latest revocation procedure) that enables evaluation keys to be functional for the current time period. Therefore, if such update material is issued, any non-revoked evaluation key may be updated for the current, challenge time period q_t and may be used to evaluate

computations and return valid results that are accepted by the challenger. If the updated evaluation key belongs to a server that was listed on $\overline{R}$, then this would count as a win for the adversary (as the adversary claimed this server would be revoked at this point). This counts as a trivial win as the accepted result was not generated by a revoked server.

Oracle 3.9 first increments the time parameter t and returns $\perp$ if the queried token is $(\mathsf{accept}, \cdot)$ i.e. there is no server to revoke; this replicates the expected behaviour of the Revoke algorithm. Since t is still incremented, the adversary may query acceptance tokens to Revoke in order to progress the system time without altering the revocation list if desired. To avoid trivial wins, if a query is made at the challenge time i.e. $t = t^\star$, the challenger must return $\perp$ if the challenge revocation list $\overline{R}$ is not a subset of the current revocation list Q_{Rev} (including the queried server S as this is about to be revoked). That is, $\perp$ is returned if there exists a server, other than S, listed on $\overline{R}$ (and hence that should be revoked at the challenge time period i.e. the current time period), but is not actually on the list of currently revoked servers. This requirement stems from the same issue regarding update material as above.

The adversary finishes this query phase on line 7 after making a polynomial number of queries, q, including exactly q_t Revoke queries. It does not return a value other than signalling the challenger that it may proceed with the remainder of the game. The challenger checks that the queries made by the adversary has indeed generated a list of revoked entities that is a superset of $\overline{R}$. If not (i.e. there is a server that the adversary included on $\overline{R}$ but is not currently revoked), then the adversary loses the game as it did not choose $\overline{R}$ or its queries appropriately. Otherwise, the challenger generates the challenge by running ProbGen on $x^\star$. The adversary is given the resulting encoded input and oracle access again, and wins if it outputs *any* result (even a correct encoding of $F(x^\star)$) that is accepted as a valid response from any server that was revoked at the time of the challenge, which the adversary chose to be (at least) those servers on $\overline{R}$.

Definition 3.10. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number, q, of oracle queries, of which q_t are* Revoke *queries, in the* sSS-Rev *game for an RPVC construction, $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{sSS-Rev}}(\mathcal{RPVC}, 1^\ell, \mathcal{F}, q_t) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{sSS-Rev}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}, q_t\right]\right].$$

Game 3.9 $\mathbf{Exp}_{\mathcal{A}}^{\text{sVINDM}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(F, x^{\star}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{F})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
4: $S \xleftarrow{\$} \mathcal{U}_{\text{ID}}$
5: $SK_S \xleftarrow{\$} \mathsf{Register}(S, \text{MK}, \text{PP})$
6: $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \text{MK}, \text{PP})$
7: $(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(x^{\star}, PK_F, \text{PP})$
8: $\theta_{F(x^{\star})} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x^{\star}}, EK_{F,S}, SK_S, \text{PP})$
9: $(RT_{F(x^{\star})}, \tau_{\theta_{F(x^{\star})}}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^{\star}}, \theta_{F(x^{\star})}, VK_{F,x^{\star}}, PK_F, \text{PP})$
10: $y \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x^{\star})}}, RT_{F(x^{\star})}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, \text{PP})$
11: **if** $((y \neq F(x^{\star}))$ **and** $(y \neq \perp))$ **then**
12: **return** 1
13: **else return** 0

An RPVC scheme, $\mathcal{RPVC}$, is secure with respect to selective, semi-static revocation *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{sSS-REV}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}, q_t) \leqslant \text{negl}(\ell).$$

3.4.2.3 Selective Vindictive Managers

As with the public verifiability notion, vindictive managers does not rely on revocation for the winning condition and so we only require the selective restriction, which is presented in Game 3.9.

The adversary first selects its challenge pair of F and $x^{\star}$. The challenger sets up the system, runs FnInit for F and then selects a server uniformly at random from the space of server identities $\mathcal{U}_{\text{ID}}$. This server will be used to generate the challenge parameters for the adversary. The challenger registers and certifies S for F, and runs $\mathsf{ProbGen}$ on the challenge input, before finally running $\mathsf{Compute}$ to generate an encoded output $\theta_{F(x^{\star})}$. The adversary is then given the encoded input, verification key and $\theta_{F(x^{\star})}$, as well as oracle access to the functions $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, \text{MK}, \text{PP})$ and $\mathsf{Revoke}(\cdot, \text{MK}, \text{PP})$, denoted by $\mathcal{O}$. It must output a retrieval token $RT_{F(x^{\star})}$ and an acceptance token $\tau_{\theta_{F(x^{\star})}}$. The challenger runs $\mathsf{Retrieve}$ on $RT_{F(x^{\star})}$ to get an output value y; the adversary wins if the challenger accepts this output and $y \neq F(x^{\star})$.

Definition 3.11. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* sVINDM *game for an RPVC construction, $\mathcal{RPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

An RPVC scheme, $\mathcal{RPVC}$, is secure with respect to selective vindictive managers *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv^{\text{sVIndM}}_{\mathcal{A}}(\mathcal{RPVC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

3.5 Construction

We now provide an instantiation of an RPVC scheme. Our construction is based on that used by Parno et al. [84] (summarised in Section 3.2.1) which uses key-policy attribute-based encryption (KP-ABE) as a black-box to outsource the computation of a (monotone) Boolean function. We restrict our attention to (monotone) Boolean functions closed under complement, and in particular the complexity class NC^1 which includes all circuits of depth $\mathcal{O}(\log n)$ [10]. Thus, functions we can outsource can be built from common operations such as AND and OR gates, NOT gates (if using a non-monotonic ABE scheme [80]), equality and comparison operators, arithmetic operators and regular expressions. Note that although our scheme only admits Boolean functions (with single bit output) and therefore seems somewhat restrictive, it is possible to outsource the evaluation of functions with n-bit outputs by outsourcing n different functions, each of which returns the single bit in position i, $1 \leqslant i \leqslant n$.

Clearly, different function families will require different constructions from that presented here for Boolean functions. As a trivial example, verifiable outsourced evaluation of the identity function may only require the server to sign the input. On the other hand, despite it seemingly being a natural choice for outsourcing, it is not clear how a VC scheme for determining if a statement is in an NP-complete language could be instantiated. A solution for such problems is by definition difficult to find so should be outsourced, whilst a candidate solution can be verified efficiently. However, a malicious server could simply claim that a solution cannot be found for the given problem instance, and the restricted client could not verify the correctness of this statement without searching the solution space itself.

Recall from Section 3.2.1 that, using a single ABE system, if a computational server returns

$\perp$ in response to a Boolean computation request then the verifier is unable to determine whether $F(x) = 0$ *or* whether the server misbehaved. To avoid this issue, we restrict the family of functions $\mathcal{F}$ we can evaluate to be the set of Boolean functions closed under complement. That is, if F belongs to $\mathcal{F}$ then $\overline{F}$, where $\overline{F}(x) = F(x) \oplus 1$, also belongs to $\mathcal{F}$. The client encrypts two random messages m_0 and m_1 and the server must decrypt each using keys associated with policies for the functions F and $\overline{F}$ respectively. Since exactly *one* of F and $\overline{F}$ will be satisfied by any given input, exactly *one* plaintext will be successfully recovered by the Decrypt algorithm. Note that, to achieve blind verification in our construction, the retrieval key $RK_{F,x}$ will be a single bit b which permutes the order of the ciphertexts in $\sigma_{F,x}$ and hence the order of the plaintexts in $\theta_{F(x)}$. Thus, a well-formed response $\theta_{F(x)}$, comprising recovered plaintexts (d_b, d_{1-b}), satisfies the following:

$$(d_b, d_{1-b}) = \begin{cases} (m_b, \perp), & \text{if } F(x) = 1; \\ (\perp, m_{1-b}), & \text{if } F(x) = 0. \end{cases} \tag{3.2}$$

Hence, the client will be able to detect whether the server has misbehaved or deduce the value of $F(x)$ otherwise.

3.5.1 Technical Details

We use an *indirectly revocable KP-ABE scheme* comprising the algorithms ABE.Setup, ABE.KeyGen, ABE.KeyUpdate, ABE.Encrypt and ABE.Decrypt. We also use a signature scheme with algorithms Sig.KeyGen, Sig.Sign and Sig.Verify, and a one-way function g. Let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\mathcal{F}}$ be the universe of attributes for the ABE scheme, formed as the union of the following 'sub-universes':

- $\mathcal{U}_{\text{attr}}$ comprises attributes that form characteristic tuples for input data, as detailed in Section 3.2.1;
- $\mathcal{U}_{\text{ID}}$ comprises attributes representing entity identifiers;
- $\mathcal{U}_{\text{time}}$ comprises attributes representing time periods output by a time source $\mathbb{T}$;
- $\mathcal{U}_{\mathcal{F}}$ comprises attributes that represent functions in $\mathcal{F}$.

3.5.1.1 Handling Multiple Servers

The PVC scheme of Parno et al. [84], which permitted the evaluation of only a single function, required a one-key IND-CPA notion of security for the underlying KP-ABE scheme. This is a more relaxed notion than considered in the vast majority of the ABE literature, in which the adversary is limited to learning just one decryption key. Parno et al. could use this property due to their restricted system model where a single server is certified for a single function per set of public parameters (the client must set up a new ABE system per function and per server).

In our setting, we aim to accommodate multiple computational servers and the outsourced evaluation of multiple functions, and as such the adversary is given a KeyGen oracle which allows the generation of polynomially many decryption keys for different servers and different functions. The scheme must prevent collusion between holders of different decryption keys and prevent keys for a particular function being used to 'compute' different functions and have the results accepted by a client. Collusion is prevented by the standard IND-CPA notions of security for ABE schemes, whereas the misuse of evaluation keys for different functions in discussed in the next section.

3.5.1.2 Handling Multiple Functions

As we discussed in Section 3.3.2, we wish to handle *multiple functions* within a single PVC system. To achieve multi-function VC (in the non-publicly verifiable setting), Parno et al. required the somewhat complex primitive of KP-ABE with Outsourcing [67]. In this work, we take a different approach. We believe that, in practical environments, it is unrealistic to expect a server to compute just a single function (as this would presumably have limited marketable applications for a cloud service provider), and we also believe that it is a reasonable expectation of cost to prepare an encoded input per computation (assuming the cost of doing so is reasonably low), especially given that the input data to different functions may well differ. Thus, whereas Parno et al. use complex primitives to allow an encoded input to be used for computations of different functions on the same data, we apply a simple encoding trick which allows servers to hold decryption keys for multiple functions in the *publicly verifiable* setting and which requires the more standard, well-studied multi-key notion of security usually considered for ABE schemes.

We now describe our encoding to ensure that an adversary holding an evaluation key for a function G cannot use this to forge a valid looking result for $F(x)$. First note that if one were to extend the scheme of Parno et al. trivially using KP-ABE and allow an adversary access to multiple keys (through a KeyGen oracle), then this solution would indeed be vulnerable to the above attack. A client would encrypt two random messages both associated with the same attribute set x within different ABE systems. The malicious server must successfully decrypt exactly one of these messages using its evaluation key (comprising decryption keys for G and $\overline{G}$). Now, any such pair of keys for a function and its complement will be able to decrypt exactly one ciphertext because on a given input, either G or $\overline{G}$ is satisfied. Thus, even without an evaluation key for F, a malicious server can still decrypt one message and return the result $G(x)$ which the client will accept (as the plaintext matches the verification key) as the outcome of $F(x)$.

Let us instead assume a bijective mapping $\phi : \mathcal{F} \rightarrow \mathcal{U}_{\mathcal{F}}$ that maps functions in the acceptable function family $\mathcal{F}$ for the RPVC scheme to attributes in the sub-universe $\mathcal{U}_{\mathcal{F}}$. We then add a conjunctive clause (an additional AND gate) to each Boolean function $F \in \mathcal{F}$ requiring the presence of the appropriate function label $\phi(F)$ in the input attribute set — that is, the Boolean function F is encoded in a decryption key for the policy $F \wedge \phi(F)$; the complement function $\overline{F}$ is similarly encoded as an access structure for $\overline{F} \wedge \phi(F)$. Finally, we add an additional attribute $\phi(F) \in \mathcal{U}_{\mathcal{F}}$ to the characteristic attribute set A_x representing the input data x for the function F — that is, the input x is represented as $A_x \cup \phi(F)$.

Notice that the client must perform the ProbGen stage per computation as the function label in the input data will differ (either the input data or the function label will change, otherwise the computation is a repeat of a previously outsourced computation). Servers can be certified for multiple functions and may not use a key for a function G to compute on data intended for another function F since the evaluation key it holds is for the two policies $G \wedge \phi(G)$ and $\overline{G} \wedge \phi(G)$, whereas the input is associated with the attribute set $A_x \cup \phi(F)$; as $\phi(G)$ is not in A_x, neither policy is satisfied and the malicious server cannot return any correct plaintext. As a result, and unlike the single function notion of Parno et al., we are able to provide the adversary with oracle access in our security games using only a simple KP-ABE scheme.

3.5.2 Instantiation

Our RPVC scheme operates as follows.

1. VC.Setup, presented in Algorithm 3.1, first forms the attribute universe for the function family $\mathcal{F}$ and establishes the public parameters and a master secret key by calling the ABE.Setup algorithm twice. We require two distinct ABE systems to enable the security proof to go through; one system will, informally, be linked to the function F and the other to the function $\overline{F}$. To avoid trivial wins in the security proof, oracle queries are restricted such that the adversary cannot query a KeyGen oracle for a key for a function (or policy) that is satisfied by the challenge input (attribute set). Thus, we initialise two ABE systems such that one is maintained by the challenger (and hence requires oracle access) and the other is maintained by the adversary itself. We choose these systems carefully so that the one maintained by the challenger is 'linked' to the unsatisfied function F or $\overline{F}$ and therefore appropriate keys can be provided by the KeyGen oracle. We distinguish the public parameters and master secret keys for these two ABE systems with a superscript 0 or 1 respectively.

   VC.Setup also initialises a time source[6] $\mathbb{T}$, an empty list of revoked servers L_{Rev}, and a two-dimensional array L_{Reg} indexed by server identities — for a server S, $L_{\text{Reg}}[S][0]$ will store a signature verification key for S and $L_{\text{Reg}}[S][1]$ will store a list of functions that S is authorised to compute.

   The public parameters PP for the RPVC system are set to be the two sets of public parameters $MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}$ for the ABE systems, the array L_{Reg} and the time source $\mathbb{T}$ such that any entity may check the current time period. The master secret, MK, for the system is defined to be the two ABE master secrets and the revocation list L_{Rev}.

2. VC.FnInit, presented in Algorithm 3.2, 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 to maintain consistency with prior definitions; note that other instantiations may require this step.

3. VC.Register, presented in Algorithm 3.3, is run by the KDC and creates a signature

[6] $\mathbb{T}$ could be a counter that is maintained in the public parameters or a networked clock.

Algorithm 3.1 $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$

$\mathcal{U} \leftarrow \mathcal{U}_{\mathrm{attr}} \cup \mathcal{U}_{\mathrm{ID}} \cup \mathcal{U}_{\mathrm{time}} \cup \mathcal{U}_{\mathcal{F}}$
$(MPK^0_{\mathrm{ABE}}, MSK^0_{\mathrm{ABE}}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell, \mathcal{U})$
$(MPK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell, \mathcal{U})$
for $S \in \mathcal{U}_{\mathrm{ID}}$ **do**
    $L_{\mathrm{Reg}}[S][0] \leftarrow \epsilon$
    $L_{\mathrm{Reg}}[S][1] \leftarrow \{\epsilon\}$
$L_{\mathrm{Rev}} \leftarrow \epsilon$
Initialise $\mathbb{T}$
$\mathrm{PP} \leftarrow (MPK^0_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, L_{\mathrm{Reg}}, \mathbb{T})$
$\mathrm{MK} \leftarrow (MSK^0_{\mathrm{ABE}}, MSK^1_{\mathrm{ABE}}, L_{\mathrm{Rev}})$

Algorithm 3.2 $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$

$PK_F \leftarrow \mathrm{PP}$

key pair by calling the KeyGen algorithm of a digital signature scheme. It gives the signing key to the server S (privately) and updates the public list L_{Reg} to store the verification key for S.

Algorithm 3.3 $SK_S \xleftarrow{\$} \mathsf{Register}(S, \mathrm{MK}, \mathrm{PP})$

$(SK_{\mathrm{Sig}}, VK_{\mathrm{Sig}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\ell)$
$SK_S \leftarrow SK_{\mathrm{Sig}}$
$L_{\mathrm{Reg}}[S][0] \leftarrow VK_{\mathrm{Sig}}$

4. VC.Certify, presented in Algorithm 3.4, creates the evaluation key $EK_{F,S}$ that enables a server S to compute F. It first updates the lists to remove S from the revocation list and to add F to the list of functions S is authorised to compute. It reads the current time t from the time source $\mathbb{T}$ and calls the ABE.KeyGen and ABE.KeyUpdate algorithms twice — once for the policy encoding F and the ABE system parameters indexed by 0, and once for $\overline{F}$ and the ABE parameters indexed by 1.

   As mentioned previously, to prevent a server certified to perform two different functions, F and G (that differ on their output) from using the key for G to retrieve the plaintext and claim it as a result for F, we add an additional attribute to the input set in ProbGen encoding the function the input should applied to, and add a conjunctive clause for such an attribute to the key policies. Thus an input set intended for F (including the label attribute $\phi(F)$ corresponding to F) will only satisfy a key issued for F (including the $\phi(F)$ label conjunctive clause), and the policy in a key for G will not be satisfied as the label $\phi(G)$ is not in the input set. The evaluation key comprises the decryption keys and the update keys for the current time period.

5. VC.ProbGen for a computation $F(x)$, presented in Algorithm 3.5, first samples the current time period t from the time source $\mathbb{T}$ in the public parameters. It then samples two (equal length) messages m_0 and m_1 uniformly at random from the

Algorithm 3.4 $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$

1: $L_{\mathrm{Reg}}[S][1] \leftarrow L_{\mathrm{Reg}}[S][1] \cup F$
2: $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \setminus S$
3: $t \leftarrow \mathbb{T}$
4: $SK^0_{\mathrm{ABE}} \xleftarrow{\$} \mathsf{ABE.KeyGen}(S, F \wedge \phi(F), MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
5: $SK^1_{\mathrm{ABE}} \xleftarrow{\$} \mathsf{ABE.KeyGen}(S, \overline{F} \wedge \phi(F), MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
6: $UK^0_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
7: $UK^1_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
8: $EK_{F,S} \leftarrow (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$

message space and a bit b uniformly at random. This bit will form the retrieval key and will randomly permute the ciphertexts within the encoded input such that the position of a successfully recovered plaintext will not reveal whether F or $\overline{F}$ was satisfied and therefore whether $F(x) = 1$ or 0 respectively.

The encoded input $\sigma_{F,x}$ comprises two ciphertexts which we denote by c_b and c_{1-b}. The ciphertext c_b is formed by encrypting the message m_b with attributes $A_x \cup \phi(F)$ where A_x is the characteristic tuple encoding of the input data x and $\phi(F) \in \mathcal{U}_{\mathcal{F}}$ is the attribute representing the function F. This encryption is performed using the public parameters MPK^0_{ABE} for the first ABE system. Similarly, c_{1-b} is formed by encrypting the message m_{1-b} under the same attributes $A_x \cup \phi(F)$ and the public parameters MPK^1_{ABE} for the second ABE system.

The verification key $VK_{F,x}$ is created by applying a one-way function g (such as a pre-image resistant hash function [48, 73]) to the messages. The verification key also includes a copy of L_{Reg} from the public parameters incase the list is modified between the current time period and the time of verification, e.g. a server is revoked. This copy may be removed if verification is likely to be imminent or if results computed even *before* a malicious server was revoked should be rejected.

Algorithm 3.5 $(\sigma_{F,x}, VK_{F,x}, RK_{F,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, PK_F, \mathrm{PP})$

1: $t \leftarrow \mathbb{T}$
2: $(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
3: $b \xleftarrow{\$} \{0, 1\}$
4: $c_b \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_b, (A_x \cup \phi(F)), t, MPK^0_{\mathrm{ABE}})$
5: $c_{1-b} \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{1-b}, (A_x \cup \phi(F)), t, MPK^1_{\mathrm{ABE}})$
6: $\sigma_{F,x} \leftarrow (c_b, c_{1-b})$, $VK_{F,x} \leftarrow (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $RK_{F,x} \leftarrow b$

6. $\mathsf{VC.Compute}$, presented in Algorithm 3.6, is run by a server S given an encoded input $\sigma_{F,x}$ and evaluation key $EK_{F,S}$ which it parses as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$. S attempts to decrypt each ciphertext using the relevant keys. It returns $(m_0, \perp)$ if $F(x) = 1$ or $(\perp, m_1)$ if $F(x) = 0$ (ordered according to the random bit $RK_{F,x}$ chosen

in VC.ProbGen), along with the server ID and a signature on the output.

Algorithm 3.6 $\theta_{F(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,x}, EK_{F,S}, SK_S, \mathrm{PP})$

1: Parse $\sigma_{F,x}$ as (c_b, c_{1-b})
2: $d_b \leftarrow \mathsf{ABE.Decrypt}(c_b, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
3: $d_{1-b} \leftarrow \mathsf{ABE.Decrypt}(c_{1-b}, SK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, UK^1_{L_{\mathrm{Rev}},t})$
4: $\gamma \xleftarrow{\$} \mathsf{Sig.Sign}((d_b, d_{1-b}, S), SK_S)$
5: $\theta_{F(x)} \leftarrow (d_b, d_{1-b}, S, \gamma)$

7. VC.Verify either accepts the encoded output $\theta_{F(x)} = (d_b, d_{1-b}, S, \gamma)$ or rejects it. It can be viewed as two algorithms as follows. In VC.BVerif, presented in Algorithm 3.7, the verifier parses the verification key as $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$ and checks whether the server S that (it is claimed) generated the computation result is authorised to compute F — that is, whether F is listed in $L_{\mathrm{Reg}}[S][1]$. If not, the result is immediately rejected.

   Otherwise, the verifier next verifies that the signature on the computational result is valid and was, indeed, generated by S. This is done using the signature verification key stored in $L_{\mathrm{Reg}}[S][0]$. Again, if this check fails, then the result is rejected.

   Otherwise, the verifier checks whether the returned plaintext is correct. It applies the one-way function g, to the first element d_b from the computational result and compares to the first element of the verification key. If the results match, then the verifier accepts the result and returns a retrieval token which is the matched plaintext element d_b from the computation result. If not, the verifier checks the result of applying g to the second returned element d_{1-b} and comparing to the second element of the verification key. Again, if this is a match, then d_{1-b} is returned and the result accepted. If neither comparison succeeds then the verifier rejects the result and reports S for revocation.

   Parno et al. [84] gave a one line remark that permuting the key pairs *and* ciphertexts given out in ProbGen could give output privacy. We believe that doing so would require four decryptions in the Compute stage to ensure the correct keys have been used (since an incorrect key, associated with different public parameters, but for a satisfying attribute set will return an incorrect, random plaintext which is indistinguishable from a valid, random message). Since our construction fixes the order of the key pairs, we do not have this issue and only require two decryptions.

   In VC.Retrieve, presented in Algorithm 3.8, a verifier that has knowledge of $RK_{F,x}$ can check whether the output from BVerif matches m_0 or m_1. If the token $\tau_{\theta_{F(x)}}$ says

Algorithm 3.7 $(RT_{F(x)}, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{BVerif}(\theta_{F(x)}, VK_{F,x}, \mathrm{PP})$

1: **if** $F \in L_{\mathrm{Reg}}[S][1]$ **then**
2: **if** accept $\leftarrow \mathsf{Sig.Verify}((d_b, d_{1-b}, S), \gamma, L_{\mathrm{Reg}}[S][0])$ **then**
3: **if** $g(m_b) = g(d_b)$ **then return** $(RT_{F(x)} \leftarrow d_b, \tau_{\theta_{F(x)}} \leftarrow (\mathsf{accept}, S))$
4: **else if** $g(m_{1-b}) = g(d_{1-b})$ **then return** $(RT_{F(x)} \leftarrow d_{1-b}, \tau_{\theta_{F(x)}} \leftarrow (\mathsf{accept}, S))$
5: **else return** $(RT_{F(x)} \leftarrow \perp, \tau_{\theta_{F(x)}} \leftarrow (\mathsf{reject}, S))$
6: $(RT_{F(x)} \leftarrow \perp, \tau_{\theta_{F(x)}} \leftarrow (\mathsf{reject}, \perp))$

that the result should be accepted, the verifier applies the one-way function g to the retrieval token $RT_{F(x)} \in \{d_b, d_{1-b}\}$ and compares to each element $g(m_0), g(m_1)$ of the verification key (note that as the verifier possesses $RK_{F,x} = b$, it can resolve the ordering of these elements in the verification key). If a match is made with $g(m_0)$ then the function F was satisfied by the input x and the result y is set to be 1. On the other hand, if the match is made with $g(m_1)$, then the complement function was satisfied and $y = 0$.

Algorithm 3.8 $y \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x)}}, RT_{F(x)}, VK_{F,x}, RK_{F,x}, \mathrm{PP})$

1: **if** $(\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ **and** $g(RT_{F(x)}) = g(m_0))$ **then return** $y \leftarrow 1$
2: **else if** $(\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ **and** $g(RT_{F(x)}) = g(m_1))$ **then return** $y \leftarrow 0$
3: **else return** $y \leftarrow \perp$

8. VC.Revoke, presented in Algorithm 3.9, is run by the KDC and redistributes fresh keys to all non-revoked servers. If the token $\tau_{\theta_{F(x)}}$ does not specify a server S to be revoked, then $\perp$ is returned. Otherwise, the KDC first removes all functions from the list $L_{\mathrm{Reg}}[S][1]$ (as S should no longer be authorised for any computations) and adds S to the revocation list L_{Rev}. It then refreshes the time source $\mathbb{T}$ (e.g. increments $\mathbb{T}$ if it is a counter) and samples the new time period. The ABE.KeyUpdate algorithm is run twice (once for each ABE system) to generate new update key material for the current time period with respect to the revocation list L_{Rev}. Finally, for all servers, the KDC updates and redistributes the evaluation key to reflect the update key material for non-revoked servers.

It is straightforward to see that correctness of this construction follows from the correctness of the attribute-based encryption scheme and of the one-way function g.

Algorithm 3.9 $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(x)}}, \mathrm{MK}, \mathrm{PP})$

if $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ **then**
  $L_{\mathrm{Reg}}[S][1] \leftarrow \{\epsilon\}$
  $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \cup S$
  Refresh $\mathbb{T}$
  $t \leftarrow \mathbb{T}$
  $UK^0_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
  $UK^1_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
  for all $S \in \mathcal{U}_{\mathrm{ID}}$ **do**
    Parse $EK_{F,S}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t-1}, UK^1_{L_{\mathrm{Rev}},t-1})$
    $EK_{F,S} \leftarrow (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$
  return $UM \leftarrow \{EK_{(\mathbb{O},\psi),S'}\}_{S' \in \mathcal{U}_{\mathrm{ID}}}$
else
  return $UM \leftarrow \perp$

3.6 Proofs of Security

Theorem 3.2. *Given a revocable KP-ABE scheme secure in the sense of indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*)* [14] *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 computation scheme defined in Algorithms 3.1–3.9. Then $\mathcal{RPVC}$ is secure in the sense of selective public verifiability, selective semi-static revocation, vindictive servers and selective vindictive managers.*

Informally, the proofs of public verifiability and vindictive managers rely on the IND-CPA security of the underlying revocable KP-ABE scheme (note that IND-CPA security is implied by IND-sHRSS) and the one-wayness of the function g. Revocation relies on the IND-sHRSS security of the revocable KP-ABE scheme. Security against vindictive servers relies on the EUF-CMA security of the signature scheme such that a vindictive server cannot return an incorrect result with a forged signature claiming to be from an honest server (note that chosen message attack is required since the vindictive client could act like a client and submit computation requests to get a valid signature).

Lemma 3.1. *The $\mathcal{RPVC}$ construction defined by Algorithms 3.1–3.9 is secure in the sense of selective public verifiability (Game 3.7) under the same assumptions as in Theorem 3.2.*

Proof. We begin by defining the following three games:

- **Game 0.** This is the selective public verifiability game as defined in Game 3.7;

- **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** except that m' is implicitly set to be the challenge input w in the one-way function game.

Following partially in the fashion of Parno et al. [84], we aim to show that, from the point of view of an adversary, **Game 2** is indistinguishable from **Game 0** except with negligible probability. Thus, an adversary against the selective public verifiability game can, instead, be run against **Game 2**. We then show that if an adversary has a non-negligible against **Game 2** then the adversary can be used to 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 $(\mathcal{RPVC}, 1^\ell, \mathcal{F})$. To achieve a contradiction, let us suppose otherwise, that $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We show that it is possible to construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VC}$ 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 (Game 3.1) with $\mathcal{A}_{ABE}$, who in turn acts as a challenger for $\mathcal{A}_{VC}$. Given the security parameter and function family $\mathcal{F}$, the entities interact as follows.

1. $\mathcal{A}_{VC}$ declares its choice of challenge function F and challenge input $x^\star$.
2. $\mathcal{A}_{ABE}$ computes $r = F(x^\star)$. It then transforms $\mathcal{A}_{VC}$'s challenge parameters into its own choice of challenge for the IND-sHRSS game. It sets $\overline{x^\star} = A_{x^\star} \cup \phi(F)$ where $\phi(F) \in \mathcal{U}_{\mathcal{F}}$ is the attribute representing the challenge function F, and $t^\star = 1$, and sends these to $\mathcal{C}$.
3. $\mathcal{C}$ runs the ABE.Setup algorithm to generate $(MPK_{\text{ABE}}, MSK_{\text{ABE}})$ and sends MPK_{ABE} to $\mathcal{A}_{ABE}$.

4. $\mathcal{A}_{ABE}$ sends $\overline{R} = \epsilon$ (i.e. an empty list) to $\mathcal{C}$. It then simulates running VC.Setup such that the ABE system owned by $\mathcal{C}$ is used as the ABE system with parameters $(MPK^r_{\text{ABE}}, MSK^r_{\text{ABE}})$. The intuition here is that, as $\mathcal{A}_{ABE}$ does not hold MSK_{ABE} generated by $\mathcal{C}$, it will need to issue queries to oracles provided by $\mathcal{C}$ in order to generate valid parameters to pass to $\mathcal{A}_{VC}$. However, to avoid trivial wins in the IND-sHRSS game, $\mathcal{A}_{ABE}$ is prevented from querying the KeyGen oracle for a function that evaluates to 1 on the challenge input $\overline{x^\star}$ (to prevent decryption of the challenge ciphertext).

   Now, recall that in the VC.Certify algorithm (Algorithm 3.4), a decryption key for the function F is generated using the ABE parameters $(MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$ associated to the first ABE system that was initialised, while the key for the complement function $\overline{F}$ is generated using the parameters $(MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$ for the second ABE system.

   Thus, if $F(x^\star) = 1$, then the policy $F \wedge \phi(F)$ will also be satisfied by $\overline{x^\star} = A_{x^\star} \cup \phi(F)$. Therefore, one cannot make a KeyGen oracle query to $\mathcal{C}$ for the policy $F \wedge \phi(F)$. In this case, we must ensure that the ABE system "owned" by $\mathcal{C}$ is in fact $(MPK^1_{\text{ABE}}, MSK^1_{\text{ABE}})$ so that $\mathcal{A}_{ABE}$ itself holds $(MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}})$ and can therefore generate a key for $F \wedge \phi(F)$ itself.

   On the other hand, if $F(x^\star) = 0$ then the function $\overline{F} \wedge \phi(F)$ may not be queried to the KeyGen oracle. The key associated with this function should, according to Algorithm 3.4 be generated using the parameters $(MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$. Therefore, we require $\mathcal{A}_{ABE}$ to "own" these parameters and that $\mathcal{C}$ "owns" $(MPK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$.

   Notice that in both cases, $\mathcal{C}$ must own the parameters $(MPK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$ where $r = F(x^\star)$.

   $\mathcal{A}_{ABE}$ simulates running VC.Setup by running Algorithm 3.1 as written, with the exception of Line 2 and 3 where it sets MPK^r_{ABE} to be that provided by $\mathcal{C}$, and implicitly sets MSK^r_{ABE} to be that held by the challenger (by 'implicitly', we mean that any subsequent uses of MSK^r_{ABE} will be simulated using oracle queries to $\mathcal{C}$, but from the point of view of $\mathcal{A}_{VC}$, MSK^r_{ABE} will appear to have been generated by $\mathcal{A}_{ABE}$).

5. $\mathcal{A}_{ABE}$ runs VC.FnInit as written and must then generate a challenge for $\mathcal{A}_{VC}$. To do so, it samples three distinct messages m_0, m_1 and m' uniformly at random from the messagespace, and flips a random coin $RK_{F,x^\star} = b \xleftarrow{\$} \{0,1\}$. It submits m_0 and m_1

as its choice of challenge plaintexts to $\mathcal{C}$, and receives back the encryption, $CT^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$), under attributes $\overline{x^\star}$ and time $t^\star$.

$\mathcal{A}_{ABE}$ must assign $CT^\star$ to be one of the ciphertexts c_b, c_{1-b} that form the encoded challenge input $\sigma_{F,x^\star}$ according to the correct ABE systems parameters (decryption will fail if the wrong parameters are used). If $r = 0$, then $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$. Otherwise, $r = 1$ and $\mathcal{A}_{ABE}$ sets c_{1-b} to be $CT^\star$.

$\mathcal{A}_{ABE}$ generates the remaining ciphertext (c_{1-b} or c_b respectively) itself by running $\mathsf{ABE.Encrypt}(m', \overline{x^\star} = (A_{x^\star} \cup \phi(F)), t^\star = 1, MPK^{1-r}_{\text{ABE}})$.

Finally, $\mathcal{A}_{ABE}$ chooses a random bit $s \xleftarrow{\$} \{0,1\}$ (intuitively, s is a guess of the value $b^\star$ chosen by $\mathcal{C}$). It forms a verification key correctly ordered according to $RK_{F,x^\star} = b$. Let $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$. Then the verification key $VK = (VK_0, VK_1, L_{Reg})$.

6. $\mathcal{A}_{VC}$ is given all outputs from the above $\mathsf{ProbGen}$ simulation, and is given oracle access, to which $\mathcal{A}_{ABE}$ can respond as follows:

   - Queries to $\mathsf{VC.FnInit}$ and $\mathsf{VC.Register}$ are performed as in Algorithms 3.2 and 3.3 respectively (as these do not rely on ABE parameters held by $\mathcal{C}$).
   - Queries of the form $\mathsf{VC.Certify}(S, F', \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Algorithm 3.4 as written with the exception that the KeyGen and $\mathsf{KeyUpdate}$ operations for the ABE system with parameters $(MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$ (owned by $\mathcal{C}$) are replaced by queries to the corresponding oracles provided by $\mathcal{C}$.

     To generate SK^r_{ABE}, $\mathcal{A}_{ABE}$ queries the $\mathsf{ABE.KeyGen}$ oracle. If $r = 0$, the query is of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, F' \wedge \phi(F'), MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$, and if $r = 1$, the query is of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, \overline{F'} \wedge \phi(F'), MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$. In both cases, $\mathcal{C}$ will return the decryption key *unless* $\overline{x^\star}$ satisfies the queried policy.

     Notice that if the queried function F' is *not* the challenge function F then, due to the bijective mapping ϕ, $\phi(F') \neq \phi(F)$. Thus, neither of the possible queries (which both require the presence of $\phi(F')$) will be satisfied. If, however, the query is for F, recall that we chose the ABE system owned by the challenger to be unsatisfied by exactly this queried policy. Hence the first check performed in Oracle 3.1 will never evaluate to `true` and therefore $\mathcal{C}$ will always be able to return a valid key in response to a KeyGen query.

     To generate $UK^r_{L_{\text{Rev}},t}$, $\mathcal{A}_{ABE}$ makes a query to the $\mathsf{ABE.KeyUpdate}$ oracle of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ returns a valid update

key *unless* the current time is the challenge time (which $\mathcal{A}_{ABE}$ chose to be 1) *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$ which $\mathcal{A}_{ABE}$ chose to be empty. Since, $\overline{R} = \epsilon$ is a subset of any L_{Rev}, the second clause will not be satisfied and $\mathcal{C}$ may generate a valid update key.

Thus, $\mathcal{A}_{ABE}$ can request valid decryption keys and update keys from $\mathcal{C}$ and can simulate Algorithm 3.4 exactly.

- Queries of the form $\mathsf{VC.Revoke}(\tau_{\theta_{F(x)}}, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Algorithm 3.9 as written with the exception of generating $UK^{\tau}_{L_{\text{Rev}},t}$ on line 6 or 7. To simulate this line, $\mathcal{A}_{ABE}$ queries for $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, MSK^{\tau}_{\text{ABE}}, MPK^{\tau}_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* $t = 1$ *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. However, as $\mathcal{A}_{ABE}$ chose $\overline{R} = \epsilon$, this second clause is never satisfied and a valid update key is returned.

7. Eventually, $\mathcal{A}_{VC}$ outputs a guess $\theta^{\star}$. Let y be the non-$\perp$ plaintext contained in $\theta^{\star}$. If $g(y) = g(m_s)$, $\mathcal{A}_{ABE}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{ABE}$ guesses $b' = 1 - s$.

Notice that if $s = b^{\star}$ (the challenge bit chosen by $\mathcal{C}$), 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 plaintexts corresponding to the ciphertexts in the encoded input (exactly one of which $\mathcal{A}_{VC}$ may recover). Otherwise, $s = 1 - b^{\star}$ and the distribution coincides with **Game 1** since the verification key comprises the one-way function applied to a legitimate plaintext m' and a random message m_{1-s} that is unrelated to both ciphertexts.

Now, we consider the advantage of this constructed adversary $\mathcal{A}_{ABE}$ playing the IND-sHRSS game: Recall that by assumption, $\mathcal{A}_{VC}$ has a non-negligible advantage δ in distinguishing **Game 0** from **Game 1** — that is

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

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{VC}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}\right]$ denotes running $\mathcal{A}_{VC}$ in Game i.

The probability of $\mathcal{A}_{ABE}$ guessing $b^\star$ correctly, by the law of total probability, is:

$$\begin{aligned}
\Pr[b' = b^\star] &= \Pr[s = b^\star] \Pr\left[b' = b^\star | s = b^\star\right] + \Pr[s \neq b^\star] \Pr\left[b' = b^\star | s \neq b^\star\right] \\
&= \frac{1}{2} \Pr[g(y) = g(m_s) | s = b^\star] + \frac{1}{2} \Pr[g(y) \neq g(m_s) | s \neq b^\star] \\
&= \frac{1}{2} \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{VC}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] + \frac{1}{2}\left(1 - \Pr[g(y) = g(m_s) | s \neq b^\star]\right) \\
&= \frac{1}{2} \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{VC}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] + \frac{1}{2}\left(1 - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{VC}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{VC}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{VC}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] + 1\right) \\
&\geq \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

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

Since δ is assumed to be non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ 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 $\mathcal{A}_{VC}$ 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 (Game 2.12). 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, $\mathcal{A}_{VC}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof We now show that using $\mathcal{A}_{VC}$ in **Game 2**, $\mathcal{A}_{ABE}$ can invert the one-way function g — that is, given a challenge $z = g(w)$, $\mathcal{A}_{ABE}$ can recover w. Specifically, during

ProbGen, $\mathcal{A}_{ABE}$ chooses the messages as follows:

- if $F(x^\star) = 1$, we implicitly set m_{1-b} to be w and the corresponding verification key component to be z. We randomly choose m_b and compute the remainder of the verification key as usual.
- if $F(x^\star) = 0$, we implicitly set m_b to be w and set the verification key component to z. m_{1-b} is chosen randomly and the remainder of the verification key computed as usual.

Now, if $\mathcal{A}_{VC}$ is 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). $\mathcal{A}_{ABE}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{VC}$ 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 3.1–3.9 is secure in the sense of selective public verifiability. □

Lemma 3.2. *The $\mathcal{RPVC}$ construction defined by Algorithms 3.1–3.9 is secure in the sense of selective, semi-static revocation (Game 3.8) under the same assumptions as in Theorem 3.2.*

Proof. We perform a reduction from selective, semi-static revocation to the IND-sHRSS security of the underlying revocable KP-ABE scheme. To achieve a contradiction, let us suppose that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the selective, semi-static revocation game (Game 3.8) when instantiated by Algorithms 3.1–3.9. We show that we can construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VC}$ 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 (Game 3.1) with $\mathcal{A}_{ABE}$, who in turn acts as a challenger for $\mathcal{A}_{VC}$. Given the security parameter, function family $\mathcal{F}$ and q_t, the number of queries that $\mathcal{A}_{VC}$ will make to the Revoke oracle, the entities interact as follows.

1. $\mathcal{A}_{VC}$ selects its challenge inputs F and $x^\star$.

2. $\mathcal{A}_{ABE}$ initialises $Q_{\text{Rev}} = \epsilon$ (an empty list) and $t = 1$. It then forms its own challenge input to be $t^\star = q_t$ and $A^\star = A_x \cup \phi(F)$ where A_x is the attribute encoding of x and $\phi(F) \in \mathcal{U}_{\mathcal{F}}$ is the attribute representing the function F, and sends these to $\mathcal{C}$.

3. $\mathcal{C}$ runs ABE.Setup and returns the resulting parameters MPK_{ABE} to $\mathcal{A}_{ABE}$.

4. $\mathcal{A}_{ABE}$ simulates running Setup by running Algorithm 3.1 as written with the exception of line 2. It sets MPK^0_{ABE} to be MPK_{ABE} generated by $\mathcal{C}$, and implicitly sets MSK^0_{ABE} to be that held by the challenger. As it does not possess MSK_{ABE}, it will make use of oracle queries to $\mathcal{C}$ wherever MSK^0_{ABE} is required. $\mathcal{A}_{ABE}$ also runs FnInit as written.

5. $\mathcal{A}_{VC}$ is given PK_F and PP and chooses a challenge revocation list $\overline{R}$, which $\mathcal{A}_{ABE}$ forwards to $\mathcal{C}$.

6. $\mathcal{A}_{VC}$ may now make oracle queries to which $\mathcal{A}_{ABE}$ can respond as follows:

   - Queries to VC.FnInit and VC.Register are run as written in Algorithms 3.2 and 3.3.
   - Queries of the form VC.Certify$(S, F', \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Oracle 3.8. If F' is the challenge function F and S is not listed in $\overline{R}$ then it returns $\perp$ to $\mathcal{A}_{VC}$. It also returns $\perp$ if the current time is the challenge time period q_t and there is a server (other than S) that is not currently revoked but should be in accordance with the challenge revocation list. Otherwise, it removes S from Q_{Rev} and simulates running Certify. To do so, it runs Algorithm 3.4 as written with the exception of lines 4 and 6.

     To simulate line 4, $\mathcal{A}_{ABE}$ queries $\mathcal{C}$ for a query of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, F' \wedge \phi(F'), MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ returns the decryption key *unless* the policy $F' \wedge \phi(F')$ is satisfied by $A^\star = A_x \cup \phi(F)$ *and* $S \notin \overline{R}'$. First, observe that this policy will never be satisfied unless $F' = F$ (since ϕ is a bijective mapping and $\phi(F')$ is required for the policy to be 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 $\mathcal{A}_{ABE}$ at the beginning of Oracle 3.8, 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 $\mathcal{A}_{VC}$ in accordance with the Certify oracle.

To simulate line 6, $\mathcal{A}_{ABE}$ queries $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* the current time is the challenge time q_t *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. However, if this was the case then $\mathcal{A}_{ABE}$ would already have returned $\perp$ by the second clause of the if statement in Oracle 3.8. Hence, $\mathcal{C}$ will always return a key which $\mathcal{A}_{ABE}$ can use to successfully simulate the Certify algorithm.

- Queries of the form $\mathsf{VC.Revoke}(\tau_{\theta_{F(x)}}, \mathrm{MK}, \mathrm{PP})$: As specified in Oracle 3.9, $\mathcal{A}_{ABE}$ first increments t. If the token does not identify a server to revoke, it outputs $\perp$ (as would the Revoke algorithm). If the current time is q_t, then $\mathcal{A}_{ABE}$ returns $\perp$ if Q_{Rev} does not contain all servers listed on the challenge revocation list $\overline{R}$. Otherwise, S is added to Q_{Rev}. $\mathcal{A}_{ABE}$ now simulates running the VC.Revoke algorithm by running Algorithm 3.9 as written with the exception of line 6. To simulate this line, $\mathcal{A}_{ABE}$ makes a query of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* $t = q_t$ *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. However, if this was so, $\mathcal{A}_{ABE}$ would have returned $\perp$ above, and so a valid update key is always returned which $\mathcal{A}_{ABE}$ can forward to $\mathcal{A}_{VC}$.

7. Eventually (after q_t Revoke queries), $\mathcal{A}_{VC}$ finishes the query phase. $\mathcal{A}_{ABE}$ checks if $\mathcal{A}_{VC}$ 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 stops.

8. $\mathcal{A}_{ABE}$ must now generate the challenge for $\mathcal{A}_{VC}$. It chooses three distinct, equal length messages m_0, m_1 and m' uniformly at random, and chooses a random bit $RK_{F,x^\star} = b \xleftarrow{\$} \{0,1\}$. It sends m_0 and m_1 to $\mathcal{C}$ as its choice of challenge. $\mathcal{C}$ chooses a random bit $b^\star \xleftarrow{\$} \{0,1\}$ and returns

$$CT^\star \leftarrow \mathsf{ABE.Encrypt}(m_{b^\star}, A^\star = (x^\star \cup \phi(F)), q_t, MPK^0_{\mathrm{ABE}}).$$

$\mathcal{A}_{ABE}$ sets $c_b = CT^\star$ and

$$c_{1-b} \leftarrow \mathsf{ABE.Encrypt}(m', A^\star = (x^\star \cup \phi(F)), q_t, MPK^1_{\mathrm{ABE}}).$$

$\mathcal{A}_{ABE}$ selects another bit $s \xleftarrow{\$} \{0,1\}$ and, if $b = 0$, sets $VK_{F,x^\star} = (g(m_s), g(m'), L_{Reg})$.

Otherwise, $VK_{F,x^\star} = (g(m'), g(m_s), L_{Reg})$. Note that s is $\mathcal{A}_{ABE}$'s guess for $b^\star$.

9. The resulting parameters from ProbGen are sent to $\mathcal{A}_{VC}$ who is also given oracle access. These queries are handled in the same way as previously, and eventually $\mathcal{A}_{VC}$ outputs its guess $\theta^\star$.

10. Let y be the non-$\perp$ plaintext returned in $\theta^\star$. If $g(y) = g(m_s)$, $\mathcal{A}_{ABE}$ guesses $b' = s$. If $g(y) = g(m')$, $\mathcal{A}_{ABE}$ makes a random guess $b' = \tilde{b} \xleftarrow{\$} \{0, 1\}$ (as $\mathcal{A}_{VC}$ did not forge a result for either m_0 or m_1 and hence is of no use in breaking the IND-sHRSS game). Else, $\mathcal{A}_{ABE}$ aborts (since $\mathcal{A}_{VC}$ did not succeed, it could be that $\mathcal{A}_{VC}$ simply did not win against the correctly formed challenge or $\mathcal{A}_{ABE}$ guessed incorrectly which message was chosen by $\mathcal{C}$ and issued a malformed challenge to $\mathcal{A}_{VC}$).

Now, consider the advantage of $\mathcal{A}_{ABE}$ playing the IND-sHRSS game: By assumption, $\mathcal{A}_{VC}$ has a non-negligible advantage δ against the selective, semi-static revocation game. That is, $\Pr[g(y) = g(m_s)] + \Pr[g(y) = g(m')] = \delta$. Therefore, by the law of total probability,

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

Hence,

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

Since δ is non-negligible, $\frac{1}{2}(\delta - 1)$ is also non-negligible. If $\mathcal{A}_{VC}$ has advantage δ at breaking the selective, semi-static revocation game then $\mathcal{A}_{ABE}$ can win the IND-sHRSS game with

non-negligible probability. However, since the ABE scheme was assumed IND-sHRSS secure, such an $\mathcal{A}_{VC}$ cannot exist. We conclude that if the ABE scheme is IND-sHRSS secure then the $RPVC$ scheme defined by Algorithms 3.1–3.9 is secure in the sense of selective, semi-static revocation. □

Lemma 3.3. *The $\mathcal{RPVC}$ construction defined by Algorithms 3.1–3.9 is secure against vindictive servers (Game 3.5) under the same assumptions as in Theorem 3.2.*

Proof. Suppose, for a contradiction, that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the vindictive servers game (Game 3.5) when instantiated by Algorithms 3.1–3.9. We show that an adversary $\mathcal{A}_{Sig}$ with non-negligible advantage δ in the EUF-CMA signature game (Game 2.11) can be constructed using $\mathcal{A}_{VC}$ as a subroutine. $\mathcal{A}_{Sig}$ interacts with the challenger $\mathcal{C}$ in the EUF-CMA security game and acts as the challenger for $\mathcal{A}_{VC}$ in the security game for vindictive servers for a function F as follows. The basic idea is that $\mathcal{A}_{Sig}$ can create an RPVC instance and play the vindictive servers game with $\mathcal{A}_{VC}$ by executing Algorithms 3.1–3.9 himself. $\mathcal{A}_{Sig}$ will guess a server identity that he thinks the adversary will select to vindictively revoke. The signature signing key that would be generated during the $\mathsf{Register}$ algorithm for this server will be implicitly set to be the signing key in the EUF-CMA game and any signatures for this identity will be formed using oracle queries to $\mathcal{C}$. Then, assuming that $\mathcal{A}_{Sig}$ guessed the correct server identity, $\mathcal{A}_{VC}$ will output a forged signature that $\mathcal{A}_{Sig}$ may output as its guess in the EUF-CMA game.

1. $\mathcal{C}$ initialises $Q \leftarrow \epsilon$ to be an empty list of messages queried to the $\mathsf{Sig.Sign}$ oracle. It runs $\mathsf{Sig.KeyGen}(1^\ell)$ to generate a challenge signing key $\overline{SK}$ and verification key $\overline{VK}$, and sends $\overline{VK}$ to $\mathcal{A}_{Sig}$.

2. $\mathcal{A}_{Sig}$ initialises the revocation list $Q_{\text{Reg}} \leftarrow \epsilon$ and $\tilde{S}$ to be $\perp$. Furthermore, it chooses a server identity from $\mathcal{U}_{\mathsf{ID}}$ which will be denoted by $\overline{S}$. $\overline{S}$ will be a guess of the target identity $\tilde{S}$ that $\mathcal{A}_{VC}$ shall choose later. If this guess is correct, then any signing operations related to $\overline{S}$ can be performed using the oracle provided by $\mathcal{C}$, and hence when $\mathcal{A}_{VC}$ attacks this identity, his output can be used to break the EUF-CMA game.

3. $\mathcal{A}_{Sig}$ runs $\mathsf{VC.Setup}$ and gives the resulting public parameters to $\mathcal{A}_{VC}$. $\mathcal{A}_{VC}$ is also provided with oracle access to which $\mathcal{A}_{Sig}$ can respond as follows

- Queries to VC.FnInit, VC.Register2, VC.Certify and VC.Revoke can all be handled simply by running the relevant algorithm as these do not require signatures.
- Queries to VC.Register: Queries for a server $S \neq \overline{S}$ can be handled by $\mathcal{A}_{Sig}$ running Oracle 3.5 as written. Note that $\perp$ is returned if the query is for the target server $\tilde{S}$. If, on the other hand, $S = \overline{S}$, then $\mathcal{C}$ aborts the game, since $\mathcal{A}_{VC}$ may not choose its target server $\tilde{S}$ to be a server for which it previously learnt the signing key, to avoid trivial wins. Therefore, $\mathcal{A}_{VC}$ cannot choose $\tilde{S} = S = \overline{S}$ and hence, $\mathcal{A}_{Sig}$'s guess of the target identity was wrong and the EUF-CMA challenge parameters have been embedded incorrectly in the reduction.

4. Eventually, $\mathcal{A}_{VC}$ finishes this query phase and outputs its choice of challenge parameters F and $x^\star$. $\mathcal{A}_{Sig}$ runs $\mathsf{VC.FnInit}(F, \mathrm{MK}, \mathrm{PP})$, as specified in Algorithm 3.2 and VC.ProbGen on the challenge $x^\star$ as in Algorithm 3.5.

5. $\mathcal{A}_{VC}$ is given the resulting values and oracle access which is handled as above. $\mathcal{A}_{VC}$ eventually outputs a target server identity $\tilde{S}$. If $\mathcal{A}_{VC}$ has previously queried for the signing key $SK_{\tilde{S}}$ from the Register oracle then $\mathcal{A}_{Sig}$ returns $\perp$.

6. If $\tilde{S} \neq \overline{S}$ then $\mathcal{A}_{Sig}$ outputs $\perp$ and stops, as it guessed incorrectly. Else, $\mathcal{A}_{VC}$ continues with oracle access as in Step 5 as well as a Compute oracle. $\mathcal{A}_{VC}$ submits queries $\mathcal{O}^{\mathsf{Compute}}(\sigma_{F',x}, EK_{F',\tilde{S}}, SK_{\tilde{S}}, \mathrm{PP})$ for its choice of computation $F'(x)$ (note that $\mathcal{A}_{VC}$ may generate a valid $\sigma_{F',x}$ using the public delegation key). If $S \neq \overline{S}$ then $\mathcal{A}_{Sig}$ simply follows Algorithm 3.6 using the decryption and signing keys generated during the oracle queries. Otherwise, $S = \overline{S}$ and $\mathcal{A}_{Sig}$ does not have access to the signing key $SK_{\overline{S}}$. Thus, he runs the ABE.Decrypt operations correctly to generate plaintexts d_b and d_{1-b}, and submits $m \leftarrow (d_b, d_{1-b}, \overline{S})$ as a Sig.Sign oracle query to $\mathcal{C}$. $\mathcal{C}$ adds m to the list Q and returns $\gamma \xleftarrow{\$} \mathsf{Sig.Sign}(m, \overline{SK})$, which $\mathcal{A}_{Sig}$ uses to return $\theta_{F(x)} \leftarrow (d_b, d_{1-b}, \overline{S}, \gamma)$, as a valid response to the Compute query.

7. $\mathcal{A}_{VC}$ finally outputs $\theta^\star = (d_b^\star, d_{1-b}^\star, \tilde{S}, \gamma)$ which appears to be an invalid result computed by $\tilde{S}$. Thus, Verify will output a reject token for $\tilde{S}$. However, $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}((d_b^\star, d_{1-b}^\star, \tilde{S}), \gamma, \overline{VK})$. Thus, γ is a valid signature under key $SK_{\tilde{S}} = \overline{SK}$.

8. $\mathcal{A}_{Sig}$ outputs $m^\star = (d_b^\star, d_{1-b}^\star, \tilde{S})$ and $\gamma^\star = \gamma$ to $\mathcal{C}$.

Note that due to the check performed on line 1 of Oracle 3.7, Compute was not simulated

(i.e. $\mathcal{A}_{Sig}$ did not make use of the Sig.Sign oracle provided by $\mathcal{C}$) for the computation $F(x^\star)$. Thus the forgery $(m^\star, \gamma^\star)$ output by $\mathcal{A}_{Sig}$ will satisfy the requirement in Game 2.11 that $m^\star \notin Q$. We argue that, assuming $\overline{S} = \tilde{S}$ (i.e. $\mathcal{A}_{Sig}$ correctly guessed the challenge identity) then $\mathcal{A}_{Sig}$ succeeds with the same non-negligible advantage δ as $\mathcal{A}_{VC}$. We assume that $n = |\mathcal{U}_{\mathsf{ID}}|$ is polynomial (else the KDC could not efficiently search the list L_{Reg}). The probability that $\mathcal{A}_{Sig}$ correctly guesses $\overline{S} = \tilde{S}$ is $\frac{1}{n}$ and

$$\begin{aligned} Adv_{\mathcal{A}_{Sig}} &\geqslant \Pr\left[\overline{S} = \tilde{S}\right] Adv_{\mathcal{A}_{VC}} \\ &\geqslant \frac{1}{n} Adv_{\mathcal{A}_{VC}} \\ &\geqslant \frac{\delta}{n} \\ &\geqslant negl(\ell) \end{aligned}$$

We conclude that, since n is polynomial, if $\mathcal{A}_{VC}$ has a non-negligible advantage δ in the vindictive servers game then $\mathcal{A}_{Sig}$ has non-negligible advantage in the EUF-CMA game, but since the signature scheme is assumed EUF-CMA secure, $\mathcal{A}_{VC}$ may not exist. □

We note that we lose a polynomial factor in the advantage due to having to guess the server $\tilde{S}$ that the adversary will attempt to revoke. This factor could be removed if we formulated the security model in a selective fashion such that $\mathcal{A}_{VC}$ must declare up front which server he will target, and then $\mathcal{A}_{Sig}$ 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.

Lemma 3.4. *The $\mathcal{RPVC}$ construction defined by Algorithms 3.1–3.9 is secure in the sense of selective vindictive managers (Game 3.9) under the same assumptions as in Theorem 3.2.*

Proof. This proof proceeds in a similar way to that of selective public verifiability. We begin by defining the following three games:

- **Game 0.** This is the selective vindictive managers game as defined in Game 3.9.
- **Game 1.** This is the same as **Game 0** except that in ProbGen, a random message $m' \neq m_0, m_1$ is chosen and, if $F(x^\star) = 1$, we replace c_1 by the encryption of m',

and otherwise we replace c_0. In other words, the ciphertext associated with the unsatisfied function is replaced by the encryption of a separate random message unrelated 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 adversarial point of view, **Game 2** is indistinguishable from **Game 0** except with negligible probability. Thus, an adversary can be run against **Game 2**. We show that if an adversary has a non-negligible advantage against **Game 2** then the adversary can be used to 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 $(\mathcal{RPVC}, 1^\ell, \mathcal{F})$. For a contradiction, let us suppose that $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VC}$ 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 (Game 3.1) with $\mathcal{A}_{ABE}$, who in turn acts as a challenger for $\mathcal{A}_{VC}$:

1. $\mathcal{A}_{VC}$ declares its choice of challenge inputs F and $x^\star$.
2. $\mathcal{A}_{ABE}$ transforms this into its own challenge input $\overline{x^\star} = A_{x^\star} \cup \phi(F)$ where $A_{x^\star}$ is the characteristic tuple of attributes representing $x^\star$ and $\phi(F) \in \mathcal{U}_{\mathcal{F}}$ is the attribute representing the challenge function F. It then sends this choice to $\mathcal{C}$ along with a challenge time period $t^\star = 1$. It also computes $r = F(x^\star)$ which will determine which of the two ABE systems will be used for 'positive' functions and which for the complement functions (since $\mathcal{C}$ will not issue a decryption key for a function satisfied by the challenge input and so $\mathcal{A}_{ABE}$ must be sure that it will only be queried for the non-satisfied function). In the following, we 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 $F_r(x^\star) = 0$. In other words, $F_0 = F$ and $F_1 = \overline{F}$ and we select between them based on r.

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

4. $\mathcal{A}_{ABE}$ returns an empty challenge revocation list $\overline{R} = \epsilon$ to $\mathcal{C}$.

5. $\mathcal{A}_{ABE}$ simulates running VC.Setup by running Algorithm 3.1 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 $r = F(x)$. $\mathcal{A}_{ABE}$ 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 $\mathcal{C}$ (any subsequent use of MSK^r_{ABE} will be simulated using oracle access to $\mathcal{C}$). It runs ABE.Setup to generate $MPK^{1-r}_{\text{ABE}}, MSK^{1-r}_{\text{ABE}}$ as usual.

6. $\mathcal{A}_{ABE}$ runs VC.FnInit as written. It must then generate a challenge encoded output and to do so it must simulate a computation server. It first picks a server S uniformly at random and runs Algorithm 3.3 as written to register S. It must then simulate running the Certify algorithm for the sever S and challenge function F. However, as it does not hold the full master secret key MK, it must use the oracle access provided by $\mathcal{C}$. It runs Algorithm 3.4 as written with the following exceptions:

   - SK^r_{ABE} is generated querying the ABE.KeyGen oracle with a query of the form $(S, F_r \wedge \phi(F), MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ will return a valid decryption key unless $\overline{x^\star} \in F_r \wedge \phi(F)$ *and* $S \notin \overline{R}$.

     Clearly, S is never in $\overline{R}$ as this was set to be empty so the second clause is always satisfied. However, we chose r specifically such that $F_r(x^\star) = 0$. Hence $A_{x^\star} \notin F_r$ and $\overline{x^\star} = A_{x^\star} \cup \phi(F) \notin F_r \wedge \phi(F)$. Thus, $\mathcal{C}$ will return a valid decryption key.

   - SK^{1-r}_{ABE} is generated by $\mathcal{A}_{ABE}$ running ABE.KeyGen using MSK^{1-r}_{ABE} for the function $F_{1-r} \wedge \phi(F)$ as usual.

   - $UK^r_{L_{\text{Rev}},t}$ is generated by making a query to the ABE.KeyUpdate oracle for parameters $(L_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key unless the current time period t is the challenge time $t^\star$ and $\overline{R} \not\subseteq Q_{\text{Rev}}$. Now, $t^\star$ was chosen to be 1 and therefore, as no Revoke operations have been performed, t does indeed equal $t^\star$. However, as $\overline{R}$ was chosen to be empty, $\overline{R}$ is certainly

a subset of L_{Rev} (in fact, both are empty). Therefore, $\mathcal{C}$ will return a valid update key.

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

7. $\mathcal{A}_{ABE}$ must now run ProbGen to generate a challenge for either **Game 0** or **Game 1**. To do so, it samples three distinct messages m_0, m_1 and m' uniformly at random from the message space, and flips a random coin $RK_{F,x^\star} = b \xleftarrow{\$} \{0,1\}$. It submits m_0 and m_1 to $\mathcal{C}$ as its challenge, and receives back the encryption, $CT^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$), under attributes $\overline{x^\star}$, time $t^\star = 1$ and public parameters MPK^r_{ABE}. If $r = 0$, then $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$. Otherwise, $r = 1$ and $\mathcal{A}_{ABE}$ sets c_{1-b} to be $CT^\star$.

   $\mathcal{A}_{ABE}$ generates the remaining ciphertext itself by running ABE.Encrypt$(m', \overline{x^\star} = (A_{x^\star} \cup \phi(F)), t^\star = 1, MPK^{1-r}_{\text{ABE}})$.

   $\mathcal{A}_{ABE}$ then selects a random bit $s \xleftarrow{\$} \{0,1\}$ which functions as $\mathcal{A}_{ABE}$'s guess of the bit $b^\star$ chosen by $\mathcal{C}$. Let $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$. Then $\mathcal{A}_{ABE}$ computes the verification key $VK = (VK_0, VK_1, L_{Reg})$.

8. $\mathcal{A}_{ABE}$ now simulates the server S performing the computation to output $\theta_{F(x^\star)}$ by running Algorithm 3.6 as written, since valid keys have been generated for S in the preceding steps.

9. $\mathcal{A}_{VC}$ is given $\sigma_{F,x^\star}$, $\theta_{F(x^\star)}$, $VK_{F,x^\star}$, PK_F and PP, as well as oracle access which is handled as follows:

   - Queries to VC.FnInit and VC.Register are performed as in Algorithms 3.2 and 3.3.
   - Queries of the form VC.Certify$(S, F', \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Algorithm 3.4 with the following exceptions. To generate SK^r_{ABE}, $\mathcal{A}_{ABE}$ queries the ABE.KeyGen oracle for $\mathcal{O}^{\mathsf{KeyGen}}(S, F'_r \wedge \phi(F'), MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$ $\mathcal{C}$ will return the decryption key *unless* $\overline{x^\star}$ satisfies the queried policy.

     Notice that if the queried function F' is *not* the challenge function F then, due to the bijective mapping ϕ, $\phi(F') \neq \phi(F)$. Thus, neither of the possible queries (which both require the presence of $\phi(F')$) will be satisfied. If, however, the query *is* for F, recall that we chose the ABE system owned by the challenger to be unsatisfied by precisely this queried policy. Hence the first check performed

in Oracle 3.1 will never evaluate to `true` and therefore $\mathcal{C}$ will always be able to return a valid key in response to a KeyGen query.

To generate $UK^r_{L_{\text{Rev}},t}$, $\mathcal{A}_{ABE}$ makes a query to the $\mathsf{ABE.KeyUpdate}$ oracle of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* the current time is the challenge time (which $\mathcal{A}_{ABE}$ chose to be 1) *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$ which $\mathcal{A}_{ABE}$ chose to be empty. Since, $\overline{R} = \epsilon$ is always a subset of any L_{Rev}, the second clause will not be satisfied and $\mathcal{C}$ may generate a valid update key.

Thus, $\mathcal{A}_{ABE}$ can request valid decryption keys and update keys from $\mathcal{C}$ and can simulate Algorithm 3.4 exactly.

- Queries of the form $\mathsf{VC.Revoke}(\tau_{\theta_{F(x)}}, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Algorithm 3.9 as written except that, to form $UK^r_{L_{\text{Rev}},t}$, $\mathcal{A}_{ABE}$ makes a query of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* $t = 1$ *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. However, as $\mathcal{A}_{ABE}$ chose $\overline{R} = \epsilon$, this second clause is never satisfied and a valid update key is returned.

10. Eventually, $\mathcal{A}_{VC}$ finishes this query phase and outputs its guesses $RT_{F(x)}$ and $\tau_{\theta_{F(x)}}$. If $g(RT_{F(x)}) = g(m_s)$, $\mathcal{A}_{ABE}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{ABE}$ guesses $b' = 1 - s$.

Notice that if $s = b^\star$ (the challenge bit chosen by $\mathcal{C}$), then the distribution of the above coincides with **Game 0** (since the verification key comprises $g(m')$ where m' is the message a legitimate server could recover, and $g(m_s)$ where m_s is the other plaintext). Otherwise, $s = 1 - b^\star$ the distribution coincides with **Game 1** (since the verification key comprises the legitimate message and a random message m_{1-s} that is unrelated to the ciphertexts).

Now, we consider the advantage of this constructed adversary $\mathcal{A}_{ABE}$ playing the IND-sHRSS game: recall that by assumption, $\mathcal{A}_{VC}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1** — that is

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

where $\mathbf{Exp}_{\mathcal{A}_{VC}}^{\mathbf{Game\ i}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]$ denotes running $\mathcal{A}_{VC}$ in Game i.

$$\begin{aligned}
\Pr[b' = b^\star] &= \Pr\left[s = b^\star\right] \Pr\left[b' = b^\star | s = b^\star\right] + \Pr\left[s \neq b^\star\right] \Pr\left[b' = b^\star | s \neq b^\star\right] \\
&= \frac{1}{2} \Pr\left[g(RT_{F(x)}) = g(m_s) | s = b^\star\right] + \frac{1}{2} \Pr\left[g(RT_{F(x)}) \neq g(m_s) | s \neq b^\star\right] \\
&= \frac{1}{2} \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}_{VC}}^{\mathbf{Game\ 0}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] + \frac{1}{2}\left(1 - \Pr\left[g(RT_{F(x)}) = g(m_s) | s \neq b^\star\right]\right) \\
&= \frac{1}{2} \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}_{VC}}^{\mathbf{Game\ 0}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] + \frac{1}{2}\left(1 - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}_{VC}}^{\mathbf{Game\ 1}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}_{VC}}^{\mathbf{Game\ 0}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}_{VC}}^{\mathbf{Game\ 1}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}\right]\right] + 1\right) \\
&\geq \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

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

Since δ is assumed to be non-negligible, $\frac{\delta}{2}$ is also non-negligible. If $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ 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 $\mathcal{A}_{VC}$ 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 to implicitly be the challenge w in the one-way function inversion game (Game 2.12). 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, $\mathcal{A}_{VC}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof We now show that using $\mathcal{A}_{VC}$ in **Game 2**, $\mathcal{A}_{ABE}$ can invert the one-way function g — that is, given a challenge $z = g(w)$ $\mathcal{A}_{ABE}$ can recover w. Specifically, during

ProbGen, $\mathcal{A}_{ABE}$ chooses the messages as follows:

- if $F(x^\star) = 1$, we implicitly set m_{1-b} to be w and the corresponding verification key component to be z. We randomly choose m_b and compute the remainder of the verification key as usual.
- if $F(x^\star) = 0$, we implicitly set m_b to be w and set the verification key component to z. m_{1-b} is chosen randomly and the remainder of the verification key computed as usual.

Now, if $\mathcal{A}_{VC}$ is successful, it will output a retrieval key 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). $\mathcal{A}_{ABE}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{VC}$ has against the selective vindictive managers 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 3.1–3.9 is secure in the sense of selective vindictive managers. □

Theorem 3.2 follows as a corollary of Lemmas 3.1–3.4.

3.7 Conclusion

In this chapter, we have introduced the new notion of 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 a computation service for many different functions and for many different clients and any client may submit multiple requests to any available servers, whereas prior work considered just one server able to compute just a single function.

The consideration of this new model leads to new functionality 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 cheating and that, by permuting elements within messages, we achieve output privacy (as hinted at by Parno et al., although seemingly with two fewer decryptions than their brief description implies). We have shown that this blind verification could be used when a manager runs a pool of servers and rewards correct work — he needs to verify correctness but is not entitled to learn the result. 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 in real environments and provides a natural set of baseline definitions from which to add future functionality. For example, in Chapter 4 we will introduce an access control framework (using our scheme as a black box construction) to restrict the set of functions that clients may outsource, or to restrict (using the blind verification property) the set of verifiers that may learn the output. In this scenario, the KDC entity may, in addition to certifying servers and registering clients, determine access rights for such entities.

Chapter 4

Access Control in Publicly Verifiable Outsourced Computation

Contents

This chapter extends the model of revocable publicly verifiable outsourced computation introduced in Chapter 3, in order to enforce access control policies over delegators, servers and verifiers such that entities are restricted over which operations they may perform within the system. We discuss policies of interest in this setting and provide a solution with a pragmatic blend of symmetric and asymmetric cryptographic enforcement mechanisms, where attribute-based encryption is used as a proof of correctness for computations while symmetric key assignment schemes are used to enforce access control policies.

4.1 Introduction

In Chapter 3 we introduced revocable publicly verifiable outsourced computation (RPVC) which includes a trusted key distribution centre (KDC) that is an authority on entities and performs expensive setup operations and key management duties. This leads to a more decentralised system architecture comprising a pool of delegators and a pool of servers (*the*

manager model), with the delegator not necessarily knowing the server chosen to perform a computation. Any delegator can submit a request for work (or job) to the server pool and some system-dependent mechanism allocates it to an available server (based on availability, suitability or on some bidding process if operating on a price per computation basis). This contrasts with prior models where the delegator chose a single server with whom to set up a verifiable computation (VC) system; in such settings, the server could be explicitly selected and authenticated beforehand. Comparatively, then, delegators now have less control over the servers that perform work on their behalf and that may access their data or computation results.

As with any multi-user setting, we may wish to control access to resources in an RPVC system. The contribution of this chapter, therefore, is to 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 to (i) restrict the computations that may be outsourced by delegators; and (ii) 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 functionality. We may restrict the set of delegators that may outsource a computation to those that would be authorised to compute it (if given sufficient resources) by the organisation's policies. The second requirement arises from the fact that in the RPVC setting, delegators may not be able to authenticate servers beforehand. 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.

Some VC settings, as in Chapter 3, distinguish between delegators and verifiers. Delegators (or distinguished verifiers) may learn the result of the computation, whereas standard verifiers may only confirm that the result was computed correctly; this is characterised by the *blind verification* property for standard verifiers. Again, in a multi-user VC setting, we may wish to restrict the users authorised to: (i) verify the result; and (ii) learn the result. When operating on sensitive data, this second restriction ensures that read access to the generated results is limited to those that satisfy the access control policy, e.g. only entities that may read the input data may read the output.

A third motivation for access control in the VC setting is that computational services

may be charged for (e.g. in subscription-based utility computing [65, 85]) and that service providers may offer different levels of service to different clients (e.g. different subscription tiers 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 [29, 87], servers enforce access control policies by authenticating users and granting or denying access based on *access control lists* or *capability lists* (see Section 2.3). This reference monitor approach is not appropriate in the multi-user VC setting since the servers are assumed to be untrusted and may have a vested interest in violating the policies. We instead use cryptographic mechanisms to enforce access control policies, where cryptographic keys are used to encrypt objects and thereby restrict access to only authorised users —- the access control mechanism thus reduces to the appropriate distribution of keys to authorised entities. We use the trusted key distribution centre (KDC) introduced in the revocable PVC model (RPVC) and, as an authority on entities within the system, we extend its duties to also instantiate the access control mechanism. In RPVC, the KDC issued keys that enable the delegation and evaluation of particular functions; we additionally require it to issue keys appropriate to the access control policy that enable read and write access to certain components of the RPVC 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 an RPVC scheme, the KDC implicitly provides some access control in 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). In particular, a delegator may request a computation that the delegator itself is not authorised to perform locally.

Cryptographic enforcement mechanisms are particularly appropriate when objects and policies are relatively static (so object re-encryption and user revocation are rare events). In the context of verifiable computation, 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, $\mathcal{F}$) 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. However, VC leads to a somewhat novel application of these mechanisms as we will see in Section 4.2.

We begin by briefly reviewing some related work. Clear et al. [43] 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. In independent and concurrent work, Xu et al. [96] 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. We believe that the PVC setting also necessitates that delegators may specify access control requirements on those *servers* that may be selected for a given computation and, especially, limits must be placed on *verifiers* that may learn the output. Xu et al. discuss their notion purely in terms of using CP-ABE as the enforcement mechanism and did not discuss the form of the policies; in contrast, we discuss in detail the types of policies that may be of interest in these settings and present these in terms of generic graph-based access control policies. Such policies may be enforced by a variety of enforcement mechanisms, including symmetric KASs as used here (which may well be more efficient than the pairing-based CP-ABE approach). We believe that we present a more generic treatment which, importantly, extends to the multi-user PVC setting.

In Section 4.2, we discuss example access control policies that are relevant to the setting of multi-user VC. We formulate policies in a generic fashion (irrespective of the cryptographic enforcement mechanism employed) in terms of authorisation labels and graph-based policies over entities and computations. Then, in Section 4.3, we extend the work of Chapter 3 to formally define a framework for revocable PVC with access control (PVC-AC). This results in novel security notions that we introduce in Section 4.4. Finally, in Section 4.5, we provide an example construction that generically extends any RPVC scheme to use a symmetric Key Assignment Scheme (KAS), and in 4.6 show that this construction meets the stated security goals.

4.2 Access Control Policies for PVC Environments

We consider graph-based policies where "objects" to be protected are not data files, as in traditional access control policies, but outsourced computations and their results. The "user" population comprises the sets of delegators $\mathcal{C}$, computational servers $\mathcal{S}$, and verifiers $\mathcal{V}$. We are interested in specifying and enforcing policies that restrict: (i) which computations a delegator may outsource; (ii) which computations a server may evaluate; (iii) which outputs a verifier may read. As noted in the introduction to this chapter, we distinguish between entities that may learn a computational result and those that may only verify correctness (using the blind verification property from Chapter 3). In this chapter, we assume that *any* entity may perform the blind verification step but that only a restricted subset of verifiers should be able to retrieve the actual output value.[1]

There has been considerable research in recent years on the cryptographic enforcement of access control policies [1,44,46,51]. Informally, access is regulated in a distributed fashion by issuing appropriate cryptographic keys to authorised subjects, rather than a centralised reference monitor mediating all attempts to access protected resources. Cryptographic access control generally focuses on read access and is regarded as being particularly appropriate when the protected content is read often but written rarely (since this would require re-encryption). In the context of verifiable computation, we will use cryptographic access control in somewhat unusual ways. Rather than storing static encrypted data, we will encrypt dynamic messages within a protocol execution. In particular, to enforce policies restricting the computations that may be outsourced, a delegator must use an appropriate key to encrypt input data. Without the appropriate encryption, the input will be discarded by the server. The enforcement of policies for performing computations is achieved by distributing keys to servers that can be used to decrypt the (encrypted) inputs. Without decryption, the server will be unable to read the input data and evaluate the function; an unauthorised server cannot determine anything about the input data and so we additionally achieve input privacy in this case. The enforcement of (read) policies on outputs uses cryptographic access control in a more conventional fashion; results are published and protected via encryption with an appropriate key.

Cryptographic access control has been particularly widely studied in the context of in-

[1] The same techniques that we present could also be applied to protect the (blind) verification keys to restrict the entities that may validate correctness of a computation.

formation flow and graph-based access control policies (see Section 2.3). We restrict our focus to graph-based policies as these have been shown to encompass many notions of access control that are desirable in practice including information flow policies, role-based access control and attribute-based access control [44]. Recall that we define the "user" population as the sets of delegators $\mathcal{C}$, computational servers $\mathcal{S}$, and verifiers $\mathcal{V}$. We define a security labelling function $\lambda : \mathcal{C} \cup \mathcal{S} \cup \mathcal{V} \cup \mathcal{O} \to L$ where $\mathcal{O}$ is the family of computations that may be outsourced and $(L, \leqslant)$ is a poset of security labels. This function assigns a label from L, representing the security classification, to each delegator, server and computation in the PVC-AC system. The access control policy requires that $\lambda(E) \geqslant \lambda(o)$, for an entity $E \in \mathcal{C} \cup \mathcal{S} \cup \mathcal{V}$ attempting to operate on a computation $o \in \mathcal{O}$.

Throughout this chapter, we refer to computations as $o = (F, x, \texttt{aux}) \in \mathcal{O}$ (in reference to their role as protected objects in the access control system). Each computation o specifies all information required to formulate the access control policy: the function F to be computed, the input data x, and any relevant contextual information, which we denote by `aux`. Thus, the labelling function λ considers all these factors, including any specified contextual information, when determining the security label $\lambda(o)$. Observe that it may not always be the case that a computation may be considered purely in terms of the function and input data alone. Indeed, although the evaluation result will be uniquely determined by such factors, the level of protection required may depend on other contextual information. As a simple motivating example, consider a summation function over the integers. The semantic meaning of the integers in question may determine the overall classification of this computation — if the integers are city populations then this may not be classified at all, but troop deployments in different regions may be much more sensitive.

Although the format of access control policies, throughout this chapter, remains constant (requiring $\lambda(E) \geqslant \lambda(o)$), different choices for the sets $\mathcal{O}$ and L support different types of policies, particularly in terms of granularity. In Section 4.2.1, we consider several different choices for these sets to restrict the computations a delegator may outsource and that a server may compute. We begin by examining the simple case where policies are defined only over the choice of functions — that is, $\mathcal{O} = \mathcal{F}$. Thus entities are authorised to operate on sets of functions, similarly to how servers are certified to compute specific sets of functions in RPVC. We then, in Section 4.2.1.2, consider more fine-grained policies defined over functions *and* input data to restrict entities to only operating on specific, au-

thorised functions with specific inputs. This provides a general solution for settings where contextual information regarding computations is not relevant in terms of formulating access control policies. Finally, in Section 4.2.1.3, we consider the integration of additional contextual information when defining objects, and also consider alternative definitions for the poset of security labels, L, to integrate with existing access control policies operated by an organisation; thus, outsourced computations can be protected in the same way that internal operations are protected.

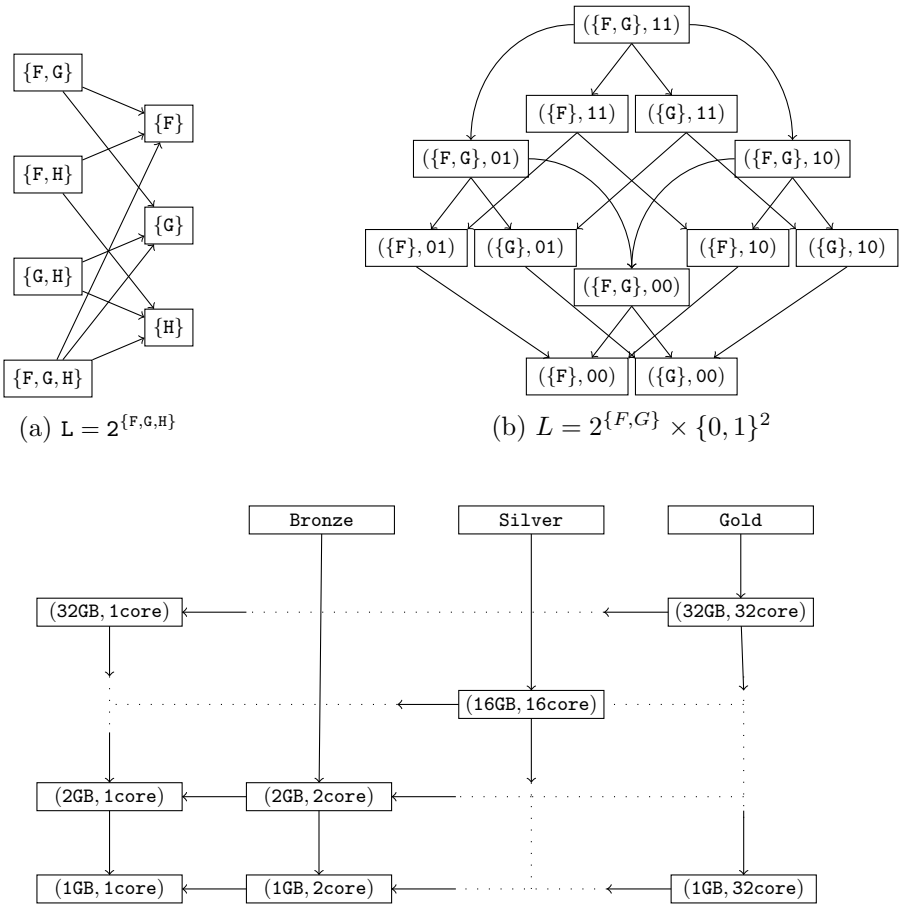
In Section 4.2.2, we discuss policies that restrict the results a *verifier* may learn. We differentiate between computation policies over delegators and servers, defined in terms of a labelling function $\lambda^C(\cdot)$, and verification policies over delegators and verifiers, defined in terms of $\lambda^V(\cdot)$. For ease of notation, we use λ to denote λ^C in Section 4.2.1, and to denote λ^V in Section 4.2.2.

4.2.1 Delegation and Computation Policies

4.2.1.1 Policies over Functions

We begin by considering a simple case where policies are formulated purely in terms of the functions being computed. Specifically, in this section, objects (or computations) are synonymous with functions ($\mathcal{O} = \mathcal{F}$) while security labels represent sets of function ($L = 2^{\mathcal{F}}$). In simple terms, a computation $o \in \mathcal{O}$ is labelled by the function being computed, and 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. We also define a *security criterion* that requires $\lambda(C) \supseteq \lambda(o)$ and $\lambda(S) \supseteq \lambda(o)$ in order to delegate or compute the computation o respectively, and that a set of unauthorised entities cannot collude to perform an operation that any of them couldn't 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 $<$



(a) $\mathtt{L} = 2^{\{\mathtt{F},\mathtt{G},\mathtt{H}\}}$

(b) $L = 2^{\{F,G\}} \times \{0,1\}^2$

(c) Subscription based computation: $L = (\{\mathtt{1GB},\ldots,\mathtt{32GB}\} \times \{\mathtt{1core},\ldots,\mathtt{32core}\}) \cup \{\mathtt{Gold},\mathtt{Silver},\mathtt{Bronze}\}$

Figure 4.1: Example posets for publicly verifiable outsourced computation with access control

such that $x < y$ if and only if $x \in \mathcal{F}, y \subseteq \mathcal{F}$ and $x \in y$; that is, x must be a singleton set (or in our case, a computation for a particular function), y must be a set of functions (i.e. be a label assigned to an entity authorising it to compute the functions in y), and x must be a subset of y (to ensure the entity is authorised for the particular function x). The corresponding Hasse diagram with $\mathcal{F} = \{F, G, H\}$ is shown in Figure 4.1a.[2] Any entity E authorised for $\lambda(E)$ is, by the correctness criterion, authorised to operate on all computations o labelled by $\lambda(o) < \lambda(E)$. For example, in Figure 4.1a, if $\lambda(E) = \{F, G\}$ then E is authorised for the functions F and G as expected. In a cryptographic enforcement mechanism, 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 $\kappa_o = \kappa_{\{F\}}$ associated to the label $\{F\}$ 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; S may do this if and only if $\{F\} \in \lambda(S)$.

[2] Nodes for empty sets are excluded from these figures.

4.2.1.2 Policies over Function Inputs

As well as limiting the functions that may be outsourced, we may wish to implement a more fine-grained access control policy determined by input values to functions. For ease of exposition let us assume that all functions have the same domain — for all $F \in \mathcal{F}$, $\mathsf{Dom}(F) = \{0,1\}^n$ for a positive, non-zero integer n.[3] We then redefine the security function such that "objects" (computations) are now considered to be pairs (F, x) where $F \in \mathcal{F}$ and $x \in \mathsf{Dom}(F)$ — that is, $\lambda : \mathcal{C} \cup \mathcal{S} \cup (\mathcal{F} \times \{0,1\}^n) \to L$.

Different settings may require different access control functionality, which can be achieved through different choices of security labels. For example, the first choice of poset we present in this section corresponds to the idea of assigning a data range to each delegator and then authorising them to outsource a specific set of functions on that data. For instance, a Chinese wall policy may result in employees being provided with separate partitions of a database. Then, depending on the employee's role, they are permitted to evaluate a specific set of functions on that data partition. The second choice of poset we present is more fine-grained and authorises delegators to outsource a single, specified computation — that is, it permits the entity to evaluate $F(x)$ for a specific choice of function and input data. For example, one could imagine an employee being given restricted access to some sensitive data to perform only a particular task, and they should not be able to use the data for other purposes.

For the first case, where entities are assigned a set of data and are authorised to compute specific functions on that data, we could define L to be $2^{\mathcal{F}} \times \{0,1\}^n$. To define the order relation on L we must first define an ordering on the input data. One choice is to define a co-ordinatewise ordering on $\{0,1\}^n$; that is $(x_1, \ldots, x_n) \leqslant (y_1, \ldots, y_n)$ if and only if $x_i \leqslant y_i$ for all $i \in [n]$. Then, for two labels $(\mathcal{G}, x)$ and $(\mathcal{G}', x')$ in L, $(\mathcal{G}, x) \leqslant (\mathcal{G}', x')$ if and only if $\mathcal{G} = \{F\}$ for some $F \in \mathcal{F}$, $F \in \mathcal{G}'$ and $x \leqslant x'$.

For an entity E, we define $\lambda(E) = (\mathcal{G}, x)$, where $\mathcal{G} \subseteq \mathcal{F}$ and $x \in \{0,1\}^n$. For a computation o of F on input x, we define $\lambda(o) = (\{F\}, x)$. Then E is authorised to operate on $G(y)$ for all $G \in \mathcal{G}$ and all $y \leqslant x$. The Hasse diagram for this poset with $\mathcal{F} = \{F, G\}$ and $n = 2$ is shown in Figure 4.1b. Different choice of orderings over inputs lead to different

[3] If this is not the case then we can define $n = \max_{F \in \mathcal{F}} |\mathsf{Dom}(F)|$ and then redefine all functions F with $\mathsf{Dom}(F) < n$ to satisfy the required property by, for example, adding fixed points $F(x) = x$ for all $x \in \{0,1\}^n \setminus \mathsf{Dom}(F)$.

restrictions; for example, one could consider bit strings as integers and use the natural ordering over integers for $x \leqslant y$.

For the second case, where entities are authorised to compute functions on specific inputs, define L to be the power set $2^{\mathcal{F}\times\{0,1\}^n}$. Then each function may be associated with a different range of permissible inputs. An entity E, is labelled by a set of pairs, each comprising a function label and an associated input label — $\lambda(E) = \{(F_1, x_1), \ldots, (F_m, x_m)\}$. A computation of $F(x)$ is labelled $\lambda(o) = (\{F\}, x)$. Then, for any two labels,

$$\{(F_1, x_1), \ldots, (F_m, x_m)\} \leqslant \{(F'_1, x'_1), \ldots, (F'_{m'}, x'_{m'})\}$$

if and only if $m = 1$ (the first label is a single pair), $F_1 = F'_i$ for some $i \in [m']$ and $x_1 \leqslant x'_i$.

Finally, we observe that the choice of $L = 2^{\mathcal{F}} \times 2^{\{0,1\}^n}$ extends this second case to a situation where each function can be associated with an arbitrary set of input values.

4.2.1.3 Policies for Other Security Labels

We have seen how sets of functions and inputs can be mapped to a graph-based access control policy to restrict the functions a delegator may outsource and that a server may compute. However, in practical applications, outsourced computation functionality will be required to integrate with existing workflows and existing access control policies. As an example, consider a company that operates role-based access control (RBAC) (see Section 2.3.2) on their local network and wishes to provide access to an external VC system. The company must ensure that the same access control requirements are adhered to within this new environment.

As mentioned in Section 4.2, it is not always appropriate to classify computations based purely on the function and input — other contextual information may be required. We now briefly discuss how to formulate access control policies for multi-user VC settings that incorporate additional security labels relying on the environment or external access control policies.

We can use any poset of security labels to classify each outsourced computation. Consider the total order $M = \{\texttt{Top-Secret}, \texttt{Secret}, \texttt{Classified}, \texttt{Unclassified}\}$ representing the

Bell-LaPadula clearance levels [22], and let K be a set of need-to-know categories. Then we can enforce the security function $\lambda : \mathcal{C} \cup \mathcal{S} \cup \mathcal{O} \rightarrow M \times 2^K$ where K specifies the nature, and M specifies the sensitivity, of the computation $o \in \mathcal{O}$. To delegate or compute $F(x)$, we require that the entity E's clearance level is at least the classification of the computation: $\lambda(E) \geqslant \lambda(o)$.

It has been shown [44] that the set of roles for a role-based access control policy can be encoded as a poset. Thus, we could similarly define L to be this role poset to integrate with existing RBAC policies within an organisation.

As well as changing the set of security labels, we could also add additional criteria in the mapping to security labels. For example, we could formulate policies dependent on time by setting $\lambda : \mathcal{C} \cup \mathcal{S} \cup (\mathcal{F} \times \mathcal{T}) \rightarrow L$, where $\mathcal{T}$ is a set of time periods. We could then set $L = 2^{\mathcal{F} \times \mathcal{T}}$ to allow entities to operate on specific functions only during specified time periods. On the other hand, by defining $L = M \times 2^K$ as above, certain functions can be more highly protected during certain times of the day. In place of a temporal poset $\mathcal{T}$ we could also apply a geo-spatial poset to classify function evaluations differently depending on location data;, e.g. a function may be more sensitive if being outsourced from a battlefield as opposed to within a secured building. Finally, policies over function inputs can be extended to include characteristics of the input data (as in the summation example in Section 4.2, contextual information often changes the level of protection required). Recall that we define computations $o \in \mathcal{O}$ in terms of F, x and some auxiliary information `aux` which provides contextual information regarding the computation that may be relevant in determining the classification of the computation, e.g. the semantic meaning of input data. Then, if A is the set of possible auxiliary information and L is defined to be $2^{\mathcal{F} \times (\{0,1\}^n \times A)}$, for example, then the same input data to the same function can be classified differently, and hence require different authorisation from the delegator and the server, based on contextual information.

As mentioned in the introduction to this chapter, one interesting setting for RPVC is when users pay for computation-as-a-service [65,85]. Users may pay a price per computation, in which case they could be issued a relevant key such as those discussed in Section 4.2.1.2, perhaps with a short time window in which to perform the computation. Then, when outsourcing the computation, it is protected by this key to prove that the user has paid for this particular computation. An alternative model [65] would be for a server to allow

subscriptions to different tiers of service. A user may pay more to subscribe to a higher tier, and then may submit multiple computations relating to this tier or lower. For example, consider a set $T = \{\texttt{Gold}, \texttt{Silver}, \texttt{Bronze}\}$ comprising tiers that users may subscribe to; `Gold` service may allow access to more computational resources, or access during busier periods, than `Silver`, and so on. For simplicity, suppose the resources to be considered are just RAM capacity and the number of processor cores available, and let R and C be sets comprising the available quantities of each. Then, the set of security labels L is set to be $L = (R \times C) \cup T$ —- that is, the cartesian product of the available resources, with the addition of labels for each service tier. A user that pays for `Gold` membership can be assigned the label $\lambda(C) = \{\texttt{Gold}\}$. Each computation is labelled by the resources that it requires. The poset ordering over L allows each tier to access different subsets of resources (e.g. if a computation is particularly memory intensive then it may require a higher subscription fee to compute), as illustrated in Figure 4.1c.

4.2.2 Verification Policies

Thus far, we have discussed policies restricting the entities that may delegate and evaluate given computations. In the multi-user VC setting, it is equally important to apply access control to the act of verifying the results and learning the computational output. Clearly, when considering computations over confidential data, it is not always appropriate to publish the results as in a publicly verifiable computation scheme. As a trivial example, outsourcing the identity function could "legitimately" leak confidential data. In Chapter 3, we distinguished between the actions of blind verification (to ensure correctness) and output retrieval (to learn results). In this chapter, we assume that all entities are able to blindly verify a result, but the set of entities that may learn the actual output should be restricted[4]. Thus, the set of verifiers we refer to in the following are just those distinguished verifiers that are able to retrieve computation results, and we wish to restrict the results that each may read.

In this section, we consider two forms of verification policy:

1. The notion of "no write down" is an important property in many traditional access control policies. In the VC setting, this amounts to ensuring that any verifier should

[4]However, we could use the same techniques to restrict blind verification if desired.

have at least the access rights of the delegating or computing entity, that is $\lambda(C) \leqslant \lambda(V)$ and $\lambda(S) \leqslant \lambda(V)$. In short, a verifier may not read a result arising from input data that he is not also authorised to read. We consider this form of policy in Section 4.2.2.1.

2. In some cases, however, the KDC may decide that the results of some computations are not as highly classified as the input data, or indeed the act of computing it. For example, statistics of company spending averaged across all departments may be less sensitive than the spending of the research department alone. Alternatively, the results of a computation over classified data may be included in a public document, despite the input data remaining classified.

   In these cases, the encoded output from the computation may be published along with a public verification key so that recipients can very the legitimacy of the computational result; however, the retrieval key can also be published alongside the computation, but be cryptographically protected (encrypted) such that only authorised verifiers (e.g. trusted reviewers of a document) may access it and therefore retrieve the actual output of the computation. This setting is explored in Section 4.2.2.2.

4.2.2.1 Enforcement of No Write Down Policies

As mentioned, "no write down" is an important requirement of many access control policies. It ensures that an entity C may not write (encrypt) an object to a lower classification level $\lambda(o) < \lambda(C)$ as this could constitute a leak of classified data. It is, of course, possible for a delegator to write the result at his (maximal) clearance level and then any verifiers with higher access rights, by the correctness criterion of the cryptographic enforcement mechanism, will be able to read the result. However, we may want to protect a result at a higher classification level (write up), e.g. when preparing a report that should only be read by management.

To enforce no-write down policies, we encrypt the retrieval key RK_o for the computation under a suitable key according to the verification policy. Recall from Chapter 3 that, in an RPVC scheme, the retrieval key is generated by the delegator during the ProbGen stage, and is used in the Retrieve algorithm run by a verifier to reveal the actual output value of the computation.

Within a PVC-AC system, we define two posets: a computation poset $\mathcal{P}_C = (L, \leqslant)$ (as per Section 4.2.1) to encode computational access control policies, and a verification poset $\mathcal{P}_V$ which encodes the no-write down verification policy. $\mathcal{P}_V$ is constructed by inverting the order relation on $\mathcal{P}_C$ — that is, if $\mathcal{P}_C = (L, \leqslant)$ then $\mathcal{P}_V = (L, \geqslant)$ where $x \geqslant y$ in $\mathcal{P}_V$ if and only if $x \leqslant y$ in $\mathcal{P}_C$. Then, delegation and computation of $o \in \mathcal{O}$ are performed using a key associated to $\lambda^C(o) \in \mathcal{P}_C$. To retrieve the result of a computation, a verifier must recover the retrieval key which is encrypted using a key related to $\lambda^V(o) \in \mathcal{P}_V$. Thus, a verifier V may successfully decrypt, and recover, RK_o and hence read the computation result if and only if $\lambda^V(V) \geqslant \lambda^V(o)$.

4.2.2.2 Published Retrieval Key

In the second form of verification policy we consider in this section, where retrieval keys are published according to a more general policy that may allow write-down, we can use similar posets and security functions to those in Section 4.2.1. The security function is defined to be $\lambda^V : \mathcal{V} \cup \mathcal{C} \cup \mathcal{O} \to L$, where $\mathcal{O}$ is the set of outsourced computations and $(L, \leqslant)$ is a poset of security labels. The user population is the set of delegators $\mathcal{C}$ and verifiers $\mathcal{V}$, and objects, $o \in \mathcal{O}$ are retrieval keys for particular computations. We will protect (encrypt) the retrieval key for each computation using an appropriate key. We require the verifier to satisfy the verification policy in order to derive the corresponding decryption key — that is, we require $\lambda(V) \geqslant \lambda(o)$ for $o \in \mathcal{O}$. Since the delegator generates the retrieval key, and hence may also use it to retrieve the output, we also assume $\lambda(C) \geqslant \lambda(o)$. This is reasonable given that the delegator must have the right to compute any function that he outsources and hence (resources permitting) could certainly learn the result locally. As before we can extend the definition of λ and of L to accommodate more fine-grained policies over outputs if required.

4.3 PVC with Access Control

We now formally define the notion of PVC-AC to enforce graph-based access control policies over delegators, servers and verifiers. Recall that we wish to impose restrictions on three activities: first, we wish to specify a (write) policy determining the computations a

client is authorised to delegate — this means restricting the inputs a client may correctly prepare; second, we specify a (read) policy that determines the computations a server may compute — that is, encoded inputs he may access; lastly, we specify a (read) policy dictating the outputs a verifier may read. We differentiate between computation policies over delegators and servers, denoted $\lambda^C(\cdot)$, and verification policies over delegators and verifiers, denoted $\lambda^V(\cdot)$. We may also use $\lambda(\cdot)$ to denote both labels where no ambiguity arises. Finally, $\mathcal{P}_C$ and $\mathcal{P}_V$ denote posets encoding the computation and verification policies respectively. Recall that the set of outsourced computations is denoted by $\mathcal{O}$. A computation $o \in \mathcal{O}$ comprises the function F, the input data x and possibly some additional contextual information.

The entities and trust relations remain the same as in Chapter 3. We extend the functionality of the KDC from Chapter 3 to grant access control credentials to delegators, servers and verifiers. We may split the responsibilities between two KDCs: a *computation* KDC (CKDC) that generates function keys for the VC functionality, and an *authorisation* KDC (AKDC) that manages access control policies. The AKDC could be a trusted authority on entities to grant relevant permissions, and may be used by multiple systems (e.g. as a form of federated identity management [36]). For ease of exposition, here we use only one KDC that performs both duties.

Definition 4.1. A *revocable publicly verifiable outsourced computation scheme with access control* (PVC-AC) comprises the following algorithms:

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V)$: run by the KDC to generate public parameters PP for the PVC-AC system as well as secret information MK used to generate keys. The inputs to this algorithm are the security parameter, the family of functions $\mathcal{F}$ that may be outsourced, and the computational and verification posets encoding the graph-based access control policies to be enforced;
- $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate a delegation key, PK_F, that enables the outsourcing of computations for a function F;
- $SK_{\mathrm{ID}} \xleftarrow{\$} \mathsf{Register}(\mathrm{ID}, \lambda(\mathrm{ID}), \mathrm{MK}, \mathrm{PP})$: run by the KDC to issue a key SK_{ID} for an entity with identifier ID[5] which grants access rights for the label $\lambda(\mathrm{ID})$ according to the computation or verification policy;

[5]In future algorithms this will be S, C or V to denote a server, delegator or verifier respectively.

- $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$: run by the KDC to create an evaluation key $EK_{F,S}$ that enables a server S to compute a function F;

- $(\sigma_o, VK_o, RK_o) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \mathrm{PP})$: run by a delegator C to outsource the computation o of $F(x)$. The algorithm takes the input x, C's secret key SK_C, the public delegation key PK_F for F, the security label $\lambda(C)$ of the delegator, the security label $\lambda(o)$ of the computation to be outsourced and the public parameters PP. If C satisfies the computation policy (i.e. $\lambda^C(C) \geqslant \lambda^C(o)$) then the algorithm outputs a valid encoded input σ_o, a public verification key VK_o to verify correctness, and a protected output retrieval key RK_o which only verifiers satisfying $\lambda^V(V) \geqslant \lambda^V(o)$ may use to read the result $F(x)$;

- $\theta_o \xleftarrow{\$} \mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), \mathrm{PP})$: run by a server S, this algorithm takes as input an encoded input σ_o for a computation $F(x)$, an evaluation key for F, a secret key SK_S, the security labels of the server and computation and the public parameters. It generates a valid encoding, θ_o, of the result $F(x)$ if and only if $\lambda^C(S) \geqslant \lambda^C(o)$;

- $(y, \tau_{\theta_o}) \leftarrow \mathsf{Verify}(\theta_o, SK_V, VK_o, RK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})$: verification comprises two steps:

  - $(RT_o, \tau_{\theta_o}) \leftarrow \mathsf{BVerif}(\theta_o, VK_o, \mathrm{PP})$: run by *any* verifier holding an encoded output and a verification key to produce a retrieval token RT_o and a token $\tau_{\theta_o} = (\mathsf{accept}, S)$ for a correct result, or $\tau_{\theta_o} = (\mathsf{reject}, S)$ to signify a cheating server S;

  - $y \leftarrow \mathsf{Retrieve}(SK_V, RT_o, \tau_{\theta_o}, VK_o, RK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})$: run by a verifier V in possession of the retrieval token RT_o from BVerif *and* the output retrieval key RK_o. If $\lambda^V(V) \geqslant \lambda^V(o)$ then V should be able to read the actual result $y = F(x)$ or $\perp$ (if the result is incorrect);

- $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_o}, F, \mathrm{MK}, \mathrm{PP})$: if a verifier reports a misbehaving server (i.e. Verify returned $\tau_{\theta_o} = (\mathsf{reject}, S)$), the KDC issues updated evaluation keys $UM = EK_{F,S'}$ to all servers, preventing S from performing further computations. If $\tau_{\theta_o} = (\mathsf{accept}, S)$ then this algorithm should output $UM = \perp$;

Intuitively, we say that a publicly verifiable outsourced computation scheme with access control (PVC-AC) is *correct* if, when all algorithms are run honestly in any arbitrary

execution sequence by authorised entities and the result is computed by a non-revoked server, the verifying party always accepts the returned result *and* the result is correct.. As in Chapter 3, we can model this as a game played between a challenger and a PPT adversary.

For each algorithm in Definition 4.1, we define an oracle which executes the corresponding algorithm on arguments provided by the adversary, and returns the output of the algorithm to the adversary, as well as maintaining some internal lists, which we shall detail below. The adversary may query the Setup oracle only once (before making any other oracle queries), but can thereon call the remaining oracles any number of times and in any order.

The challenger maintains two lists, L_{Reg} and L_F. L_{Reg} is a list of tuples of the form (ID, SK_{ID}) of entity identifiers with the associated signing key; these are added by the Register oracle. L_F, on the other hand, comprises tuples of the form $(S, F, EK_{F,S})$ denoting that $EK_{F,S}$ was generated as a result of the server S being queried to the Certify oracle for the function F — that is, S has been certified to compute F. If the adversary queries the Revoke oracle with an input $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ for some S, the challenger removes all entries of the form $(S, \cdot, \cdot)$ (i.e. all entries for S for any function) from L_F.

The challenger also creates and maintains a table T which records the parameters and values relating to each computation performed through the oracle queries. T is updated in the following oracles:

- ProbGen: if $\lambda^C(C) \geqslant \lambda^C(o)$, the challenger creates a new row in T comprising 8 components, all of which are initialised to be empty; it then assigns x, F, the result $F(x)$ (computed by the challenger itself), $\sigma_{F,x}, VK_{F,x}$ and $RK_{F,x}$ to the first 6 components;

- Compute: if $\lambda^C(S) \geqslant \lambda^C(o)$, the challenger first searches T for all rows that contain the queried $\sigma_{F,x}$ in the 4^{th} component and where the 7^{th} component is empty (i.e. those rows relating to computations on this encoded input that have not yet been performed).

  For each such row, r, the challenger takes the second component (the function identifier, $\tilde{F}$), and checks that there exists a server identity $\tilde{S}$ such that the tuple $(\tilde{S}, SK_{\tilde{S}}) \in L_{\text{Reg}}$ (where $SK_{\tilde{S}}$ is that given as input to the Compute oracle) *and*

such that the tuple $(\tilde{S}, \tilde{F}, EK_{F,\tilde{S}}) \in L_F$ (where $EK_{F,\tilde{S}}$ is also that given as input to the Compute oracle). This check ensures that there is a currently un-revoked server (as the entries of L_F for $\tilde{S}$ have not been removed) that holds the signing key and evaluation key being used to perform the computation and which is certified for a function $\tilde{F}$ for which the encoded input $\sigma_{F,x}$ for this computation was generated.

The challenger then performs the Compute algorithm on the queried $\sigma_{F,x}, EK_{F,S}$ and SK_S to produce an output $\theta_{F(x)}$. For each of the rows r of T found above, the challenger writes $\theta_{F(x)}$ and $\tilde{S}$ to the 7^{th} and 8^{th} components of r respectively. Thus, a row of T will only have a (non-empty) value in the 7^{th} component if there exists a non-revoked, certified, authorised server to perform the computation for which $\sigma_{F,x}$ was generated.

Thus, when complete, the entries of T will be of the form

$$(x, F, F(x), \sigma_{F,x}, VK_{F,x}, RK_{F,x}, \theta_{F(x)}, S).$$

Note that the oracles only update entries in T when the given arguments relate to an authorised entity for the computation (we are not interested in the output of algorithms run by unauthorised entities as these should be rejected in accordance with the security models we define shortly).

After a polynomial number of queries, the adversary will return a value $\theta^{\star}_{F(x)}$ which he believes either encodes an incorrect computational result or which encodes a correct computational result yet which the Verify algorithm will reject. The challenger first performs a look up in T for all entries containing $\theta^{\star}_{F(x)}$ in the 7^{th} position of the tuple, and stores any such entries as another table $\tilde{T}$. Note that this means that $\theta^{\star}_{F(x)}$ must have been honestly generated by the Compute oracle (else it would not be in T).

For each such row, the challenger uses the 5^{th} and 6^{th} components of the row (the verification key and retrieval key) to run Verify on $\theta^{\star}_{F(x)}$ to generate the outputs y and $\tau_{\theta_{F(x)}}$[6]. The challenger first checks whether y matches the 3^{rd} component of the row (that is, whether y is the correct computational result $F(x)$). If so, it then checks whether $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$, and if so it ends the game by returning 1 to indicate that the adversary has won the game (the adversary has found a valid encoding of a correct result, computed by a certified,

[6]Note that the challenger can simulate a verifier that is authorised to perform this verification.

authorised, non-revoked server, that the Verify algorithm is incorrectly rejecting).

On the other hand, if y did *not* match the correct value of $F(x)$, the challenger also ends the game by returning 1 to indicate that the adversary has won the game (the adversary in this case has found an incorrect result that was computed honestly by authorised entities).

If no row in $\tilde{T}$ allows the adversary to win, then the challenger outputs 0 to indicate that the adversary has lost.

A PVC-AC scheme is *correct* if, for all PPT adversaries, the probability that the adversary wins the game described above is 0.

4.4 Security Models

We now introduce several security models capturing requirements of PVC-AC. These models capture the intuition that (i) only authorised entities can perform each type of restricted operation, and (ii) unauthorised servers learn nothing about the input data. These are important both to ensure that the enforcement mechanism for the access control policies correctly captures the required properties, and that the access control mechanism and the PVC implementation (if built from separate primitives) interact securely. As noted by Ferrara et al. [51], it is not always immediate that (probabilistic) cryptographic enforcement mechanisms can safely implement access control policies, which are generally specified in absolute terms; hence formally defining and proving the required security goals is necessary. Notions of public verifiability, revocation and vindictive servers follow naturally from Chapter 3 with additional inputs for policy declarations. As these do not rely on the access control properties, the proofs follow naturally, and we do not replicate them here. Throughout the remainder of this chapter, the notation $\mathcal{A}^{\mathcal{O}}$ denotes an adversary $\mathcal{A}$ being given oracle access to the following functions: $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{ProbGen}(\cdot, SK_{(\cdot)}, \cdot, \cdot, \cdot, \mathrm{PP})$, $\mathsf{Compute}(\cdot, \cdot, SK_{(\cdot)}, \cdot, \cdot, PP)$ and $\mathsf{Revoke}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$. Each oracle, unless otherwise given alongside the game, simply runs the relevant algorithm.

Game 4.1 $\mathbf{Exp}_{\mathcal{A}}^{\text{AUTHO}}\left[\mathcal{PVC}_{\text{AC}}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]$

$L_R \leftarrow \epsilon, L_P \leftarrow \epsilon, o \leftarrow \perp$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V)$
$o = (F, x, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
for all $\lambda^C(v_i) \in L_R$ **do**
  if $(\lambda^C(o) \leqslant \lambda^C(v_i))$ **then return** 0
for all $\lambda^C(o') \in L_P$ **do**
  if $(\lambda^C(o') = \lambda^C(o))$ **then return** 0
$(S, V) \xleftarrow{\$} \mathcal{U}_{\text{ID}} \times \mathcal{U}_{\text{ID}}$ *s.t.* $\lambda^C(S) \geqslant \lambda^C(o)$ and $\lambda^V(V) \geqslant \lambda^V(o)$
$SK_S \xleftarrow{\$} \mathsf{Register}(S, \lambda^C(S), \text{MK}, \text{PP})$
$SK_V \xleftarrow{\$} \mathsf{Register}(V, \lambda^V(V), \text{MK}, \text{PP})$
$PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
$EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \text{MK}, \text{PP})$
$(\sigma_o, VK_o, RK_o) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(o, \lambda(o), PK_F, \text{PP})$
$\theta_o \xleftarrow{\$} \mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), \text{PP})$
$(y, \tau_{\theta_o}) \leftarrow \mathsf{Verify}(\theta_o, SK_V, VK_o, \lambda^V(V), \lambda^V(o), \text{PP})$
if $((y, \tau_{\theta_o}) \neq (\perp, (\mathsf{reject}, \cdot)))$ **then return** 1
else return 0

Oracle 4.1 $\mathcal{O}^{\mathsf{Register}}(ID, \lambda(ID), \text{MK}, \text{PP})$

if $(o \neq \perp)$ **and** $(\lambda(ID) \geqslant \lambda^C(o))$ **then**
  return $\perp$
$L_R \leftarrow L_R \cup \lambda(ID)$
return $\mathsf{Register}(ID, \lambda(ID), \text{MK}, \text{PP})$

Oracle 4.2 $\mathcal{O}^{\mathsf{ProbGen}}(x', SK_C, PK_{F'}, \lambda(C), \lambda(o'), \text{PP})$

if $(\lambda^C(o') = \lambda^C(o))$ **then return** $\perp$
$L_P \leftarrow L_P \cup \lambda(o')$
return $\mathsf{ProbGen}(x', SK_C, PK_{F'}, \lambda(C), \lambda(o'), \text{PP})$

4.4.1 Authorised Outsourcing

The notion of authorised outsourcing formalises that a delegator may not outsource any computation for which he is not authorised — that is, any computation o where $\lambda^C(o) \not\leqslant \lambda^C(C)$. Clearly, within our framework we cannot make any guarantees about what an entity may do externally. For instance, we cannot enforce that a client does not circumvent the access control policies using an external server not subject to these controls; however we do enforce that to use the provided functionality (i.e. to contract an available server registered in this system) they must prove authorisation. This assumption seems reasonable, taking into consideration organisational boundaries — external control should perhaps be enforced by limiting external communication channels (e.g. using a strict firewall). Similarly, we cannot enforce that an entity does not share key content, but we do ensure that such collusion does not enable access that either entity alone could not access. Due to the revocation functionality introduced in Chapter 3, it may be undesirable for a server to share key material as he must trust the additional server not to cheat.

The authorised outsourcing game is presented in Game 4.1 and uses Oracles 4.1 and 4.2. First the challenger initialises an empty list L_R of security labels that have been registered, an empty list L_P of computations that have been given as input to ProbGen, and a challenge computation o which is initially set to a distinguished symbol $\perp$. It runs the Setup procedure for the scheme and gives the adversary the public parameters and oracle access as specified in Section 3.4.

Eventually, the adversary outputs its choice of challenge outsourced computation o to attack, which specifies F and x and any auxiliary information that should be considered when computing its security label. On line 4, we ensure that no security label that $\mathcal{A}$ has queried to the Register oracle may allow the derivation of a key for $\lambda^C(o)$ as this would be a trivial win, and we return 0 since the adversary has not found a valid attack target. We also require that any ID that $\mathcal{A}$ queries to its oracles must previously have been queried to the Register oracle i.e. SK_{ID} will be well-defined for any ID queried to an oracle since ID will already have been queried to Register (which is in keeping with realistic operation). On line 6, we check that the ProbGen oracle has not been queried on the chosen challenge computation o as the oracle output could be used to trivially win the game.

The challenger then selects two entities S and V at random such that S is a server authorised to compute the challenge computation, and V is a verifier authorised to verify the challenge computation. These will be used by the challenger to simulate processing the challenge computation. It registers both entities, initialises the challenge function F and certifies S to compute F. The adversary is then challenged, given all information that a real attacker may learn and oracle access, to output an encoded input that the Compute and Verify algorithms will accept — that is, an unauthorised adversary must produce an input that is accepted and computed on by honest entities.

In Oracle 4.1, the challenger checks that the security label for the queried identity is not an ancestor of the challenge computation label and returns $\perp$ otherwise; if such a key was issued then the adversary would be able to use it to legitimately act on the challenge computation and trivially win. If $\perp$ was not returned, the security label is added to the list L_R and the challenger returns the output of running Register. Similarly, in Oracle 4.2, the challenger returns $\perp$ if queried for a computation with the challenge computation label $\lambda^C(o)$ as the resulting values would form a winning adversarial output. Otherwise, the queried computation is added to the list L_P and the results of running ProbGen are

returned.

Definition 4.2. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* AuthO *game for an PVC-AC construction, $\mathcal{PVC}_{AC}$, for a family of functions $\mathcal{F}$ and computational and verification access control posets $\mathcal{P}_C$ and $\mathcal{P}_V$ respectively, is defined as:*

$$Adv_{\mathcal{A}}^{\text{AuthO}}(\mathcal{PVC}_{AC}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{AuthO}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]\right].$$

An PVC-AC scheme, $\mathcal{PVC}_{AC}$, is secure with respect to authorised outsourcing *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{AuthO}}(\mathcal{PVC}_{AC}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) \leqslant \text{negl}(\ell).$$

4.4.2 Authorised Computation

Authorised computation, given in Game 4.2 and Oracles 4.3 and 4.4, proceeds in a similar way to the authorised outsourcing game and captures the notion that a computational result should only be considered valid if generated by an authorised party.

Game 4.2 $\mathbf{Exp}_{\mathcal{A}}^{\text{AuthC}}\left[\mathcal{PVC}_{\text{AC}}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]$

$L \leftarrow \epsilon, o \leftarrow \perp$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{P}_C, \mathcal{P}_V)$
$o = (F, x, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
for all $\lambda^C(v_i) \in L$ **do**
  if $(\lambda^C(o) \leqslant \lambda^C(v_i))$ **then return** 0
$(C, V) \xleftarrow{\$} \mathcal{U}_{\text{ID}} \times \mathcal{U}_{\text{ID}}$ *s.t.* $\lambda^C(C) \geqslant \lambda^C(o)$ and $\lambda^V(V) \geqslant \lambda^V(o)$
$SK_C \xleftarrow{\$} \mathsf{Register}(C, \lambda^C(C), \text{MK}, \text{PP})$
$SK_V \xleftarrow{\$} \mathsf{Register}(V, \lambda^V(V), \text{MK}, \text{PP})$
$PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
$(\sigma_o, VK_o, RK_o) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \text{PP})$
$\theta_o \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_o, VK_o, o, \lambda^C(o), PK_F, RK_o, \text{PP})$
$(y, \tau_{\theta_o}) \leftarrow \mathsf{Verify}(\theta_o, SK_V, VK_o, \lambda^V(V), \lambda^V(o), RK_o, \text{PP})$
if $(((y, \tau_{\theta_o}) \neq (\perp, (\mathsf{reject}, \cdot))))$ **then return** 1
else return 0

The game begins with the challenger setting up the system and providing oracle access for the adversary. $\mathcal{A}$ chooses a target computation o that it is not authorised to compute by *any* of the keys it holds. The challenger then simulates two entities: a delegator and a verifier that are authorised for this computation, and creates an encoded input by running ProbGen on the adversary's target computation. This is given to $\mathcal{A}$ who must produce an

Oracle 4.3 $\mathcal{O}^{\mathsf{Register}}(ID, \lambda(ID), \mathrm{MK}, \mathrm{PP})$

1: **if** $(o \neq \perp)$ **and** $(\lambda(ID) \geqslant \lambda^C(o))$ **then**
2: **return** $\perp$
3: $L_R \leftarrow L_R \cup \lambda(ID)$
4: **return** $\mathsf{Register}(ID, \lambda(ID), \mathrm{MK}, \mathrm{PP})$

Oracle 4.4 $\mathcal{O}^{\mathsf{Compute}}(\sigma_{o'}, EK_{F',S}, SK_S, \lambda^C(S), \lambda^C(o'), \mathrm{PP})$

1: **if** $(o' = o)$ **then return** $\perp$
2: **return** $\mathsf{Compute}(\sigma_{o'}, EK_{F',S}, SK_S, \lambda^C(S), \lambda^C(o'), \mathrm{PP})$

encoded output that is accepted by the verifier. Note that although the adversary chooses the input to the computation, and therefore can certainly compute $F(x)$, it still should not be able to convince the verifier to accept (since the Verify algorithm should enforce the computational access control policy and reject responses from unauthorised servers).

The Register oracle is identical to that for authorised outsourcing and prevents the adversary learning a secret key for the challenge label (or any ancestors that enable derivation of the challenge label). The Compute oracle, in Oracle 4.4, returns $\perp$ if queried for the challenge computation as this would trivially verify correctly and win the game.

Definition 4.3. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* AuthC *game for an PVC-AC construction, $\mathcal{PVC}_{AC}$, for a family of functions $\mathcal{F}$ and computational and verification access control posets $\mathcal{P}_C$ and $\mathcal{P}_V$ respectively, is defined as:*

$$Adv^{\text{AuthC}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\text{AuthC}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]\right].$$

An PVC-AC scheme, $\mathcal{PVC}_{AC}$, is secure with respect to authorised computation *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv^{\text{AuthC}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) \leqslant \mathrm{negl}(\ell).$$

4.4.3 Authorised Verification

In Game 4.3 and Oracles 4.5 and 4.6, we capture the notion that an unauthorised verifier should not be able to learn the output of a computation. This is formulated in an indistinguishability game, where the adversary, on line 3, chooses two computations o_0 and o_1 to give to the challenger. To avoid a trivial win, we require that neither of the associated labels in the verification poset are less than or equal to a label queried to the Register ora-

Game 4.3 $\mathbf{Exp}_{\mathcal{A}}^{\text{AUTHV}}\left[\mathcal{PVC}_{\text{AC}}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]$

1: $L \leftarrow \epsilon, o_0 \leftarrow \perp, o_1 \leftarrow \perp$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{P}_C, \mathcal{P}_V)$
3: $(o_0, o_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}, \mathsf{Retrieve}}(\text{PP})$
4: **for** all $\lambda^V(v_i) \in L$ **do**
5: **if** $((\lambda^V(o_0) \leqslant \lambda^V(v_i))$ **or** $(\lambda^V(o_1) \leqslant \lambda^V(v_i)))$ **then return** 0
6: $b \xleftarrow{\$} \{0,1\}$
7: $(C, S) \xleftarrow{\$} \mathcal{U}_{\text{ID}} \times \mathcal{U}_{\text{ID}}$ *s.t.* $\lambda^C(C) \geqslant \lambda^C(o_b)$ and $\lambda^C(S) \geqslant \lambda^C(o_b)$
8: $SK_C \xleftarrow{\$} \mathsf{Register}(C, \lambda^C(C), \text{MK}, \text{PP})$
9: $SK_S \xleftarrow{\$} \mathsf{Register}(V, \lambda^C(S), \text{MK}, \text{PP})$
10: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
11: $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \text{MK}, \text{PP})$
12: $(\sigma_{o_b}, VK_{o_b}, RK_{o_b}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o_b), \text{PP})$
13: $\theta_{o_b} \xleftarrow{\$} \mathsf{Compute}(\sigma_{o_b}, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o_b), \text{PP})$
14: $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}, \mathsf{Retrieve}}(\theta_{o_b}, VK_{o_b}, RK_{o_b}, \lambda(o_0), \lambda(o_1), PK_F, \text{PP})$
15: **return** $b' = b$

Oracle 4.5 $\mathcal{O}^{\mathsf{Register}}(ID, \lambda(ID), \text{MK}, \text{PP})$

1: **if** $((o_b \neq \perp)$ **and** $((\lambda(ID) \geqslant \lambda^V(o_0))$ **or** $(\lambda(ID) \geqslant \lambda^V(o_1)))$ **then**
2: **return** $\perp$
3: $L \leftarrow L \cup \lambda(ID)$
4: **return** $SK_{ID} \xleftarrow{\$} \mathsf{Register}(ID, \lambda(ID), \text{MK}, \text{PP})$

Oracle 4.6 $\mathcal{O}^{\mathsf{Retrieve}}(SK_V, RT_{o'}, \tau_{\theta_{o'}}, VK_{o'}, RK_{o'}, \lambda^V(V), \lambda^V(o'), \text{PP})$

1: **if** $((o' = o_0)$ **or** $(o' = o_1))$ **then return** $\perp$
2: **return** $\mathsf{Retrieve}(SK_V, RT_{o'}, \tau_{\theta_{o'}}, VK_{o'}, RK_{o'}, \lambda^V(V), \lambda^V(o'), \text{PP})$

cle (otherwise, the adversary is in fact authorised to verify the chosen computation). The challenger chooses *one* of these at random and simulates outsourcing and computing this computation. To do so, on line 7, the challenger chooses two random entities, one to act as a delegator and one to act as a server. These identities will be registered in the system and used to simulate running ProbGen and Compute as in a real system. The adversary is provided with the encoded output from this computation, along with the verification keys and other public information, and must guess which computation was chosen. As the adversary chose both computations, it can certainly work out the results; however, no information about the result should leak from the encoded output unless the verifier is authorised, and hence the adversary should be unable to tell which result the output corresponds to.

As with the other games in this chapter, the adversary is given oracle access to the functions $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, \text{MK}, \text{PP})$, $\mathsf{ProbGen}(\cdot, SK_{(\cdot)}, \cdot, \cdot, \cdot, \text{PP})$, $\mathsf{Compute}(\cdot, \cdot, SK_{(\cdot)}, \cdot, \cdot, PP)$ and $\mathsf{Revoke}(\cdot, \cdot, \text{MK}, \text{PP})$, which is denoted by $\mathcal{O}$. As previously, the Register oracle, given in Oracle 4.5, returns $\perp$ if queried for a label that would

authorise the adversary for the challenge computation. Unlike prior Register oracles, this check is performed on the verification poset rather than the computational poset, and checks against *both* challenge computations o_0 and o_1 (otherwise, the adversary could determine which computation was chosen based on which labels are restricted in these queries). The adversary is also given access to a Retrieve oracle, specified in Oracle 4.6, so that it can observe outputs for the challenge label. However, to avoid giving the adversary the information required to form a trivial win or to reveal which computation was chosen, the challenger returns $\perp$ if queried for either challenge computation o_0 or o_1.

Definition 4.4. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* AuthV *game for an PVC-AC construction, $\mathcal{PVC}_{AC}$, for a family of functions $\mathcal{F}$ and computational and verification access control posets $\mathcal{P}_C$ and $\mathcal{P}_V$ respectively, is defined as:*

$$Adv_{\mathcal{A}}^{\text{AuthV}}(\mathcal{PVC}_{AC}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{AuthV}}\left[\mathcal{RPVC}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]\right] - \tfrac{1}{2}.$$

An PVC-AC scheme, $\mathcal{PVC}_{AC}$, is secure with respect to authorised verification *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{AuthV}}(\mathcal{PVC}_{AC}, 1^{\ell}, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) \leqslant \text{negl}(\ell).$$

4.4.4 Weak Input Privacy

The notion of weak input privacy, captured in Game 4.4 and Oracles 4.7 and 4.8, is not as strong as the input privacy often considered in PVC settings where computational servers learn nothing about the data they are working on. Instead, we are interested in ensuring that servers (or other entities) that are *not authorised* to access (compute on) the input data may not learn x. If full input privacy is required then the underlying PVC scheme, such as the KP-ABE based one used as a black box in the construction, could be replaced by one built from a predicate encryption scheme for the same class of functions.

The game begins in a similar way to the authorised verification game with the adversary selecting two computations that he is not authorised to compute by *any* key that he has queried to the Register oracle (Oracle 4.7). The Compute oracle in Oracle 4.8 can be queried for any input except the challenge inputs. The challenger then simulates the outsourcing of one of these computations using a simulated delegator C, and gives the

Game 4.4 $\mathbf{Exp}^{\text{wIP}}_{\mathcal{A}}\left[\mathcal{PVC}_{\text{AC}}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]$

$L \leftarrow \epsilon, o_0 \leftarrow \perp, o_1 \leftarrow \perp$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{P}_C, \mathcal{P}_V)$
$(o_0, o_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
for all $\lambda^C(v_i) \in L$ **do**
  if $((\lambda^C(o_0) \leqslant \lambda^C(v_i))$ **or** $(\lambda^C(o_1) \leqslant \lambda^C(v_i)))$ **then return** 0
$b \xleftarrow{\$} \{0,1\}$
$C \xleftarrow{\$} \mathcal{U}_{\text{ID}}$ *s.t.* $\lambda^C(C) \geqslant \lambda^C(o_b)$
$SK_C \xleftarrow{\$} \mathsf{Register}(C, \lambda^C(C), \text{MK}, \text{PP})$
$PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
$(\sigma_{o_b}, VK_{o_b}, RK_{o_b}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o_b), \text{PP})$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{o_b}, VK_{o_b}, \lambda(o_0), \lambda(o_1), PK_F, \text{PP})$
return $(b' = b)$

Oracle 4.7 $\mathcal{O}^{\mathsf{Register}}(ID, \lambda(ID), \text{MK}, \text{PP})$

if $((o_0, o_1 \neq \perp)$ **and** $((\lambda(ID) \geqslant \lambda^C(o_0))$ **or** $(\lambda(ID) \geqslant \lambda^C(o_1))))$ **then**
  return $\perp$
$L_R \leftarrow L_R \cup \lambda(ID)$
return $\mathsf{Register}(ID, \lambda(ID), \text{MK}, \text{PP})$

Oracle 4.8 $\mathcal{O}^{\mathsf{Compute}}(\sigma_{o'}, EK_{F',S}, SK_S, \lambda^C(S), \lambda^C(o'), \text{PP})$

if $(o' = o)$ **then return** $\perp$
return $\mathsf{Compute}(\sigma_{o'}, EK_{F',S}, SK_S, \lambda^C(S), \lambda^C(o'), \text{PP})$

adversary the resulting encoded input. The adversary must guess which computation the input corresponds to; i.e. which input data is encoded in the input.

Definition 4.5. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* wIP *game for an PVC-AC construction, $\mathcal{PVC}_{AC}$, for a family of functions $\mathcal{F}$ and computational and verification access control posets $\mathcal{P}_C$ and $\mathcal{P}_V$ respectively, is defined as:*

$$Adv^{\text{wIP}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\text{wIP}}_{\mathcal{A}}\left[\mathcal{RPVC}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V\right]\right] - \tfrac{1}{2}.$$

An PVC-AC scheme, $\mathcal{PVC}_{AC}$, is secure with respect to weak input privacy *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv^{\text{wIP}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V) \leqslant \text{negl}(\ell).$$

4.5 Construction

We now provide an instantiation of PVC-AC that is provably secure with respect to the security definitions in Section 3.4. The approach we take makes generic, black-box use of any RPVC scheme, such as that presented in Chapter 3, and introduces the use of a

symmetric key assignment scheme (KAS) (see Section 2.4) to restrict the behaviour of entities.

4.5.1 Informal Overview

The graph-based access control policies discussed in Section 4.2 assign labels to each entity and outsourced computation. The ordering relation and the correctness criterion ensures that entities are authorised for all appropriate computations (those that are descendants of their label in the Hasse diagram of the poset). A KAS is designed to enforce such policies, and assigns a key to each label. Each entity is provided with a key corresponding to their label, and they may derive keys for all descendants. As per the security criterion, users may not collude to derive keys for which they are not authorised.

In our setting, we restrict the computations a delegator may outsource, the computations a server may perform, and the results a verifier may learn. We use two independent KASs instantiated over the computation and verification posets, $\mathcal{P}_C$ and $\mathcal{P}_V$ respectively. Delegators and servers are issued a key according to their label in the computation poset, while delegators and verifiers are given a key from the verification KAS.

Appropriate keys from the computation KAS are used to encrypt the encoded input for a computation. To encode an input in a manner that will be accepted by a server, delegators must encrypt the encoded input using the key, $\kappa_{\lambda^C(o)}$, associated to the computation within the computation poset. To do so, the delegator must be able to derive $\kappa_{\lambda^C(o)}$ and hence must hold a key at that level or higher i.e. $\lambda(C) \geqslant \lambda(o)$ — delegators must be authorised by the KDC to outsource the computation. Similarly, only authorised servers can derive the decryption key to access the encoded input and perform the computation. By the IND-CPA security of the symmetric encryption scheme used, no information about the encoded input is learnt by an unauthorised entity.

To enforce verification policies, delegators generate a retrieval key during delegation, and encrypt this using an appropriate key from the verification KAS. Then, only verifiers that can derive this key may decrypt the ciphertext to recover the retrieval key and learn the output.

4.5.2 Instantiation

Let $\mathcal{RPVC}$ = (RPVC.Setup, RPVC.FnInit, RPVC.Register, RPVC.Certify, RPVC.ProbGen, RPVC.Compute, RPVC.BVerif, RPVC.Retrieve, RPVC.Revoke) be an RPVC scheme, as defined in Chapter 3, for a class of functions $\mathcal{F}$. Let $\mathcal{SE}$= (SE.KeyGen, SE.Encrypt, SE.Decrypt) be an authenticated symmetric encryption scheme secure in the sense of IND-CPA $\wedge$ INT-PTXT, and let $\mathcal{KAS}$ = (KAS.MakeKeys, KAS.MakeSecrets, KAS.MakePublicData, KAS.GetKey) be a key assignment scheme secure in the sense of strong-key indistinguishability[7] whose keys are compatible with $\mathcal{SE}$ (i.e. the keys generated in the KAS have the same distribution as those generated by SE.KeyGen). Finally, let $\mathcal{P}_C$ denote the poset encoding computation policies (e.g. $(L, \leqslant)$), and similarly let $\mathcal{P}_V$ denote the poset encoding verification policies (e.g. $(L, \geqslant)$)[8]. Then, for the same class of functions $\mathcal{F}$, there is a PVC-AC scheme $\mathcal{PVC}_{\mathrm{AC}}$ = ($\mathsf{PVC_{AC}}$.Setup, $\mathsf{PVC_{AC}}$.FnInit, $\mathsf{PVC_{AC}}$.Register, $\mathsf{PVC_{AC}}$.Certify, $\mathsf{PVC_{AC}}$.ProbGen, $\mathsf{PVC_{AC}}$.Compute, $\mathsf{PVC_{AC}}$.BVerif, $\mathsf{PVC_{AC}}$.Retrieve, $\mathsf{PVC_{AC}}$.Revoke) defined in Algorithms 4.1-4.9 which operates as follows.

1. $\mathsf{PVC_{AC}}$.Setup, presented in Algorithm 4.1, runs the setup algorithm for the underlying RPVC scheme. It also instantiates two KASs: one for the computational poset $\mathcal{P}_C$ and one for the verification poset $\mathcal{P}_V$. These will form the cryptographic enforcement mechanisms for computational and verification access control policies respectively. The public parameters for the PVC-AC system comprise the RPVC public parameters and the public labels for the two KASs. The master secret key for the PVC-AC system comprises the master secret of the RPVC scheme and the keys and secrets for the two KASs.

2. $\mathsf{PVC_{AC}}$.FnInit, presented in Algorithm 4.2, simply outputs the public delegation key for the RPVC scheme.

3. $\mathsf{PVC_{AC}}$.Register, presented in Algorithm 4.3, differs based on whether the queried identity ID is a delegator, server, or verifier. If ID is a server, then it must be registered in the underlying RPVC system (servers are the only entities that need

[7] It has been shown that S-KI is equivalent to KI [35]. We make use of the additional queries in the S-KI game, but due to the equivalence this is not a strengthening of the assumptions. It is interesting to note that the form of the S-KI game is useful as a proof technique besides the original motivation to reflect realistic attacks.

[8] We use $\kappa_{\mathcal{P}_C}$ to denote the set of all keys generated by the KAS for the computation poset $\mathcal{P}_C$ and, for a label $\lambda^C(\mathrm{ID}) \in \mathcal{P}_\mathrm{C}$, the associated key is $\kappa_{\lambda^C(\mathrm{ID})} \in \kappa_{\mathcal{P}_C}$.

Algorithm 4.1 $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{P}_C, \mathcal{P}_V)$

1: $(\mathrm{PP}', \mathrm{MK}') \xleftarrow{\$} \mathsf{RPVC.Setup}(1^\ell)$
2: $\kappa_{\mathcal{P}_C} \xleftarrow{\$} \mathsf{KAS.MakeKeys}(1^\ell, \mathcal{P}_C)$
3: $\omega_{\mathcal{P}_C} \xleftarrow{\$} \mathsf{KAS.MakeSecrets}(1^\ell, \mathcal{P}_C)$
4: $Pub_{\mathcal{P}_C} \xleftarrow{\$} \mathsf{KAS.MakePublicData}(1^\ell, \mathcal{P}_C)$
5: $\kappa_{\mathcal{P}_V} \xleftarrow{\$} \mathsf{KAS.MakeKeys}(1^\ell, \mathcal{P}_V)$
6: $\omega_{\mathcal{P}_V} \xleftarrow{\$} \mathsf{KAS.MakeSecrets}(1^\ell, \mathcal{P}_V)$
7: $Pub_{\mathcal{P}_V} \xleftarrow{\$} \mathsf{KAS.MakePublicData}(1^\ell, \mathcal{P}_V)$
8: $\mathrm{PP} \leftarrow (\mathrm{PP}', Pub_{\mathcal{P}_C}, Pub_{\mathcal{P}_V})$
9: $\mathrm{MK} \leftarrow (\mathrm{MK}', \kappa_{\mathcal{P}_C}, \omega_{\mathcal{P}_C}, \kappa_{\mathcal{P}_V}, \omega_{\mathcal{P}_V})$

Algorithm 4.2 $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$

1: $PK_F \xleftarrow{\$} \mathsf{RPVC.FnInit}(F, \mathrm{MK}', \mathrm{PP}')$

registering in RPVC). The secret key for the server comprises the signing key produced by the RPVC Register algorithm, as well as the KAS key and secret for the computational label of the server (such that it can derive keys for all computations it is authorised to compute).

If ID is a delegator then it must be able to encrypt both encoded inputs (protected by computational policies over $\mathcal{P}_C$) and retrieval keys (protected by verification policies over $\mathcal{P}_V$). Therefore, it is issued the KAS keys and secrets for the security labels assigned to ID in both KASs.

Finally, if ID is a verifier, it must be able to decrypt ciphertexts using KAS keys associated to the verification poset (to recover valid retrieval keys). Therefore, it is issued the KAS key and secret for the label of ID in the verification poset.

Algorithm 4.3 $SK_{\mathrm{ID}} \xleftarrow{\$} \mathsf{Register}(\mathrm{ID}, \lambda(\mathrm{ID}), \mathrm{MK}, \mathrm{PP})$

if ID is a server **then**
    $SK'_{\mathrm{ID}} \xleftarrow{\$} \mathsf{RPVC.Register}(\mathrm{ID}, \mathrm{MK}', \mathrm{PP}')$
    $SK_{\mathrm{ID}} \leftarrow (SK'_{\mathrm{ID}}, \kappa_{\lambda^C(\mathrm{ID})}, \omega_{\lambda^C(\mathrm{ID})})$
else if ID is a delegator **then**
    $SK_{\mathrm{ID}} \leftarrow (\kappa_{\lambda^C(\mathrm{ID})}, \omega_{\lambda^C(\mathrm{ID})}, \kappa_{\lambda^V(\mathrm{ID})}, \omega_{\lambda^V(\mathrm{ID})})$
else
    $SK_{\mathrm{ID}} \leftarrow (\kappa_{\lambda^V(\mathrm{ID})}, \omega_{\lambda^V(\mathrm{ID})})$

4. $\mathsf{PVC_{AC}.Certify}$, presented in Algorithm 4.4, simply runs the Certify algorithm of the underlying RPVC scheme as the cryptographic enforcement mechanism is not required to certify servers for particular functions.

5. $\mathsf{PVC_{AC}.ProbGen}$, presented in Algorithm 4.5, first runs the ProbGen algorithm for the underlying RPVC scheme. It must then protect the resulting encoded input and retrieval key in accordance with the computational and verification access control

Algorithm 4.4 $EK_{F,S} \xleftarrow{\$} \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$

1: $EK_{F,S} \xleftarrow{\$} \mathsf{RPVC.Certify}(S, F, \mathrm{MK}', \mathrm{PP}')$

policy respectively. To do so, it derives the keys for the labels $\lambda^C(o)$ and $\lambda^V(o)$ from the secret issued to the delegator running the algorithm. If both keys are derived successfully (i.e. the delegator is authorised to outsource this computation) then these keys are used to encrypt the encoded input and retrieval key respectively using the symmetric encryption scheme.

Algorithm 4.5 $(\sigma_o, VK_o, RK_o) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \mathrm{PP})$

1: $(\sigma'_o, VK_o, RK'_o) \xleftarrow{\$} \mathsf{RPVC.ProbGen}(x, PK_F, \mathrm{PP}')$
2: $\kappa_{\lambda^C(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^C(C), \lambda^C(o), \omega_{\lambda^C(C)}, \mathrm{PP})$
3: $\kappa_{\lambda^V(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^V(C), \lambda^V(o), \omega_{\lambda^V(C)}, \mathrm{PP})$
4: **if** $(\kappa_{\lambda^C(o)} \neq \perp)$ **and** $(\kappa_{\lambda^V(o)} \neq \perp)$ **then**
5: $\quad \sigma_o \leftarrow (\lambda^C(o), \mathsf{SE.Encrypt}(\sigma'_o, \kappa_{\lambda^C(o)}))$
6: $\quad RK_o \leftarrow (\lambda^V(o), \mathsf{SE.Encrypt}(RK'_o, \kappa_{\lambda^V(o)}))$

6. $\mathsf{PVC_{AC}.Compute}$, presented in Algorithm 4.6, first attempts to derive the key $\kappa_{\lambda^C(o)}$ for the computation, using the secret issued to the server by the KDC. If derivation is unsuccessful then the algorithm terminates as the server is unauthorised for the computation. Otherwise, the encoded input is decrypted and the resulting plaintext used as input to the Compute algorithm of the underlying RPVC scheme.

Algorithm 4.6 $\theta_o \xleftarrow{\$} \mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), \mathrm{PP})$

1: Parse σ_o as $(\lambda^C(o), c)$
2: $\kappa_{\lambda^C(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^C(S), \lambda^C(o), \omega_{\lambda^C(S)}, \mathrm{PP})$
3: **if** $(\kappa_{\lambda^C(o)} = \perp)$ **then**
4: $\quad$ **return** $\theta_o \leftarrow \perp$
5: **else**
6: $\quad \sigma'_o \leftarrow \mathsf{SE.Decrypt}(c, \kappa_{\lambda^C(o)})$
7: $\quad$ **if** $(\sigma'_o = \perp)$ **then return** $\theta_o \leftarrow \perp$
8: $\quad$ **else** $\theta_o \xleftarrow{\$} \mathsf{RPVC.Compute}(\sigma'_o, EK_{F,S}, SK_S, PP')$

7. $\mathsf{PVC_{AC}.BVerif}$, presented in Algorithm 4.7, simply runs the corresponding algorithm of the underlying RPVC scheme, as any entity is authorised to perform this step in our model.

8. $\mathsf{PVC_{AC}.Retrieve}$, presented in Algorithm 4.8, first attempts to derive the key associated to the verification label $\lambda^V(o)$ of the computation. If unsuccessful, then the verifier is unauthorised and the algorithm terminates. Otherwise, the derived key is used with the symmetric decryption algorithm to recover the retrieval key which

Algorithm 4.7 $(RT_o, \tau_{\theta_o}) \leftarrow \mathsf{BVerif}(\theta_o, VK_o, \mathrm{PP})$:

1: $(RT_o, \tau_{\theta_o}) \leftarrow \mathsf{RPVC.BVerif}(\theta_o, VK_o, \mathrm{PP}')$

can then be used in the RPVC.Retrieve algorithm to produce the output of the computation.

Algorithm 4.8 $y \leftarrow \mathsf{Retrieve}(SK_V, RT_o, \tau_{\theta_o}, VK_o, RK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})$:

1: Parse RK_o as $(\lambda^V(o), e)$
2: $\kappa_{\lambda^V(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^V(V), \lambda^V(o), \omega_{\lambda^V(V)}, \mathrm{PP})$
3: **if** $\kappa_{\lambda^V(o)} = \perp$ **then**
4: **return** $y \leftarrow \perp$
5: **else**
6: $RK'_o \leftarrow \mathsf{SE.Decrypt}(e, \kappa_{\lambda^V(o)})$
7: $y \leftarrow \mathsf{RPVC.Retrieve}(RT_o, \tau_{\theta_o}, VK_o, RK'_o, \mathrm{PP}')$

9. $\mathsf{PVC_{AC}.Revoke}$, presented in Algorithm 4.9, simply runs the RPVC.Revoke algorithm. As this is run by the KDC, access control restrictions do not need to be applied.

Algorithm 4.9 $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_o}, F, \mathrm{MK}, \mathrm{PP})$

1: **return** $\mathsf{RPVC.Revoke}(\tau_{\theta_o}, F, \mathrm{MK}, \mathrm{PP}')$

It is straightforward to see that correctness of this construction follows from the correctness of the RPVC scheme.

Theorem 4.1. *Given any strong-key-indistinguishability secure KAS, any RPVC scheme secure in the sense of public verifiability and revocation for a class of functions $\mathcal{F}$, and an authenticated symmetric encryption scheme secure in the sense of* IND-CPA $\wedge$ INT-PTXT*, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 4.1-4.9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of public verifiability, revocation, authorised outsourcing, authorised computation, weak input privacy, and authorised verification for the same class of functions $\mathcal{F}$.*

4.6 Proofs of Security

Informally, the security proofs follow from the security of the underlying RPVC scheme and the S-KI, INT-PTXT and IND-CPA security properties. The proofs for public verifiability and revocation are similar to the proofs given in Chapter 3 up to some syntactical changes, and so we do not replicate these here.

Lemma 4.1. *Given a secure RPVC scheme, a symmetric authenticated encryption scheme secure in the sense of* IND-CPA ∧ INT-PTXT *and a KAS secure with respect to strong-key indistinguishability, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 4.1–4.9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of authorised outsourcing (Game 4.1).*

Proof. Let $\mathcal{A}_{SE}$ be an adversary playing the IND-CPA game with a challenger $\mathcal{C}$ against $\mathcal{SE}$. We first transition to a slightly modified version of the authorised outsourcing game, showing a negligible difference between the two. We can then use an adversary against this modified game to break the IND-CPA security of the symmetric encryption scheme.

- **Game 0.** This is the authorised outsourcing game as defined in Section 4.4.1.
- **Game 1.** This is identical to **Game 0**, except that we replace the key $\kappa_{\lambda^C(o)}$ for the challenge computation o with a key $\kappa^\star$ drawn uniformly at random from the keyspace.

Game 0 to Game 1 This game hop relies on the strong key indistinguishability (S-KI) of the KAS. Suppose, for a contradiction, that an adversary $\mathcal{A}_{VC}$ exists that can distinguish **Game 0** from **Game 1** with non-negligible advantage δ. Then we show that there exists an adversary $\mathcal{A}_{SE}$ that, using $\mathcal{A}_{VC}$ as a sub-routine, breaks the S-KI of the KAS also with advantage δ. $\mathcal{A}_{VC}$ will play either **Game 0** or **Game 1** with $\mathcal{A}_{SE}$ acting as the challenger, and must guess correctly which game he is playing. $\mathcal{A}_{SE}$ in turn will play the S-KI game with a challenger $\mathcal{C}$. This reduction is important to ensure that no additional information about the encryption key is leaked through the construction.

1. $\mathcal{C}$ begins by selecting a bit $b \xleftarrow{\$} \{0,1\}$ which will determine whether the challenge key is real or random, and hence whether $\mathcal{A}_{VC}$ plays **Game 0** or **Game 1**. $\mathcal{C}$ continues by instantiating the KAS in lines 2 to 5 of Game 2.4 and giving the public information to $\mathcal{A}_{SE}$.
2. $\mathcal{A}_{SE}$ runs lines 1 and 2 of the authorised outsourcing game to initialise L_R, L_P, o and the PVC-AC system, and sends $\mathcal{A}_{VC}$ the generated public parameters. During the Setup algorithm in line 2, $\mathcal{A}_{SE}$ will not run lines 2 to 4 of Algorithm 4.1. Instead, it will set $Pub_{\mathcal{P}_C}$ to be the challenge public information from the S-KI challenger $\mathcal{C}$ and will make use of oracle queries to $\mathcal{C}$ for the remaining parameters.

3. $\mathcal{A}_{VC}$ is then given oracle access. Queries to the FnInit, Certify and Revoke functions can be answered simply by running the corresponding algorithm. Queries to the remaining algorithms may need to make use of KAS derived keys. $\mathcal{A}_{SE}$ can use its knowledge of the verification poset and the corresponding KAS (which it owns), where appropriate, but it does not know anything other than the public information for the computational poset, because this is the challenge poset for the S-KI game. Thus, $\mathcal{A}_{SE}$ will issue Corrupt oracle queries to retrieve the necessary keys and secret information for the queried identity label. Since $\mathcal{A}_{VC}$ has not yet chosen a challenge computation o, and $\mathcal{A}_{SE}$ has not yet chosen a challenge node $v^\star$, the Corrupt oracle will always return a valid key and secret pair, which $\mathcal{A}_{SE}$ can use to form a full, valid response for $\mathcal{A}_{VC}$.

4. Eventually, $\mathcal{A}_{VC}$ will output a challenge computation o. $\mathcal{A}_{SE}$ will output 0 and end the game if this choice is invalid — that is, if $\mathcal{A}_{VC}$ is authorised to outsource o by any key $\lambda^C(v_i)$ queried to the Register oracle. $\mathcal{A}_{SE}$ will send $\lambda^C(o)$ to $\mathcal{C}$ as the challenge node $v^\star$ for the S-KI game. Note that this is a valid challenge in the S-KI game because the only queries made to the Corrupt oracle were as a result of queries to the Register, ProbGen and Compute queries. Recall that we require any identity ID queried to an oracle to have previously been queried to Register such that SK_{ID} is well-defined (as it would be in a realistic system evolution). Therefore, any KAS key for a label $\lambda^C(ID)$ required in a ProbGen or Compute oracle query, would previously have been queried to Register. Now, in the authorised outsourcing game, $\mathcal{A}_{VC}$ may not choose a challenge computation o for which he is authorised — that is, any o such that $\lambda^C(o) \leqslant \lambda^C(v_i)$ for any $\lambda^C(v_i)$ queried to the Register oracle. This corresponds precisely to the condition on the choice of $v^\star$ in the S-KI game.

   $\mathcal{C}$ will return a key $\kappa^\star$ which corresponds either to the real key $\kappa_{\lambda^C(o)}$ or a random key from the keyspace, chosen according to the bit b chosen at the beginning of the game.

5. $\mathcal{A}_{SE}$ must now simulate a server and a verifier. It does this by first registering two such entities, S and V respectively, that are authorised to compute and verify o respectively. It chooses a random S such that $\lambda^C(S) = \lambda^C(o)$ and hence $\kappa_{\lambda^C(S)} = \kappa^\star$. Note that this is the only part of SK_S that is required, and in particular $\omega_{\lambda^C(S)}$ is not required (as this is only needed for deriving keys, and S will only need to use $\kappa^\star$). To register S, $\mathcal{A}_{SE}$ runs Register and sets $SK_S = (SK'_S, \kappa^\star, \perp)$. Furthermore, to register V, $\mathcal{A}_{SE}$ sets $SK_V = (\kappa_{\lambda^V(V)}, \omega_{\lambda^V(V)})$.

6. $\mathcal{A}_{SE}$ then runs lines 11 and 12 as written in the authorised outsourcing game (Game 4.1) to initialise F and certify the selected server S for F. $\mathcal{A}_{VC}$ is given PK_F and oracle access. As before, for queries to FnInit, Certify and Revoke, $\mathcal{A}_{SE}$ simply runs the corresponding algorithm.

   For queries to Register for an identity ID, $\mathcal{A}_{SE}$ will return $\perp$ if $\lambda(ID) \geqslant \lambda^C(o)$ as specified in Oracle 4.1 (since $\mathcal{A}_{VC}$ should not be authorised to outsource o). Otherwise, $\mathcal{A}_{SE}$ will issue a Corrupt oracle query for $(\lambda(ID), \lambda^C(o))$. Now, since $\mathcal{A}_{SE}$ did not output $\perp$ already, $v^\star \not\leqslant v_i$ in the S-KI game (or equivalently, $\lambda^C(o) \not\leqslant \lambda(ID)$) and hence $\mathcal{C}$ will always return a valid key and secret pair which $\mathcal{A}_{SE}$ can use to simulate running Register.

   For queries to ProbGen, observe that only the KAS keys and not the KAS secrets are required to form a correct output, and that Oracle 4.2 returns $\perp$ if queried for $\lambda^C(o)$ which is the challenge label in the S-KI game. The Corrupt oracle (Oracle 2.3) returns (at least) a key for all queries except the challenge label. Therefore, for all queries that $\mathcal{A}_{VC}$ does not return $\perp$ for, $\mathcal{C}$ will be able to provide a key that can be used to simulate running ProbGen.

   Finally, for queries to Compute, again observe that only the KAS key for the queried label is required to form a valid output, and not the KAS secret. The Corrupt oracle will issue a valid key for all labels but the challenge label $\lambda^C(o)$, and so in these cases, $\mathcal{A}_{SE}$ can successfully simulate the Compute algorithm. If, on the other hand, a query is made to the Compute oracle for the challenge computation, then the adversary has either submitted a malformed encoded input (and $\perp$ should be returned), or the adversary has submitted a correctly formed encoded input, and hence has already won the game and needn't make this query.

7. Eventually, $\mathcal{A}_{VC}$ outputs its encoded output, verification key and retrieval key for the challenge computation. $\mathcal{A}_{SE}$ runs the Compute algorithm as written, as it knows $\kappa_{\lambda^C(o)}$ from $\kappa_{\lambda^C(C)}$, and the Verify algorithm as written since it owns the verification KAS. $\mathcal{A}_{SE}$ can therefore return the result of the game to $\mathcal{A}_{VC}$ as expected.

Now, $\mathcal{A}_{VC}$ has been provided all information that an adversary against the authorised outsourcing game would be given and will guess which game he is playing with advantage δ. Notice that the distribution of the game generated by $\mathcal{A}_{SE}$ is precisely that of **Game 0** if $b = 0$ (i.e. the real KAS key is used), and is precisely that of **Game 1** otherwise

(i.e. a random key was chosen). Thus, $\mathcal{A}_{SE}$ can simply forward $\mathcal{A}_{VC}$'s guess to $\mathcal{C}$ as its guess in the S-KI game. We conclude that if $\mathcal{A}_{VC}$ guesses correctly with non-negligible advantage δ, then $\mathcal{A}_{SE}$ can break the strong-key indistinguishability of the KAS also with non-negligible advantage δ. However, since we assumed the KAS was S-KI secure, such an adversary cannot exist and hence **Game 0** is indistinguishable from **Game 1** except with at most a negligible advantage $\epsilon \leqslant 1 - \delta$.

Reduction to INT-PTXT We have shown that, from the adversary's point of view, **Game 1** is almost (with negligible distinguishing advantage) identical to **Game 0**. Thus, we may run the adversary against **Game 1** instead with at most an ϵ loss in the tightness of the reduction. In essence, we have now removed any information leakage from the KAS.

We now consider the security of **Game 1**. To achieve a contradiction, suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ in **Game 1**. We show that, using this adversary $\mathcal{A}_{VC}$ as a subroutine, we can construct an adversary $\mathcal{A}_{SE}$ that breaks the INT-PTXT security of the authenticated symmetric encryption scheme $\mathcal{SE}$. Let $\mathcal{C}$ be the INT-PTXT challenger for $\mathcal{A}_{SE}$ who in turn acts as the challenger in **Game 1** for $\mathcal{A}_{VC}$.

1. $\mathcal{C}$ begins by initialising the list L and running $\mathsf{SE.KeyGen}(1^\ell)$ to generate a key $\kappa^\star$, as specified in Game 2.7. It sends the security parameter 1^ℓ to $\mathcal{A}_{SE}$.

2. $\mathcal{A}_{SE}$ must now initialise **Game 1** for $\mathcal{A}_{VC}$. Informally, it will set the KAS key for the label $\lambda^C(o)$ to be the random key $\kappa^\star$ chosen by $\mathcal{C}$. However, the challenge label is unknown until $\mathcal{A}_{VC}$ chooses it, whilst the public parameters and oracle access must be provided before this choice. Thus, we require $\mathcal{A}_{SE}$ to guess the challenge label during Setup so that the correct key can be implicitly set to be the INT-PTXT challenge key (and all encryptions under that key can be formed using oracle access to $\mathcal{C}$). If the number of labels in the poset is N, where N is polynomial in the security parameter (as the scheme must be efficiently instantiable), then $\mathcal{A}_{SE}$ may guess $\lambda^C(o)$ with probability at least $\frac{1}{N}$. Assuming that the guess is correct, we proceed as follows.

3. $\mathcal{A}_{SE}$ runs $\mathsf{PVC_{AC}.Setup}$ as given in Algorithm 4.1 with the modification that the key for the guessed label in the computation poset, $\kappa_{\lambda^C(o)}$ is implicitly set to be the key generated by $\mathcal{C}$ in the IND-CPA game and the KAS is constructed in such a

way to be consistent with this choice. Of course, $\mathcal{A}_{SE}$ does not hold $\kappa_{\lambda^C(o)}$, but will ensure that any operations using it will be performed using its oracles to $\mathcal{C}$. Since the challenge key, and hence any corresponding secret derivation information, is unknown, it is not trivial to construct a KAS incorporating this key. However, notice that the authorised outsourcing game (and by extension **Game 1**) does not permit the adversary to query any label that is an ancestor of the challenge label in the computation poset. Thus, KAS keys for the set of ancestors will not be needed, and a KAS can simply be instantiated over the remaining nodes (and the public information for the ancestor set simulated; as the keys cannot be derived, the public information need not be functionally correct). Remaining keys (for the set of ancestors of the challenge label) can simply be generated using the security parameter and the symmetric KeyGen algorithm. From the adversarial point of view, this will be indistinguishable from the real games.

4. $\mathcal{A}_{VC}$ is given the generated public parameters and oracle access which $\mathcal{A}_{SE}$ may respond to as follows:

   - FnInit, Certify, and Revoke queries can be handled by simply calling the relevant algorithm as these have no dependence on the KAS.
   - If a Register query is made for a label $\lambda(ID) \geqslant \lambda^C(o)$ as guessed by $\mathcal{A}_{SE}$ then $\mathcal{A}_{SE}$ aborts the game since $\mathcal{A}_{VC}$ would not then be able to choose $\lambda^C(o)$ as its challenge computation; hence $\mathcal{A}_{SE}$'s guess was incorrect. Otherwise, $\mathcal{A}_{SE}$ holds the relevant KAS keys and may respond by running as specified in Oracle Query 4.1.
   - By the restriction specified on line 6 of the authorised outsourcing game, $\mathcal{A}_{VC}$ may not choose a challenge computation whose label has been queried to the ProbGen oracle. Hence, if such a query is made at this point, then $\lambda^C(o)$ could not be chosen as the challenge computation and $\mathcal{A}_{SE}$'s guess would be incorrect; therefore, if this occurs, the game is aborted. For any other choice of computation queried to the ProbGen oracle, $\mathcal{A}_{SE}$ holds the KAS key (or generated symmetric key) and may run Algorithm 3.5 as written.
   - Observe that if a query is made to the Compute oracle for the challenge computation, then the adversary has either submitted a malformed encoded input, and $\perp$ should be returned, or the adversary has submitted a correctly formed encoded input, and hence has already won the game. For all other query inputs,

$\mathcal{A}_{SE}$ can use one of the keys it holds to respond to the challenge by running Algorithm 3.6 as written.

5. $\mathcal{A}_{VC}$ eventually outputs a choice of challenge computation, and if this is not the computation chosen by $\mathcal{A}_{SE}$ at the beginning of the game, the game is aborted. Otherwise, the choice of computation label is valid since the game has not already been aborted during the oracle queries.

6. $\mathcal{A}_{SE}$ should now register two entities: a computational server S and a verifier V. However, these will not be required in the following, so $\mathcal{A}_{SE}$ can simulate registering them with labels $\lambda^C(o)$ and $\lambda^V(o)$ respectively and update the public parameters accordingly without actually requiring the correct keys for these entities (as the adversary will not see any output from these entities other than their presence in the lists in the public parameters). $\mathcal{A}_{SE}$ can also run the FnInit and Certify algorithms as written.

7. $\mathcal{A}_{VC}$ is then provided with all relevant information and is given oracle access again. Queries can be handled as above, but Register queries now return $\perp$ if the queried label is an ancestor of the computation label in the poset i.e. exactly when $\mathcal{A}_{SE}$ does not hold a KAS key for the queried label.

8. $\mathcal{A}_{VC}$ finally outputs a forged encoded input σ_o.

Now, observe that from the point of view of $\mathcal{A}_{VC}$, the game has been simulated correctly up to this point and that $\mathcal{A}_{SE}$ has not made any queries to its Encrypt oracle in the INT-PTXT game. Thus, if the ciphertext c within σ_o decrypts successfully (which it must for $\mathcal{A}_{VC}$ to win the authorised outsourcing game), then the verification oracle $\mathrm{Ver}(c)$ will output 1 *and* the decrypted message has certainly not been queried to the Encrypt oracle, and thus $\mathcal{A}_{SE}$ can forward c as its answer in the INT-PTXT game and win with exactly the advantage of $\mathcal{A}_{VC}$ in the authorised outsourcing game. Thus, for a poset of polynomial size N, the advantage of $\mathcal{A}_{SE}$ is $\frac{\delta}{N}$ which is non-negligible if $\mathcal{A}_{VC}$ has non-negligible advantage δ in the authorised outsourcing game. However, since we assumed the authenticated symmetric encryption scheme to be secure, such an adversary with non-negligible advantage against the authorised outsourcing game cannot exist.

Thus, we conclude, the overall advantage against the authorised outsourcing game is the sum of the distinguishing advantage between **Game 0** and **Game 1**, and the advantage

in the reduction to INT-PTXT, both of which we have shown to be negligible. Therefore, the overall advantage against the authorised outsourcing game is negligible. □

We remark that in the above proof we required $\mathcal{A}_{SE}$ to correctly guess the label that $\mathcal{A}_{VC}$ would select ahead of time. This is very similar to the notions of security commonly found in functional encryption primitives, particularly identity-based encryption [32] and attribute-based encryption [27,66]. In these settings, it is common to refer to the method of guessing the correct label as *complexity leveraging* which results in a polynomial loss in the tightness of the reduction. An alternative, which could equally be taken here, is to formulate a weaker *selective* notion of security in which the adversary must select the challenge label at the beginning of the game before seeing the public parameters (in a similar fashion to the selective notions of security for ABE schemes discussed in Section 2.8.5).

We also note that the restriction on queries to the ProbGen oracle (i.e. that the challenge computation cannot be queried) is stronger than required for our particular construction due to the Encrypt oracle available in the INT-PTXT game. An alternative would be to allow queries to ProbGen for the challenge computation but require that the final adversarial output is distinct from those given in response to these queries.

The remaining proofs follow a similar line of argument and can be found in Appendix A. The proof of Theorem 4.1 follows as a corollary of Lemma 4.1 and Lemmas A.1 to A.3.

4.7 Conclusion

In this chapter, we have motivated the need for the cryptographic enforcement of access control policies in the setting of outsourced computation, particularly in the multi-user setting that we developed in Chapter 3. As developments in VC continue towards such settings, it is vital to enable restrictions to be placed on: the computations that delegators can outsource (both from the perspective of separation of duties, and considering a server providing differing levels of service for different users); the computations a server may perform (such that certain computations, over sensitive data say, may only be performed by a server satisfying a policy); and the verifiers that may learn the output of the result

(e.g. ensuring that read access to the newly generated data is handled in a way that is consistent with the sensitivity of the inputs). We have shown example graph-based access control policies for these scenarios, as well as providing a formal definitional framework, security models, and a provably secure construction built from key assignment schemes.

It may also be interesting to consider alternative enforcement mechanisms, such as authentication protocols that enforce graph-based authorisation policies to achieve ticket-based access control to a computational service. Such protocols have been explored by Alderman and Crampton [2] where they show that standardised authentication protocols can be easily extended (for example, using a KAS) to authenticate a user as possessing a certain set of access rights rather than to authenticate individual identities. They also show that this model of authentication can lead to novel authentication protocols by choosing appropriate posets of security labels. In Chapter 6, we will consider the use of dual-policy ABE to outsource computations (using the KP-ABE policy) and enforcing access control (using the CP-ABE policy).

Specific VC scenarios may lead to interesting access control models. One particularly applicable setting for the enforcement of access control policies is *verifiable searchable encryption* [37, 99]. Consider a remote database host that returns verifiably correct results to user queries (computations). In practice it is unlikely that all users should have unrestricted access to the entire database. It is imperative that only authorised users may perform specific queries (those relating solely to their duties and to data for which they have clearance) and that results remain protected to prevent data leakage.

Chapter 5

Verifiable Delegable Computation

Contents

In this chapter, we explore a novel form of publicly verifiable computation which we call VDC. Here, rather than clients submitting input data to computational servers, the servers themselves hold data which they allow to be verifiably queried by delegators. We see that this setting has natural applications such as verifiable queries on remote data and verifiable MapReduce operations. In Chapter 3, we explored techniques to achieve publicly verifiable outsourced computation using key-policy attribute-based encryption. In this chapter, we consider the similar primitive of ciphertext-policy attribute-based encryption (CP-ABE) which, too, was originally designed as a cryptographic enforcement mechanism for access control policies. We see that CP-ABE can be used to construct VDC.

5.1 Introduction

In this chapter, we consider an alternate model of outsourced computation which we call *verifiable delegable computation* (VDC). VDC can be considered as a reversal of the usual

model for publicly verifiable computation in which remote servers make available a static database over which any delegator may request computations (or queries) to be performed. Data is static and stored by the server and may be embedded in a server's secret key, whilst the computation of many different functions can be requested by using *only* public information. Thus, in this setting, the servers act as the data owners and delegators are more akin to data users, whereas in PVC, the delegators are the data owners and request servers to perform work on their behalf. The relationship between servers and delegators in VDC is more akin to the traditional client-server model.

We also see that VDC has very natural applications. For example, VDC can be used to perform verifiable queries on remote databases without access to the data itself, or to construct verifiable MapReduce operations for parallel computing problems. In the latter case, worker nodes hold some input data whilst multiple delegators can request each worker to compute some "sub-problem" F on their data portion. Verifiably correct results are returned to the delegator who can aggregate the results to form a correct result for the entire dataset. Note that it is not necessary to distribute the data to each delegator to allow them to request computations. We explain these example applications in more detail in Section 5.4.

Recall, from Chapter 3, that key-policy attribute-based encryption can be used to provide a proof mechanism for the correctness of outsourced computations. KP-ABE was designed, primarily, as a cryptographic enforcement mechanism for access control policies. Data objects to be protected are associated with a descriptive set of attributes and are encrypted, whilst users are issued a key corresponding to policies (i.e. Boolean formulae) defined over attributes. Users may decrypt an object, and thereby gain read access, if and only if the policy associated to their key is satisfied by the attributes associated to the object. The key insight of Parno et al. [84] was to observe that successful decryption of a ciphertext can be used as a proof that a Boolean formula F is satisfied by a set of attributes x (that is, that $F(x) = 1$), and can therefore be used for verifiable computation. In this setting, computational servers are certified to compute certain functions by being issued a KP-ABE decryption key for the policy representing F, whilst delegators outsource computations by encrypting random messages (which act as verification tokens) using attributes encoding the input data x.

Ciphertext-policy attribute-based encryption has very similar functionality to KP-ABE,

with the association of attribute sets and policies to ciphertexts and keys reversed. In this chapter, we see that CP-ABE can also be meaningfully employed in the verifiable computation setting to achieve VDC.

The *efficiency* requirement for this model is very different from the classical PVC setting: outsourcing a computation is no longer merely an attempt to gain efficiency since the delegator is never in possession of the input data and cannot execute the computation himself (even if it had the necessary resources). Thus, although a scheme should aim to be efficient, we do not have the stringent efficiency requirement present in PVC (that outsourcing and verifying computations be more efficient than performing the computation itself, to make outsourcing a worthwhile investment). Indeed, in the similar setting of Backes et al. [17], 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. Our solution achieves constant time public verification and the communication costs to delegate computations depends on the function F, while the size of the server's response depends only on the size of the computational result itself and *not* on the size of the input which may be large, particularly when querying remote databases. Future work in this area should focus on reducing the cost of outsourcing computations.

We begin this chapter by reviewing related work. In Section 5.3, we describe and formally define our model of publicly verifiable delegable computation. In Section 5.4, we consider several example applications for VDC: namely, verifiable MapReduce operations, verifiable queries on remote databases and three-party computations as introduced by Backes et al. [17]. We consider security of VDC in terms of public verifiability and blind verification in Section 5.5. We provide an example construction based on CP-ABE in Section 5.6, which is relevant to our *hybrid model* of publicly verifiable computation (Chapter 6). Finally, we prove that our construction is secure according to our security models.

5.2 Related Work

Work from the realm of authenticated data also lends itself to the concept of verifiable computations over outsourced data. Backes et al. [17] consider computations over outsourced data based on privacy-preserving proofs over authenticated data outsourced by

a trusted client, which we refer to as *three-party computation*. In this setting, a trusted source produces and authenticates some data which is given to a server. Other parties, that do not trust this server, can then request computations on this data and efficiently verify the results, but should learn nothing more than the computation results and their validity (i.e. should not learn the data itself). In our setting, the source can be thought of as the trusted KDC and then the same trust relations hold between the parties in both VDC and in three-party computations. The solution of Backes et al. [17] makes uses of homomorphic MACs and succinct non-interactive arguments (SNARGs) [78]. Similar results were presented in [91] using public logs. It is notable that the work by [17] and [30] also achieve the notion of public verifiability. In this chapter, we too achieve public verifiability but using very different techniques; we use the decryption mechanism of the CP-ABE scheme to achieve the same goal as the SNARGs.

Chung et al. [42] introduced *memory delegation* where a client uploads his memory to a server who can update and compute a function F over the entire memory. In our setting, the data owner need not be the client and computations can be performed on a subset of the data, but we only consider static data. Backes et al. [18] consider a client that outsources a large amount of data and requests computations on a data portion. The client can efficiently verify the correctness of the result without holding the input data. Most work in this realm of outsourced data requires the client to know the data to verify, e.g. in SNARG-based approaches [25, 30, 58] and signatures of correct computation [81]. Apon et al. [7] propose a notion of *verifiable oblivious storage* to ensure data confidentiality, access pattern privacy, integrity and freshness of data accesses.

5.3 Verifiable Delegable Computation

Informally, a VDC scheme for a family of functions $\mathcal{F}$ comprises a set of n computation servers S_i, $1 \leqslant i \leqslant n$. Each server S_i owns a dataset D_i comprising m_i data points — that is, $D_i = \{x_{i,j}\}_{j=1}^{m_i}$. S_i publishes a unique descriptive label $l(x_{i,j})$ of each data point $x_{i,j} \in D_i$.

S_i may also specify a list of functions $\mathcal{F}_i \subseteq \mathcal{F}$ that it is willing to compute, on behalf of any delegator, on specified portions of its data set (those data points that are in the domain of

Table 5.1: Example database for VDC

User ID	Name	Age	Height
001	Alice	26	165
002	Bob	22	172

Table 5.2: Example list $\mathcal{F}_i$

F	Dom(F)
Average	Age of record 1, Height of record 1, Age of record 2, Height of record 2
Most common value	Name of record 1, Age of record 1, Height of record 1, Name of record 2, Age of record 2, Height of record 2

a given function). As delegators are not the data owners, labels should *not* reveal the data values themselves in order to preserve the confidentiality of D_i. Delegators may select servers and data using *only* knowledge of these labels. Delegators may ask S_i to compute $F(X)$ for any function $F \in \mathcal{F}_i$ on a set of data points $X \subseteq \mathsf{Dom}(F)$ by specifying the set of labels $\{l(x_{i,j})\}_{x_{i,j} \in X}$. Note that F has $|X|$ inputs, e.g. if $X = \{x_1, x_3, x_7\}$ then the server computes $F(x_1, x_3, x_7)$.

Example 5.1. *Consider a server S_i that owns the database shown in Table 5.1. The dataset D_i represents this database as the set of field values for each record in turn — $D_i = \{001, \text{Alice}, 26, 165, 002, \text{Bob}, 22, 172\}$.*

S_i publishes a list of data labels {User ID of record 1, Name of record 1, Age of record 1, Height of record 1, User ID of record 2, Name of record 2, Age of record 2, Height of record 2}. S_i also publishes a list of functions $\mathcal{F}_i$ that it is willing to compute over D_i along with the domain of each function, as in Table 5.2.

A delegator may then query for the "Average" on the data labelled by the set $X =$ {Age of record 1, Age of record 2}.

Note that, although it may be tempting to suggest that S_i simply caches the results of computing each $F \in \mathcal{F}_i$ on its dataset, the choice of data points $X \subseteq \mathsf{Dom}(F)$ for each computation could vary; hence the number of results that must be cached could be large, making this an unattractive solution.

Also note that the list $\mathcal{F}_i$ could be defined to include computations such as "Average height

of entities over the age of 21" whereby the list of available inputs $\mathsf{Dom}(F)$ is filtered by the server to only include such records. However, in these cases, care must be taken that these lists of data labels, with associated queries for which they are acceptable inputs, do not leak too much information about the data or allow data items to be linked across multiple queries — by observing the same data label satisfying multiple queries, it may be possible to learn detailed information about the data item. In these cases, the data labels could be randomised per function by hashing the data and the function identifier with a pre-image resistant hash function.

A VDC scheme begins with a key distribution centre (KDC) (e.g. a trusted third party as in Chapter 3, a trusted data source or a delegator) running Setup to produce public parameters and a master secret key. The KDC also registers each server S_i to provide a 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 enrols with the KDC using the $\mathsf{Certify}$ algorithm to make their dataset D_i available for computations. The KDC issues a (single) evaluation key EK_{D_i,S_i} which enables S_i to perform computations on D_i (or, indeed, subsets of D_i). As the server is the data owner in the setting of VDC, it should be able to specify which functions are (i) meaningful on their dataset (some data may not be in the domain of all functions in $\mathcal{F}$), and (ii) permissible in accordance with their own access control policies — that is, servers may not wish to evaluate and reveal the results of *all* computations over their data. Thus, each server provides a list of functions, $\mathcal{F}_i \subseteq \mathcal{F}$, that they are willing to evaluate on their data. In some settings, such as the MapReduce example, $\mathcal{F}_i$ may well be the full set $\mathcal{F}$ since the delegator is the original data owner (of the entire input dataset) and issues each worker with a static portion of that data. In this case, the delegator should be able to request *any* meaningful computation from the worker nodes. However when servers are remote data providers, they themselves own the input dataset and should be able to specify what that data be used for.

Furthermore, not all data points $x_{i,j} \in D_i$ may be appropriate for each function (e.g. if $F \in \mathcal{F}$ is an averaging function, then only numeric data in D_i should be used as input). For the purposes of this chapter, we define the set $\mathcal{F}_i$ to comprise elements of the form $(F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$ describing each function and the associated permissible inputs.

Throughout this chapter, we also assume (to aid the client in encoding the function F in our construction), that the set of labels $\{l(x_{i,j})\}_{x_{i,j} \in D_i}$ reveals the order of each data point as stored in the server's memory and the size of each data point when represented as a bitstring; thus, clients may know the memory location from which data should be used, but *do not* know the value of each bit. Note that data can be padded to achieve a uniform length so that revealing the size of each data point also does not reveal information about the data value.

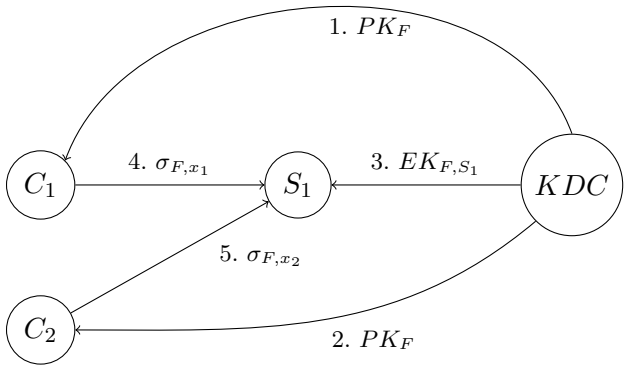
To request a computational result from a server, a delegator runs the $\mathsf{ProbGen}$ algorithm. The delegator chooses a function $F \in \mathcal{F}_i$ and a set of data points $X \subseteq \mathsf{Dom}(F)$. In the MapReduce example, $\mathsf{ProbGen}$ would be run once per worker for the same function F (which is acceptable by all workers because $\mathcal{F}_i = \mathcal{F}$ for all $i \in [n]$). $\mathsf{ProbGen}$ generates an encoded input $\sigma_{F,X}$, verification key $VK_{F,X}$ and output retrieval key $RK_{F,X}$. A server S_i uses their evaluation key EK_{D_i,S_i} to compute $\theta_{F(X)}$ encoding the computational result $F(X)$.

Verification is divided into two algorithms. We consider public verifiability where *any* party (including an adversary) may verify a result. An example motivation of where public verifiability may be useful is in a grid computing environment where some management software ensures results are correct before returning them to the user. Blind verification is performed by *any* user using a verification key $VK_{F,X}$ to verify correctness. It generates a retrieval token $RT_{F(X)}$ for valid outputs (or $\perp$ if the output is invalid) but does *not* reveal the actual output value. Finally, $\mathsf{Retrieve}$ takes a (non-$\perp$) token $RT_{F(X)}$ and the output retrieval key $RK_{F,X}$ to reveal the final result $y_{F(X)} = F(X)$. This is also illustrated and compared to publicly verifiable outsourced computation in simplified form in Figure 5.1. More formally:

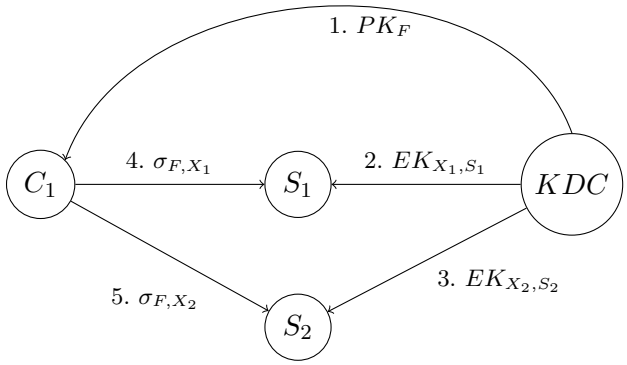
Definition 5.1. A *publicly verifiable delegable computation (VDC) scheme* comprises the following algorithms[1]:

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$: run by the KDC to initialise the system. The input values are the unary representation of the security parameter, 1^ℓ, and the family of functions, $\mathcal{F}$, that may be outsourced;
- $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate a public delegation key

[1] We retain the algorithm names from prior PVC schemes for consistency.



(a) PVC



(b) VDC

Figure 5.1: Comparison between PVC and VDC

PK_F allowing clients to request computations of a function F;

- $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to enrol a computational server S_i in the system and to generate a signing key SK_{S_i} for S_i;

- $EK_{D_i,S_i} \xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{l(x_{i,j})\}_{x_{i,j} \in D_i}, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate an evaluation key EK_{D_i,S_i} which enables the server S_i to perform computations of all functions $F \in \mathcal{F}_i$ chosen by the server, on the server's input data $D_i = \{x_{i,j}\}_{j=1}^{m_i}$, comprising m_i data points, each uniquely labelled by $l(x_{i,j})$;

- $(\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(F, \{l(x_{i,j})\}_{x_{i,j} \in X}, PK_F, \mathrm{PP})$: run by a client to request the computation of the function F on a set of data points $X \subseteq D_i$ belonging to the server S_i. The input values are the function F to be computed, a set of labels identifying each data point $x_{i,j} \in X$ to be computed on (note that the data itself is not required), the public delegation key PK_F for the function and the public parameters. The output values are an encoded input $\sigma_{F,X}$, a public verification key $VK_{F,X}$, and a retrieval key $RK_{F,X}$;

- $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,X}, EK_{D_i,S_i}, SK_{S_i}, \mathrm{PP})$: run by a server S_i to compute $F(X)$. The input values are the encoded input $\sigma_{F,X}$ prepared by a client for a set of input data points $X \subseteq D_i$, an evaluation key EK_{D_i,S_i} permitting S_i to compute on its dataset D_i, a signing key SK_{S_i} for the server, and the public parameters. The algorithm produces an encoded output $\theta_{F(X)}$ of $F(X)$;

- $y_{F(X)} \leftarrow \mathsf{Verify}(\theta_{F(X)}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$: verification comprises two sub-algorithms as follows. These could be performed together as a single Verify algorithm if blind verification is not required:

  - $(RT_{F(X)}, \tau_{\theta_{F(X)}}) \leftarrow \mathsf{BVerif}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$: run by a verifier in possession of the encoded output $\theta_{F(X)}$, a verification key for the computation and the public parameters. The output of this algorithm is a retrieval token $RT_{F(X)}$ encoding the actual output value of the computation (this can be thought of as a partial translation from $\theta_{F(X)}$ to $F(X)$) and a token $\tau_{\theta_{F(X)}}$, the value of which is accept if $\theta_{F(X)}$ is a valid server response for $F(X)$ and reject otherwise. Note that the verifier learns only whether $\tau_{\theta_{F(X)}}$ is correct, it cannot learn the value of $F(X)$;

  - $y_{F(X)} \leftarrow \mathsf{Retrieve}(RT_{F(X)}, \tau_{\theta_{F(X)}}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$: run by an authorised verifier (i.e. a verifier holding the retrieval key $RK_{F,X}$) generated by the client during ProbGen, the verification key $VK_{F,X}$ and the outputs from the BVerif stage. This algorithm returns the actual computational result $y_{F(X)}$ which is either $y_{F(X)} = F(X)$ if the server performed correctly, or a distinguished failure symbol $y_{F(X)} = \perp$ otherwise.

We do not consider revocation here but observe that an indirectly revocable CP-ABE scheme could be employed in a similar fashion to the indirectly revocable KP-ABE scheme in Chapter 3. In Chapter 6, we will introduce a hybrid scheme which incorporates VDC functionality with a revocation mechanism. Although not explicitly stated, the KDC may update the public parameters PP during any algorithm, in order to reflect changes in the user population (e.g. servers are added or removed from the system, or granted the ability to compute additional functions).

In this model, the server must provide its input data to the trusted KDC in order to be certified. In some application settings, this is expected behaviour. For example, if the manager of an organisation acts as the KDC and divides a database according to a

separation of duty policy between different servers. A similar partitioning and distribution of data also occurs in MapReduce applications. In the three-party computation model introduced by Backes et al. [17], a trusted data source acts as the KDC and issues data directly to the server.

In other settings, the server may be the data owner and may have to trust the KDC with its data (but not the delegators). By requiring a trusted KDC to issue keys, servers are prevented from creating a key for data they do not own (or indeed a subset of their data). Future work will attempt to relax this requirement.

Intuitively, we say that a VDC scheme is *correct* if all arbitrary execution sequences of the algorithms where a computational result is honestly computed by a non-revoked server results in the verification algorithm accepting the correct result of the computation. We can model this as a cryptographic game between a challenger and a PPT adversary; the adversary aims to find an (honestly generated) encoded output which either does not encode the correct result or which does encode the correct result but which will not be accepted by the verification algorithm. For each algorithm in Definition 5.1, we define a corresponding oracle that executes the algorithm on inputs provided by the adversary, and returns the output of the algorithm to the adversary. The adversary may query the Setup oracle only once (before making any other oracle queries), but can thereon call the remaining oracles any number of times and in any order.

The challenger maintains two lists, L_{Reg} and L_F; L_{Reg} is a list of tuples comprising server identities, S_i, and the resulting signing keys, SK_{S_i}, that have been queried to the Register oracle, whilst L_F comprises tuples of the form $(S_i, D_i, \{l(x_{i,j})\}_{x_{i,j} \in D_i}, \mathcal{F}_i, EK_{D_i,S_i})$ denoting that the server S_i has been queried to the Certify oracle for the set of functions $\mathcal{F}_i$ and the data set D_i, and that EK_{D_i,S_i} was generated.

The challenger also creates and maintains a table T which records the parameters relating to each computation performed through the oracle queries. T is updated in the following oracles:

- ProbGen: the challenger creates a new row in T comprising 8 components, all of which are initialised to be empty; it then assigns X (which can be found as a subset of D_i in L_F), F, the result $F(X)$ (computed by the challenger itself), $\sigma_{F,X}, VK_{F,X}$

and $RK_{F,X}$ to the first 6 components;

- Compute: the challenger first searches T for all rows that contain the queried $\sigma_{F,X}$ in the 4^{th} component and where the 7^{th} component is empty (i.e. those rows relating to computations on this encoded input that have not yet been performed). For each such row, r, the challenger takes the second component (the function identifier, $\tilde{F}$), and checks that there exists a server identity $\tilde{S}_i$ such that the tuple $(\tilde{S}_i, SK_{\tilde{S}_i}) \in L_{\text{Reg}}$ (where $SK_{\tilde{S}_i}$ is that given as input to the Compute oracle) *and* such that the tuple $(\tilde{S}_i, \cdot, \cdot, \mathcal{F}_i, EK_{D_i,S_i}) \in L_F$, where EK_{D_i,S_i} is also that given as input to the Compute oracle and where $\tilde{F} \in \mathcal{F}_i$. This check ensures that there is a server that holds the signing key and evaluation key being used to perform the computation and which is certified for a function $\tilde{F}$ for which the encoded input $\sigma_{F,X}$ was generated.

  The challenger then performs the Compute algorithm on the queried $\sigma_{F,X}, EK_{D_i,S_i}$ and SK_{S_i} to produce an output $\theta_{F(X)}$. For each of the rows r of T found above, the challenger writes $\theta_{F(X)}$ and $\tilde{S}$ to the 7^{th} and 8^{th} components of r respectively.

Thus, when complete, the entries of T will be of the form

$$(x, F, F(X), \sigma_{F,X}, VK_{F,X}, RK_{F,X}, \theta_{F(X)}, S_i).$$

After a polynomial number of queries, the adversary will return a value $\theta^{\star}_{F(X)}$ which he believes encodes a correct computational result, yet which the Verify algorithm will reject (that is, an output for which the protocol execution will not be correct). The challenger first performs a look up in T for all entries containing $\theta^{\star}_{F(X)}$ in the 7^{th} position of the tuple, and stores any such entries as another table $\tilde{T}$. Note that this means that $\theta^{\star}_{F(X)}$ must have been honestly generated by the Compute oracle (else it would not be in T).

For each such row, the challenger uses the 5^{th} component (the verification key) to run BVerif on $\theta^{\star}_{F(X)}$ to generate the outputs $RT_{F(X)}$ and $\tau_{\theta_{F(X)}}$, and then uses the 5^{th} and 6^{th} components (the verification key and retrieval key) and $\tau_{\theta_{F(X)}}$ to run Retrieve on $\theta^{\star}_{F(X)}$ to generate $y_{F(X)}$.

The challenger first checks whether $y_{F(X)}$ matches the 3^{rd} component of the row (that is, whether $y_{F(X)}$ is the correct computational result $F(x)$). If so, it then checks whether $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$, and if so it ends the game by returning 1 to indicate that the adversary has won the game (the adversary has found a valid encoding of a correct result, computed

by a certified server, that the Verify algorithm is incorrectly rejecting).

On the other hand, if $y_{F(X)}$ did *not* match the correct value of $F(x)$, the challenger also ends the game by returning 1 to indicate that the adversary has won the game (the adversary in this case has found an incorrect result that was computed honestly by the algorithms).

If no row in $\tilde{T}$ allows the adversary to win, then the challenger outputs 0 to indicate that the adversary has lost. An VDC scheme is *correct* if, for all PPT adversaries, the probability that the adversary wins the game described above is 0.

5.4 Potential Applications for VDC

We consider the following example applications to motivate our study of VDC.

- **MapReduce** [50] (or Hadoop [93]) 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 set of worker nodes each compute subproblems on portions of the data and report to a manager who combines the results. A MapReduce problem comprises two stages:
  - **Map:** a master node (or manager) takes the input and divides it into smaller sub-problems which are distributed to the worker nodes. The worker computes the function associated with the sub-problem and passes the result back to the manager;
  - **Reduce:** the manager collects the answers to all sub-problems and combines them to form the output to the original problem.

  VDC enables verifiable MapReduce such that only *valid* results are combined. The manager acts as the key distribution centre (KDC) to distribute evaluation keys for partitions of the data to workers. He can then request multiple sub-problems to be solved over this data partitioning.

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

vice. Any delegator may use public information to query *any* function allowed by the server (within the family allowed by the VDC scheme) on these databases. Data is remotely stored and delegators see nothing more than the results of queries which they are assured are 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. It is important to note that the data owner is not required to retain any knowledge of the data after it has been outsourced.

- **Three-party computation**. The model of Backes et al. [17] considers a trusted data source that provides authenticated data to a service provider who can then provide verifiably correct results to computations requested by third parties. As mentioned in Section 5.2, in the context of VDC, the source can be thought of as the KDC, the service provider as the computational server and the third parties as delegators.

  Backes et al. [17] considered several applications for this model of computation. Firstly, trusted sensors could be placed in client premises (e.g. a smart energy meter or a sensor placed in a car to monitor driving habits) or worn by the client (e.g. a wearable health band). These sensors collect data which is authenticated (due to the trusted nature of the collection devices) and given to the client who acts as the service provider. Because this data could be sensitive (e.g. revealing the habits and lifestyle of the client), the service provider may be reluctant to release the data to third parties. Nevertheless, there exist legitimate business cases that require access to compute on the data (e.g. for billing purposes, or to produce an insurance quote if the sensor is a health monitor or monitors driving habits). Therefore, these third parties may request appropriate computations on the data from the service provider itself, and can verify that the service provider performs the computation correctly on the correct data.

  Secondly, Backes et al. suggest a company whose finances must be periodically audited. The trusted sensor in this setting would be the official bookkeeper for the company who is legally responsible for maintaining correct financial records. The company keeps the records private as they are business-critical, but provides verifiable computational results to the auditors.

Game 5.1 $\mathbf{Exp}_{\mathcal{A}}^{\text{PubVerif}}\left[\mathcal{VDC}, 1^{\ell}, \mathcal{F}\right]$

1: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
2: $(F, X^{\star}, \{l(x_j)\}_{x_j \in X^{\star}}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\text{PP})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
4: $(\sigma_{F,X^{\star}}, VK_{F,X^{\star}}, RK_{F,X^{\star}}) \xleftarrow{\$} \mathsf{ProbGen}(F, \{l(x_j)\}_{x_j \in X^{\star}}, PK_F, \text{PP})$
5: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{F,X^{\star}}, VK_{F,X^{\star}}, RK_{F,X^{\star}}, PK_F, \text{PP})$
6: $(RT_{F(X^{\star})}, \tau_{\theta^{\star}}) \leftarrow \mathsf{BVerif}(\theta^{\star}, VK_{F,X^{\star}}, \text{PP})$
7: $y_{F(X^{\star})} \leftarrow \mathsf{Retrieve}(RT_{F(X^{\star})}, \tau_{\theta^{\star}}, VK_{F,X^{\star}}, RK_{F,X^{\star}}, \text{PP})$
8: **if** $(((y_{F(X^{\star})}, \tau_{\theta^{\star}}) \neq (\perp, \mathsf{reject}))$ **and** $(y_{F(X^{\star})} \neq F(X^{\star})))$ **then**
9: **return** 1
10: **else return** 0

5.5 Security Model

In the context of VDC, we consider security in the sense of *public verifiability* to ensure that a server cannot return incorrect results without being detected. This is a natural extension of that discussed in Chapter 3; we do not require the other notions introduced in Chapter 3 as we do not consider revocation in this chapter.

The notion of public verifiability is captured in Game 5.1 to ensure that a server (even having corrupted other servers and holding verification keys) may not cheat by returning an incorrect result without being detected, even without the original delegator or the verifier possessing the input data. The game begins with the challenger setting up the system and providing the resulting public parameters to the adversary. The adversary is also given oracle access, denoted $\mathcal{O}$, to the following functions: $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$ and $\mathsf{Certify}(\cdot, \cdot, \cdot, \cdot, \text{MK}, \text{PP})$.

The adversary will select the challenge inputs which are the function F to be computed, the input data points $X^{\star}$ to be computed on, and the labels $l(x_j)$ for each data point $x_j \in X^{\star}$ (as the adversary is acting as a malicious server, it creates and owns the data $X^{\star}$ available to be computed on and may query the $\mathsf{Certify}$ oracle for a data set $D \supseteq X^{\star}$). The challenger runs FnInit to initialise the challenge function F and forms a challenge by running $\mathsf{ProbGen}$ on the challenge input labels and function. It sends the resulting parameters to $\mathcal{A}$ and again provides oracle access as above. The adversary wins the game if it is able to produce an encoded output that verifies correctly but does not correspond to the actual result $F(X^{\star})$.

Definition 5.2. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* PubVerif *game for a VDC construction, $\mathcal{VDC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

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

$$Adv_{\mathcal{A}}^{\text{PubVerif}}(\mathcal{VDC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

5.6 Construction

Informally, our construction to compute the family of (monotone) Boolean formulas, closed under complement, operates similarly to that for RPVC in Chapter 3.

As with our RPVC construction, to encode an n-bit binary input string $\vec{x} = x_1 x_2 \ldots x_n$ as an attribute set A_x, we define a universe $\mathcal{U}_x = \{\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}\}$ of n attributes and let $\mathtt{x_i} \in A_x$ if and only if the i^{th} bit of the input string is 1 — that is, $A_x = \{\mathtt{x_i} : x_i = 1\}$. Note that the inputs to our computations are sets of input data points $X \subseteq D_i$ belonging to the server S_i; D_i can be thought of as a bitstring made from the concatenation of the bitstring representation of each input data point $x_{i,j} \in D_i$, and then D_i can be encoded as above. Again, as in Section 3.2.1, we may include attributes representing 0 bit values if desired.

To request a computation of $F(X)$, a client must encode a function $F \in \mathcal{F}_i$ in terms of attributes, and in particular those attributes that encode $X \subseteq D_i$ — that is, a meaningful computation will require particular input values to be used at particular points and hence, when encoding F, the client must be able to specify these inputs. Recall that we assumed that the set of labels $\{l(x_{i,j})\}_{x_{i,j} \in D_i}$ published by the server including information about the size of each data point $x_{i,j} \in D_i$ and its order within the dataset. Thus, since A_{D_i} is the concatenation of the binary representation of these ordered data points, clients may encode Boolean functions (which operate on the binary data points) in terms of the bit positions within the concatenated bitstring, or equivalently, in terms of the attributes $\{\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x}_{\sum_{x_{i,j} \in D_i} |x_{i,j}|}\} \in \mathcal{U}_x$. Note that the delegator does not need to know the value of each attribute (or whether the server holds that attribute) in order to form this function.

The delegator 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 is given a decryption key for the data D_i that they hold. 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, servers learn nothing about the message if $F(D_i) = 0$ since this corresponds to an access structure not being satisfied. Thus, if the correct message is returned, the delegator 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 delegator of a false negative result. Thus, we produce two CP-ABE ciphertexts. As above, one corresponds to F, whilst the other corresponds to $\overline{F} = F(X) \oplus 1$. 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 is the subset of D_i that the function is actually required to operate over. Note that the function F is encoded in terms of attributes, and is specific to each input (that is, the encoding of a function F will differ to compute $F(X)$ and $F(X')$). As a result, if the function F is evaluated on an input set $X' \supset X$ then the server will discard the additional data points $x \in X' \setminus X$ and the outcome of the computation remains $F(X)$.

A well formed response, (d_0, d_1), from a server, therefore, satisfies the following:

$$(d_0, d_1) = \begin{cases} (m_0, \perp), & \text{if } F(D_i) = 1; \\ (\perp, m_1), & \text{if } F(D_i) = 0. \end{cases} \tag{5.1}$$

If the returned plaintext does not match one of the chosen random messages then the server has returned an incorrect result (also if both results are $\perp$ but no rational malicious server would return this).

Public verifiability is achieved by publishing a token comprising the output of 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. For blind verification, a random bit b chosen by the delegator permutes the ciphertexts, hiding whether the matched plaintext is associated with F or $\overline{F}$. The public parameters contain a two-dimensional array L_{Reg} with two columns: the first column, $L_{Reg}[S_i][0]$, is indexed by server identities and contains signature verification keys; the second column, $L_{Reg}[S_i][1]$, lists functions and labels for which S_i is certified.

Let $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{ID}$, where $\mathcal{U}_l$ is a disjoint (from $\mathcal{U}_x$) universe comprising attributes representing each unique data label, $l(x_{i,j})$, and $\mathcal{U}_{ID}$ comprises server identities.[2] If such a universe becomes too large it is, of course, possible to use a *large universe* CP-ABE scheme [66] where attributes need not be defined ahead of time. However, it is very likely that, for efficiency of the system, the inputs and number of servers will be polynomially sized in the security parameter and can therefore be accommodated by a *small universe*.

As we consider adversaries that have access to multiple keys, we must ensure that a key for different data cannot produce a response that will be accepted as valid. We do this by labelling each data point $x_{i,j} \in D_i$ with a unique (across the entire system) label $l(x_{i,j})$ and define an attribute in $\mathcal{U}_l$ for each label; for ease of exposition, we refer to these attributes simply as $l(x_{i,j})$ as well — in practice, one could hash the text label into the bilinear group to form the attribute.

For a data set D_i, the decryption key for a server S_i is formed over the attribute set $(A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j}))$. During $\mathsf{ProbGen}$ for a computation of $F(X)$ for a set of data points $X \subseteq \mathsf{Dom}(F), X \subseteq D_i$ for some i, the encryption uses the access structure encoding of the conjunction $(F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j}))$ specified in terms of attributes corresponding to the required input data (and similarly for the complement function $\overline{F}$). Thus, decryption only succeeds if $F(D_i) = 1$ (i.e. $F(X) = 1$ as the remaining $x_{i,j} \notin X$ are ignored by the decryption procedure) *and* all the labels $l(x_{i,j})$, for $x_{i,j} \in X$, are matched in the key and ciphertext — a key for different data will not include the correct labels. Note that, because the encoding of the function specifies the attributes that should be provided for the function to be evaluated and because the evaluation key is formed over the entire (ordered) dataset, the server may not use different data points within his own dataset to forge a result either.

Our instantiation of VDC is more efficient than that for PVC since we do not require the setup of two independent ABE systems or two (expensive) key generations. However, in contrast to PVC, the delegator must perform work roughly equivalent to performing the computation itself twice, in order to prepare an encoded input (that is, to encrypt two messages with the access structure corresponding to F or $\overline{F}$ using a CP-ABE scheme). Whereas in PVC environments, this would be unacceptable (as it negates the purpose of

[2] We assume that the following algorithms check, where relevant, that all functions and input data are formed over $\mathcal{U}_x$ and each additionally contains exactly one attribute/clause over the label universe $\mathcal{U}_l$.

outsourcing to more powerful servers), in the context of VDC, this is not such an issue — the delegator does not possess the input data but would like to learn $F(X)$; clearly, it is not possible for it to compute $F(X)$ itself without the data, but the delegator can perform approximately the same amount of work as computing $F(X)$ to request the computation from the data owner.

Let $\mathcal{CPABE}$ = (ABE.Setup, ABE.KeyGen, ABE.Encrypt and 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 5.1–5.8 define a VDC scheme for the class of functions $\mathcal{F}$, which operates as follows:

1. VDC.Setup, presented in Algorithm 5.1, first forms the attribute universe for the function family, accommodating all possible input labels and server identities. It then initialises the CP-ABE system by calling the ABE.Setup algorithm, and initialises the two-dimensional array L_{Reg} indexed by server identities — for a server S, $L_{\text{Reg}}[S][0]$ will store a signature verification key for S and $L_{\text{Reg}}[S][1]$ will store a list of functions that S is willing to compute. The public parameters for the VDC system comprise the public parameters of the CP-ABE scheme and the array L_{Reg}. The master secret, MK, for the system is the master secret of the CP-ABE scheme.

Algorithm 5.1 $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$

$\mathcal{U} \leftarrow \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{ID}$
$(MPK_{\text{ABE}}, MSK_{\text{ABE}}) \xleftarrow{\$} \mathsf{ABE.Setup}(1^\ell, \mathcal{U})$
for $S_i \in \mathcal{U}_{\text{ID}}$ **do**
    $L_{\text{Reg}}[S_i][0] \leftarrow \epsilon$, $L_{\text{Reg}}[S_i][1] \leftarrow \{\epsilon\}$
PP $\leftarrow (MPK_{\text{ABE}}, L_{\text{Reg}})$, MK $\leftarrow (MSK_{\text{ABE}})$

2. VDC.FnInit, presented in Algorithm 5.2, simply outputs the public parameters for the VDC system, and is the same for all functions. In our particular construction, this step is not required due to the use of public key CP-ABE — the delegation of *all* functions $F \in \mathcal{F}$ simply use the public parameters of the CP-ABE scheme.

Algorithm 5.2 $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$

$PK_F \leftarrow \text{PP}$

3. VDC.Register, presented in Algorithm 5.3, creates a key pair using the digital signature KeyGen algorithm. The signing key is issued to the server S_i being registered,

and the verification key is added to the public array L_{Reg} so that anyone can verify signatures produced by S_i.

Algorithm 5.3 $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \text{MK}, \text{PP})$

1: $(SK_{\text{Sig}}, VK_{\text{Sig}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\ell)$
2: $SK_{S_i} \leftarrow SK_{\text{Sig}}$
3: $L_{\text{Reg}}[S_i][0] \leftarrow VK_{\text{Sig}}$

4. VDC.Certify, presented in Algorithm 5.4, certifies that a server S_i may provide computational services on the dataset D_i and is willing to perform any computation in the set $\mathcal{F}_i \subseteq \mathcal{F}$. The data set D_i comprises m_i data points $x_{i,j}$, each of which is uniquely represented by an attribute $l(x_{i,j}) \in \mathcal{U}_l$. For each function $F \in \mathcal{F}_i$, the pair $(F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$ is added to the array $L_{\text{Reg}}[S_i][1]$ to indicate to prospective clients that S_i is willing to compute F on any set comprising data points $x_{i,j} \in D_i$ which are in the domain of F. The algorithm then generates a CP-ABE decryption key for the attribute set $A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j})$ that forms the evaluation key for the server.

Algorithm 5.4 $EK_{D_i,S_i} \xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{l(x_{i,j})\}_{x_{i,j} \in D_i}, \mathcal{F}_i, \text{MK}, \text{PP})$

1: **for** $F \in \mathcal{F}_i$ **do**
2: $\quad L_{\text{Reg}}[S_i][1] \leftarrow L_{\text{Reg}}[S_i][1] \cup (F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$
3: $SK_{\text{ABE},D_i} \xleftarrow{\$} \mathsf{ABE.KeyGen}((A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j})), MSK_{\text{ABE}}, MPK_{\text{ABE}})$
4: $EK_{D_i,S_i} \leftarrow SK_{\text{ABE},D_i}$

5. VDC.ProbGen, presented in Algorithm 5.5, chooses two messages uniformly at random from the message space and a random bit b. For a computation request for $F(X)$ to a server S_i, where $X \subseteq D_i$ and $X \subseteq \mathsf{Dom}(F)$, the algorithm forms two CP-ABE ciphertexts that encrypt the chosen messages under the policies $(F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j}))$ and $(\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j}))$ respectively. These ciphertexts form the encoded input $\sigma_{F,X}$ while the bit b forms the retrieval key $RK_{F,X}$. The verification key for the computation is created by applying the one-way function g to each chosen message.

Algorithm 5.5 $(\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(F, \{l(x_{i,j})\}_{x_{i,j} \in X}, PK_F, \text{PP})$

1: $(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
2: $b \xleftarrow{\$} \{0, 1\}$
3: $c_b \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_b, (F \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})), MPK_{\text{ABE}})$
4: $c_{1-b} \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{1-b}, (\overline{F} \wedge \bigwedge_{x_{i,j} \in X} l(x_{i,j})), MPK_{\text{ABE}})$
5: $\sigma_{F,X} \leftarrow (c_b, c_{1-b})$, $VK_{F,X} \leftarrow (g(m_b), g(m_{1-b}))$, $RK_{F,X} \leftarrow b$

6. VDC.Compute, presented in Algorithm 5.6, first decrypts both CP-ABE ciphertexts

in the encoded input using the decryption key issued as the evaluation key for $D_i \supseteq X$ to server S_i. It then signs the resulting plaintexts using the server's signing key and returns the plaintexts, server ID and signature as the encoded output. Note that the recovered plaintexts are $(m_b, \perp)$ if $F(X) = 1$ or $(\perp, m_{1-b})$ if $F(X) = 0$ (ordered according to the random bit $RK_{F,X} = b$ chosen in VC.ProbGen).

Algorithm 5.6 $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{F,X}, EK_{D_i,S_i}, SK_{S_i}, \mathrm{PP})$

1: Parse $\sigma_{F,X}$ as (c_b, c_{1-b})
2: $d_b \leftarrow \mathsf{ABE.Decrypt}(c_b, EK_{D_i,S_i}, MPK_{\mathrm{ABE}})$
3: $d_{1-b} \leftarrow \mathsf{ABE.Decrypt}(c_{1-b}, EK_{D_i,S_i}, MPK_{\mathrm{ABE}})$
4: $\gamma \xleftarrow{\$} \mathsf{Sig.Sign}((d_b, d_{1-b}, S_i), SK_{S_i})$
5: $\theta_{F(X)} \leftarrow (d_b, d_{1-b}, S_i, \gamma)$

7. VDC.BVerif, presented in Algorithm 5.7, first checks that the signature over the encoded output verifies correctly under the verification key for the server S_i. It then applies the one-way function g to each plaintext returned in $\theta_{F(X)}$ and compares it to the corresponding element of the verification key. If neither comparison matches, then the verifier outputs $RT_{F(X)} = \perp$ and $\tau_{\theta_{F(X)}} = \mathsf{reject}$ to show that the server did not provide a valid computational result, and otherwise outputs the matched plaintext as the retrieval token and $\tau_{\theta_{F(X)}} = \mathsf{accept}$.

Algorithm 5.7 $(RT_{F(X)}, \tau_{\theta_{F(X)}}) \leftarrow \mathsf{BVerif}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$

1: Parse $VK_{F,X}$ as (VK, VK') and $\theta_{F(x)}$ as (d, d', S_i, γ)
2: **if** accept $\leftarrow \mathsf{Sig.Verify}((d, d', S_i), \gamma, L_{\mathrm{Reg}}[S][0])$ **then**
3: **if** $(VK = g(d))$ **then return** $(RT_{F(X)} \leftarrow d, \tau_{\theta_{F(X)}} \leftarrow \mathsf{accept})$
4: **if** $(VK' = g(d'))$ **then return** $(RT_{F(X)} \leftarrow d', \tau_{\theta_{F(X)}} \leftarrow \mathsf{accept})$
5: **return** $(RT_{F(X)} \leftarrow \perp, \tau_{\theta_{F(X)}} \leftarrow \mathsf{reject})$

8. VDC.Retrieve, presented in Algorithm 5.8, orders the verification key according to the bit b that forms the retrieval key. It can then determine whether m_0 or m_1 was matched in VDC.BVerif, and hence whether $F(X) = 1$ or $F(X) = 0$ respectively.

Algorithm 5.8 $y_{F(X)} \leftarrow \mathsf{Retrieve}(RT_{F(X)}, \tau_{\theta_{F(X)}}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$

1: Parse $VK_{F,X}$ as $(g(m_b), g(m_{1-b}))$ and $RK_{F,X}$ as b
2: **if** $(g(RT_{F(X)}) = g(m_0))$ **then return** $y_{F(X)} \leftarrow 1$
3: **if** $(g(RT_{F(X)}) = g(m_1))$ **then return** $y_{F(X)} \leftarrow 0$
4: **return** $y_{F(X)} \leftarrow \perp$

It is straightforward to see that correctness of this construction follows from the correctness of the attribute-based encryption scheme and of the one-way function g.

Theorem 5.1. *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 in the sense of* EUF-CMA*, let $\mathcal{VDC}$ be the verifiable delegable computation scheme defined in Algorithms 5.1–5.8. Then $\mathcal{VDC}$ is secure in the sense of selective public verifiability (Game 5.1).*

5.7 Proof of Security

Informally, public verifiability relies on the IND-CPA property of the CP-ABE encryption and the one-wayness of g. The proof proceeds by showing that, for the unsatisfied function F or $\overline{F}$, an adversary cannot observe if the plaintext is altered. Thus, the verification key can be the one-way function challenge $g(w)$ and the plaintext can implicitly be set to w.

Proof. We begin by defining the following three games:

- **Game 0.** This is the public verifiability game as defined in Game 5.1.
- **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 $\mathsf{ABE.Encrypt}(m', (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j)), MPK_{\text{ABE}})$. Otherwise, we replace c_0 by $\mathsf{ABE.Encrypt}(m', (F \wedge \bigwedge_{x_j \in X^\star} l(x_j)), MPK_{\text{ABE}})$. 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 (Game 2.12).

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, for a contradiction, that $\mathcal{A}_{VDC}$ can distinguish the two games with non-negligible advantage δ. We then construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VDC}$ 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 (Game 2.8) with attribute universe $\mathcal{U}$ with $\mathcal{A}_{ABE}$, who in turn acts as a challenger in the public verifiability game for $\mathcal{A}_{VDC}$:

1. $\mathcal{C}$ runs the $\mathsf{ABE.Setup}$ algorithm on the security parameter and $\mathcal{U}$ to generate MPK_{ABE} and MSK_{ABE}. He gives MPK_{ABE} to $\mathcal{A}_{ABE}$.

2. $\mathcal{A}_{ABE}$ now simulates running $\mathsf{VDC.Setup}$ such that the outcome is consistent with MPK_{ABE}. It initialises the list L_{Reg} and sets $\text{PP} = (MPK_{\text{ABE}}, L_{Reg})$. The master key is implicitly set to be MSK_{ABE}, and will be simulated using oracle queries to $\mathcal{C}$.

3. $\mathcal{A}_{VDC}$ is given PP and oracle access, which $\mathcal{A}_{ABE}$ can handle as follows.

   - $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$ and $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$ can be run as written.
   - $\mathsf{Certify}(\cdot, \cdot, \cdot, \cdot, \text{MK}, \text{PP})$: To generate the evaluation key for a queried data set D_i, $\mathcal{A}_{ABE}$ makes use of the KeyGen oracle in the CP-ABE game. It first updates L_{Reg} as in lines 1–2 of the $\mathsf{Certify}$ algorithm. Then it sets $D' = (A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j}))$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(D', \text{MK}, \text{PK})$. $\mathcal{C}$ shall generate a CP-ABE decryption key $SK_{D'}$ if and only if $D' \notin \mathbb{A}^\star$. However, since at this point $\mathbb{A}^\star$ is still initialised to $\{\emptyset\}$, $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as EK_{D_i,S_i}.

4. Eventually, $\mathcal{A}_{VDC}$ outputs its choice of challenge parameters: the function F and input data $X^\star$ comprising data points with labels $\{l(x_j)\}_{x_j \in X^\star}$.

5. $\mathcal{A}_{ABE}$ runs FnInit as given in the construction. To generate the challenge input, $\mathcal{A}_{ABE}$ begins by choosing a random bit b, three equal length random messages m_0, m_1 and m' from the message space, and another random bit t.

   It must then choose its challenge access structure for the CP-ABE IND-CPA game. It first computes $r = F(X^\star)$. If $r = 0$, it sets $\mathbb{A}^\star = (F \wedge \bigwedge_{x_j \in X^\star} l(x_j))$. Otherwise, it sets $\mathbb{A}^\star = (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j))$. $\mathcal{A}_{ABE}$ sends the messages m_0 and m_1, as well as $\mathbb{A}^\star$ to $\mathcal{C}$ as the challenge parameters for the CP-ABE game. Note that $\mathbb{A}^\star$ is a valid challenge access structure as the only queries made to the KeyGen oracle resulted

from Certify oracle queries to $\mathcal{A}_{ABE}$. Now, due to the inclusion of the (unique) labels $\{l(x_j)\}_{x_j \in X^\star}$ in the challenge access structure, no Certify query for input data points $X' \subset X^\star$, with labels $\{l(x_j)\}_{x_j \in X'} \subset \{l(x_j)\}_{x_j \in X^\star}$, would result in a KeyGen query for attributes that satisfy $\mathbb{A}^\star$.

If queried for a dataset $X' \supseteq X^\star$, observe that $\mathbb{A}^\star$ is 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 c and returns $CT^\star \xleftarrow{\$} \mathsf{Encrypt}(m_c, \mathbb{A}^\star, MPK_{\text{ABE}})$.

- If $r = 1$ (that is, $\mathbb{A}^\star = (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j))$, $\mathcal{A}_{ABE}$ generates

$$c_b \xleftarrow{\$} \mathsf{Encrypt}(m', (F \wedge \bigwedge_{x_j \in X^\star} l(x_j)), MPK_{\text{ABE}})$$

  and sets $c_{1-b} = CT^\star$ (formed over $\mathbb{A}^\star$ by $\mathcal{C}$). It also sets $VK_b = g(m')$ and $VK_{1-b} = g(m_t)$.
- Else $r = 0$, and $\mathcal{A}_{ABE}$ sets $c_b = CT^\star$ and computes

$$c_{1-b} \xleftarrow{\$} \mathsf{Encrypt}(m', (\overline{F} \wedge \bigwedge_{x_j \in X^\star} l(x_j)), MPK_{\text{ABE}}).$$

  It sets $VK_b = g(m_t)$ and $VK_{1-b} = g(m')$.

$\mathcal{A}_{ABE}$ sets $\sigma_{F,X^\star} = (c_b, c_{1-b})$, $VK_{F,X^\star} = (VK_b, VK_{1-b})$ and $RK_{F,X^\star} = b$.

6. $\mathcal{A}_{ABE}$ sends the output from ProbGen along with the public information to $\mathcal{A}_{VDC}$, who is also given oracle access to which $\mathcal{A}_{ABE}$ responds as follows:
   - $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$ and $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$ can be run as written.
   - $\mathsf{Certify}(\cdot, \cdot, \cdot, \cdot, \text{MK}, \text{PP})$: For a query for a dataset D_i, $\mathcal{A}_{ABE}$ first updates L_{Reg} as in lines 1–2 of the Certify algorithm. It sets $D' = (A_{D_i} \cup \bigcup_{x_{i,j} \in D_i} l(x_{i,j}))$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(D', \text{MK}, \text{PK})$.

     $\mathcal{C}$ shall generate a CP-ABE decryption key $SK_{D'}$ for D_i if and only if $D' \notin \mathbb{A}^\star$. By the definition of $\mathbb{A}^\star$ and the uniqueness of data labels, D' will satisfy $\mathbb{A}^\star$ only if $\{l(x_{i,j})\}_{x_{i,j} \in D_i} \supseteq \{l(x_j)\}_{x_j \in X^\star}$, hence only if $X^\star \subseteq D_i$. Now, if $X^\star \subseteq D_i$, then additionally, D_i must satisfy either F or $\overline{F}$ to satisfy $\mathbb{A}^\star$. 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 $D' \notin \mathbb{A}^\star$ and $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as EK_{D_i,S_i}.

7. Eventually, $\mathcal{A}_{VDC}$ outputs $\theta_{F(X^\star)}$, which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $F(X^\star)$).

8. $\mathcal{A}_{ABE}$ parses $\theta_{F(X^\star)}$ as $(d_b, d_{1-b}, S_i, \gamma)$ and using the retrieval key $RK_{F,X^\star} = b$, finds d_0 and d_1. One of d_0 and d_1 will be $\perp$ (by construction) and we denote the other value by Y.

   Observe that, if $\mathcal{A}_{VDC}$ is successful against the selective public verifiability game, the non-$\perp$ value, Y, that it will return the plaintext m_c since the challenge access structure was always set to be unsatisfied on the challenge input.

   Thus, if $g(Y) = g(m_t)$, $\mathcal{A}_{ABE}$ outputs a guess $c' = t$ and otherwise guesses $c' = 1 - t$.

If $t = c$ (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 $g(m_c)$ where m_c is the other plaintext). Alternatively, $t = 1 - c$ 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-c} that is unrelated to the ciphertext).

Now, we consider the advantage of this constructed $\mathcal{A}_{ABE}$ playing the IND-CPA game for CP-ABE. By assumption, $\mathcal{A}_{VDC}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1** :

$$\left|\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right] - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right]\right| \geqslant \delta,$$

where $\mathbf{Exp}^{\mathbf{Game\ i}}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]$ denotes running $\mathcal{A}_{VDC}$ in Game i.

The probability of $\mathcal{A}_{ABE}$ guessing c correctly, by the law of total probability, is:

$$\begin{aligned}
\Pr[c' = c] &= \Pr[t = c]\Pr\left[c' = c | t = c\right] + \Pr[t \neq c]\Pr\left[c' = c | t \neq c\right] \\
&= \frac{1}{2}\Pr\left[g(Y) = g(m_t) | t = c\right] + \frac{1}{2}\Pr\left[g(Y) \neq g(m_t) | t \neq c\right] \\
&= \frac{1}{2}\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{0}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right] + \frac{1}{2}\left(1 - \Pr\left[g(Y) = g(m_t) | t \neq c\right]\right) \\
&= \frac{1}{2}\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{0}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right] + \frac{1}{2}\left(1 - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{1}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right]\right) \\
&= \frac{1}{2}\left(\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{0}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right] - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{1}_{\mathcal{A}_{VDC}}\left[\mathcal{VDC}, 1^\ell, \mathcal{F}\right]\right] + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned} Adv_{\mathcal{A}_{ABE}} &\geqslant \left| \Pr\left[c = c'\right] - \frac{1}{2} \right| \\ &\geqslant \left| \frac{1}{2}(\delta + 1) - \frac{1}{2} \right| \\ &\geqslant \frac{\delta}{2} \end{aligned}$$

Hence, if $\mathcal{A}_{VDC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the IND-CPA game for the CP-ABE scheme with non-negligible probability. Thus, since we assumed the CP-ABE scheme to be secure, we conclude that $\mathcal{A}_{VDC}$ 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, $\mathcal{A}_{VDC}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof. We now show that using $\mathcal{A}_{VDC}$ in **Game 2**, $\mathcal{A}_{ABE}$ 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-b} to be w and set the verification key component $VK_{1-b} = z$. We choose m_b and VK_b randomly as usual.
- if $F(X^\star) = 0$, we implicitly set m_b to be w and set the verification key component $VK_b = z$. We choose m_{1-b} and VK_{1-b} randomly as usual.

$\mathcal{A}_{VDC}$ 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). $\mathcal{A}_{ABE}$ can therefore forward this

result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{VDC}$ has against **Game 2**.

We conclude that if the ABE scheme is IND-CPA secure and the one-way function is hard-to-invert, then $\mathcal{VDC}$ as defined by Algorithms 5.1–5.8 is secure in the sense of public verifiability. □

5.8 Conclusion

This chapter has shown that CP-ABE can be used in the setting of verifiable outsourced computation in a similar fashion to the use of KP-ABE to construct PVC schemes. The resulting system model reverses the role of the delegator and the server as the data owner, in a similar way that the assignment of attribute sets changes from the key to ciphertexts in CP-ABE compared to KP-ABE. In PVC, the delegator is the data owner and employs a server to perform work that it does not have the resources to do itself. In VDC, the server itself acts as the data owner (or is provided with data from a trusted source), and clients simply request the verifiably correct results of computations on the data held by each server.

We have seen that VDC can have natural applications and is also similar to, although arguably more general than, existing models of verifiable computation developed by Backes et al. [17]. Our VDC construction, however, uses attribute-based encryption as a proof method, whereas Backes et al. considered the use of succinct non-interactive arguments (SNARGs) and homomorphic MACs. In future work, we would like to compare these solutions more closely and investigate whether ABE can fully provide the functionality of SNARGs, as well as to investigate methods to allow dynamic updates of the data owned by servers as a result of computations.

Future work could also look at imposing cryptographic access control mechanisms on the queries that may be made to a server. It may be that by including some cryptographic key material in the set $\mathcal{F}_i$, a delegator would be unable to form a valid computation request for $F(X)$ if both F and X are not permissible according to $\mathcal{F}_i$. In this regard, the application to querying on remote databases could lead to interesting solutions when considering sensitive data — only computations that retain a certain degree of anonymity

are permitted. Of course, even with the solution presented in this chapter, a server can manually validate any requested function to ensure it complies with its own access control policy.

The motivation for this chapter was primarily to extend the work of Chapter 3 to use an alternative ABE primitive and to see whether the resulting solution yielded meaningful applications of verifiable outsourced computation. We have seen that, indeed, it does, and in Chapter 6 we will combine this notion of VDC with RPVC to provide a flexible solution to verifiable outsourced computation, again using a different form of attribute-based encryption as the proof mechanism.

Chapter 6

Hybrid Publicly Verifiable Outsourced Computation

Contents

This chapter draws together the work from the previous chapters of this thesis, to provide a flexible system for publicly verifiable computation. In large multi-user systems, individual user requirements may be diverse and change over time. We therefore introduce HPVC which, with a single set up, allows entities to both outsource computations and provide computation services as required.

6.1 Introduction

In Chapter 3, we extended the PVC construction of Parno et al. [84] to create a revocable, publicly verifiable outsourced computation (RPVC) scheme built on KP-ABE. In Chapter 4, we motivated the need for access control in multi-user PVC environments and provided a construction using key assignment schemes. Finally, in Chapter 5, we switched the attribute-based encryption scheme, which acts as the proof mechanism, from KP-ABE to CP-ABE and saw that it yielded a verifiable computation system wherein servers hold data and make it available for verifiable, public querying. Thus, we have seen that cryptographic primitives designed to act as enforcement mechanisms for access control policies can be used, instead, in the context of verifiable computation. The VC systems that arise

can be viewed as large, multi-user systems comprising many servers and many delegators. However, in such systems, the individual user requirements may be diverse and require different forms of outsourced computations whereas current PVC schemes support only a single form.

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 dataset that is of interest to other parties, and hence want to to selectively share this information by allowing others to query the dataset 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 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. In this chapter, we introduce *Hybrid PVC* (HPVC) which is a single mechanism (with the associated costs of a single setup operation and a single set of system parameters to publish and maintain) which simultaneously satisfies all of the above requirements. Entities may play the role of both delegators and servers, in the following modes of operation, dynamically as required:

• **Revocable PVC** (RPVC) where clients with limited resources outsource computations on data of their choosing to more powerful, untrusted servers using only public information. Multiple servers can compute multiple 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;

• **RPVC with access control** (PVC-AC) which restricts the servers that may perform a given computation. Outsourced computations may be distributed amongst a pool of available servers that are not individually authenticated and known by the delegator. In Chapter 4, we used symmetric primitives and required all entities to be registered in the system (including delegators) but here we achieve a fully public system where only servers need be registered (as usual in PVC);

• **Verifiable Delegable Computation** (VDC) where servers are the data owners and make a static dataset available for verifiable querying. Clients request computations on subsets of the dataset using public, descriptive labels.

Continuing our investigation of the use of attribute-based encryption in verifiable computation settings, we consider a third form of ABE, namely *dual-policy attribute-based encryption* [16] (see Section 2.6.3). We see that HPVC provides a natural application for DP-ABE, which combines both KP- and CP-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 cryptographically enforced access control (where objects are encrypted and can only be read by users holding keys satisfying some decryption policy), it is unclear that the policies enforced by DP-ABE are natural choices for access control. Thus an interesting side-effect of this work is to show that additional applications for DP-ABE exist. In independent and concurrent work, Shi et al. [90] considered a similar use of DP-ABE to combine keyword search on encrypted data with the enforcement of an access control policy.

As DP-ABE combines both KP- and CP-ABE, it is not so surprising that it is possible to use DP-ABE to capture RPVC and VDC. However, we observe that this does not use the full power of DP-ABE; instead, it merely uses the ability of DP-ABE to separately enforce key and ciphertext policies. We show that, by using both forms of policy *simultaneously*, we can enforce access control policies in a similar fashion to Chapter 4 (where the data owner specifies the policy).

In Section 6.2, formally introduce our hybrid model of publicly verifiable outsourced computation. As the notions of security for HPVC primarily combine the games seen in the previous chapters, we defer them to Appendix B.2. In Section 6.3.2, we provide a construction of an HPVC scheme. To implement the RPVC functionality, we first must develop a new form of DP-ABE that includes the ability to revoke entities and prevent further decryptions. Hence, in Section 6.3.1, we combine the revocation techniques from the indirectly revocable KP-ABE scheme [14] we used in Chapter 3 with the DP-ABE scheme of Attrapadung and Imai (see Section 2.6.3). We define the primitive and security model in Section 6.3.1, and give a concrete construction and proof of security in Appendix B.1. Finally, we proof our HPVC scheme secure in Appendix B.3.

6.2 Hybrid Publicly Verifiable Computation

We now define our umbrella framework of *hybrid publicly verifiable computation* (HPVC). This is a single system (with the associated costs of a single setup operation and system parameters) that supports multiple modes of operation. Thus, a single KDC may initialise an HPVC system that provides functionality for many users with diverse requirements. Our construction in Section 6.3.2 is based on a novel use of DP-ABE. The key observation is that DP-ABE can, using special attribute tokens, implement KP-ABE, CP-ABE or DP-ABE policies. In more detail, we capture the following functionality:

- **RPVC:** uses objective (KP-ABE) policies only to achieve RPVC as introduced in Chapter 3. Clients with limited resources available may outsource computations to more powerful entities within the system and receive verifiably correct results. Multiple servers can enrol within the system and each may evaluate multiple functions. Entities that misbehave and return incorrect computational results can be detected and prevented from further operating within the system;
- **RPVC with access control:** uses both subjective (CP-ABE) and objective (KP-ABE) policies to achieve RPVC with restrictions on the servers that may perform a computation. This can be seen as an alternative to PVC-AC introduced in Chapter 4. Within a large system comprising many servers, delegators may have limited knowledge about the server selected to perform a given computation. The set of servers that may evaluate a given computation can therefore be restricted based on

factors such as sensitivity of the input data, physical server location etc. The construction in Chapter 4, although able to enforce a wider range of policies, required all entities to be registered in the system (including delegators); with HPVC, we achieve a fully public system where only servers need be registered (as required in usual publicly verifiable outsourced computation);

- **VDC:** uses subjective (CP-ABE) policies only to allow entities to make a dataset available over which other entities may request particular computations (or queries) be performed, as discussed in Chapter 5.

Note that within an HPVC system, entities may play the role of both delegators and servers as required. This provides a flexible solution to interactions between different entities within a large system or organisation where individual workflows may require different forms of outsourced computation.

6.2.1 Informal Overview

An HPVC scheme for a family of functions $\mathcal{F}$ begins with a trusted key distribution centre (KDC) (e.g. a trusted third party) setting up the system by producing public parameters and a master secret key. For each function of interest F, the KDC provides a public delegation key PK_F. Next, the KDC registers each entity S_i, that wishes to act as a server, by deriving a private signing key SK_{S_i}.

The Certify algorithm generates an evaluation key $EK_{(\mathbb{O},\psi),S_i}$ that enables a server S_i to evaluate computations either of the function $\mathbb{O}$ or on the input data ψ, depending on the mode in which the algorithm is run. The server S_i is able to choose a set of labels L_i — in RPVC or PVC-AC modes L_i will uniquely represent the function F that the server is certified to compute; in VDC mode on the other hand, L_i will be a set of labels, each representing a data point contained in the server's dataset D_i. S_i also provides a list of computations $\mathcal{F}_i$ he is willing to evaluate.

A delegator runs ProbGen to make a request for the computation of $F(X)$ from a server S_i; again the input values depend on the mode. The delegator provides a set of labels $L_{F,X} \subseteq L_i$. In RPVC or PVC-AC modes, $L_{F,X}$ simply represents the function F that should be computed on the provided inputs; in VDC mode, $L_{F,X}$ represents the data points

Table 6.1: Parameter definitions for different modes

mode	$\mathbb{O}$	ψ	ω	$\mathbb{S}$
RPVC	F	$\{T_S\}$	X	$\{\{T_S\}\}$
VDC	$\{\{T_O\}\}$	D_i	$\{T_O\}$	F
PVC-AC	F	s	X	P

mode	L_i	$L_{F,X}$	$\mathcal{F}_i$
RPVC	$\{l(F)\}$	$\{l(F)\}$	$\{F\}$
VDC	$\{l(x_{i,j})\}_{x_{i,j}\in D_i}$	$\{l(x_{i,j})\}_{x_{i,j}\in X}$	$\{F, \bigcup_{x_{i,j}\in \mathsf{Dom}(F)} l(x_{i,j})\}_{i=1}^{m}$
PVC-AC	$\{l(F)\}$	$\{l(F)\}$	$\{F\}$

$X \subseteq D_i, X \subseteq \mathsf{Dom}(F)$ held by the server that should be computed on. The algorithm generates an encoded input $\sigma_{F,X}$, a public verification key $VK_{F,X}$ and an output retrieval key $RK_{F,X}$.

A server may use the encoded input with its evaluation key $EK_{(\mathbb{O},\psi),S_i}$ to compute an output $\theta_{F(X)}$ encoding the computational result $F(X)$. Verification comprises two steps. Blind verification is performed by *any* user using the verification key $VK_{F,X}$ to verify correctness of the result *without* learning the output value, and generates an output retrieval token $RT_{F(X)}$. The algorithm also generates a token $\tau_{F(X)}$ indicating the correctness and the server ID. If verification failed, this token is sent to the KDC and the server is revoked from performing further evaluations and hence incurs a penalty. Otherwise, $RT_{F(X)}$ can be used in the Retrieve algorithm to reveal the final result $y_{F(X)} = F(X)$.

6.2.2 Supporting Different Modes

We define HPVC generically in terms of objective (RPVC) and subjective (VDC) policies ($\mathbb{O}$ and $\mathbb{S}$) with corresponding attribute sets (ω and ψ respectively). The values of these parameters depend upon the mode in which an algorithm is being run, and are detailed in Table 6.1. We define two additional parameters T_O and T_S. The meaning of these will become clearer in Section 6.3.2; for now, it suffices to think of them as dummy objects where $\{T_S\} \in \{\{T_S\}\}$ and so the subjective function $\mathbb{S} = \{\{T_S\}\}$ is always trivially satisfied by the input $\psi = \{T_S\}$, and similarly for T_O. These parameter choices will be used to "disable" the functionality of a mode if not required.

6.2.2.1 RPVC

For RPVC functionality, only the objective access structure $\mathbb{O}$ and the objective attribute set ω are required, and are set to be F and the input X respectively to outsource the computation of $F(X)$; in this context, X is a set comprising a single element, corresponding to the single input data point x in Chapter 3. The unneeded subjective parameters $\mathbb{S}$ and ψ are defined in terms of the dummy parameter T_S such that $\mathbb{S}$ is trivially satisfied. The set of functions $\mathcal{F}_i$ that a server is certified for during a single run of the $\mathsf{Certify}$ algorithm is simply the function F, and the sets of labels L_i and $L_{F,X}$ both comprise a single element labelling F. Note that in this setting, the label $l(F)$ corresponds to the bijective mapping $\phi : \mathcal{F} \to \mathcal{U}_{\mathcal{F}}$ defined in Section 3.5.1.2.

6.2.2.2 VDC

For VDC functionality, on the other hand, only the subjective policy $\mathbb{S}$ and attribute set ψ are required. $\mathbb{S}$ is set to represent a function F that is queried to a server, whilst ψ is defined to be the dataset D_i held by the server S_i comprising m_i data points — that is, $D_i = \{x_{i,j}\}_{j=1}^{m_i}$. The remaining objective parameters are defined in terms of the dummy parameter T_O to be trivially satisfied. The set of functions $\mathcal{F}_i$ is defined to be a set of functions that the server is willing to compute on behalf of queriers along with the labels of each permissible input to the function (which is given as input to the $\mathsf{Certify}$ algorithm). Finally, the set of labels L_i comprise a data label labelling each data point $x_{i,j} \in D_i$ held by the server (that is, all data points in ψ), whilst the set of labels $L_{F,X} \subseteq L_i$ represent data points that a particular computation should be performed over (i.e. the input $X \subseteq D_i, X \subseteq \mathsf{Dom}(F)$ to the computation $F(X)$).

6.2.2.3 PVC-AC

The above parameter choices allow us to define a single HPVC system to support both RPVC and VDC functionality. However, we can also use *both* sets of parameters (objective *and* subjective) simultaneously to achieve a form of PVC-AC such that only servers that meet an authorisation policy attached to the encoded input may produce valid (acceptable) results (and hence be rewarded for their effort).

Recall that, in Chapter 4, we introduced PVC-AC with the motivation that servers could be selected from a (large) pool of available servers based on a system-dependent mechanism (e.g. resource availability or a bidding process). (This contrasts with prior models [84] where a client chose a server up-front with which to set up a PVC system.) Thus, in this setting, delegators have less knowledge about the selected server and may not authenticate them beforehand. The PVC-AC construction in Chapter 4 used symmetric encryption and key assignment schemes as the cryptographic enforcement mechanisms, such that only authorised entities could derive decryption keys. However, delegators and verifiers had to be registered by the KDC. This is partly due to the policies being enforced (e.g. such that delegators may outsource only certain computations) but also due to the use of symmetric primitives — to encrypt an input that only authorised servers can decrypt, delegators must be privately issued the symmetric key. Thus, the scheme is not strictly *publicly delegable* — any entity may be registered as a delegator but delegation does not depend *only* on public information, and similarly for verification.

Here, we give a more relaxed framework that retains public verifiability and public delegability, but for a limited class of access control policies placing restrictions only on the set of servers that may compute a given outsourced computation. In some sense, servers are already authorised for functions by virtue of being issued evaluation keys. However, we believe not all outsourced computations should be considered purely in terms of functions and that access control policies in this setting should allow for additional context. For example, a government contractor that subscribes to a verifiable software-as-a-service system may, due to the nature of its work, require that servers be physically located within the same country. Alternatively, as discussed in Chapter 4, the semantic meaning of input data may affect the access control requirements.

Informally, to define PVC-AC in the context of an HPVC system, we use the objective policy to evaluate an outsourced computation (as in Section 6.2.2.1 for RPVC) whilst the subjective policy will additionally be used to enforce access control on the server. Servers are assigned both an evaluation key for a function F *and* a set of descriptive attributes describing their authorisation rights, $s \subseteq \mathcal{U}_S$, where $\mathcal{U}_S$ is a universe of attributes used solely to define authorisation — in the terminology of Chapter 4, $\lambda^C(S) = (F, s)$. ProbGen operates on both the input data X *and* an authorisation policy $P \subseteq 2^{\mathcal{U}_S} \setminus \{\emptyset\}$ which dictates the sets of necessary authorisation attributes to perform this computation — again, in the terminology of Chapter 4, $\lambda^C(o) = (X, P)$ for the computation

o. A server may produce a valid output that is accepted by the Verify algorithms if and only if $s \in P$ — that is, the server satisfies the authorisation policy. For example, s may be $\{\texttt{Country} = \texttt{UK}, \texttt{Capacity} = \texttt{3TB}\}$ to describe the location and resources of the server, while $P = (\texttt{Country} = \texttt{UK}) \vee ((\texttt{clearance} = \texttt{Secret}) \wedge (\texttt{Country} = \texttt{USA})) = \{\{\texttt{UK}\},\{\texttt{clearance} = \texttt{Secret}, \texttt{Country} = \texttt{USA}\}\}$ defines a policy dictating the necessary clearances for servers in different locations.

It is interesting to note that by discussing DP-ABE policies in terms of a labelling function, it becomes clear that this is *not* the same as the dual-policy graph-based access control policies discussed by Crampton [44]. Here the range of the labelling function is $2^{2^{\mathcal{U}_C}} \times 2^{2^{\mathcal{U}_S}}$ (where $\mathcal{U}_S$ is the universe of attributes used to describe access control policies, as discussed above, and $\mathcal{U}_C$ comprises attributes from which functions and computational inputs can be formed), whilst Crampton used the range $2^{2^{U_C \times U_S}}$. There, instead of assigning to each entity an access structure and an attribute set, each entity is instead given just a policy and access is granted if and only if there exists an attribute set that satisfies both policies.

In the VDC setting, we could allow the subjective policy to specify the function F whilst the objective policy contains the authorisation policy P. Then, delegators may request a computation only if they can provide certain attributes (e.g. those authorised to learn the computational results over sensitive data). As this mechanism is symmetric to that for PVC-AC, we do not discuss it in detail here.

6.2.3 Formal Definition

Definition 6.1. A *hybrid publicly verifiable computation (HPVC) scheme* for a family of functions $\mathcal{F}$ comprises the following algorithms:

1. $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$: Run by the KDC to establish public parameters PP and a master secret key MK for the system. The inputs are the security parameter and the family of functions $\mathcal{F}$ that may be computed;

2. $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$: Run by the KDC to generate a public delegation key, PK_F, allowing entities to outsource or request computations of F;

3. $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \mathrm{MK}, \mathrm{PP})$: 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. $EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate an evaluation key $EK_{(\mathbb{O},\psi),S_i}$ enabling the server S_i to compute on the pair $(\mathbb{O}, \psi)$. If mode is VDC then ψ is a dataset owned by S_i comprising m_i data points. Each data point $x_{i,j} \in \psi$ is uniquely represented by a label in the set $L_i = \{l(x_{i,j})\}_{x_{i,j} \in \psi}$. $\mathcal{F}_i$ is a set of functions that S_i is willing to compute on subsets of ψ. Otherwise, $\mathbb{O}$ represents a function F, the set of labels L_i comprises just a single element $l(F)$ representing F, and $\mathcal{F}_i = \{F\}$;

5. $(\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, PK_F, \mathrm{PP})$: run by an entity to request a computation of $F(X)$ from a server S_i.

   The inputs are the mode (RPVC, PVC-AC or VDC), the pair $(\omega, \mathbb{S})$ representing the computation request, a set of labels $L_{F,X} \subseteq L_i$, the public delegation key for F and the public parameters.

   If mode is VDC then the labels $L_{F,X} = \{l(x_j)\}_{x_{i,j} \in X} \subseteq L_i$ represent the data items $X \subseteq D_i, X \subseteq \mathsf{Dom}(F)$ held by a server S_i which form the inputs to F. Otherwise, the set of labels comprises a single label labelling F which should be computed on the provided inputs ω. The outputs of this algorithm are an encoded input $\sigma_{F,X}$, a verification key $VK_{F,X}$ and an output retrieval key $RK_{F,X}$;

6. $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \mathrm{PP})$: run by an entity S_i to perform a computation. The inputs are the mode (RPVC, PVC-AC or VDC), an encoded input, an evaluation key and signing key for S_i and the public parameters. The output is $\theta_{F(X)}$ which encodes the result of the computation;

7. $y_{F(X)} \leftarrow \mathsf{Verify}(\theta_{F(X)}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$: verification consists of two steps.

   - $(RT_{F(X)}, \tau_{F(X)}) \leftarrow \mathsf{BVerif}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$: run by any verifier possessing an encoded output and verification key for a computation, and the public parameters. It produces a retrieval token $RT_{F(X)}$ encoding the actual output of the computation, and a token $\tau_{F(X)}$ which is (accept, S_i) if $\theta_{F(X)}$ is a correct result computed by the entity S_i, or (reject, S_i) if S_i misbehaved;
   - $y \leftarrow \mathsf{Retrieve}(RT_{F(X)}, \tau_{F(X)}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$: run by a verifier holding the outputs from BVerif, the verification key and retrieval key for the computation

of $F(X)$ and the public parameters. This algorithm produces a result $y = F(X)$ if the result was computed correctly, or $y = \perp$ otherwise;

8. $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{F(X)}, \mathrm{MK}, \mathrm{PP})$: run by the KDC if a verifier reports a misbehaving server i.e. that Verify returned $\tau_{F(X)} = (\mathsf{reject}, S_i)$. If the algorithm is run with $\tau_{F(X)} = (\mathsf{accept}, S_i)$ then it returns $UM = \perp$ as no entity is to be revoked. Otherwise, all evaluation keys $EK_{(\cdot,\cdot),S_i}$ for the server S_i are rendered non-functional. The update material UM is a set of updated evaluation keys $\{EK_{(\mathbb{O},\psi),S'}\}$ which are issued to all servers.

The KDC may additionally update the public parameters PP during any algorithm to reflect any changes in the user population.

We say that an HPVC scheme is *correct* if, when all algorithms are run honestly in any order and the result is computed by a non-revoked server, the result is correct and the verification algorithm accepts the result. We can model this as a cryptographic game between a challenger and a PPT adversary; the adversary aims to find an encoded output (generated honestly by a non-revoked server) which either does not encode the correct result, or which does encode the correct result yet which will not be accepted by the verification algorithm.

The adversary is given access to a set of oracles; for each algorithm in Definition 6.1, we define a corresponding oracle which executes the corresponding algorithm on arguments provided by the adversary, and returns the output of the algorithm to the adversary. The adversary may query the Setup oracle only once (before making any other oracle queries), but can thereon call the remaining oracles any number of times and in any order.

The challenger maintains two lists, L_{Reg} and L_F. L_{Reg} is a list of tuples comprising server identities, S_i, and the resulting signing keys, SK_{S_i}, that have been queried to the $\mathsf{Register}$ oracle. L_F comprises tuples of the form $(S_i, \psi, L_i, \mathcal{F}_i, EK_{(\mathbb{O},\psi),S_i})$ denoting that the server S_i has been queried to the $\mathsf{Certify}$ oracle for the set of functions $\mathcal{F}_i$ and that $EK_{(\mathbb{O},\psi),S_i}$ was generated. When the adversary makes a Revoke query with a revocation token that identifies a server S_i to be revoked (that is, if $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ is given as input to the Revoke oracle), the challenger removes all entries of the form $(S_i, \cdot, \cdot, \cdot, \cdot, \cdot)$ (i.e. all entries for S_i for any function) from L_F.

The challenger also creates and maintains a table T which records the parameters and values relating to each computation performed through the oracle queries. T is updated in the following oracles:

- ProbGen: the challenger creates a new row in T comprising 8 components, all of which are initialised to be empty; it then assigns X (which is either given explicitly in ω or can be found by searching L_F for the labels $L_{F,X}$), F, the result $F(x)$ (computed by the challenger itself), $\sigma_{F,X}, VK_{F,X}$ and $RK_{F,X}$ to the first 6 components;

- Compute: the challenger first searches T for all rows that contain the queried $\sigma_{F,X}$ in the 4^{th} component and where the 7^{th} component is empty (i.e. those rows relating to computations on this encoded input that have not yet been performed). For each such row, r, the challenger takes the second component (the function identifier, $\tilde{F}$), and checks that there exists a server identity $\tilde{S}_i$ such that the tuple $(\tilde{S}_i, SK_{\tilde{S}_i}) \in L_{\text{Reg}}$ (where $SK_{\tilde{S}_i}$ is that given as input to the Compute oracle) *and* such that the tuple $(\tilde{S}_i, \cdot, \cdot, \mathcal{F}_i, EK_{(\mathbb{O},\psi),S_i}) \in L_F$ (where $EK_{(\mathbb{O},\psi),S_i}$ is also that given as input to the Compute oracle) and where $\tilde{F} \in \mathcal{F}_i$. This check ensures that there is a currently un-revoked server (as the entries of L_F for $\tilde{S}_i$ have not been removed) that holds the signing key and evaluation key being used to perform the computation and which is certified for a function $\tilde{F}$ for which the encoded input $\sigma_{F,X}$ was generated.

  The challenger then performs the Compute algorithm on the queried $\sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}$ and SK_{S_i} to produce an output $\theta_{F(X)}$. For each of the rows r of T found above, the challenger writes $\theta_{F(X)}$ and $\tilde{S}_i$ to the 7^{th} and 8^{th} components of r respectively. Thus, a row of T will only have a (non-empty) value in the 7^{th} component if there exists a non-revoked, certified server to perform the computation for which $\sigma_{F,X}$ was generated.

Thus, when complete, the entries of T will be of the form

$$(X, F, F(X), \sigma_{F,X}, VK_{F,X}, RK_{F,X}, \theta_{F(X)}, S).$$

After a polynomial number of queries, the adversary will return a value $\theta^{\star}_{F(X)}$ which he believes either encodes an incorrect computational result or which encodes a correct computational result yet which the Verify algorithm will reject (that is, an output for which the protocol execution will not be correct). The challenger first performs a look

up in T for all entries containing $\theta^{\star}_{F(X)}$ in the 7^{th} position of the tuple, and stores any such entries as another table $\tilde{T}$. Note that this means that $\theta^{\star}_{F(X)}$ must have been honestly generated by the Compute oracle (else it would not be in T).

For each such row, the challenger uses the 5^{th} component (the verification key) to run BVerif on $\theta^{\star}_{F(X)}$ to generate the outputs $RT_{F(X)}$ and $\tau_{\theta_{F(X)}}$, and then uses the 5^{th} and 6^{th} components (the verification key and retrieval key) and $\tau_{\theta_{F(X)}}$ to run Retrieve on $\theta^{\star}_{F(X)}$ to generate y.

The challenger first checks whether y matches the 3^{rd} component of the row (that is, whether y is the correct computational result $F(X)$). If so, it then checks whether $\tau_{\theta_{F(X)}} = (\mathsf{reject}, S_i)$, and if so it ends the game by returning 1 to indicate that the adversary has won the game (the adversary has found a valid encoding of a correct result, computed by a certified, non-revoked server, that the Verify algorithm is incorrectly rejecting).

On the other hand, if y did *not* match the correct value of $F(X)$, the challenger also ends the game by returning 1 to indicate that the adversary has won the game (the adversary in this case has found an incorrect result that was computed honestly by the algorithms).

If no row in $\tilde{T}$ allows the adversary to win, then the challenger outputs 0 to indicate that the adversary has lost. An HPVC scheme is *correct* if, for all PPT adversaries, the probability that the adversary wins the game described above is 0.

6.3 Construction

Before giving our construction of HPVC, we must first introduce a new cryptographic primitive which will form the basic building block of our construction.

6.3.1 Revocable Dual-policy Attribute-based Encryption

In Section 2.6.3 we reviewed the notation and properties of *dual-policy ABE*, introduced by Attrapadung and Imai [16], that conjunctively combines KP-ABE and CP-ABE such that both the decryption key *and* the ciphertext comprise an attribute set and an access

structure. As discussed in Section 3.2.2, Attrapadung and Imai [14] introduced the formal notion of revocation in ABE schemes supporting two different modes: *direct revocation* and *indirect revocation*. Direct revocation allows users to specify a revocation list at the point of encryption such that periodic re-keying is not required but encryptors must have knowledge of the current revocation list. In contrast, indirect revocation requires a time period to be specified at the point of encryption and an authority to issue updated key material at each time period to enable non-revoked entities to update their key to be functional during that time period. As in Chapter 3, with the setting of verifiable outsourced computations in mind, we choose to focus on *indirect* revocation to minimise the workload of the client devices in terms of maintaining synchronised revocation lists.

To implement a revocation mechanism in the KP-ABE setting, Attrapadung and Imai amended the policy of the ABE scheme to include an identifier of the entity owning the key, and then embedded the current time period into the ciphertext. Update keys were issued for all non-revoked identities at each time period which were used in combination with the decryption key to decrypt ciphertexts formed for particular time periods — only if the entity was issued an update key for time t (i.e. was not revoked) could they decrypt ciphertexts formed using t. We observe that, to define a *revocable DP-ABE* scheme, the revocation mechanism can be embedded either, as above, in the KP-ABE functionality *or* in the CP-ABE functionality. In more detail, decryption in DP-ABE is successful if and only if *both* attribute sets satisfy their corresponding access structure. In order to prevent decryption, therefore, at least one attribute set should not satisfy the corresponding access structure. Here we present a formal definition of revocable DP-ABE using indirect revocation in the key-policy. It will be the subject of future work to compare the efficiency of the two approaches.

We refer to the access structure associated to a decryption key as an *objective* access structure, denoted $\mathbb{O}$, and the attribute set associated to the ciphertext as an *objective* attribute set, denoted ω; these are associated with the KP-ABE functionality. Similarly, the access structure associated to a ciphertext is referred to as a *subjective* access structure, denoted $\mathbb{S}$, and the attribute set associated to the key as an *subjective* attribute set, denoted ψ; these are associated with the CP-ABE functionality.

Definition 6.2. A *revocable key dual-policy attribute-based encryption scheme* (rkD-PABE) comprises five algorithms:

- $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U})$: takes the security parameter and the attribute universe as input and generates public parameters PP for the system and a master secret key MK;

- $CT_{(\omega,\mathbb{S}),t} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, \mathrm{PP})$: takes as input a message to be encrypted, an objective attribute set ω, a subjective policy $\mathbb{S}$, a time period t and the public parameters. It outputs a ciphertext that is valid for time t;

- $SK_{(\mathbb{O},\psi),\mathrm{ID}} \xleftarrow{\$} \mathsf{KeyGen}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PP})$: takes an identity ID, an objective access structure $\mathbb{O}$, a subjective attribute set ψ, as well as the master secret key and the public parameters. It outputs a secret decryption key $SK_{(\mathbb{O},\psi),\mathrm{ID}}$;

- $UK_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP})$: takes a revocation list R that contains the identities of revoked entities, the current time period, as well as the master secret key and public parameters. It outputs updated key material $UK_{R,t}$ which makes the decryption keys $SK_{(\mathbb{O},\psi),\mathrm{ID}}$, for all non-revoked identities $\mathrm{ID} \notin \mathrm{R}$, functional to be able to decrypt ciphertexts encrypted for the time t.

- $PT \leftarrow \mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), SK_{(\mathbb{O},\psi),\mathrm{ID}}, (\mathbb{O}, \psi), UK_{R,t}, \mathrm{PP})$: takes as input a ciphertext formed for the time period t and the associated pair (ω,$\mathbb{S}$) comprising an objective attribute set and a subjective access structure, a decryption key for entity ID and the associated objective access structure $\mathbb{O}$ and subjective attribute set ψ, an update key for the time t and the public parameters.

  It outputs a plaintext PT which is the encrypted message m, if and only if the objective attributes ω satisfies the objective access structure $\mathbb{O}$ *and* the subjective attributes ψ satisfies the subjective policy $\mathbb{S}$ *and* the value of t in the update key matches that specified during encryption. If not, PT is set to be a distinguished failure symbol $\perp$.

Definition 6.3. *An rkDPABE scheme is* correct *if for all messages $m \in \mathcal{M}$, for all access*

Game 6.1 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{RKDPABE}, 1^{\ell}, \mathcal{U}\right]$

1: $(t^{\star}, (\omega^{\star}, \mathbb{S}^{\star})) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{U})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U})$
3: $\bar{R} \xleftarrow{\$} \mathcal{A}(\text{PP})$
4: $(m_0, m_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,(\cdot,\cdot),\text{MK},\text{PP})\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,\text{MK},\text{PP})}(\text{PP})$
5: **if** $(|m_0| \neq |m_1|)$ **then return** 0
6: $b \xleftarrow{\$} \{0,1\}$
7: $CT^{\star} \xleftarrow{\$} \mathsf{Encrypt}(m_b, (\omega^{\star}, \mathbb{S}^{\star}), t^{\star}, \text{PP})$
8: $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,(\cdot,\cdot),\text{MK},\text{PP})\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,\text{MK},\text{PP})}(CT^{\star}, \text{PP})$
9: **return** $b' = b$

Oracle 6.1 $\mathcal{O}^{\mathsf{KeyGen}}(\text{ID}, (\mathbb{O}, \psi), \text{MK}, \text{PP})$:

1: **if** $((\omega^{\star} \in \mathbb{O})$ **and** $(\psi \in \mathbb{S}^{\star})$ **and** $(\text{ID} \notin \tilde{\text{R}}))$ **then return** $\perp$
2: **return** $\mathsf{KeyGen}(\text{ID}, (\mathbb{O}, \psi), \text{MK}, \text{PP})$

Oracle 6.2 $\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, \text{MK}, \text{PP})$:

1: **if** $((t = t^{\star})$ **and** $(\tilde{R} \not\subseteq R))$ **then return** $\perp$
2: **return** $\mathsf{KeyUpdate}(R, t, \text{MK}, \text{PP})$

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* $\psi \in \mathbb{S}$*,*

$$
\begin{aligned}
&\Pr[(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{U}),\\
&\quad SK_{(\mathbb{O},\psi),\text{ID}} \xleftarrow{\$} \mathsf{KeyGen}(\text{ID}, (\mathbb{O}, \psi), \text{MK}, \text{PP}),\\
&\quad CT_{(\omega,\mathbb{S}),t} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, \text{PP}),\\
&\quad UK_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, \text{MK}, \text{PP}),\\
&\quad m \leftarrow \mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), SK_{(\mathbb{O},\psi),\text{ID}}, (\mathbb{O}, \psi), UK_{R,t}, \text{PP})]\\
&\quad = 1 - \text{negl}(\ell).
\end{aligned}
$$

The security model for rkDPABE is a natural extension of the IND-sHRSS game for an indirectly revocable KP-ABE scheme (see Section 3.2.2), and is presented in Game 6.1 and Oracles 6.1 and 6.2.

Definition 6.4. *The* advantage *of a PPT adversary* $\mathcal{A}$ *in the* IND-sHRSS *game for an rkDPABE construction* $\mathcal{RKDPABE}$ *is defined as:*

$$
Adv_{\mathcal{A}}^{\text{IND-sHRSS}}(\mathcal{RKDPABE}, 1^{\ell}, \mathcal{U}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{RKDPABE}, 1^{\ell}, \mathcal{U}\right]\right] - \tfrac{1}{2}.
$$

An rkDPABE scheme is secure in the sense of indistinguishability against selective-target

with semi-static query attack (IND-sHRSS) *if for all PPT adversaries* $\mathcal{A}$*,*

$$Adv_{\mathcal{A}}^{\text{IND-sHRSS}}(\mathcal{RKDPABE}, 1^{\ell}, \mathcal{U}) \leqslant \text{negl}(\ell).$$

In the remainder of this chapter, we shall use an rkDPABE scheme in a black-box manner; therefore to comprehend the rest of the chapter, it is sufficient to refer to Definitions 6.2 and 6.4 only. For completeness, we provide a construction and security proof for an rkDPABE scheme in Appendix B.1.

6.3.2 Instantiation of HPVC

We construct a HPVC scheme for a family $\mathcal{F}$ of monotone Boolean formulas closed under complement using a revocable key dual-policy ABE in a black-box manner. Consider a function to be delegated $F : \{0,1\}^n \to \{0,1\}$ and its complement function $\overline{F} = F(x) \oplus 1$. As in Section 5.6, n-bit binary input strings x are encoded as attribute sets $A_x \subseteq \mathcal{U}_x$. Let $\mathcal{U}_l$ be a set of attributes (disjoint from $\mathcal{U}_x$) that uniquely label each function and each data item, and let $\mathcal{U}_{ID}$ represent server identities. Let g be a one-way function and $\mathcal{DPABE} =$ (DPABE.Setup, DPABE.Encrypt, DPABE.KeyGen, DPABE.KeyUpdate, DPABE.Decrypt) be a revocable key DP-ABE scheme for $\mathcal{F}$ (see Section 6.3.1) with attribute universe $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{ID} \cup T_O \cup T_S$.

T_O and T_S allow a DP-ABE scheme to efficiently function as either KP-ABE or CP-ABE [15]. For KP-ABE, the subjective policy $\mathbb{S} = \{\{T_S\}\}$ is satisfied by the presence in ψ of the special attribute T_S — thus, $\mathbb{S}$ is always trivially satisfied and decryption only depends on the objective attributes and policy. Similarly, for CP-ABE, $\omega = \{T_O\}$ and $\mathbb{O} = \{\{T_O\}\}$. We will initialise two independent DP-ABE systems over $\mathcal{U}$. Hence, we define a total of four additional attributes: T_O^0, T_S^0 relating to the first DP-ABE system, and T_O^1, T_S^1 for the second DP-ABE system. We denote the complement functions in different modes as follows: In RPVC and PVC-AC, $\mathbb{O} = F$ and $\mathbb{S} = \{\{T_S^0\}\}$; we define $\overline{\mathbb{O}} = \overline{F}$ and $\bar{\mathbb{S}} = \{\{T_S^1\}\}$. Similarly, for VDC, $\overline{\mathbb{O}} = \{\{T_0^1\}\}$ and $\bar{\mathbb{S}} = \overline{F}$.

Each mode operates by encrypting a pair of random messages and issuing keys such that the recovery of one message implies whether the ciphertext was linked to F or $\overline{F}$, and hence if $F(X) = 1$ or 0. Ciphertext indistinguishability ensures an adversary cannot cheat by

returning the other message.

1. Setup, presented in Algorithm 6.1, initialises two revocable DP-ABE schemes over the universe $\mathcal{U}$, an empty two-dimensional array L_{Reg}, a list of revoked servers and a time source $\mathbb{T}$ (e.g. a networked clock or counter updated by Revoke) to index update keys.

Algorithm 6.1 $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{HPVC.Setup}(1^\ell, \mathcal{F})$

$(MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}}, T^0_O, T^0_S) \xleftarrow{\$} \mathsf{DPABE.Setup}(1^\ell, \mathcal{U})$
$(MPK^1_{\text{ABE}}, MPK^1_{\text{ABE}}, T^1_O, T^1_S) \xleftarrow{\$} \mathsf{DPABE.Setup}(1^\ell, \mathcal{U})$
for $S_i \in \mathcal{U}_{\text{ID}}$ **do**
    $L_{\text{Reg}}[S_i][0] \leftarrow \epsilon$, $L_{\text{Reg}}[S_i][1] \leftarrow \{\epsilon\}$
Initialise $\mathbb{T}$
$L_{\text{Rev}} \leftarrow \epsilon$
$\text{PP} \leftarrow (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{\text{Reg}}, T^0_O, T^1_O, T^0_S, T^1_S, \mathbb{T})$
$\text{MK} \leftarrow (MSK^0_{\text{ABE}}, MSK^1_{\text{ABE}}, L_{\text{Rev}})$

2. FnInit, presented in Algorithm 6.2, sets the public delegation key PK_F to be the public parameters for the system (since we use public key primitives). This algorithm is the same for all functions F.

Algorithm 6.2 $PK_F \xleftarrow{\$} \mathsf{HPVC.FnInit}(F, \text{MK}, \text{PP})$

$PK_F \leftarrow \text{PP}$

3. Register, presented in Algorithm 6.3, runs a signature KeyGen algorithm and adds the verification key to $L_{Reg}[S_i][0]$. Signatures ensure that honest servers are neither impersonated nor maliciously revoked.

Algorithm 6.3 $SK_{S_i} \xleftarrow{\$} \mathsf{HPVC.Register}(S_i, \text{MK}, \text{PP})$

$(SK_{\text{Sig}}, VK_{\text{Sig}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\ell)$
$SK_{S_i} \leftarrow SK_{\text{Sig}}$
$L_{\text{Reg}}[S_i][0] \leftarrow L_{\text{Reg}}[S_i][0] \cup VK_{\text{Sig}}$

4. Certify, presented in Algorithm 6.4, first adds an element $(F, \bigcup_{l_j \in L_i} l_j)$ to the list $L_{Reg}[S_i][1]$ for every $F \in \mathcal{F}_i$; this publicises the computations that S_i is able to perform (either the functions in RPVC and PVC-AC modes, or the functions and data labels in VDC mode). The algorithm then removes S_i from the revocation list, initialises the time source $\mathbb{T}$ and runs KeyGen for the first DP-ABE system to generate a decryption key for $(\mathbb{O}, \psi \cup \bigcup_{l_j \in L_i} l_j)$.

   The inclusion of the labels $l_j \in \mathcal{U}_l$ as additional attributes ensures that a key may not be used to evaluate computations that do not correspond to these labels. In

the RPVC and PVC-AC settings, this means that a key for a function G may not be used to evaluate a computation on input $X^\star$ when the delegator outsources the computation of $F(X^\star)$. In VDC, this means that the evaluation key used by the server to produce a computational result must have been issued for a dataset D_i that includes (at least) the specified input data $X^\star$. We note that it is sufficient to only include the labels 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 can amend the servers key (in the subjective attributes) but need not take similar measures against the delegator. Delegators can then specify, in the subjective policy that they create, the labels that are required. Even though the subjective attribute set is defined to be a dummy parameter in RPVC and PVC-AC modes, we still add the additional attributes — then the delegator can specify the required labels during ProbGen and these labels must still be present in the server's key for a successful evaluation (decryption) to occur. (Trivially applying the labels to both the objective and subjective policies would require L_i to equal $L_{F,X^\star}$ in order for both policies to be satisfiable.)

The KDC should check that the label actually corresponds to the input (for example, by defining and applying a one-way, injective label mapping from $\mathcal{F} \times \mathcal{U}_x$ to $\mathcal{U}_l$) to ensure that a server does not fraudulently advertise data that he does not actually own. It also generates an update key for the current time period to prove that S_i is not currently revoked. If operating in a publicly verifiable outsourced computation mode, another pair of keys is generated using the second DP-ABE system for the complement inputs.

Algorithm 6.4 $EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{HPVC.Certify}(\mathtt{mode}, S_i, (\mathbb{O},\psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

for $F \in \mathcal{F}_i$ **do**
  $L_{\mathrm{Reg}}[S_i][1] \leftarrow L_{\mathrm{Reg}}[S_i][1] \cup (F, \bigcup_{l_j \in L_i} l_j)$
$L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \setminus S_i$, $t \leftarrow \mathbb{T}$
$SK^0_{\mathrm{ABE}} \xleftarrow{\$} \mathsf{DPABE.KeyGen}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_j \in L_i} l_j), MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
$UK^0_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
if ($\mathtt{mode} = RPVC$) **or** $\mathtt{mode} = \text{PVC-AC}$) **then**
  $SK^1_{\mathrm{ABE}} \xleftarrow{\$} \mathsf{DPABE.KeyGen}(S_i, (\overline{\mathbb{O}}, A_\psi \cup \bigcup_{l_j \in L_i} l_j), MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
  $UK^1_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
else
  $SK^1_{\mathrm{ABE}} \leftarrow \perp$, $UK^1_{L_{\mathrm{Rev}},t} \leftarrow \perp$
$EK_{(\mathbb{O},\psi),S_i} \leftarrow (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$

5. ProbGen, presented in Algorithm 6.5, first chooses two messages m_0 and m_1 uniformly at random from the message space. A random bit b randomly permutes

the messages such that a verifier that does not know b does not know which message was recovered and hence the value of $F(X)$. Message m_b is encrypted with $(A_\omega, \mathbb{S} \wedge \bigwedge_{l_j \in L_{F,X}} l_j)$ and the first system parameters (where A_ω is the attribute set encoding of the input ω), whilst m_{1-b} is encrypted using the complement policy and either the first system parameters for VDC or the second for RPVC (the attribute set remains the same as it is either the same input data X in RPVC, or the same special attribute T_O^0 in VDC). The verification key is computed by applying the one-way function g to the messages (the one-wayness allows the key to be published), and b forms the output retrieval key (as this reveals the order of the decrypted results and hence the output).

Algorithm 6.5 $(\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{HPVC.ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, PK_F, \mathrm{PP})$

$(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
$b \xleftarrow{\$} \{0,1\}$
$t \leftarrow \mathbb{T}$
$c_b \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_b, (A_\omega, \mathbb{S} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), t, MPK^0_{\mathrm{ABE}})$
if $\mathtt{mode} = VDC$ **then** $c_{1-b} \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_{1-b}, (A, \bar{\mathbb{S}} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), t, MPK^0_{\mathrm{ABE}})$
else $c_{1-b} \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_{1-b}, (A_\omega, \bar{\mathbb{S}} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), t, MPK^1_{\mathrm{ABE}})$
return $\sigma_{F,X} \leftarrow (c_b, c_{1-b})$, $VK_{F,X} \leftarrow (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $RK_{F,X} \leftarrow b$

6. Compute, presented in Algorithm 6.6, decrypts the two ciphertexts and signs the results, again ensuring that the different modes use the correct system parameters.

Algorithm 6.6 $\theta_{F(X)} \xleftarrow{\$} \mathsf{HPVC.Compute}(\mathtt{mode}, \sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \mathrm{PP})$

Parse $EK_{(\mathbb{O},\psi),S_i}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$ and $\sigma_{F,X}$ as (c, c')
$d_b \leftarrow \mathsf{DPABE.Decrypt}(c, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
if $\mathtt{mode} = VDC$ **then** $d_{1-b} \leftarrow \mathsf{DPABE.Decrypt}(c', SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
else $d_{1-b} \leftarrow \mathsf{DPABE.Decrypt}(c', SK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, UK^1_{L_{\mathrm{Rev}},t})$
$\gamma \xleftarrow{\$} \mathsf{Sig.Sign}((d_b, d_{1-b}, S_i), SK_{S_i})$
$\theta_{(\omega,\mathbb{S}),(\mathbb{O},\psi)} \leftarrow (d_b, d_{1-b}, S_i, \gamma)$

7. BVerif, presented in Algorithm 6.7, verifies the signature using the verification key for the server S_i. If correct, it applies 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), that plaintext is returned as the retrieval token and the output token is accept. Otherwise the result is rejected and the server is reported for revocation.

8. Retrieve, presented in Algorithm 6.8, orders the components of the verification key according to the retrieval key $RK_{F,X} = b$ and checks which message was returned correctly, and hence determines the result of the computation. If m_0 was returned

Algorithm 6.7 $(RT_{F(X)}, \tau_{F(X)}) \leftarrow \mathsf{HPVC.BVerif}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$

Parse $VK_{F,X}$ as $(VK, VK', L_{\mathrm{Reg}})$ and $\theta_{F(X)}$ as (d, d', S_i, γ)
if accept $\leftarrow \mathsf{Sig.Verify}((d, d', S_i), \gamma, L_{\mathrm{Reg}}[S_i][0])$ **then**
  if $VK = g(d)$ **then return** $(RT_{F(X)} \leftarrow d, \tau_{F(X)} \leftarrow (\mathsf{accept}, S_i))$
  else if $VK' = g(d')$ **then return** $(RT_{F(X)} \leftarrow d', \tau_{F(X)} \leftarrow (\mathsf{accept}, S_i))$
  else return $(RT_{F(X)} \leftarrow \perp, \tau_{F(X)} \leftarrow (\mathsf{reject}, S_i))$
return $(RT_{F(X)} \leftarrow \perp, \tau_{F(X)} \leftarrow (\mathsf{reject}, \perp))$

then $F(X) = 1$ as m_0 was encrypted for the non-complemented input set; if m_1 was returned then $F(X) = 0$.

Algorithm 6.8 $y \leftarrow \mathsf{HPVC.Retrieve}(RT_{F(X)}, \tau_{F(X)}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$

Parse $VK_{F,X}$ as $(g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $\theta_{F(X)}$ as $(d_b, d_{1-b}, S_i, \gamma)$, $RK_{F,X}$ as b
if $(\tau_{F(X)} = (\mathsf{accept}, S_i)$ **and** $g(RT_{F(X)}) = g(m_0))$ **then return** $y \leftarrow 1$
if $(\tau_{F(X)} = (\mathsf{accept}, S_i)$ **and** $g(RT_{F(X)}) = g(m_1))$ **then return** $y \leftarrow 0$
return $y \leftarrow \perp$

9. Revoke, presented in Algorithm 6.9, first checks whether a sever should in fact be revoked. If so, it deletes the list $L_{\mathrm{Reg}}[S][1]$ of computations that the server, S_i, to be revoked, may perform. It also adds S_i to the revocation list, and refreshes the time source. It then generates new update keys for all non-revoked entities according to the updated revocation list, and distributes updated evaluation keys such that non-revoked keys are still functional in the new time period.

Algorithm 6.9 $UM \xleftarrow{\$} \mathsf{HPVC.Revoke}(\tau_{F(X)}, \mathrm{MK}, \mathrm{PP})$

if $(\tau_{F(X)} \neq (\mathsf{reject}, S_i))$ **then return** $UM \leftarrow \perp$
$L_{\mathrm{Reg}}[S][1] \leftarrow \{\epsilon\}$, $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \cup S_i$
Refresh $\mathbb{T}$, $t \leftarrow \mathbb{T}$
$UK^0_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
if $(\mathtt{mode} = RPVC)$ **or** $\mathtt{mode} = \mathrm{PVC\text{-}AC})$ **then**
  $UK^1_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
for all $S' \in \mathcal{U}_{\mathrm{ID}}$ **do**
  Parse $EK_{(\mathbb{0},\psi),S'}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t-1}, UK^1_{L_{\mathrm{Rev}},t-1})$
  $EK_{(\mathbb{0},\psi),S'} \leftarrow (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$
return $UM \leftarrow \{EK_{(\mathbb{0},\psi),S'}\}_{S' \in \mathcal{U}_{\mathrm{ID}}}$

It is straightforward to see that correctness of this construction follows from the correctness of the attribute-based encryption scheme and of the one-way function g.

Theorem 6.1. *Given a secure* IND-sHRSS *rkDPABE 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*, then $\mathcal{HPVC}$, defined by Algorithms 6.1 to 6.9, is secure in the sense of selective public verifiability and selective semi-static revocation.*

A proof of Theorem 6.1 can be found in Appendix B.3.

6.4 Conclusion

In this chapter, we have drawn together the work of previous chapters to create a unified notion of verifiable outsourced computation, HPVC, which supports multiple modes of operation to meet the diverse user requirements of a large multi-user system. We capture the notions of RPVC, RPVC with access control on servers (maintaining public delegability and verification), and VDC. We have seen that HPVC leads to a natural and novel use of DP-ABE.

In future work, we will consider the use of multiple key generation authorities (or KDCs) in a DP-ABE scheme such that, in an HPVC scheme, the responsibilities for assigning function evaluation keys and for assigning security attributes are not borne by a single entity. We believe that in practice, it is more likely that entities will be authoritative on only one of these areas (and that the KDC that assigns security attributes could also be used in other systems as a form of federated identity management). This amounts to splitting the KeyGen operations for the KP- and CP- parts of the DP-ABE scheme, yet ensuring that the scheme remains secure by tying these keys together by using a global identifier [38, 77]. We will also further investigate our revocable DP-ABE scheme to compare the efficiency of revoking the key- and ciphertext-policies. Finally, interesting open questions are whether predicate encryption could enable private information retrieval [41], and whether multi-authority techniques could allow servers themselves to generate evaluation keys for *only* their own data. Furthermore we will investigate techniques allowing updatable data to be stored on the server.

Chapter 7

Conclusion

Throughout this thesis, we have considered methods in which attribute-based encryption can be employed, not just as a cryptographic enforcement mechanism for access control policies, but to enable the delegation of computation to external entities. We have tried to develop results that are interesting in practical settings, considering the system models and the necessary interactions between parties, whilst still maintaining formal and rigorous cryptographic security notions and proofs.

Whilst verifiable computation has recently attracted intense interest in the theoretical community, the motivation for this area of research remains firmly entrenched in the practical, real-world requirements of available computational resources. As user devices become increasingly mobile-oriented (and as a result, possibly weaker), and communication channels over mobile networks become faster and more widely available, the notion of outsourcing intensive processes to more capable servers is only going to become more attractive. Indeed, already much processing on smart-phones is outsourced to external servers; for example, Apple's Siri [8, 94] and Google Voice Search [64], amongst many other popular services, transmit user requests to powerful data centres in order to be processed, and results are returned to the user device. When the network that such requests and results are transmitted over is insecure or may introduce faults, or when the data centre may not be trusted, it is vital to efficiently verify the results of such computations before presenting information to the user. It is therefore important that research in this area maintains a focus on practical system models and not just theoretically interesting cryptographic results. Of course, a rigorous security treatment is still absolutely necessary to ensure the security of systems once deployed.

We began this thesis by considering the work of Parno et al. who first realised that KP-

ABE could be used as a verifiable proof mechanism for the satisfaction a Boolean formula. We considered this notion of *publicly verifiable outsourced computation* with a view to the practical environments in which such systems may be deployed. As such, we developed a system model wherein multiple servers can be certified to compute multiple functions, and where misbehaving servers can be removed from the system to act as a deterrent and to avoid wasting delegator resources. We considered two example architectures, namely the standard model and the manager model. In the case of the latter, servers can be selected from a large pool based on availability or other desirable properties (e.g. resources, location, latency or cost). We formalised the notion of blind verification to allow the manager to verify the work of these servers before returning the results to the client, and also considered novel security models arising from the revocation mechanism — both to ensure that the cryptographic revocation achieved the desired properties and to prevent vindictive entities from "gaming" the system to revoke otherwise honest servers.

In Chapter 4, we observed that we had transitioned to a potentially large, multi-user distributed system and, as such, our setting of RPVC was a natural environment in which to apply cryptographic access control mechanisms. We discussed several forms of policy that are of interest in these settings, defined generically in terms of graph-based information flow policies. We also defined appropriate security models to ensure that only authorised entities could participate in valid interactions and gave a provably secure instantiation built on symmetric key assignment schemes.

Chapter 5 considered the use of CP-ABE in the setting of verifiable computation. We observed that the resulting system model, although somewhat different to PVC built on KP-ABE, had very natural applications to verifiable parallel processing of large datasets, verifiable queries to remote databases and to another interesting line of research into three-party computation on authenticated data.

Finally, in Chapter 6, we brought together the work of the previous chapters to create a single unified publicly verifiable outsourced computation system in which entities may outsource computations to servers with available resources, as well as providing computational services on its own local data. We saw that the use of DP-ABE allowed for both RPVC and VDC functionality, and also enabled a form of access control to be applied to the set of servers that may perform a given outsourced computation, using only public key primitives.

Throughout the thesis, we have highlighted, where appropriate, possible directions for future work. One particularly important task would be to implement the schemes developed in this work and to compare the efficiency of the different approaches and to evaluate the schemes against other work in the literature. Another important area to focus future attention on, if these schemes were to be deployed, would be to strengthen the primitives on which we base our constructions so that we can achieve the full, ideal, notions of security. Several of our currently achievable security notions include selective or semi-static restrictions, which could be removed by developing fully secure [76] and adaptive revocable ABE schemes. Finally, formal security models based on indistinguishability games should be developed to prove that the blind verification mechanism used in our constructions does indeed provide the level of security required.

Furthermore, it would be nice to explore further applications of VDC, which appears to be a flexible computational model, particularly with regards to searching on remote databases. Searchable encryption is one research area that has looked at this problem, and requires additional privacy properties that we have not considered for VDC in this thesis.

Another important area for future research is general-purpose verifiable outsourced computations. Many current VC schemes are limited to very specific operations, such as matrix multiplication or polynomial evaluation [53, 98] or, as in this thesis, to the evaluation of restrictive classes of function. In this thesis, we have restricted our attention to (monotone) Boolean formulae. Whilst these are important results, and cover many useful functions that may be of interest in particular outsourcing applications, it is hard to envisage verifiable computation being widely deployed until a broad range of computations can be efficiently supported. Indeed, an ultimate goal may be to transparently compile applications designed to be executed locally into secure, verifiable programs that can be outsourced.

Bibliography

[1] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, Aug. 1983.

[2] J. Alderman and J. Crampton. On the use of key assignment schemes in authentication protocols. In J. Lopez, X. Huang, and R. Sandhu, editors, *Network and System Security*, volume 7873 of *Lecture Notes in Computer Science*, pages 607–613. Springer Berlin Heidelberg, 2013.

[3] 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.

[4] J. Alderman, C. Janson, C. Cid, and J. Crampton. Hybrid publicly verifiable computation. Cryptology ePrint Archive, Report 2015/320, 2015. `http://eprint.iacr.org/`, to appear in Topics in Cryptology-CT-RSA 2016, The Cryptographer's Track at RSA Conference 2016.

[5] 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*, volume 8957 of *Lecture Notes in Computer Science*, pages 51–71. Springer International Publishing, 2015.

[6] ANSI InterNational Committee for Information Technology Standards (INCITS). INCITS 359-2012 information technology - role based access control, 2012.

[7] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. Verifiable oblivious storage. In H. Krawczyk, editor, *Public-Key Cryptography - PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 131–148. Springer Berlin Heidelberg, 2014.

[8] Apple. Siri. `https://www.apple.com/uk/ios/siri/`. Accessed: 20/07/2015.

[9] 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.

[10] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[11] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, Jan. 1998.

[12] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–18:43, Jan. 2009.

[13] M. J. Atallah, M. Blanton, and K. B. Frikken. Efficient techniques for realizing geospatial access control. In *Proceedings of the 2Nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS '07, pages 82–92, New York, NY, USA, 2007. ACM.

[14] N. Attrapadung and H. Imai. Attribute-based encryption supporting direct/indirect revocation modes. In M. Parker, editor, *Cryptography and Coding*, volume 5921 of *Lecture Notes in Computer Science*, pages 278–300. Springer Berlin Heidelberg, 2009.

[15] N. Attrapadung and H. Imai. Dual-policy attribute based encryption. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 168–185. Springer Berlin Heidelberg, 2009.

[16] 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.

[17] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *Proc. 36th IEEE Symposium on Security & Privacy (S&P), page to appear. IEEE Computer Society Press*, 2015.

[18] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, pages 863–874, New York, NY, USA, 2013. ACM.

[19] 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.

[20] A. Beimel. *Secure schemes for secret sharing and key distribution.* PhD thesis, Technion-Israel Institute of technology, Faculty of computer science, 1996.

[21] 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.

[22] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, MITRE Corporation, 1973.

[23] 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.

[24] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions andanalysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.

[25] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: Extended abstract. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 401–414, New York, NY, USA, 2013. ACM.

[26] 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.

[27] 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.

[28] M. Bishop. *Introduction to computer security.* Addison-Wesley Boston, MA, 2005.

[29] M. A. Bishop. *Computer Security. Art and Science.* Addison-Wesley Professional, 2002.

[30] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, New York, NY, USA, 2012. ACM.

[31] A. Boldyreva, V. Goyal, and V. Kumar. Identity-based encryption with efficient revocation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 417–426, New York, NY, USA, 2008. ACM.

[32] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin Heidelberg, 2001.

[33] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer Berlin Heidelberg, 2005.

[34] 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.

[35] A. Castiglione, A. De Santis, and B. Masucci. Key indistinguishability vs. strong key indistinguishability for hierarchical key assignment schemes. *IEEE Transactions on Dependable and Secure Computing*, 1(1):1–1, 2014.

[36] D. Chadwick. Federated identity management. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 96–120. Springer Berlin Heidelberg, 2009.

[37] Q. Chai and G. Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Communications (ICC), 2012 IEEE International Conference on*, pages 917–922. IEEE, 2012.

[38] M. Chase. Multi-authority attribute based encryption. In S. Vadhan, editor, *Theory of Cryptography*, volume 4392 of *Lecture Notes in Computer Science*, pages 515–534. Springer Berlin Heidelberg, 2007.

[39] D. Chaum and T. P. Pedersen. Wallet databases with observers. 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 89–105. Springer, 1992.

[40] S. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In A. Sahai, editor, *Theory of Cryptography*, volume 7785 of *Lecture Notes in Computer Science*, pages 499–518. Springer Berlin Heidelberg, 2013.

[41] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, Nov. 1998.

[42] K.-M. Chung, Y. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer Berlin Heidelberg, 2011.

[43] 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.

[44] J. Crampton. Cryptographic enforcement of role-based access control. In P. Degano, S. Etalle, and J. Guttman, editors, *Formal Aspects of Security and Trust*, volume 6561 of *Lecture Notes in Computer Science*, pages 191–205. Springer Berlin Heidelberg, 2011.

[45] J. Crampton. Practical and efficient cryptographic enforcement of interval-based access control policies. *ACM Trans. Inf. Syst. Secur.*, 14(1):14:1–14:30, June 2011.

[46] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, CSFW '06, pages 98–111, Washington, DC, USA, 2006. IEEE Computer Society.

[47] T. W. Cusick and P. Stanica. *Cryptographic Boolean functions and applications*. Academic Press, 2009.

[48] Q. Dang. Recommendation for applications using approved hash algorithms. *NIST Special Publication*, 107(February), 2008.

[49] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order (2. ed.)*. Cambridge University Press, 2002.

[50] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[51] A. L. Ferrara, G. Fuchsbauer, and B. Warinschi. Cryptographically enforced RBAC. In *Proceedings of the 2013 IEEE 26th Computer Security Foundations Symposium*, CSF '13, pages 115–129, Washington, DC, USA, 2013. IEEE Computer Society.

[52] A. Fiat and M. Naor. Broadcast encryption. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[53] 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.

[54] M. J. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9):948–960, 1972.

[55] E. S. V. Freire, K. G. Paterson, and B. Poettering. Simple, efficient and strongly ki-secure hierarchical key assignment schemes. In E. Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2013.

[56] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[57] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *Advances in Cryp-*

tology - CRYPTO 2010, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer Berlin Heidelberg, 2010.

[58] 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.

[59] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, pages 169–178. ACM, 2009.

[60] S. Goldwasser, S. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer Berlin Heidelberg, 2014.

[61] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 113–122, New York, NY, USA, 2008. ACM.

[62] 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.

[63] 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.

[64] Google. How Google uses pattern recognition. `http://www.google.com/policies/technologies/pattern-recognition/`. Accessed: 20/07/2015.

[65] Google. Google Compute Engine – Cloud Computing & IaaS – Google Cloud Platform. `http://cloud.google.com/compute/`, 2014. Accessed 23/10/2014.

[66] 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.

[67] M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of ABE ciphertexts. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 34–34, Berkeley, CA, USA, 2011. USENIX Association.

[68] 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.

[69] International Organization for Standardization (ISO). Information technology—Security techniques—Encryption algorithms—Part 1: General, 2005.

[70] International Organization for Standardization (ISO). Information technology—Security techniques—Encryption algorithms—Part 2: Asymmetric ciphers, 2006.

[71] International Organization for Standardization (ISO). Information technology—Trusted platform module—Part 1: Overview, 2009.

[72] International Organization for Standardization (ISO). ISO/IEC 9797-1:2011: Information technology - Security techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher, 2011.

[73] International Organization for Standardization (ISO). ISO/IEC DIS 10118-1 Information technology – Security techniques – Hash-functions – Part 1: General, 2014.

[74] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[75] J. Kilian. Improved efficient arguments (preliminary version). In D. Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 1995.

[76] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer Berlin Heidelberg, 2010.

[77] A. Lewko and B. Waters. Decentralizing attribute-based encryption. In K. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 568–588. Springer Berlin Heidelberg, 2011.

[78] S. Micali. CS proofs. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 436–453. IEEE, 1994.

[79] F. Monrose, P. Wyckoff, and A. D. Rubin. Distributed execution with remote audit. In *NDSS*, volume 99, pages 3–5, 1999.

[80] 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.

[81] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In A. Sahai, editor, *Theory of Cryptography*, volume 7785 of *Lecture Notes in Computer Science*, pages 222–242. Springer Berlin Heidelberg, 2013.

[82] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Optimal verification of operations on dynamic sets. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 91–110. Springer Berlin Heidelberg, 2011.

[83] 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.

[84] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer Berlin Heidelberg, 2012.

[85] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[86] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[87] R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.

[88] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. Van Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, volume 173, 2005.

[89] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[90] 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.

[91] 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.

[92] 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.

[93] T. White. *Hadoop: the definitive guide: the definitive guide.* " O'Reilly Media, Inc.", 2009.

[94] Wired. Apple finally reveals how long Siri keeps your data. `http://www.wired.com/2013/04/siri-two-years/`. Accessed: 20/07/2015.

[95] XACML-V3.0 – eXtensible Access Control Markup Language (XACML). http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.

[96] L. Xu and S. Tang. Verifiable computation with access control in cloud computing. *The Journal of Supercomputing*, 69(2):528–546, 2014.

[97] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.

[98] L. F. Zhang and R. Safavi-Naini. Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. In M. Abdalla, C. Nita-Rotaru, and

R. Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 329–348. Springer, 2013.

[99] Q. Zheng, S. Xu, and G. Ateniese. VABKS: Verifiable attribute-based keyword search over outsourced encrypted data. In *INFOCOM, 2014 Proceedings IEEE*, pages 522–530. IEEE, 2014.

Appendix A

Additional Material for Access Control in Publicly Verifiable Outsourced Computation

This appendix includes additional security proofs for the revocable publicly verifiable outsourced computation scheme with access control introduced in Chapter 4. These largely follow a similar pattern to the proof of Lemma 4.1.

A.1 Proof of Authorised Computation

Lemma A.1. *Given a secure RPVC scheme, an authenticated symmetric encryption scheme secure in the sense of* IND-CPA $\wedge$ INT-PTXT *and a KAS secure against strong key-indistinguishability, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 4.1–4.9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of authorised computation (Game 4.2).*

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage δ in the authorised computation game and let $\mathcal{A}_{SE}$ be an adversary playing the IND-CPA game with a challenger $\mathcal{C}$ against $\mathcal{SE}$. We define the following sequence of games and show that we can perform a sequence of game hops with negligible difference between each successive pairs of games, and thereby construct an adversary $\mathcal{A}_{SE}$ that, using $\mathcal{A}_{VC}$ as a subroutine, can break the IND-CPA property with non-negligible advantage.

- **Game 0.** This is the authorised computation game as defined in Section 4.4.2.
- **Game 1.** This is identical to **Game 0**, except that we replace the key $\kappa_{\lambda^C(o)}$ for

the challenge computation o with a key $\kappa^\star$ drawn uniformly at random from the keyspace.

- **Game 2.** This is the same as **Game 1** with the modification that, in $\mathsf{ProbGen}$, the encoded input is generated by either encrypting the proper encoded input from the RPVC functionality σ'_o, or a random message of the same length as σ'_o.

Game 0 to Game 1 This game hop relies on the strong key-indistinguishability of the KAS and is very similar to the corresponding hop in the proof of Lemma 4.1. As such, we do not reproduce it here. If $\mathcal{A}$ distinguishes **Game 0** from **Game 1** with non-negligible advantage δ, then an adversary can use $\mathcal{A}$ as a subroutine to break the strong key-indistinguishability of the KAS also with non-negligible advantage δ. Since the KAS is assumed S-KI secure, such an adversary cannot exist and hence **Game 0** is indistinguishable from **Game 1** except with at most a negligible advantage $\epsilon \leqslant 1 - \delta$.

Game 1 to Game 2 We have shown that, from the adversary's point of view, **Game 1** is almost (with negligible distinguishing advantage) identical to **Game 0**. Thus, we may run the adversary against **Game 1** instead to remove any information leakage from the KAS. We now show that an adversary cannot distinguish **Game 1** from **Game 2** with more than a negligible probability, which then removes any information leakage from the encrypted, encoded input. Suppose, for a contradiction, that an adversary $\mathcal{A}_{VC}$ exists that can distinguish **Game 1** from **Game 2** with non-negligible advantage δ. Then we show that there exists an adversary $\mathcal{A}_{SE}$ that breaks the IND-CPA security of the symmetric encryption scheme $\mathcal{SE}$ also with advantage δ. $\mathcal{A}_{VC}$ will play either **Game 1** or **Game 2** with $\mathcal{A}_{SE}$ acting as the challenger, and must guess correctly which game he is playing. $\mathcal{A}_{SE}$ in turn will play the IND-CPA game with a challenger $\mathcal{C}$.

1. $\mathcal{C}$ begins by choosing a random bit b (which ultimately will determine which of **Game 1** and **Game 2** is being played) and running $\mathsf{SE.KeyGen}(1^\ell)$ to generate a key $\kappa^\star$. It sends the security parameter 1^ℓ to $\mathcal{A}_{SE}$.

2. $\mathcal{A}_{SE}$ must now initialise **Game b+1** for $\mathcal{A}_{VC}$. Informally, it will set the KAS key for the label $\lambda^C(o)$ to be the random key $\kappa^\star$ chosen by $\mathcal{C}$. However, the challenge label is unknown until $\mathcal{A}_{VC}$ chooses it, whilst the public parameters and oracle access must

be provided before this choice. Thus, we require $\mathcal{A}_{SE}$ to guess the challenge label during Setup so that the corresponding key can be implicitly set to be the IND-CPA challenge key (that is, the key for the guessed label will be defined to be the IND-CPA challenge key but, because $\mathcal{A}_{SE}$ does not hold this key, all encryptions under it shall be formed using oracle queries to $\mathcal{C}$). If the number of labels in the poset is N, where N is polynomial in the security parameter (as the scheme must be efficiently instantiable), then $\mathcal{A}_{SE}$ may guess $\lambda^C(o)$ with probability at least $\frac{1}{N}$. Assuming that the guess is correct, we proceed as follows.

3. $\mathcal{A}_{SE}$ runs $\mathsf{PVC}_{\mathsf{AC}}$.Setup as given in Algorithm 4.1 except that the key for the guessed label in the computation poset, $\kappa_{\lambda^C(o)}$ is implicitly set to be $\kappa^\star$ (i.e. any use of $\kappa^\star$ must be performed using oracle queries to $\mathcal{C}$) and the KAS is constructed to be consistent with this choice. Note that the authorised computation game (and by extension **Game b+1**) does not permit the adversary to query any label that is an ancestor of the challenge label in the computation poset. Thus a KAS can be instantiated over the remaining nodes (and the public information for the ancestor set simulated — as the keys cannot be derived, the public information need not be functionally correct). Remaining keys can simply be generated using the security parameter. From the adversarial point of view, this will be indistinguishable from the real games.

4. $\mathcal{A}_{VC}$ is given the generated public parameters and oracle access which $\mathcal{A}_{SE}$ responds to as follows:

   - FnInit, Certify, and Revoke queries can be handled by simply calling the relevant algorithm.
   - If a Register query is made for a label $\lambda(ID) \geqslant \lambda^C(o)$ then $\mathcal{A}_{SE}$ aborts the game since $\mathcal{A}_{VC}$ would not then be able to choose $\lambda^C(o)$ as its challenge computation, and hence $\mathcal{A}_{SE}$'s guess was incorrect. Otherwise, $\mathcal{A}_{SE}$ holds the relevant KAS keys and may respond by running Oracle Query 4.1.
   - ProbGen queries for a computation labelled by the challenge label $\lambda(o)$ can be handled by running Algorithm 3.5 with the exception that lines 2 and 5 are simulated as follows. Line 2 is not run at all as $\mathcal{A}_{SE}$ does not hold the challenge key, and line 5 is run using an oracle query to the IND-CPA LoR oracle for the choice of messages $m_0 = m_1 = \sigma'_o$ (which will return the encryption of σ'_o as both message options are the same). For any other queried computation, $\mathcal{A}_{SE}$

holds the KAS key (or generated symmetric key) and may run Algorithm 3.5 as written.

- Observe that by the INT-PTXT property of the authenticated symmetric encryption scheme, $\mathcal{A}_{VC}$ cannot form a valid ciphertext (one that will decrypt to anything other than $\perp$) without holding the encryption key. The encryption keys that $\mathcal{A}_{VC}$ may hold are precisely those revealed through Register queries, and since queries may not be made for labels $\lambda(ID) \geqslant \lambda^C(o)$, each query to the Compute oracle will be for a label belonging to the KAS instantiated by $\mathcal{A}_{SE}$. Hence, if $\lambda(ID) \geqslant \lambda^C(o)$, $\mathcal{A}_{SE}$ returns $\perp$ and otherwise it uses its knowledge of the KAS to construct a genuine response (by running Algorithm 3.6).

5. $\mathcal{A}_{VC}$ eventually outputs a challenge computation, and if this is not the computation chosen by $\mathcal{A}_{SE}$ in Step 2, then the game is aborted. Otherwise, the choice of computation label is valid since the game has not already been aborted during the oracle queries.

6. $\mathcal{A}_{SE}$ should now register two entities: a delegator C and a verifier V. However, these will not be required in the following, so $\mathcal{A}_{SE}$ may simulate registering them with labels $\lambda^C(o)$ and $\lambda^V(o)$ respectively and update the public parameters accordingly, without actually requiring the correct keys for these entities (as the adversary will not see any output from these entities other than their presence in the lists in the public parameters). $\mathcal{A}_{SE}$ can also run the FnInit algorithm as written.

7. $\mathcal{A}_{SE}$ must now run ProbGen for the challenge computation o to generate an encoded input σ_o. To do so, it will make use of the LoR oracle provided by $\mathcal{C}$ in the IND-CPA game. $\mathcal{A}_{SE}$ will run lines 1, 3 and 4 as written to generate a problem encoding σ'_o. It sets $m_0 = \sigma'_o$ and chooses another message m_1 of the same length uniformly at random from the message space. These are queried to the LoR oracle which will return the encryption of message m_b for the challenger's choice of b from Step 1. $\mathcal{A}_{SE}$ can then use this response to form σ_o and form RK_o using the verification KAS key which it holds.

8. All relevant information is passed to $\mathcal{A}_{VC}$ who is also given oracle access. This is again handled in the same manner as in Step 4. Eventually, $\mathcal{A}_{VC}$ will produce a challenge output, which $\mathcal{A}_{SE}$ can verify using previously generated parameters.

Observe that if the challenger's random bit $b = 0$, then this is precisely **Game 1** (since the

real encoded input was encrypted by $\mathcal{C}$). If, on the other hand, $b = 1$ then $\mathcal{A}_{VC}$ is provided with the encryption of a random message which does not relate to the computation at all, which is precisely the setting of **Game 2**. Now, we assumed that $\mathcal{A}_{VC}$ could distinguish **Game 1** from **Game 2** with non-negligible probability δ. Since these two games correspond directly to the challenger's choice of bit b, $\mathcal{A}_{SE}$ can simply forward $\mathcal{A}_{VC}$'s guess b' to $\mathcal{C}$ and win the IND-CPA game with non-negligible advantage δ. However, as we assumed that SE was IND-CPA $\wedge$ INT-PTXT secure, this is a contradiction and hence an adversary with non-negligible distinguishing advantage cannot exist.

Reduction to public verifiability We have shown negligible distinguishing advantages in the transitions from the authorised computation game to **Game 2**. Thus we may run an adversary against **Game 2** instead. By moving to this modified game, we have removed any information leakage from the KAS and from the ciphertext encrypting the encoded input. Thus the only information that remains that could aid an adversary in the authorised computation game is the verification key and the output retrieval key. Intuitively, however, the underlying RPVC scheme is designed such that these components *do not* leak information that aids the adversary in producing a fraudulent result.

We now show that, since the adversary is not authorised for the challenge computation, even if it holds a valid evaluation key it cannot produce a valid response in **Game 2**. If it could do so, then an adversary could be constructed to break the public verifiability of the RPVC scheme. More formally, to achieve a contradiction, let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage δ against **Game 2**. Then we construct an adversary $\mathcal{A}_{PV}$ that breaks the public verifiability of the RPVC scheme with advantage $\frac{\delta}{2}$. Let $\mathcal{C}$ play the public verifiability game (Game 3.2) with $\mathcal{A}_{PV}$ who acts as the challenger for $\mathcal{A}_{VC}$ in **Game 2**.

1. $\mathcal{C}$ runs $\mathsf{RPVC.Setup}$ on the security parameter and gives $\mathcal{A}_{PV}$ the resulting public parameters PP' and oracle access.

2. $\mathcal{A}_{PV}$ initialises the list L and the variable o and must then simulate running the $\mathsf{PVC}_{\mathsf{AC}}.\mathsf{Setup}$ algorithm. It uses the public parameters given by $\mathcal{C}$ and implicitly sets MK' to be that generated by $\mathcal{C}$ (oracle queries will be made to $\mathcal{C}$ whenever MK' is required in the following stages). It sets up the KASs in the same manner as in the transition from **Game 1** to **Game 2** (i.e. by guessing the correct challenge label

with probability at least $\frac{1}{N}$).

3. $\mathcal{A}_{PV}$ sends $\mathcal{A}_{VC}$ the public parameters PP and provides oracle access as follows:

   - Queries to the FnInit, Certify and Revoke oracles can be forwarded to $\mathcal{C}$ and the response returned to $\mathcal{A}_{VC}$.
   - Queries to the Register oracle can be run as in Oracle 4.3 with the exception of line 2 of the Register algorithm for which $\mathcal{A}_{PV}$ makes a RPVC.Register oracle query for the queried identity.
   - Queries to the ProbGen oracle can be handled simply by running Algorithm 3.5. Note that $\mathcal{A}_{PV}$ owns all information related to SK_C for all delegators, and the RPVC.ProbGen algorithm only requires public information.
   - Queries to the Compute oracle can be run as written in Oracle 4.4 using parameters generated in other oracle queries.

4. Eventually, $\mathcal{A}_{VC}$ outputs a choice of challenge computation o, including the function F and input data x. If this is not the same as the guess made by $\mathcal{A}_{PV}$ in Step 2 then the game is aborted. Otherwise, $\mathcal{A}_{PV}$ checks that the label of this computation is valid in accordance with the queries made above. If so, it forwards x to $\mathcal{C}$ as the challenge input $x^\star$ in the public verifiability game. $\mathcal{A}_{PV}$ also computes $F(x)$. If $F(x) = 0$ then it sends $F^\star = F$ as its challenge function to $\mathcal{C}$. Otherwise, it sends $F^\star = \overline{F}$, the complement function, to $\mathcal{C}$. In other words, $\mathcal{A}_{PV}$ chooses the unsatisfied function as its challenge input. Thus, by the winning condition of **Game 2** (and by extension, the authorised computation game), $\mathcal{A}_{VC}$ will, if successful, output a result for the correct value of $F(x)$, which he is unauthorised to compute. To win the public verifiability game, $\mathcal{A}_{PV}$ must output a valid response for the unsatisfied function (i.e. an incorrect result), which is exactly the output that $\mathcal{A}_{VC}$ provides; $\mathcal{A}_{PV}$ will be able to simply forward the output of $\mathcal{A}_{VC}$ to $\mathcal{C}$ to win the public verifiability game with the same (non-negligible) advantage δ of $\mathcal{A}_{VC}$.

5. $\mathcal{C}$ runs RPVC.FnInit for $F^\star$ and RPVC.ProbGen on $x^\star$ and gives the resulting parameters to $\mathcal{A}_{PV}$ and again provides oracle access.

6. $\mathcal{A}_{PV}$ must now register a delegator and a verifier to act in the following stages. To do so, it simply runs Algorithm 3.3 since it knows all required information from the KASs it instantiated. It simulates running FnInit either using the $PK_{F^\star}$ provided by $\mathcal{C}$ in Step 2 (if $F^\star = F$) or by querying the RPVC.FnInit oracle (if $F^\star \neq F$).

Finally, $\mathcal{A}_{PV}$ runs ProbGen as given in Algorithm 3.5 with the exception of choosing a random message instead of the encoded input, as per **Game 2**. All generated parameters are passed to $\mathcal{A}_{VC}$, and oracle access is provided as before.

7. Eventually, $\mathcal{A}_{VC}$ finishes querying and outputs θ_o which it believes will appear to be a valid result despite being unauthorised. $\mathcal{A}_{PV}$ forwards θ_o to $\mathcal{C}$ as its guess in the public verifiability game.

Now, in order for $\mathcal{A}_{PV}$ to win the public verifiability game, it must produce a valid output for the unsatisfied function F or $\overline{F}$ on input $x^\star$. Assuming that $\mathcal{A}_{VC}$ is a successful adversary against **Game 2**, θ_o is a valid result with non-negligible probability δ. Thus, if the game is not aborted (i.e. $\mathcal{A}_{PV}$ guessed the challenge computation label correctly), $\mathcal{A}_{PV}$ wins with probability $\Pr[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 2}}_{\mathcal{A}_{VC}}[\mathcal{PVC}_{\mathrm{AC}}, 1^\ell, \mathcal{F}, \mathcal{P}_C, \mathcal{P}_V]] = \delta$. Thus the overall probability of $\mathcal{A}_{PV}$ winning, including guessing the computation correctly is at least $\frac{\delta}{N}$ which is non-negligible if δ is non-negligible, as assumed. However, since we assumed the underlying RPVC scheme to be secure in the sense of public verifiability, such an adversary $\mathcal{A}_{VC}$ with non-negligible advantage in **Game 2** cannot exist.

Finally, we observe that each transition to **Game 2** had a negligible distinguishing advantage and the final reduction showed a negligible advantage against **Game 2**, and so we conclude that the scheme is secure in the sense of authorised computation. □

A.2 Proof of Authorised Verification

Lemma A.2. *Given a secure RPVC scheme, a KAS secure in the sense of strong-key indistinguishability and an authenticated symmetric encryption scheme secure in the sense of* IND-CPA∧INT-PTXT*, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 4.1–4.9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of authorised verification (Game 4.3).*

Proof. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ in the authorised verification game and $\mathcal{A}_{SE}$ is an adversary playing the IND-CPA game with a challenger $\mathcal{C}$ against $\mathcal{SE}$. We first define the following modified game and show that an adversary has negligible distinguishing advantage between the two. We can therefore employ an

adversary against this modified game to break the IND-CPA security of the symmetric encryption scheme.

- **Game 0.** This is the authorised verification game as defined in Section 4.4.3.
- **Game 1.** This is identical to **Game 0**, except that we replace the key $\kappa_{\lambda^V(o)}$ for the challenge computation o in the verification poset with a random key $\kappa^\star$ drawn uniformly from the keyspace.

Game 0 to Game 1 This transition relies on the strong key-indistinguishability of the KAS. The proof is very similar to that given in the proof of Lemma 4.1, and so we omit the details. If $\mathcal{A}$ can distinguish **Game 0** from **Game 1** with non-negligible advantage δ, then an adversary using $\mathcal{A}$ as a subroutine can break the strong key-indistinguishability of the KAS with the same non-negligible advantage δ. However, as the KAS is assumed S-KI secure, such an adversary may not exist and **Game 0** is indistinguishable from **Game 1** except with at most a negligible advantage $\epsilon \leqslant 1 - \delta$.

Reduction to IND-CPA Suppose, for a contradiction, that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the authorised verification game. We show that we can construct an adversary $\mathcal{A}_{SE}$ that breaks the IND-CPA security (Game 2.5) of the symmetric encryption scheme $\mathcal{SE}$ using $\mathcal{A}_{VC}$ as a subroutine. From the adversary's point of view, **Game 1** is indistinguishable from **Game 0** with negligible distinguishing advantage. Therefore, $\mathcal{A}_{SE}$ may simulate **Game 1** instead to remove any information leakage from the KAS. Let $\mathcal{C}$ be the IND-CPA challenger for $\mathcal{A}_{SE}$ who in turn acts as the challenger in **Game 1** for $\mathcal{A}_{VC}$ who succeeds with non-negligible probability δ.

1. $\mathcal{C}$ first chooses a bit β uniformly at random[1] and generates a key $\kappa^\star \xleftarrow{\$} \mathsf{SE.KeyGen}(1^\ell)$. It sends the security parameter 1^ℓ to $\mathcal{A}_{SE}$.
2. $\mathcal{A}_{SE}$ must now initialise **Game 1** for $\mathcal{A}_{VC}$. It will implicitly set the KAS key for the label $\lambda^V(o_b)$ to be $\kappa^\star$. However, as $\mathcal{A}_{VC}$ has not yet chosen his challenge labels, $\mathcal{A}_{SE}$ must guess one of these labels in order to generate the public parameters and

[1] Note that β corresponds to the challenge bit b in Game 2.5, but we use the notation β in this proof to avoid a notational conflict with the bit b which selects a challenge computation in **Game 1** (and by extension, the authorised verification game in Game 4.3).

provide oracle access. $\mathcal{A}_{SE}$ may guess $\lambda^V(o_b)$ with probability at least $\frac{2}{N}$ (as $\mathcal{A}_{VC}$ will select two labels, and $\mathcal{A}_{SE}$ must match one of them) for a poset comprising N nodes. Assuming that the guess is correct, we proceed as follows.

3. $\mathcal{A}_{SE}$ runs $\mathsf{PVC}_{\mathsf{AC}}.\mathsf{Setup}$ as given in Algorithm 4.1 except that the key for the guessed label in the verification poset, $\kappa_{\lambda^V(o)}$ is implicitly set to be $\kappa^\star$ (i.e. any subsequent use of $\kappa^\star$ will be simulated using oracle queries to $\mathcal{C}$), and the KAS is constructed in a manner consistent with this choice. Note that in the authorised verification game (and by extension **Game 1**) the adversary may not query any label that is an ancestor of the challenge label $\lambda^V(o_b)$ in the verification poset. Thus $\mathcal{A}_{SE}$ can instantiate a KAS over the remaining nodes and simulate the public information for the ancestor set — as keys for this set cannot be derived, the public information need not be functionally correct. $\mathcal{A}_{SE}$ can also use the security parameter to generate remaining keys in this set. From the adversarial point of view, this is indistinguishable from the real games.

4. $\mathcal{A}_{SE}$ sends the resulting public parameters to $\mathcal{A}_{VC}$ and provides oracle access as follows:

   - For queries to FnInit, $\mathsf{Certify}$, and Revoke, $\mathcal{A}_{SE}$ can simply call the relevant algorithm.
   - If a $\mathsf{Register}$ query is made for a label $\lambda(ID)$ such that $\lambda(ID) \geqslant \lambda^V(o_0)$ and $\lambda(ID) \geqslant \lambda^V(o_1)$ then $\mathcal{A}_{SE}$ aborts the game as $\mathcal{A}_{VC}$ will not be able to choose $\lambda^V(o_b)$ as one of its challenge computations, and hence $\mathcal{A}_{SE}$'s guess was incorrect. Otherwise, $\mathcal{A}_{SE}$ holds the relevant KAS keys and may respond by running Oracle 4.5.
   - A $\mathsf{ProbGen}$ query for a computation labelled $\lambda^V(o)$ can be handled by running Algorithm 4.5 with the exception of lines 3 and 6 which are simulated by making an oracle query to the IND-CPA LoR oracle for the message pair $m_0 = m_1 = RK'_o$. For all other queries, $\mathcal{A}_{SE}$ holds the correct keys and may honestly run Algorithm 4.5.
   - $\mathsf{Compute}$ queries can be handled by running Algorithm 4.6 since it relies only on the computation poset which is owned by $\mathcal{A}_{SE}$.

5. Eventually, $\mathcal{A}_{VC}$ chooses two challenge computations o_0 and o_1 and, if neither is the same as that chosen earlier by $\mathcal{A}_{SE}$, the game is aborted. Otherwise, the choices are

valid since the game has not already been aborted during the oracle queries. Instead of choosing the bit b at random, it can be set such that o_b corresponds to $\mathcal{A}_{SE}$'s guess of challenge computation.

6. $\mathcal{A}_{SE}$ now simulates registering two entities: a delegator C and a server S. However, as these will not be required in the following, $\mathcal{A}_{SE}$ may simulate registering the delegator with label $\lambda^C(o_b)$ and update the public parameters accordingly without requiring valid keys for the delegator (as the adversary will not see any output from the delegator other than being listed in the public parameters). For the server, $\mathcal{A}_{SE}$ runs Register and sets $SK_S = (SK'_S, \kappa^\star, \perp)$ $\mathcal{A}_{SE}$ can also run the FnInit and Certify algorithms as written.

7. $\mathcal{A}_{SE}$ now runs ProbGen for the challenge computation o_b to encode σ_{o_b}. It runs lines 1, 2, 4 and 5 as written to generate a problem encoding σ_o and retrieval key RK'_o. It sets $m_0 = RK'_o$ and randomly chooses another message m_1 of the same length from the message space. These are given as input to the LoR oracle which returns the encryption of message m_β for the challenger's random choice of β. Using this response, $\mathcal{A}_{SE}$ can create RK_{o_b}.

8. $\mathcal{A}_{SE}$ must finally run Compute on the resulting encoded input, which it can do honestly as the algorithm does not rely on the verification poset. $\mathcal{A}_{VC}$ then receives all relevant information and oracle access, which is again handled as before. Eventually, $\mathcal{A}_{VC}$ outputs a guess b' of b.

Observe that if $\mathcal{A}_{SE}$ guessed the challenge label correctly, and $\beta = 0$, then this is precisely **Game 1**, and $\mathcal{A}_{VC}$ wins with non-negligible probability δ. Thus $\mathcal{A}_{SE}$ wins with non-negligible probability $\frac{2\delta}{N}$. If $\beta = 1$, on the other hand, the encoded input provided to $\mathcal{A}_{VC}$ is the encryption of a random message completely unrelated to the retrieval key. In this case, by the blind verification property of the underlying RPVC construction, $\mathcal{A}_{VC}$ may only have a negligible advantage ϵ (over random guessing) at learning the result of the computation. Thus, if $\mathcal{A}_{VC}$ outputs $b' = b$, then $\mathcal{A}_{SE}$ should output a guess of $\beta' = 0$, and otherwise should guess $\beta' = 1$. If $\beta = 0$ then $\mathcal{A}_{SE}$ wins with probability $\frac{2\delta}{N}$, and if $\beta = 1$ then $\mathcal{A}_{SE}$ wins with probability $1 - \epsilon$. Thus, $\mathcal{A}_{SE}$ wins in both cases with non-negligible probability. However, as the symmetric encryption scheme was assumed to be IND-CPA secure, an adversary $\mathcal{A}_{VC}$ with non-negligible advantage cannot exist.

The overall advantage against the authorised verification game is the sum of the distinguishing advantage between **Game 0** and **Game 1**, and the advantage in the reduction to IND-CPA, both of which we have shown to be negligible. Therefore, the overall advantage against the authorised verification game is negligible.

□

A.3 Proof of Weak Input Privacy

Lemma A.3. *Given a secure RPVC scheme, a KAS secure in the sense of strong-key indistinguishability and an* IND-CPA *secure symmetric encryption scheme, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 4.1–4.9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of weak input privacy (Game 4.4).*

Proof. Assume that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ in the weak input privacy game. We show that we can use this to construct an adversary, $\mathcal{A}_{SE}$, with non-negligible advantage in the IND-CPA game. Let $\mathcal{C}$ be the challenger for $\mathcal{A}_{SE}$ and let $\mathcal{A}_{SE}$ act as the challenger for $\mathcal{A}_{VC}$. We first transition to a modified version of the weak input privacy game, and show that $\mathcal{A}_{VC}$ has negligible distinguishing advantage between these games. We then construct $\mathcal{A}_{SE}$ which uses $\mathcal{A}_{VC}$ against this modified game to break the IND-CPA security of the symmetric encryption scheme.

- **Game 0:** This is the weak input privacy game as given in Game 4.4.
- **Game 1:** This is identical to **Game 0** except that the key $\kappa_{\lambda^C(o_b)}$ for the challenge computation o_b is replaced by a random key $\kappa^\star$ drawn uniformly from the keyspace.

Game 0 to Game 1 This game hop relies on the strong-key indistinguishability of the KAS and is very similar to that given in the proof of authorised outsourcing in Lemma 4.1; as such, we do not replicate this proof in full here. Intuitively, suppose an adversary $\mathcal{A}_{VC}$ exists that can distinguish **Game 0** from **Game 1** with non-negligible advantage ϵ. Then we construct an adversary $\mathcal{A}_{KI}$ that breaks the S-KI security of the KAS also with advantage ϵ. A challenger $\mathcal{C}$ will choose either **Game 0** or **Game 1** and interact with

$\mathcal{A}_{KI}$ in the S-KI game; $\mathcal{A}_{KI}$ will act as the challenger in either **Game 0** or **Game 1** for $\mathcal{A}_{VC}$ who must guess which game he is playing. $\mathcal{A}_{KI}$ will choose one of $\mathcal{A}_{VC}$'s challenge computation choices at random (as expected in the weak input privacy game) and forward this to $\mathcal{C}$ as its own challenge in the S-KI game.

Reduction to IND-CPA Given that no adversary can distinguish between **Game 0** and **Game 1** with non-negligible advantage, we may run an adversary against **Game 0** against **Game 1** instead with at most a negligible loss ϵ in the tightness of the reduction. That is, we can use a truly random key to form the challenge ciphertext which removes any information leakage from the KAS. Let us now suppose, for a contradiction, that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ against **Game 1**. We construct an adversary $\mathcal{A}_{SE}$ that breaks the IND-CPA security of the symmetric encryption scheme using $\mathcal{A}_{VC}$ as a subroutine. Let $\mathcal{C}$ be the IND-CPA challenger for $\mathcal{A}_{SE}$ who in turn acts as the challenger for $\mathcal{A}_{VC}$.

1. $\mathcal{C}$ chooses a bit $\beta \xleftarrow{\$} \{0,1\}$ and generates a key $\kappa^\star$ by running $\mathsf{SE.KeyGen}$. It sends the security parameter to $\mathcal{A}_{SE}$ and provides oracle access to the LoR function.

2. $\mathcal{A}_{SE}$ must now initialise **Game 1** for $\mathcal{A}_{VC}$. Informally, it will set the challenge KAS key $\kappa_{\lambda^C(o_\beta)}$ to be the random $\kappa^\star$ chosen by $\mathcal{C}$. However, as this label is currently unknown and the public parameters and oracle access must be granted before the choice is made, $\mathcal{A}_{SE}$ must make a guess, $\widetilde{\lambda^C}(o_\beta)$, for the correct challenge label to assign the random key to. This choice is correct with probability at least $\frac{1}{N}$ where the computation poset comprises N labels and where N is polynomial in the security parameter (to enable efficient instantiation).

3. $\mathcal{A}_{SE}$ initialises the parameters L and o, and also initialises an empty list Q that will be used to store messages queried to the LoR oracle in the IND-CPA game. It runs Setup as written with the following modification. The KAS key for the guessed challenge label $\widetilde{\lambda^C}(o_\beta)$ is implicitly set to be $\kappa^\star$ and the rest of the computational KAS is made consistent with this choice. Since the adversary in **Game 1** is not permitted to query the $\mathsf{Register}$ function for any label that is an ancestor of the challenge label, keys for ancestors of the challenge label will not need to be derivable and can simply be generated from $\mathsf{SE.KeyGen}$. Therefore, a KAS can simply be instantiated over the remaining (non-ancestor) nodes and the public information

for ancestor nodes can be simulated (as keys will not be derived, this need not be functionally correct but must appear to be distributed correctly).

4. $\mathcal{A}_{VC}$ is given the resulting public parameters and access to oracle functionality which $\mathcal{A}_{SE}$ responds to as follows:

   - Queries to FnInit, Certify and Revoke can be handled by simply running the relevant algorithms.
   - If $\mathcal{A}_{VC}$ queries Register for a label $\lambda(ID) \geqslant \widetilde{\lambda^C}(o_\beta)$, as guessed by $\mathcal{A}_{SE}$, then $\mathcal{A}_{SE}$ will abort the game (since $\mathcal{A}_{VC}$ would no longer be able to choose $\widetilde{\lambda^C}(o_\beta)$ as a valid challenge label and hence $\mathcal{A}_{SE}$'s guess was incorrect). For any other Register query, $\mathcal{A}_{SE}$ has generated and holds the relevant KAS keys and may respond honestly.
   - A query to ProbGen for anything other than a computation labelled with $\widetilde{\lambda^C}(o_\beta)$ can be run honestly using the keys he generated during Setup. A computation labelled by $\widetilde{\lambda^C}(o_\beta)$ will require an encryption of σ_o under $\kappa_{\widetilde{\lambda^C}(o_\beta)}$ which is not known to $\mathcal{A}_{SE}$ as it is the challenge key in the IND-CPA game. Therefore, $\mathcal{A}_{SE}$ must make a query to the LoR oracle provided by $\mathcal{C}$ where $m_0 = m_1 = \sigma_o$ to receive a valid ciphertext CT. $\mathcal{A}_{SE}$ also adds the pair (σ_o, CT) to the list Q.
   - By the restriction on Compute oracle queries, $\perp$ is returned if $\mathcal{A}_{VC}$ queries for the challenge inputs. By the INT-PTXT property of the symmetric encryption scheme, $\mathcal{A}_{VC}$ is unable to form a valid ciphertext for the challenge label without making use of the Encrypt oracle. Thus, if the input to the Compute oracle is for the challenge computation label, then the ciphertext is either malformed (and $\perp$ should be returned), or the encoded input was previously queried to the LoR oracle and $\mathcal{A}_{SE}$ may look up the received ciphertext in the list Q to recover the encrypted input σ_o. For any other computation label, $\mathcal{A}_{SE}$ holds the corresponding KAS keys and can run the Compute algorithm honestly.

5. $\mathcal{A}_{VC}$ eventually outputs a choice of two computations o_0 and o_1. If neither computation matches with the label $\widetilde{\lambda^C}(o_\beta)$ chosen by $\mathcal{A}_{SE}$ earlier, then the game is aborted. Otherwise, $\mathcal{A}_{SE}$ will use the matching computation henceforth. It first checks the labels for validity and aborts if not valid.

6. $\mathcal{A}_{SE}$ must now register a delegator C. However, the role of C will be substituted by oracle queries to $\mathcal{C}$ in the following, so $\mathcal{A}_{SE}$ may simply simulate registering C with

label $\widetilde{\lambda^C}(o_\beta)$ by updating any relevant public information. $\mathcal{A}_{SE}$ can also run FnInit as written.

7. $\mathcal{A}_{SE}$ must now run ProbGen to generate the challenge input. To do so, it runs RPVC.ProbGen on both inputs x_0 and x_1 dictated by o_0 and o_1 respectively, to generate two encoded inputs labelled $m_0 = \sigma_{o_0}$ and $m_1 = \sigma_{o_1}$. It then submits m_0 and m_1 to the IND-CPA LoR oracle provided by $\mathcal{C}$. $\mathcal{C}$ will return the encryption of m_β corresponding to σ_{o_β}, under the key $\kappa_{\lambda^C(o_\beta)}$. $\mathcal{A}_{SE}$ can also encrypt the verification key VK_{o_β} for the o_β matching its guess, using the verification KAS keys that it owns.

8. The (encrypted) encoded input and the verification key are given to $\mathcal{A}_{VC}$ along with oracle access. Queries are handled as above but Register now returns $\perp$ if the queried label is an ancestor of either $\lambda^C(o_0)$ or $\lambda^C(o_1)$ (i.e. precisely when $\mathcal{A}_{SE}$ does not hold the relevant KAS keys). $\mathcal{A}_{VC}$ eventually outputs a guess b' that $o_{b'}$ was chosen.

Now, $\mathcal{A}_{SE}$ can simply forward the guess b' to $\mathcal{C}$ as its guess for β. Observe that if $\mathcal{A}_{SE}$ correctly guessed $\widetilde{\lambda^C}(o_\beta)$, then from $\mathcal{A}_{VC}$'s point of view the distribution of the above game is precisely that of **Game 1**. Thus, if $\mathcal{A}_{VC}$ can successfully distinguish which computation was chosen (by $\mathcal{C}$), then it implicitly can decide which plaintext (encoded input) was encrypted during the IND-CPA game. $\mathcal{A}_{SE}$ can correctly guess the challenge label that will be chosen by $\mathcal{C}$ with probability at least $\frac{1}{N}$. Thus, since we assumed that $\mathcal{A}_{VC}$ had non-negligible advantage δ against **Game 1**, we have shown how to construct an adversary $\mathcal{A}_{SE}$ with non-negligible advantage $\frac{\delta}{N}$ against the IND-CPA game. However, as the symmetric encryption scheme is assumed to be IND-CPA secure, such an adversary against **Game 1** may exist, and since there is a negligible distinguishing advantage between **Game 0** and **Game 1**, no adversary with non-negligible advantage against the weak input privacy game may exist either.

□

Appendix B

Additional Material for Hybrid Publicly Verifiable Outsourced Computation

This appendix includes additional material corresponding to our Hybrid PVC scheme introduced in Chapter 6. Firstly, we give a construction and security proof for our notion of revocable dual-policy attribute-based encryption, upon which our HPVC construction is based. Then, we give security notions and proofs for our HPVC instantiation, which in general are formed by combining those of the previous chapters and adapting the notation to our more general format.

B.1 Construction of Revocable Dual-policy Attribute-based Encryption

Our revocable DP-ABE scheme will be based on a combination of DP-ABE [16], which itself is a combination of CP-ABE [92] and KP-ABE [66], and an ABE scheme supporting revocation [14]. We represent a subjective access structure $\mathbb{S}$ by a linear secret sharing scheme (LSSS, see Appendix 2.6.4.1) which we denote by (M, ρ) and represent an objective access structure $\mathbb{O}$ as an LSSS denoted by (N, π).

Let $\mathcal{U}_s$ and $\mathcal{U}_o$ be the universe of subjective and objective attributes respectively. The objective attribute universe comprises disjoint sub-universes $\mathcal{N}, \mathcal{T}, \mathcal{M}$ and $\mathcal{U}_{\text{ID}}$ referring to standard ABE attributes, time periods, messages and user 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 $\mathcal{T} \cap \mathcal{X} = \emptyset$ (note that this can be achieved 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 DP-ABE scheme is defined to be $\mathcal{U} = \mathcal{U}_s \cup \mathcal{U}_o$. 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 $R \subseteq \mathcal{U}_{\mathrm{ID}}$, where $\mathsf{Cover}(R)$ is defined as in Appendix 2.1.1.

We construct each algorithm of the rkDPABE scheme as follows:

- $\mathsf{Setup}(1^\ell, \mathcal{U})$: The algorithm first picks a random generator $g \in \mathbb{G}$ and random exponents $\gamma, \alpha \in \mathbb{Z}_p$. It then 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 first randomly choosing $h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d$ and setting

$$F_s(x) = \prod_{j=0}^{m'} h_j^{x^j}, \quad F_o(x) = \prod_{j=0}^{n'} q_j^{x^j}, \quad P(x) = \prod_{j=0}^{d} u_j^{x^j}. \tag{B.1}$$

  The public parameters are defined as

$$\mathrm{PP} = (g, e(g,g)^\gamma, g^\alpha, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d).$$

  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 $\mathrm{MK} = (\gamma, \alpha, \{a_x, r_x\}_{x \in \mathcal{X}})$.

- $\mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, \mathrm{PP})$: 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 cipher-

text is then computed as $CT = (C, C^{(1)}, \{C_k^{(2)}\}_{k\in\omega}, \{C_i^{(3)}\}_{i=1,\ldots,l_s}, C^{(4)})$, where

$$C = m \cdot (e(g,g)^{\gamma})^s, \qquad C^{(1)} = g^s,$$
$$C_k^{(2)} = F_o(k)^s, \qquad C_i^{(3)} = g^{\alpha\lambda_i} F_s(\rho(i))^{-s},$$
$$C^{(4)} = P(t)^s.$$

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 (although, clearly, without revealing the value itself, unless one can solve the discrete logarithm problem which is thought to be hard). $C_k^{(2)}$ embeds each attribute in the objective set ω into the ciphertext, incorporating the encryption secret s such that attributes from prior ciphertexts cannot be used to break the security of this encryption. Similarly, $C_i^{(3)}$ embeds the subjective policy $\mathbb{S}$ into the ciphertext using the shares of s divided according to $\mathbb{S}$ — that is, s is shared over the set of attributes such that any set of attributes that satisfies $\mathbb{S}$ can be used to 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.

- $\mathsf{KeyGen}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PP})$: The key generation algorithm takes as input a LSSS access structure (N, π) for the objective policy $\mathbb{O}$ and a subjective attribute set $\psi \subset \mathcal{U}_s$. Let the dimensions of N be denoted $l_o \times k_o$. The algorithm also takes an identity $\mathrm{ID} \in \mathcal{U}$ which is a leaf in the binary tree.

  For all $x \in \mathsf{Path}(\mathrm{ID})$, the algorithm first 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 $x \in \mathsf{Path}(\mathrm{ID})$, and outputs the private key

$$SK_{(N,\pi),\mathrm{ID}} = ((D_{x,i}^{(1)}, D_{x,i}^{(2)})_{x\in\mathsf{Path}(\mathrm{ID}),\mathrm{i}=1,\ldots,\mathrm{l_o}}, (D_x, \{D_k^{(3)}\}_{k\in\psi})_{x\in\mathsf{Path}(\mathrm{ID})}),$$

where

$$D_x = g^{r_x}, \qquad D_{x,i}^{(1)} = g^{r_{x,i}},$$
$$D_{x,i}^{(2)} = g^{\sigma_{x,i}} F_o(\pi(i))^{r_{x,i}}, \qquad D_k^{(3)} = F_s(k)^{r_x}.$$

Intuitively, r_x and $r_{x,i}$ for each $x \in \mathsf{Path}(\mathrm{ID})$ randomises the key for the user ID (so that users may not collude, as their key components shall be formed using different random values). D_x and $D_{x,i}^{(1)}$ allow use of these random key values during decryption. $D_{x,i}^{(2)}$ embeds the shares of $f_x(1) = a_x + \alpha r_x + \gamma$, distributed over the attribute universe such that only the authorised sets according to $\mathbb{O}$ will be able to reconstruct $f_x(1)$. Finally, $D_k^{(3)}$ 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.

- $\mathsf{KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP})$: 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 $UK(R,t) = \left\{U_x^{(1)}, U_x^{(2)}\right\}_{x \in \mathsf{Cover}(R)}$, where

$$U_x^{(1)} = g^{f_x(t)} P(t)^{r_x}, \quad U_x^{(2)} = g^{r_x}.$$

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). The use of $P(t)$ is to embed the current time period and 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 $D_{x,i}^{(2)}$ 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.

- $\mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), SK_{(\mathbb{O},\psi),\mathrm{ID}}, (\mathbb{O}, \psi), UK_{R,t}, \mathrm{PP})$: The decryption algorithm takes as an input the ciphertext CT which contains a subjective access structure (M, ρ) for $\mathbb{S}$ and a set of objective attributes ω, and a decryption key $SK_{(N,\pi),\mathrm{ID}}$ which contains a set of subjective attributes ψ and an objective access structure (N, π) for $\mathbb{O}$. Suppose that the set ψ for subjective attribute satisfies (M, ρ), the set ω satisfies (N, π), and that $\mathrm{ID} \notin \mathrm{R}$ (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 ID $\notin$ R, the algorithm also finds a node x such that $x \in \mathsf{Path}(\mathrm{ID}) \cap \mathsf{Cover}(\mathrm{R})$. Finally, it computes the following

$$C \cdot \frac{\prod_{i \in I_s} \left(e(C_i^{(3)}, D_x) \cdot e(C^{(1)}, D_{\rho(i)}^{(3)}) \right)^{\mu_i}}{\left(\prod_{j \in I_o} \left(\frac{e(D_{x,j}^{(2)}, C^{(1)})}{e(C_{\pi(j)}^{(2)}, D_{x,j}^{(1)})} \right)^{\nu_j} \right)^{\frac{t}{t-1}} \left(\frac{e(U_x^{(1)}, C^{(1)})}{e(C^{(4)}, U_x^{(2)})} \right)^{\frac{1}{1-t}}} = m.$$

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 the 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 cancel the result of computing C' whilst the first part will cancel with C to leave only the message m, as required.

$$\begin{aligned}
C' &= \prod_{i \in I_s} \left(e(C_i^{(3)}, D_x) \cdot e(C^{(1)}, D_{\rho(i)}^{(3)}) \right)^{\mu_i} \\
&= \prod_{i \in I_s} \left(e(g^{\alpha \lambda_i} F_s(\rho(i))^{-s}, g^{r_x}) \cdot e(g^s, F_s(\rho(i))^{r_x}) \right)^{\mu_i} \\
&= \prod_{i \in I_s} \left(e(g,g)^{\alpha \lambda_i r_x} \cdot e(g, F_s(\rho(i)))^{-r_x s} \cdot e(g, F_s(\rho(i)))^{r_x s} \right)^{\mu_i} \\
&= e(g,g)^{\alpha r_x \sum_{i \in I_s} \mu_i \lambda_i} \\
&= e(g,g)^{\alpha r_x s}.
\end{aligned}$$

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 properties of the

reconstruction constants of the LSSS, namely that $\sum_{i \in I_s} \mu_i \lambda_i = s$.

$$\begin{aligned}
K' &= \prod_{j \in I_o} \left(\frac{e(D^{(2)}_{x,j}, C^{(1)})}{e(C^{(2)}_{x,\pi(j)}, D^{(1)}_{x,j})} \right)^{\nu_j} \\
&= \prod_{j \in I_o} \left(\frac{e(g^{\sigma_{x,j}} F_o(\pi(j))^{r_{x,j}}, g^s)}{e(F_o(\pi(j))^s, g^{r_{x,j}})} \right)^{\nu_j} \\
&= \prod_{j \in I_o} \left(\frac{e(g,g)^{\sigma_{x,j} s} \cdot e(g, F_o(\pi(j)))^{r_{x,j}, s}}{e(g, F_o(\pi(j)))^{r_{x,j}, s}} \right)^{\nu_j} \\
&= e(g,g)^{s \sum_{j \in I_o} \nu_j \sigma_{x,j}} \\
&= e(g,g)^{s f_x(1)}.
\end{aligned}$$

The second equality follows directly from the construction; the third equality follows from the properties of bilinear maps; the fourth equality stems from moving the product into the exponent; and the last equality follows from the set of LSSS reconstruction constants with $\sum_{j \in I_o} \nu_j \sigma_{x,j} = f_x(1) = a_x + \alpha r_x + \gamma$.

$$\begin{aligned}
K'' &= \frac{e(U^{(1)}_x, C^{(1)})}{e(C^{(4)}, U^{(2)}_x)} \\
&= \frac{e(g^{f_x(t)} P(t)^{r_x}, g^s)}{e(P(t)^s, g^{r_x})} \\
&= \frac{e(g,g)^{f_x(t) s} \cdot e(g, P(t)^{r_x s})}{e(g, P(t)^{r_x s}} \\
&= e(g,g)^{f_x(t) s}
\end{aligned}$$

Then,

$$\begin{aligned}
K &= (K')^{\frac{t}{t-1}} (K'')^{\frac{1}{1-t}} \\
&= (e(g,g)^{s f_x(1)})^{\frac{t}{t-1}} (e(g,g)^{f_x(t) s})^{\frac{1}{1-t}} \\
&= (e(g,g)^s)^{f_x(1) \frac{t}{t-1} + f_x(t) \frac{1}{1-t}}
\end{aligned}$$

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 (see Appendix 2.6.4.1). 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

$$C \cdot \frac{C'}{K} = m \cdot e(g,g)^{s\gamma} \cdot \frac{e(g,g)^{\alpha s r_x}}{e(g,g)^{s(\alpha r_x + \gamma)}} = m \cdot e(g,g)^{s\gamma} \cdot \frac{e(g,g)^{\alpha s r_x}}{e(g,g)^{s\gamma} \cdot e(g,g)^{\alpha s r_x}} = m.$$

B.1.1 Proof of Security

Theorem B.1. *The rkDPABE construction presented in Appendix B.1 is secure with respect to Indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*), as presented in Game 6.1, assuming that the Decision q-BDHE problem is hard.*

The proof follows the methodology from a combination [14] and [15] with some adjustment in the simulation of the private keys. We show that if an adversary can break the rkDPABE scheme with advantage ϵ in the IND-sHRSS game with a challenge subjective access structure matrix of size $l_s^\star \times k_s^\star$, then a simulator with advantage ϵ in solving the Decision q-BDHE problem can be constructed, where $m + k_s^\star \leqslant q$.

Proof. Suppose, to achieve a contradiction with Theorem B.1, 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 2.21) in $\mathbb{G}$. Recall that we denote 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 $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ where 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 an 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 $\gamma' \xleftarrow{\$} \mathbb{Z}_p$ and implicitly defines $\gamma = \gamma' + a^{q+1}$ by defining

$$\begin{aligned} e(g,g)^{\gamma} &= e(g_1, g_q) \cdot e(g,g)^{\gamma'} \\ &= e(g^a, g^{a^q}) \cdot e(g,g)^{\gamma'} \\ &= e(g,g)^{\gamma' + a^{q+1}}. \end{aligned}$$

   It also sets $g^\alpha = g_1 = g^a$.

   It then must define the polynomials F_s, F_o and P (as in [14] and [15]).

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):

$$p_j(x) = \begin{cases} M^\star_{i,j} & \text{for } j \in [1, k_s^\star] \\ 0 & \text{for } j \in [k_s^\star + 1, k_s^\star + m] \end{cases} \tag{B.2}$$

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

$$p(x) = \sum_{j=0}^{k_s^\star+m} p_j(x) a^j. \tag{B.3}$$

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 $m' = m + l_{s,\mathbf{max}} - 1 = m + l_s^\star - 1$,

$$\begin{aligned} F_s(x) &= \prod_{i=0}^{m'} h_i^{x^i} && \text{(by B.1)} \\ &= \prod_{i=0}^{m'} \left(\prod_{j=0}^{k_s^\star+m} g_j^{p_{j,i}} \right)^{x^i} && \text{(by definition of } h_i^{x^i}\text{)} \\ &= \prod_{i=0}^{m'} \left(\prod_{j=0}^{k_s^\star+m} g^{p_{j,i} a^j} \right)^{x^i} && \text{(by definition of } g_j = g^{a^j}\text{)} \\ &= g^{\sum_{j=0}^{k_s^\star+m} \sum_{i=0}^{m'} p_{j,i} x^i a^j} = g^{\sum_{j=0}^{k_s^\star+m} p_j(x) a^j} \\ &= g^{p(x)} && \text{(by B.3)} \end{aligned}$$

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

$j = [0, n-1]$. Finally,

$$F_o(x) = \prod_{j=0}^{n-1} q_j^{(x^j)} = g_q^{f(x)} g^{f'(x)}.$$

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,

$$P(y) = \prod_{j=0}^{d} u_j^{y^j} = (g^a)^{\hat{p}(y)} g^{\rho(y)}. \tag{B.4}$$

The public key PK for the DPABE scheme is defined to be $\text{PK} = (g, e(g,g)^\gamma, g^\alpha, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d)$, which is given to $\mathcal{A}$. Observe that due to 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, 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 KeyUpdate functions. Let $\mathcal{X}_{\overline{R}} = \{x \in \mathsf{Path}(\mathrm{ID}) : \mathrm{ID} \in \overline{\mathrm{R}}\}$. For each node label $x \in \mathcal{X}$ in the tree, $\mathcal{B}$ randomly chooses $a'_x \in \mathbb{Z}_p$ and implicitly defines

$$a_x = \begin{cases} a'_x - \alpha r_x - \gamma & \text{if } x \in \mathcal{X}_{\overline{R}} \\ a'_x - \frac{\alpha r_x - \gamma}{t^\star} & \text{if } x \notin \mathcal{X}_{\overline{R}} \end{cases} \tag{B.5}$$

Hence,

$$f_x(1) = a_x + \alpha r_x + \gamma = a'_x - \alpha r_x - \gamma + \alpha r_x + \gamma = a'_x \qquad \text{if } x \in \mathcal{X}_{\overline{R}} \tag{B.6}$$

$$f_x(t^\star) = a_x t^\star + \alpha r_x + \gamma = (a'_x - \frac{\alpha r_x - \gamma}{t^\star}) t^\star + \alpha r_x + \gamma = a'_x t^\star \qquad \text{if } x \notin \mathcal{X}_{\overline{R}} \tag{B.7}$$

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 $(\mathrm{ID} \in \overline{\mathrm{R}})$:

  For each $x \in \mathsf{Path}(\mathrm{ID})$, $\mathcal{B}$ does the following. First note that for all such x, since $\mathrm{ID} \in \overline{\mathrm{R}}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (B.6), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathrm{ID})$. $\mathcal{B}$ 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 $(\mathrm{ID} \in \overline{\mathrm{R}})$:

  For each $x \in \mathsf{Path}(\mathrm{ID})$, $\mathcal{B}$ does the following. First note that for all such x, since $\mathrm{ID} \in \overline{\mathrm{R}}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (B.6), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathrm{ID})$. $\mathcal{B}$ randomly choses $r_x \in \mathbb{Z}_p$. 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 2.1, 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 B.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 (B.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

$$D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}},$$

  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, such that $\pi(i) \notin \omega^\star$, $\mathcal{B}$ 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 [15], 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 $(\mathrm{ID} \notin \overline{\mathrm{R}})$:

For each $x \in \mathsf{Path}(\mathrm{ID})$, $\mathcal{B}$ does the following. Since ψ does not satisfy $M^\star$, by Proposition 2.1, 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 $M_i \cdot \boldsymbol{w_x} = 0$ for all i where $\rho(i) \in \psi^\star$. Now, by our definition of $p_j(x)$ in (B.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

$$r_x = r'_x + w_1(\frac{t^\star}{t^\star - 1}) \cdot \alpha^q + w_2(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-1} + \cdots + w_{k_s^\star+m}(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-(k_s^\star+m)+1}$$

by setting the key $D_x = g^{r'_x} \prod_{k=1}^{k_s^\star+m} (g_{q+1-k})^{w_k(\frac{t^\star}{t^\star-1})} = g^{r_x}$. Then, since $\gamma = \gamma' + \alpha^{q+1}$ and as $x \notin \mathcal{X}_{\overline{\mathcal{R}}}$, we have

$$\begin{aligned}
f_x(1) &= \gamma + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a'_x - \frac{\alpha r_x - \gamma}{t^\star} \quad \text{by (B.5)} \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + (\alpha(\frac{t^\star - 1}{t^\star}))r_x \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + (\alpha(\frac{t^\star - 1}{t^\star})(r'_x + w_1(\frac{t^\star}{t^\star - 1}) \cdot \alpha^q \\
&\quad + w_2(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-1} + \cdots + w_{k_s^\star+m}(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-(k_s^\star+m)+1}) \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + (\alpha r'_x + w_2(\frac{t^\star}{t^\star - 1}) \cdot \alpha^q + \cdots + w_{k_s^\star+m}(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-(k_s^\star+m)+2})
\end{aligned}$$

where the α^{q+1} term in γ has canceled out. The simulator now randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and implicitly lets $\boldsymbol{v_x} = (\gamma + \alpha r_x + a_x, z_{x,2}, \ldots, z_{x,k_o})$ as in the construction.

$\mathcal{B}$ also randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and computes for $i = 1$ to 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 $k \in \psi$.

$$\begin{aligned} D_k^{(3)} &= D_x^{p_0(k)} \prod_{j=1}^{k_s^\star+m} \left(g_j^{r'_x} \prod_{k \in [1, k_s^\star+m], k \neq j} (g_{q+1-k+j})^{w_k} \right)^{p_j(k)} \\ &= (g^{r_x})^{p_0(k)} \prod_{j=1}^{k_s^\star+m} (g^{r_x})^{\alpha^j p_j(k)} \\ &= (g^{r_x})^{p(k)} \\ &= F_s(k)^{r_x}, \end{aligned}$$

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 [15]).

- $(\omega^\star \notin \mathbb{O})$ and $(\psi \in \mathbb{S}^\star)$ and $(\mathrm{ID} \notin \overline{\mathrm{R}})$:

  For each $x \in \mathsf{Path}(\mathrm{ID})$, $\mathcal{B}$ does the following. $\mathcal{B}$ randomly choses $r_x \in \mathbb{Z}_p$. 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 2.1, 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 (B.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, such that $\pi(i) \notin \omega^\star$, $\mathcal{B}$ randomly chooses $r'_{x,i} \in \mathbb{Z}_p$. Observe

that

$$\begin{aligned}\boldsymbol{N_i}\cdot\boldsymbol{v_x} &= \boldsymbol{N_i}\cdot(-(a'_x-\frac{\alpha r_x-\gamma}{t^\star}+\alpha r_x+\gamma)\boldsymbol{a_x}+\boldsymbol{v'_x})\\ &= \boldsymbol{N_i}\cdot(-(a'_x-\frac{\alpha r_x-(\gamma'+a^{q+1})}{t^\star}+\alpha r_x+(\gamma'+a^{q+1}))\boldsymbol{a_x}+\boldsymbol{v'_x})\\ &= \boldsymbol{N_i}\cdot(\boldsymbol{v'_x}-(a'_x+\gamma'(\frac{1}{t^\star}+1))\boldsymbol{a_x})+(r_x(\frac{1}{t^\star}-1)\boldsymbol{N_i}\cdot\boldsymbol{a_x})\alpha\\ &\quad -((\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x})a^{q+1}\end{aligned}$$

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_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

$$D^{(2)}_{x,i}=g_1^{\left(r_x(\frac{1}{t^\star}-1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}-(\frac{1}{t^\star}+1)\frac{\boldsymbol{N_i}\cdot\boldsymbol{a_x}f'(\pi(i))}{f(\pi(i))}\right)}\cdot g^{\boldsymbol{N_i}\cdot(\boldsymbol{v'_x}-(a'_x+\gamma'(\frac{1}{t^\star}+1)))\boldsymbol{a_x}}F_o(\pi(i))^{r'_{x,i}}$$

To see that $D^{(2)}_{x,i}$ is valid, we observe

$$\begin{aligned}D^{(2)}_{x,i} &= g_{q+1}^{(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}\cdot D^{(2)}_{x,i}\cdot g_{q+1}^{-(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}\\ &= g_{q+1}^{(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}\cdot g_1^{r_x(\frac{1}{t^\star}-1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}\cdot g^{\boldsymbol{N_i}\cdot(\boldsymbol{v'_x}-(a'_x+\gamma'(\frac{1}{t^\star}+1)))\boldsymbol{a_x}}\\ &\quad\cdot\left(g_{q+1}^{-(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}g_1^{-(\frac{1}{t^\star}+1)\frac{\boldsymbol{N_i}\cdot\boldsymbol{a_x}f'(\pi(i))}{f(\pi(i))}}\right)\cdot F_o(\pi(i))^{r'_{x,i}}\\ &= g^{\boldsymbol{N_i}\cdot\boldsymbol{v_x}}\left(g_q^{f(\pi(i))}g^{f'(\pi(i))}\right)^{\frac{-a(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}}\cdot F_o(\pi(i))^{r'_{x,i}}\\ &= g^{\boldsymbol{N_i}\cdot\boldsymbol{v_x}}\cdot F_o(\pi(i))^{\frac{-a(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}}\cdot F_o(\pi(i))^{r'_{x,i}}\text{ by (B.4)}\\ &= g^{\boldsymbol{N_i}\cdot\boldsymbol{v_x}}\cdot F_o(\pi(i))^{r_{x,i}}\end{aligned}$$

$\mathcal{B}$ also defines

$$D^{(1)}_{x,i}=g^{r'_{x,i}}g_1^{-\frac{(\frac{1}{t^\star}+1)\boldsymbol{N_i}\cdot\boldsymbol{a_x}}{f(\pi(i))}}=g^{r_{x,i}}$$

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 $\mathsf{KeyUpdate}$ 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 $U^{(1)}_x=$

$(g^{a'_x t^\star})P(t^\star)^{r_x}$ and $U_x^{(2)} = g^{r_x}$. Both keys are valid because since $\overline{R} \subseteq R$ and thus for all $x \in \mathsf{Cover}(R)$ we have $x \notin \mathcal{X}_{\overline{R}}$. Hence, by (B.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 (B.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,

$$
\begin{aligned}
U_x^{(1)} &= g^{f_x(t)} P(t)^{r_x} \text{ by the construction} \\
&= g^{a'_x t + a r_x (1-t) + \gamma'(1-t) + a^{q+1}(1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \text{ by } f_x(t) \text{ and (B.4)} \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a r_x (1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{a\hat{p}(t) r'_x} g^{-a\hat{p}(t)B} g^{\rho(t) r'_x} g^{-\rho(t)B} \\
&\qquad \text{by } r_x = r'_x - B \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{-a\hat{p}(t)B} g^{-\rho(t)B} \text{ by (B.4)} \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)B} g^{-B a(1-t) + a\hat{p}_t} \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} (g^a)^{-(\frac{a^q(1-t)}{\hat{p}(t)+1-t})^{(1-t)+\hat{p}_t}} \\
&\qquad \text{by } B = \frac{a^q(1-t)}{\hat{p}(t)+1-t} \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} (g^a)^{-(a^q(1-t))} \\
&= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} g^{-a^{q+1}(1-t)} \\
&= g^{a'_x t} g^{\gamma'(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} \\
&= (g^{a'_x})^t (g^{\gamma'})^{(1-t)} (g_1^{r'_x})^{(1-t)} g_q^{-\frac{\rho(t)(1-t)}{\hat{p}(t)+1-t}} P(t)^{r'_x} \text{ as we defined}
\end{aligned}
$$

– 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 (B.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,

$$C = m_b \cdot Z \cdot e(h, g^{\gamma'}) = m_b \cdot e(g_{q+1}, h) \cdot e(h, g^{\gamma'}) = m_b \cdot e(g, g)^{sa^{q+1}} \cdot e(g, g)^{s\gamma'}$$
$$= m_b \cdot e(g, g)^{s(\gamma' + a^{q+1})} = m_b \cdot e(g, g)^{s\gamma}.$$

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 $\hat{p}(t^\star) = 0$, $C^{(4)} = (g^s)^{\rho(t^\star)} = ((g^a)^{\hat{p}(t^\star)} g^{\rho(t^\star)})^s = P(t^\star)^s$.

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 in $\mathbb{G}_T$.

If $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{R}_{BDHE}$ then $\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) = 0] = \frac{1}{2}$ 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, $|\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) = 0] - \frac{1}{2}| \geq \epsilon$. It follows that $\mathcal{B}$ has advantage at least ϵ in solving q-BDHE problem in $\mathbb{G}$. Hence, as we assume that the decisional q-BDHE problem is hard, an adversary with non-negligible advantage in the IND-sHRSS game cannot exist.

□

Game B.1 $\mathbf{Exp}_{\mathcal{A}}^{\text{sPubVerif}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(\omega^{\star}, \mathbb{O}^{\star}, \psi^{\star}, \mathbb{S}^{\star}, L_{F,X^{\star}}, \texttt{mode}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{F})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
3: **if** $((\texttt{mode} = VDC))$ **then** $(F \leftarrow \mathbb{S}^{\star}, X^{\star} \leftarrow \psi^{\star})$
4: **else** $(F \leftarrow \mathbb{O}^{\star}, X^{\star} \leftarrow \omega^{\star})$
5: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
6: $(\sigma^{\star}, VK^{\star}, RK^{\star}) \xleftarrow{\$} \mathsf{ProbGen}(\texttt{mode}, (\omega^{\star}, \mathbb{S}^{\star}), L_{F,X^{\star}}, PK_F, \text{PP})$
7: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^{\star}, VK^{\star}, RK^{\star}, PK_F, \text{PP})$
8: $(y, \tau_{\theta^{\star}}) \leftarrow \mathsf{Verify}(\theta^{\star}, VK^{\star}, RK^{\star}, \text{PP})$
9: **if** $(((y, \tau_{\theta^{\star}}) \neq (\perp, (\mathsf{reject}, \cdot))))$ **and** $(y \neq F(X^{\star})))$ **then**
10: **return** 1
11: **else return** 0

B.2 Security Models

In this section we discuss some security models which are of interest in HPVC, namely modified versions of *public verifiability*, *revocation* and *authorised computation*. These notions are motivated by the same scenarios considered in the previous chapters and, as in Chapter 3, we must include some additional restrictions on the public verifiability and revocation games in order to accommodate similar restrictions stemming from our current rkDPABE primitive. For brevity, we do not discuss these restrictions in detail here, but refer the reader to the discussion in Section 3.4.2. We also do not provide ideal notions of each game, as we did in Chapter 3 for RPVC, as they can be easily seen by adapting the notation in the ideal notions for RPVC to accommodate the additional HPVC parameters. Notions for vindictive servers and vindictive managers can also be easily adapted from Chapter 3 if desired.

B.2.1 Selective Public Verifiability

In Game B.1, we combine the public verifiability games of Chapters 3 and 5, to formalise in the HPVC model that no server may return an incorrect result for a computation without being detected. We allow the adversary to corrupt other servers, generate arbitrary computations, and to perform verification steps himself.

As with Game 3.7, this is a *selective* notion and, as such, the adversary first selects its challenge parameters, including the mode it wishes its challenge to be generated in and the labels associated to its choice of inputs. Note that we ask the adversary to output choices for $\omega^{\star}, \mathbb{O}^{\star}, \psi^{\star}$ and $\mathbb{S}^{\star}$, despite only $\omega^{\star}$ and $\mathbb{S}^{\star}$ forming the challenge input. This

Game B.2 $\mathbf{Exp}_{\mathcal{A}}^{\text{sSS-Rev}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}, q_t\right]$

1: $(\omega^{\star}, \mathbb{O}^{\star}, \psi^{\star}, \mathbb{S}^{\star}, L_{F,X^{\star}}, \texttt{mode}^{\star}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{F}, q_t)$
2: **if** $((\texttt{mode}^{\star} = VDC))$ **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: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
7: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
8: $\overline{R} \xleftarrow{\$} \mathcal{A}(PK_F, \text{PP})$
9: $\mathcal{A}^{\mathcal{O}}(PK_F, \text{PP})$
10: **if** $(\overline{R} \not\subseteq Q_{\text{Rev}})$ **then return** 0
11: $(\sigma^{\star}, VK^{\star}, RK^{\star}) \xleftarrow{\$} \mathsf{ProbGen}(\texttt{mode}^{\star}, (\omega^{\star}, \mathbb{S}^{\star}), L_{F,X^{\star}}, PK_F, \text{PP})$
12: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^{\star}, VK^{\star}, RK^{\star}, PK_F, \text{PP})$
13: **if** $((y, (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\theta^{\star}, VK^{\star}, RK^{\star}, \text{PP})$ **and** $(S \in \overline{R}))$ **then**
14: **return** 1
15: **else return** 0

is a notational convenience that allows us to define the challenge computation in terms of F and $X^{\star}$ in line 3; note that the same information can, however, be gleaned from the choice of mode and the set of labels $L_{F,X^{\star}}$, so this is not a weakening of the game — this information has already been determined by the choices of the adversary.

The challenger runs Setup and FnInit to create a public delegation key for the chosen challenge function F. The challenger then runs $\mathsf{ProbGen}$ on the challenge inputs to create the challenge parameters for the adversary, which are given to $\mathcal{A}$ along with the public information. The adversary is also given oracle access to the functions $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \text{MK}, \text{PP})$ and $\mathsf{Revoke}(\cdot, \text{MK}, \text{PP})$, denoted by $\mathcal{O}$. The adversary wins the game if it creates an encoded output that verifies correctly yet does not encode the correct value $F(x)$.

Definition B.1. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* sPubVerif *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{sPubVerif}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{sPubVerif}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]\right].$$

An HPVC scheme, $\mathcal{HPVC}$, is secure with respect to selective public verifiability *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{sPubVerif}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}) \leqslant \text{negl}(\ell).$$

B.2.2 Selective, Semi-static Revocation

Oracle B.1 $\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

1: **if** $((L_{F,X^\star} \subseteq L_i$ **and** $S_i \notin \overline{R})$ **or** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\mathrm{Rev}} \setminus S_i))$ **then return** $\perp$
2: $Q_{\mathrm{Rev}} \leftarrow Q_{\mathrm{Rev}} \setminus S$
3: **return** $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP}))$

Oracle B.2 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{F'(X)}, \mathrm{MK}, \mathrm{PP}))$

1: $t \leftarrow t + 1$
2: **if** $(\tau_{F'(X)} = (\mathsf{accept}, \cdot))$ **then return** $\perp$
3: **if** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\mathrm{Rev}} \cup S_i)$ **then return** $\perp$
4: $Q_{\mathrm{Rev}} \leftarrow Q_{\mathrm{Rev}} \cup S$
5: **return** $\mathsf{Revoke}(\tau_{F'(X)}, \mathrm{MK}, \mathrm{PP})$

The notion of revocation requires that if a server is detected as misbehaving, i.e. the BVerif algorithm outputs $\tau_{F(X)} = (\mathsf{reject}, S_i)$, then any subsequent evaluations by S_i should be rejected. As in Chapter 3, this notion inherits the selective, semi-static restrictions from the revocation mechanism of the underlying primitive in our construction. Hence, the adversary must first select its challenge parameters, which the challenger can parse to learn F and $X^\star$ for the challenge computation. The challenger maintains a time period t which is incremented during Revoke oracle queries, and a list Q_{Rev} of currently revoked entities. On line 4, the adversary (before receiving oracle access) must choose a list $\overline{R}$ of servers to be revoked during the challenge generation (which we assume will be at time q_t, where q_t is given as an input to the game).

The adversary is then given oracle access to the functions $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Revoke}(\cdot, \mathrm{MK}, \mathrm{PP})$, denoted by $\mathcal{O}$. Certify and Revoke queries are handled as specified in Oracles B.1 and B.2. Note that the Certify oracle returns $\perp$ if the resulting evaluation key would enable evaluation of the challenge computation. After finishing this query phase (and in particular after q_t Revoke queries), the challenge is created. The adversary wins if it outputs *any* result (even a correct encoding of $F(X^\star)$) that is accepted as a valid response from any server that was revoked at the time of the challenge.

Definition B.2. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number, q, of oracle queries, of which q_t are* Revoke *queries, in the* sSS-Rev *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{sSS-Rev}}(\mathcal{HPVC}, 1^\ell, \mathcal{F}, q_t) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{sSS-Rev}}\left[\mathcal{HPVC}, 1^\ell, \mathcal{F}, q_t\right]\right].$$

An HPVC scheme, $\mathcal{HPVC}$, is secure with respect to selective semi-static revocation *if,*

for all PPT adversaries $\mathcal{A}$*,*

$$Adv_{\mathcal{A}}^{\text{sSS-Rev}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}, q_t) \leqslant \text{negl}(\ell).$$

B.2.3 Selective Authorised Computation

The notion of *selective authorised computation*, presented in Game B.3, ensures that only a server that satisfies the additional authorisation policy specified in the encoded input may perform a given computation and hence be rewarded for correct work; this is similar to the authorised computation notion given in Game 4.2. A result generated by an unauthorised server should be rejected (even if the result itself is correct). Note that this game is only meaningful when the challenge parameters are generated in PVC-AC mode.

Game B.3 $\mathbf{Exp}_{\mathcal{A}}^{\text{sAuthC}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(F, X^{\star}, P) \xleftarrow{\$} \mathcal{A}(1^{\ell})$
2: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
4: $(\sigma^{\star}, VK^{\star}, RK^{\star}) \xleftarrow{\$} \mathsf{ProbGen}(\text{PVC-AC}, (x, P), \{l(F)\}, PK_F, \text{PP})$
5: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^{\star}, VK^{\star}, RK^{\star}, PK_F, \text{PP})$
6: $(RT^{\star}, \tau^{\star}) \leftarrow \mathsf{BVerif}(\theta^{\star}, VK^{\star}, \text{PP})$
7: **if** $(\tau^{\star} \neq (\mathsf{reject}, \cdot))$ **then return** 1
8: **else return** 0

This is a selective notion due to the selectively secure revocable key DP-ABE primitive we use in our construction; as such, the game begins with the adversary choosing a challenge function F, a challenge input $X^{\star}$ and an authorisation policy P. The challenger initialises the system and generates an encoded input for the challenge computation. The adversary is given the resulting parameters and oracle access to $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \text{MK}, \text{PP})$ and $\mathsf{Revoke}(\cdot, \text{MK}, \text{PP})$, denoted by $\mathcal{O}$. The Certify oracle is handled as specified in Oracle B.3. It returns a failure symbol $\perp$ if the queried attributes ψ satisfies the authorisation policy P, else the adversary could trivially produce a valid response as an authorised entity. Otherwise, the result of running the Certify algorithm is returned.

The adversary must return a result which is accepted by a verifier.

Definition B.3. *The* advantage *of a PPT adversary* $\mathcal{A}$ *in the* sAuthC *game for an HPVC construction,* $\mathcal{HPVC}$*, for a family of functions* $\mathcal{F}$ *is defined as:*

Oracle B.3 $\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

1: **if** $(\psi \in P)$ **then return** $\perp$
2: **return** $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

$$Adv_{\mathcal{A}}^{\text{sAuthC}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}) = \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{sAuthC}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]\right].$$

An HPVC scheme, $\mathcal{HPVC}$, is secure with respect to selective authorised computation *if, for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\text{sAuthC}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}) \leqslant \mathrm{negl}(\ell).$$

B.3 Proofs of Security

Informally, public verifiability and revocation reduce to the indistinguishability of ciphertexts within the rkDPABE scheme which allows us to replace the message for the unsatisfied function (which cannot be decrypted) with the challenge for a one-way function game. Then an adversary against these games can attack the verification token for this message.

B.3.1 Proof of Public Verifiability

Lemma B.1. *The HPVC construction defined by Algorithms 6.1–6.9 is secure with respect to selective public verifiability (Game B.1) under the same assumptions as in Theorem 6.1.*

One way to view this proof is to reduce, based on the `mode` to either the RPVC or VDC public verifiability game. Observe that the choice of dummy attributes and a dummy policy that is trivially satisfied by the presence of that dummy attribute means that one type of policy is always trivially satisfied, and hence decryption hinges entirely on the remaining policy. Thus we are in the setting of the single mode games.

We now give a unified proof.

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the selective public

verifiability game (Game B.1) when instantiated with Algorithms 6.1–6.9. We define the following three games:

- **Game 0.** This is the correct selective public verifiability game as in Game B.1.
- **Game 1.** This differs from **Game 0** in that ProbGen no longer returns an encryption of m_0 and m_1. Instead, we choose a random message $m' \neq m_0, m_1$. Note that there are two ciphertexts created during ProbGen (being the encryption of m_0 and m_1) and that one of these will be associated with the function F, and the other with the complement function $\overline{F}$. Now, only one of F and $\overline{F}$ will be satisfied by the input data $X^\star$. We replace the plaintext associated with the *unsatisfied* function by m' which is unrelated to m_0, m_1 and 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.

By hopping from **Game 0** to **Game 2**, we show that an adversary with non-negligible advantage against public verifiability can be used to construct an adversary that inverts 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 $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VC}$ as a sub-routine to break the IND-sHRSS security of the rkDPABE scheme (Game 6.1). In this proof, we consider the PVC-AC mode to be a special case of the RPVC mode (since we assume the adversary can be authorised to evaluate the challenge computation); therefore we focus only on RPVC and VDC modes. We consider a challenger $\mathcal{C}$ playing the IND-sHRSS game with $\mathcal{A}_{ABE}$, who in turn acts as a challenger in the selective public verifiability game for $\mathcal{A}_{VC}$:

1. $\mathcal{A}_{VC}$ is given the security parameter, and declares its choice of challenge input parameters $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star,)$, a set of labels $L_{F,X^\star}$ and the `mode` in which the challenge should be generated.

2. $\mathcal{A}_{ABE}$ must send a challenge input $(\tilde{\omega}, \tilde{\mathbb{S}})$ and a challenge time period $\tilde{t}$ to the challenger. It first sets $\tilde{t} = 1$. Observe that in VDC mode, $\mathbb{S}^\star$ corresponds to the function F and $\psi^\star$ is the challenge input data $X^\star$. In RPVC mode, $\mathbb{O}^\star = F$ and ω corresponds to the challenge input $X^\star$. The other inputs are either dummy attributes or corresponding policies, and these policies are trivially satisfied by the dummy attribute. Now, using the relevant inputs, $\mathcal{A}_{ABE}$ computes $r = F(X^\star)$.

   - If the challenge mode is RPVC, set

$$\tilde{\omega} = A_{\omega^\star} = A_{X^\star},$$

$$\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X}} l_j = \{\{T_S^0\}\} \wedge \{l(F)\}.$$

   - If the challenge mode is VDC, we want to set $(\tilde{\omega}, \tilde{\mathbb{S}})$ such that the pair is not satisfied by the challenge input.
     - If $r = 1$: set

$$\tilde{\omega} = A_{\omega^\star} = \{T_O^0\},$$

$$\tilde{\mathbb{S}} = \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X}} l_j = \overline{F} \wedge \{l(x_{i,j})\}_{x_{i,j} \in X^\star}.$$

     - If $r = 0$: set

$$\tilde{\omega} = A_{\omega^\star} = \{T_O^0\},$$

$$\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X}} l_j = F \wedge \{l(x_{i,j})\}_{x_{i,j} \in X^\star}.$$

3. $\mathcal{C}$ runs the DPABE.Setup algorithm on the security parameter to generate MPK_{ABE} and MSK_{ABE} and gives MPK_{ABE} to $\mathcal{A}_{ABE}$.

4. $\mathcal{A}_{ABE}$ sends $\overline{R} = \epsilon$ (i.e. an empty list) to $\mathcal{C}$ and simulates running HPVC.Setup such that the outcome is consistent with MPK_{ABE}. If `mode` = `VDC`, it runs lines 3 to 5 as written, sets $MPK^0_{\text{ABE}} = MPK_{\text{ABE}}$ as given by $\mathcal{C}$, and implicitly sets $MSK^0_{\text{ABE}} = MSK_{\text{ABE}}$ (any use of MSK^0_{ABE} will be simulated using oracle queries to $\mathcal{C}$). Otherwise, it sets MPK^r_{ABE} to be that issued by $\mathcal{C}$, and implicitly sets MSK^r_{ABE} to be that held by the challenger.

   In both cases, it also runs DPABE.Setup itself to generate a second DP-ABE system.

5. $\mathcal{A}_{ABE}$ runs HPVC.FnInit as written. To generate the challenge input, $\mathcal{A}_{ABE}$ begins by choosing two random bits, b (which it defines to be $RK_{F,X^\star}$) and s, and three

random messages m_0, m_1 and m' from the message space.

$\mathcal{A}_{ABE}$ sends m_0 and m_1 to $\mathcal{C}$ as its challenge inputs. $\mathcal{C}$ chooses $b^\star \xleftarrow{\$} \{0,1\}$ and returns $CT^\star \leftarrow \mathsf{DPABE.Encrypt}(m_{b^\star}, (\tilde{\omega}, \tilde{\mathbb{S}}), \tilde{t}, MPK_{\text{ABE}})$.

- In RPVC mode, $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates

$$c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), \tilde{t}, MPK^1_{\text{ABE}})$$

  itself. It sets $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$.

- In VDC mode:
  - If $r = 1$:

    $\mathcal{A}_{ABE}$ generates

$$c_b \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \mathbb{S}^\star) \wedge \bigwedge_{l_j \in L_{F,X}} l_j, \tilde{t}, MPK^0_{\text{ABE}})$$

    itself, and sets c_{1-b} to be $CT^\star$. It sets $VK_b = g(m')$ and $VK_{1-b} = g(m_s)$.
  - If $r = 0$:

    $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates

$$c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), \tilde{t}, MPK^0_{\text{ABE}})$$

    itself. It sets $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$.

Finally, $\mathcal{A}_{ABE}$ sets $\sigma_{F,X^\star} = (c_b, c_{1-b})$, $VK_{F,X^\star} = (VK_b, VK_{1-b}, L_{Reg})$ and $RK_{F,X^\star} = b$. Note that s is essentially $\mathcal{A}_{ABE}$'s guess for the value of $b^\star$ chosen by $\mathcal{C}$.

6. $\mathcal{A}_{VC}$ is provided the output of $\mathsf{ProbGen}$ and given oracle access which is handled as follows.

   - $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$ and $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$ can be run as written.
   - $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, S_i, \text{MK}, \text{PP})$:

     To generate the evaluation key for the queried parameters, $\mathcal{A}_{ABE}$ uses the KeyGen oracle in the rkDPABE game. It first updates the relevant list entries as specified. Then it sets $\mathbb{O}' = \mathbb{O}$ and $\psi' = A_\psi \cup \bigcup_{l_j \in L_i} l_j$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}', \psi'), \text{MK}, \text{PP})$ as in Oracle 6.1. $\mathcal{C}$ shall generate a rkDPABE decryption key $SK_{\mathbb{O}', \psi'}$ if and only if $\tilde{\omega} \notin \mathbb{O}'$ or $\psi' \notin \tilde{\mathbb{S}}$ or $S_i \in \overline{R}$.

Observe, that $S_i \notin \overline{R}$ since we set $\overline{R}$ to be empty.

Now, by construction (Step 2), $\psi' \in \tilde{\mathbb{S}}$ only if the labels $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$. If the labels *do not* satisfy this relation, then $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as SK^0_{ABE}.

If, on the other hand, the labels *do* satisfy this relation, then observe that, because each label uniquely describes a single element (either a function or a data point):

- In RPVC mode: as both L_i and $L_{F,X^\star}$ are singleton sets, and $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$, it must be that $L_i = L_{F,X^\star} = \{l(F)\}$ and hence, by uniqueness of the labels, $\mathbb{O} = \mathbb{O}^\star$ i.e. the adversary has requested an evaluation key for the challenge function F. However, in Step 4, we assigned the ABE system owned by the challenger (with master secret MSK^r_{ABE}) precisely such that $\mathbb{O}^\star$ is not satisfied by the challenge input $\tilde{\omega}$, and therefore $\mathbb{O}'$ is not satisfied either — that is, $\tilde{\omega} \notin \mathbb{O}'$ and hence $\mathcal{C}$ can generate a valid key which $\mathcal{A}_{ABE}$ will store as SK^r_{ABE}.
- In VDC mode: $\{l_k\}_{l_k \in L_{F,X^\star}} \subseteq \{l_j\}_{l_j \in L_i} \Rightarrow \{l(x_{i,k})\}_{x_{i,k} \in X^\star} \subseteq \{l(x_{i,j})\}_{x_{i,j} \in D_i} \Rightarrow X^\star \subseteq D_i$ i.e. by uniqueness of the labels, the adversary has requested an evaluation key for a superset of the challenge input data — that is, D_i contains $X^\star$ and possibly some additional data points. Now, if $X^\star \subseteq D_i$ then, additionally D_i must satisfy either F or $\overline{F}$ to satisfy $\tilde{\mathbb{S}}$. However, note that in Step 2, $\tilde{\mathbb{S}}$ was chosen specifically such that it is not satisfied by the challenge input $X^\star$ and therefore by D_i (as F will simply select the elements of $X^\star$ to evaluate on). Hence $\mathcal{C}$ may generate a valid key which $\mathcal{A}_{ABE}$ will store as SK^0_{ABE}.

$\mathcal{A}_{ABE}$ also must request an update key by making a KeyUpdate oracle query to $\mathcal{C}$. $\mathcal{C}$ will return a valid key unless the current time period is $t = \tilde{t}$ and $\overline{R} \not\subseteq Q_{Rev}$. Observe that the second clause is never satisfied since $\overline{R} = \epsilon$ and hence is a subset of any Q_{Rev}. Hence $\mathcal{C}$ may always generate a valid update key.

If in RPVC mode, $\mathcal{A}_{ABE}$ additionally generates a key SK^{1-r}_{ABE} using the second system parameters (which he owns) for $(\overline{\mathbb{O}}, \overline{\psi})$.

- Revoke$(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), \text{MK}, \text{PP})$: In response to a Revoke query, $\mathcal{A}_{ABE}$ runs Algorithm 6.9 as written except that it will make a KeyUpdate oracle query to $\mathcal{C}$ for the update key for to the ABE system owned by the

challenger ($UK^{r}_{L_{\text{Rev}},t}$ in the case of RPVC and $UK^{0}_{L_{\text{Rev}},t}$ in the case of VDC). $\mathcal{C}$ will generate a valid update key unless the time period is $\tilde{t} = q_t$ and there exists a server on $\overline{R}$ that is not currently revoked. However, as $\overline{R}$ was defined to be an empty list, this second condition will never hold and $\mathcal{C}$ can always return a valid key.

Eventually, it outputs $\theta^{\star}$ which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $F(X^{\star})$).

7. $\mathcal{A}_{ABE}$ parses $\theta^{\star}$ as $(d_b, d_{1-b}, S_{i^{\star}}, \gamma)$ and using the retrieval key $RK_{F,X^{\star}} = b$, finds d_0 and d_1. One of d_0 and d_1 will be $\perp$ (by construction) and we denote the other value by Y.

   If $g(Y) = g(m_s)$, $\mathcal{A}_{ABE}$ outputs a guess $b' = s$ and otherwise guesses $b' = (1 - s)$.

If $s = b^{\star}$ (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 $g(m_s)$ where m_s is the other plaintext). Alternatively, $s = 1 - b^{\star}$ and the distribution of the above experiment is identical to **Game 1** (since the verification key comprises the legitimate message and a random message $m_{1-b^{\star}}$ that is unrelated to the ciphertext).

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

$$\left|\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 0}}_{\mathcal{A}_{VC}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]\right] - \Pr\left[1 \xleftarrow{\$} \mathbf{Exp}^{\mathbf{Game\ 1}}_{\mathcal{A}_{VC}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]\right]\right| \geqslant \delta$$

where $\mathbf{Exp}^{i}_{\mathcal{A}_{VC}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]$ denotes running $\mathcal{A}_{VC}$ in **Game** i.

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

Hence,

$$
Adv_{\mathcal{A}_{ABE}} \geqslant \left|\Pr\left[b^\star = b'\right] - \frac{1}{2}\right| \geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \geqslant \frac{\delta}{2}.
$$

Hence, if $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the IND-sHRSS game for rkDPABE with non-negligible probability. Thus since we assumed the rkDPABE scheme to be secure, we conclude that $\mathcal{A}_{VC}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** sets the value of m' to correspond to the challenge w in the one-way function inversion game (rather than be a randomly chosen message). 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, $\mathcal{A}_{VC}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{ABE}$).

Final Proof We now show that using $\mathcal{A}_{VC}$ in **Game 2**, $\mathcal{A}_{ABE}$ 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 $r = 1$, we implicitly set m_{1-b} to be w by setting the corresponding verification key component to be z. We choose m_b and the other verification key component

randomly as usual.

- if $r = 0$, we implicitly set m_b to be w by setting the corresponding verification key component to be z. We choose m_{1-b} and the other verification key component randomly as usual.

Now, if $\mathcal{A}_{VC}$ is successful, it will output a forgery comprising the plaintext encrypted under the function F or $\overline{F}$ that evaluates to 0. By construction, this will be w (and the adversary's view in consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ can forward this result to $\mathcal{C}$ to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{VC}$ has against the selective public verifiability game.

We conclude that if the rkDPABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $HPVC$ as defined by Algorithms 6.1–6.9 is secure in the sense of selective public verifiability. □

B.3.2 Proof of Revocation

Lemma B.2. *The HPVC construction defined by Algorithms 6.1–6.9 is secure in the sense of selective, semi-static revocation (Game B.2) under the same assumptions as in Theorem 6.1.*

Proof. We reduce the security of the selective, semi-static revocation game to the IND-sHRSS security of the underlying revocable-key DPABE scheme (Game 6.1). To achieve a contradiction, let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the selective, semi-static revocation game (Game B.2) when instantiated with Algorithms 6.1–6.9 and making q_t Revoke queries. We show that, if such an $\mathcal{A}_{VC}$ exists, then it can be used to construct an adversary $\mathcal{A}_{ABE}$ that can break the IND-sHRSS security of the revocable-key DPABE scheme. Again, we consider only RPVC and VDC modes here (and view PVC-AC as a special case of RPVC) as the adversary should be authorised (in terms of the authorisation policy, if not in terms of revocation) for the challenge computation. Let $\mathcal{C}$ be a challenger for the IND-sHRSS game playing with $\mathcal{A}_{ABE}$, who in turn acts as the challenger in the selective, semi-static revocation game with $\mathcal{A}_{VC}$.

1. $\mathcal{A}_{VC}$ selects its challenge inputs $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode}^\star)$ for a challenge computation of $F(X^\star)$.

2. $\mathcal{A}_{ABE}$ initialises the list $Q_{\text{Rev}} = \epsilon$ and time parameter $t = 1$. It then forms its own challenge input as follows. It sets $t^\star = q_t$. Then, it sets $\tilde{\omega} = A_{\omega^\star}$ and sets $\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j$. It sends $t^\star, \tilde{\omega}$ and $\tilde{\mathbb{S}}$ to $\mathcal{C}$.

3. $\mathcal{C}$ runs DPABE.Setup and returns the public parameters MPK_{ABE} to $\mathcal{A}_{ABE}$ who stores them as MPK^0_{ABE}.

4. $\mathcal{A}_{ABE}$ now simulates the HPVC.Setup algorithm such that the output is consistent with the public parameters generated by $\mathcal{C}$. It runs Algorithm 6.1 as written, with the exception of line 1, since MSK^0_{ABE} and MPK^0_{ABE} was already generated by $\mathcal{C}$. $\mathcal{A}_{ABE}$ also runs HPVC.FnInit as written, and gives the public parameters and public delegation key to $\mathcal{A}_{VC}$.

5. $\mathcal{A}_{VC}$ chooses a challenge revocation list $\overline{R}$, which $\mathcal{A}_{ABE}$ forwards to $\mathcal{C}$.

6. $\mathcal{A}_{VC}$ is now given oracle access to which $\mathcal{A}_{ABE}$ can respond as follows:

   - Queries to HPVC.FnInit and HPVC.Register can be run as written.
   - Queries of the form HPVC.Certify($\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP}$): $\mathcal{A}_{ABE}$ runs Oracle B.1. To simulate running the HPVC.Certify algorithm, $\mathcal{A}_{ABE}$ runs Algorithm 6.4 as written with the exception of lines 4 and 5, as these depend on MSK^0_{ABE} held by $\mathcal{C}$.

     To simulate line 4, $\mathcal{A}_{ABE}$ makes an oracle query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_k \in L_i} l_k), MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ responds by running Oracle 6.1 which will return a valid key unless $(\tilde{\omega} \in \mathbb{O})$ *and* $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \tilde{\mathbb{S}})$ *and* $(S_i \notin \overline{R})$.

     If $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \tilde{\mathbb{S}})$ then $\mathcal{C}$ can return a valid decryption key. Otherwise, we may observe that

     $$(A_\psi \cup \bigcup_{l_k \in L_i} l_k) \in (\mathbb{S} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j)$$

     only if $\{l_k\}_{l_k \in L_i} \supseteq \{l_j\}_{l_j \in L_{F,X^\star}}$. By virtue of the fact that labels uniquely label the objects they relate to (either functions or data points), this implies $L_{F,X^\star} \subseteq L_i$. However, in this case, by the first check performed in Oracle B.1, $\mathcal{A}_{ABE}$ would have returned $\perp$ without querying KeyGen if $S_i \notin \overline{R}$, to avoid certifying $\mathcal{A}_{VC}$ for the challenge computation.

Thus, at the point of making a KeyGen query, if $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \tilde{\mathbb{S}})$ then $S_i \in \overline{R}$. Therefore, $\mathcal{C}$ can respond to all queries made to it during this phase with a valid key which $\mathcal{A}_{ABE}$ can use to simulate line 4 correctly.

To simulate line 5 of HPVC.Certify, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ responds as written in Oracle 6.2 — that is, it returns a valid update key *unless* $t = t^\star$ *and* $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, note that $\mathcal{A}_{ABE}$ chose $t^\star = q_t$, and at the point of calling the KeyUpdate oracle, the list $Q_{\text{Rev}} = Q_{\text{Rev}} \setminus S_i$. Therefore, if $\mathcal{C}$ would return $\perp$ in response to this query, then $\mathcal{A}_{ABE}$ would already have returned $\perp$ as a result of the checks performed in Oracle B.1. Hence, for all queries made to $\mathcal{C}$, a valid update key is returned and line 4 is simulated correctly.

- Queries of the form HPVC.Revoke$(\tau_{F(X)}, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Oracle B.2. To simulate running the HPVC.Revoke algorithm, $\mathcal{A}_{ABE}$ runs Algorithm 6.9 as written with the exception of line 4. Instead, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. Note that, according to Oracle B.2, $\mathcal{A}_{ABE}$ would have returned $\perp$ if $t = q_t$ (where, recall, $q_t = t^\star$ by the choice of $\mathcal{A}_{ABE}$) and $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S_i$. This corresponds directly to the conditions wherein $\mathcal{C}$ cannot form a valid update key according to Oracle 6.2 (since, if HPVC.Revoke is called, then S_i was already removed from the list Q_{Rev}). Hence, for all queries, $\mathcal{C}$ can form a valid update key and $\mathcal{A}_{ABE}$ can simulate the expected behaviour.

7. Eventually (after q_t Revoke queries), $\mathcal{A}_{VC}$ finishes this query phase. $\mathcal{A}_{ABE}$ checks whether the list of revoked entities is compatible with the challenge list $\overline{R}$ and returns 0 if not.

8. $\mathcal{A}_{ABE}$ must now generate the challenge input for $\mathcal{A}_{VC}$. To do so, it chooses three distinct messages, m_0, m_1 and m', uniformly at random from the messagespace. It also chooses a random bit b which it defines to be $RK_{F,X^\star}$. It sends m_0 and m_1 to $\mathcal{C}$ as its challenge inputs in the IND-sHRSS game. $\mathcal{C}$ chooses a bit $b^\star$ uniformly at random and returns $CT^\star \xleftarrow{\$} \mathsf{Encrypt}(m_{b^\star}, \tilde{\omega}, \tilde{\mathbb{S}}, t^\star, MPK^0_{\text{ABE}})$. $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates c_{1-b} himself by encrypting m' as specified on lines 5 and 6 of Algorithm 6.5.

   Finally, $\mathcal{A}_{ABE}$ selects another bit s uniformly at random and, if $b = 0$, it sets $VK_{F,X^\star} = (g(m_s), g(m'), L_{\text{Reg}})$, or otherwise it sets $VK_{F,X^\star} = (g(m'), g(m_s), L_{\text{Reg}})$.

Note that s can be thought of as $\mathcal{A}_{ABE}$'s guess as to the value of $b^\star$.

9. $\mathcal{A}_{VC}$ is given the resulting parameters and again given oracle access which $\mathcal{A}_{ABE}$ responds to as in Step 6.

10. Eventually, $\mathcal{A}_{VC}$ outputs its result $\theta^\star$ which, in order to appear valid, should contain exactly one non-$\perp$ plaintext; we denote this plaintext by y. If $g(y) = g(m_s)$, $\mathcal{A}_{ABE}$ guesses $b' = s$. If $g(y) = g(m')$, $\mathcal{A}_{ABE}$ guesses randomly $b' \leftarrow \{0, 1\}$ (as $\mathcal{A}_{VC}$ did not respond for either m_0 or m_1, it reveals no information to aid $\mathcal{A}_{ABE}$). Otherwise, $\mathcal{A}_{ABE}$ aborts as $\mathcal{A}_{VC}$ was not successful (either for legitimate reasons or because $\mathcal{A}_{ABE}$ chose s incorrectly and issued a malformed challenge).

By assumption, $\mathcal{A}_{VC}$ has non-negligible advantage δ against the selective, semi-static revocation game — that is, $\Pr[g(y) = g(m_s)] + \Pr[g(y) = g(m')] = \delta$. Therefore,

$$\begin{aligned}
\Pr\left[b' = b^\star\right] &= \Pr\left[b' = b^\star | g(y) = g(m_s)\right] \Pr\left[g(y) = g(m_s)\right] \\
&\quad + \Pr\left[b' = b^\star | g(y) = g(m')\right] \Pr\left[g(y) = g(m')\right] \\
&= \Pr\left[s = b^\star\right] \Pr\left[g(y) = g(m_s)\right] + \Pr\left[\tilde{b} = b^\star\right] \Pr\left[g(y) = g(m')\right] \\
&= \frac{1}{2} \Pr\left[g(y) = g(m_s)\right] + \frac{1}{2} \Pr\left[g(y) = g(m')\right] \\
&= \frac{1}{2}(\Pr\left[g(y) = g(m_s)\right] + \Pr\left[g(y) = g(m')\right]) \\
&= \frac{\delta}{2}
\end{aligned}$$

Hence,

$$\begin{aligned}
Adv_{\mathcal{A}_{ABE}} &\geq \left| \Pr\left[b^\star = b'\right] - \frac{1}{2} \right| \\
&\geq \left| \frac{\delta}{2} - \frac{1}{2} \right| \\
&\geq \frac{1}{2}(\delta - 1),
\end{aligned}$$

which is non-negligible. However, the DPABE scheme was, in fact, assumed to be secure in the sense of IND-sHRSS and hence such an adversary as $\mathcal{A}_{ABE}$ may not exist. Therefore, our assumption on $\mathcal{A}_{VC}$ must be incorrect, and no adversary with non-negligible advantage against the selective, semi-static revocation game can exist. □

B.3.3 Proof of Authorised Computation

Lemma B.3. *The HPVC construction defined by Algorithms 6.1–6.9 is secure with respect to selective authorised computation (Game B.3) under the same assumptions as in Theorem 6.1.*

Proof. We reduce the security of the selective authorised computation game to the IND-sHRSS security of the underlying revocable-key DPABE scheme (Game 6.1). To achieve a contradiction, let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the selective authorised computation game (Game B.3) when instantiated with Algorithms 6.1–6.9. We show that, if such an $\mathcal{A}_{VC}$ exists, then it can be used to construct an adversary $\mathcal{A}_{ABE}$ that can break the IND-sHRSS security of the revocable-key DPABE scheme. Note that this notion is only meaningful in PVC-AC mode. Let $\mathcal{C}$ be a challenger for the IND-sHRSS game playing with $\mathcal{A}_{ABE}$, who in turn acts as the challenger in the selective authorised computation game with $\mathcal{A}_{VC}$.

1. $\mathcal{A}_{VC}$ first selects its challenge inputs $F, X^\star$ and the authorisation policy P.
2. $\mathcal{A}_{ABE}$ defines its own challenge inputs for the IND-sHRSS game by setting $t^\star = 1$, $\omega^\star = A_{X^\star}$ and $\mathbb{S}^\star = P \wedge \{l(F)\}$, and sends these to $\mathcal{C}$.
3. $\mathcal{C}$ runs the Setup algorithm for the DPABE scheme and returns the public parameters MPK_{ABE} to $\mathcal{A}_{ABE}$ who stores them as MPK^0_{ABE}.
4. $\mathcal{A}_{ABE}$ now simulates the $\mathsf{HPVC.Setup}$ algorithm such that the output is consistent with MPK^0_{ABE}. It runs Algorithm 6.1 as written, with the exception of line 1, since MSK^0_{ABE} and MPK^0_{ABE} are defined to be those generated by $\mathcal{C}$. $\mathcal{A}_{ABE}$ also runs $\mathsf{HPVC.FnInit}$ as written and sends an empty challenge revocation list $\overline{R} = \epsilon$ to $\mathcal{C}$.
5. $\mathcal{A}_{ABE}$ must next create the challenge input for $\mathcal{A}_{VC}$. To do so, it chooses three distinct messages, m_0, m_1 and m', uniformly at random from the messagespace. It also chooses a random bit b which it defines to be $RK_{F,X^\star}$. It sends m_0 and m_1 to $\mathcal{C}$ as its challenge inputs in the IND-sHRSS game. $\mathcal{C}$ chooses a bit $b^\star$ uniformly at random and returns $CT^\star \xleftarrow{\$} \mathsf{Encrypt}(m_{b^\star}, \omega^\star, \mathbb{S}^\star, t^\star, MPK^0_{\text{ABE}})$. $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates c_{1-b} himself by encrypting m' as specified on lines 5 and 6 of Algorithm 6.5.

Finally, $\mathcal{A}_{ABE}$ chooses another bit s uniformly at random and, if $b = 0$, it sets $VK_{F,X^\star} = (g(m_s), g(m'), L_{\text{Reg}})$, or otherwise sets $VK_{F,X^\star} = (g(m'), g(m_s), L_{\text{Reg}})$.

6. $\mathcal{A}_{ABE}$gives the resulting parameters to $\mathcal{A}_{VC}$ along with oracle access which $\mathcal{A}_{ABE}$ can handle as follows:

   - Queries to HPVC.FnInit and HPVC.Register can be run as given in the construction.
   - Queries of the form HPVC.Certify($\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP}$): $\mathcal{A}_{ABE}$ runs Oracle B.3. If the queried ψ satisfies the challenge authorisation policy P then $\mathcal{A}_{ABE}$ returns $\perp$. Otherwise it simulates running HPVC.Certify by running Algorithm 6.4 as written with the exception of lines 4 and 5, as these depend on MSK^0_{ABE} held by $\mathcal{C}$.

     To simulate line 4, $\mathcal{A}_{ABE}$ queries $\mathcal{C}$ for

     $$\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_k \in L_i} l_k), MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}}).$$

     $\mathcal{C}$ will return $\perp$ if $(\omega^\star \in \mathbb{O})$ *and* $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \mathbb{S}^\star)$ *and* $(S_i \notin \overline{R})$. However, for the query to have been made, $\mathcal{A}_{ABE}$ must not have returned $\perp$ in Oracle B.3, and therefore $\psi \notin P$, and hence $\psi \notin \mathbb{S}^\star$. Therefore, $\mathcal{C}$ can always return a valid decryption key SK^0_{ABE}.

     To simulate line 5 of HPVC.Certify, $\mathcal{A}_{ABE}$ queries $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ responds as in Oracle 6.2 and returns a valid update key *unless* $t = t^\star$ *and* $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, as $\overline{R}$ was chosen to be empty, $\overline{R} \subseteq Q_{\text{Rev}}$ for any Q_{Rev}, and hence $\mathcal{C}$ can create a valid update key.

   - Queries of the form HPVC.Revoke($\tau_{F(X)}, \text{MK}, \text{PP}$): $\mathcal{A}_{ABE}$ runs Algorithm 6.9 as written with the exception of line 4. To simulate this line, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. As specified in Oracle 6.2, $\mathcal{C}$ will return a valid key unless $t = t^\star$ and $\overline{R} \not\subseteq R$). However, $\mathcal{A}_{ABE}$ chose $\overline{R}$ to be empty and so it is certainly a subset of any R, and in particular L_{Rev}. Hence, for all queries, $\mathcal{C}$ can form a valid update key and $\mathcal{A}_{ABE}$ can simulate the expected behaviour.

7. Eventually, $\mathcal{A}_{VC}$ finishes this query phase and outputs a result $\theta^\star$ corresponding to the result of $F(X^\star)$ protected by an authorisation policy P, where $\mathcal{A}_{VC}$ never

received a key for a set of authorisation attributes $s \in P$. As $\theta^\star$ should appear valid, it should comprise exactly one non-$\perp$ element which we denote by y.

8. If $g(y) = g(m_s)$, $\mathcal{A}_{ABE}$ guesses $b' = s$. If $g(y) = g(m')$, $\mathcal{A}_{ABE}$ randomly guesses $b' \xleftarrow{\$} \{0,1\}$ (as $\mathcal{A}_{VC}$ did not respond for either m_0 or m_1, it reveals no information to aid $\mathcal{A}_{ABE}$ in its IND-sHRSS game). Otherwise, $\mathcal{A}_{ABE}$ aborts as $\mathcal{A}_{VC}$ was not successful (either for legitimate reasons or because $\mathcal{A}_{ABE}$ chose s incorrectly and issued a malformed challenge).

By assumption, $\mathcal{A}_{VC}$ has non-negligible advantage δ against the selective authorised computation game — that is, $\Pr[g(y) = g(m_s)] + \Pr[g(y) = g(m')] = \delta$. Therefore,

$$\begin{aligned}
\Pr\left[b' = b^\star\right] &= \Pr\left[b' = b^\star | g(y) = g(m_s)\right] \Pr\left[g(y) = g(m_s)\right] \\
&\quad + \Pr\left[b' = b^\star | g(y) = g(m')\right] \Pr\left[g(y) = g(m')\right] \\
&= \Pr\left[s = b^\star\right] \Pr\left[g(y) = g(m_s)\right] + \Pr\left[\tilde{b} = b^\star\right] \Pr\left[g(y) = g(m')\right] \\
&= \frac{1}{2} \Pr\left[g(y) = g(m_s)\right] + \frac{1}{2} \Pr\left[g(y) = g(m')\right] \\
&= \frac{1}{2}(\Pr\left[g(y) = g(m_s)\right] + \Pr\left[g(y) = g(m')\right]) \\
&= \frac{\delta}{2}
\end{aligned}$$

Hence,

$$\begin{aligned}
Adv_{\mathcal{A}_{ABE}} &\geq \left| \Pr\left[b^\star = b'\right] - \frac{1}{2} \right| \\
&\geq \left| \frac{\delta}{2} - \frac{1}{2} \right| \\
&\geq \frac{1}{2}(\delta - 1),
\end{aligned}$$

which is non-negligible. However, as the DPABE scheme was assumed to be IND-sHRSS secure, such an adversary can not exist. Therefore, our assumption on $\mathcal{A}_{VC}$ must be incorrect, and no adversary with non-negligible advantage against the selective authorised computation game can exist. □