Revocation in Publicly Verifiable Outsourced Computation

James Alderman[*], Christian Janson, Carlos Cid[†], and Jason Crampton

Information Security Group, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, United Kingdom
{James.Alderman.2011, Christian.Janson.2012}@live.rhul.ac.uk
{Carlos.Cid, Jason.Crampton}@rhul.ac.uk

Abstract

The combination of software-as-a-service and the increasing use of mobile devices gives rise to a considerable difference in computational power between servers and clients. Thus, there is a desire for clients to outsource the evaluation of complex functions to an external server. Servers providing such a service may be rewarded per computation, and as such have an incentive to cheat by returning garbage rather than devoting resources and time to compute a valid result.

In this work, we introduce the notion of Revocable Publicly Verifiable Computation (RPVC), where a cheating server is revoked and may not perform future computations (thus incurring a financial penalty). We introduce a Key Distribution Center (KDC) to efficiently handle the generation and distribution of the keys required to support RPVC. The KDC is an authority over entities in the system and enables revocation. We also introduce a notion of blind verification such that results are verifiable (and hence servers can be rewarded or punished) without learning the value. We present a rigorous definitional framework, define a number of new security models and present a construction of such a scheme built upon Key-Policy Attribute-based Encryption.

1 Introduction

It is increasingly common for mobile devices to be used as general computing devices. There is also a trend towards cloud computing and enormous volumes of data ("big data") which means that computations 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, coupled with the increasing use of software-as-a-service, means there is a requirement for a client device to be able to delegate a computation to a server.

Consider, for example, a company that operates a "bring your own device" policy, enabling employees to use personal smartphones and tablets for work. Due to resource limitations, it may not be possible for these devices to perform complex computations locally. Instead, a

[*] The first author acknowledges support from BAE Systems Advanced Technology Centre under a CASE Award.

[†] This research was partially sponsored by US Army Research laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

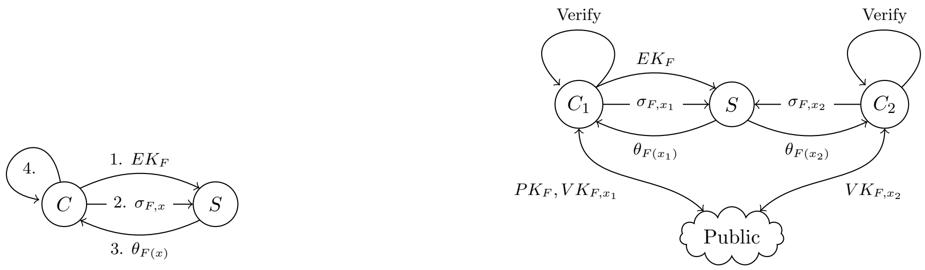
computation is outsourced over some network to a more powerful server (possibly outside the company, offering software-as-a-service, and hence untrusted) and the result of the computation is returned to the client device. Another example arises in the context of battlefield communications where each member of a squadron of soldiers is deployed 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. if the soldiers are part of a coalition network. Thus a soldier must have an assurance that the command has been computed correctly. A final example could consider sensor networks where lightweight sensors transmit readings to a more powerful base station to compute statistics that can be verified by an experimenter.

In simple terms, given a function F to be computed by a server S, the client sends input x to S, who should return $F(x)$ to the client. However, there may be an incentive for the server (or an imposter) to cheat and return an invalid result $y \neq F(x)$ to the client. The server may wish to convince a client of an incorrect result, or (particularly if servers are rewarded per computation performed) the server may be too busy or may not wish to devote resources to perform the computation. Thus, the client wishes to have some assurance that the result y returned by the server is, in fact, $F(x)$.

This problem, known as *Verifiable Outsourced Computation* (VC), has attracted a lot of attention in the community recently. In practical scenarios, it may well be desirable that cheating servers are prevented from performing future computations, as they are deemed completely untrustworthy. Thus, future clients need not waste resources delegating to a 'bad' server, and servers are disincentivised from cheating in the first place as they will incur a significant (financial) penalty from not receiving future work. Many current schemes have an expensive pre-processing stage run by the client. However, it is likely that many different clients will be interested in outsourcing computations, and that functions of interest to each client will substantially overlap, as in the "bring your own device" scenario above. It is also conceivable that the number of servers offering to perform such computations will be relatively low (limited to a reasonably small number of trusted companies with plentiful resources). Thus, it is easy to envisage a situation in which many computationally limited clients wish to outsource computations of the same (potentially large) set of functions to a set of untrusted servers. Current VC schemes do not support this kind of scenario particularly well.

Our main contribution, then, is to introduce the new notion of *Revocable Publicly Verifiable Computation* (RPVC). We also propose the introduction of a Key Distribution Center (KDC) to perform the computationally intensive parts of VC and manage keys for all clients, and we simplify the way in which the computation of multiple functions is managed. We enable the revocation of misbehaving servers (those detected as cheating) such that they cannot perform further computations until recertified by the KDC, as well as "blind verification", a form of output privacy, such that the verifier learns whether the result is valid but not the value of the output. Thus the verifier may reward or punish servers appropriately without learning function outputs. We give a rigorous definitional framework for RPVC, that we believe more accurately reflects real environments than previously considered. This new framework both removes redundancy and facilitates additional functionality, leading to several new security notions.

In the next section, we briefly review related work. In Section 3, we define our framework and the relevant security models. In Section 4, we provide an overview, technical details and a concrete instantiation of our framework using Attribute-based Encryption as well as full security proofs. Additional background details can be found in the Appendix.



(a) A VC system

(b) A PVC system

Figure 1: The operation of verifiable computation schemes

Notation. In the remainder of this paper we use the following notation. If A is a probabilistic algorithm we write $y \leftarrow A(\cdot)$ for the action of running A on given inputs and assigning the result to an output y. We denote the empty string by ϵ and use PPT to denote probabilistic polynomial-time. We say that $\text{negl}(\cdot)$ is a negligible function on its input and κ denotes the security parameter. We denote by $\mathcal{F}$ the family 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 $\mathcal{F}$. We denote the domain of F by $\mathsf{Dom}(F)$ and the range by $\mathsf{Ran}(F)$. By $\mathcal{M}$ we denote a message space and the notation $\mathcal{A}^{\mathcal{O}}$ is used to denote the adversary $\mathcal{A}$ being provided with oracle access. Finally, $[n]$ denotes the set $\{1, \ldots, n\}$.

2 Verifiable Computation Schemes and Related Work

The concept of non-interactive verifiable computation was introduced by Gennaro et al. [7] and 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 the provably correct computation of $F(x)$ by the server for an input x supplied by the client. More specifically, a VC scheme comprises the following steps [7]:

1. C computes evaluation information EK_F that is given to S to enable it to compute F (pre-processing);
2. C sends the encoded input $\sigma_{F,x}$ to S (input preparation);
3. S computes $F(x)$ using EK_F and $\sigma_{F,x}$ and returns an encoding of the output $\theta_{F(x)}$ to C (output computation);
4. C checks whether $\theta_{F(x)}$ encodes $F(x)$ (verification).

The operation of a VC scheme is illustrated in Figure 1a. Step 1 is performed once; steps 2–4 may be performed many times. Step 1 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) is amortized over multiple computations of F. A VC scheme comprises four algorithms – KeyGen, ProbGen, Compute and Verify – corresponding to the four steps described above.

Parno et al. [14] introduced *Publicly Verifiable Computation* (PVC). The operation of a Publicly Verifiable Outsourced Computation scheme is illustrated in Figure 1b. In this setting, a single client C_1 computes EK_F, as well as publishing information PK_F that enables other clients 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. It uses the same four algorithms as VC but KeyGen and ProbGen now output public values that other clients may use to encode inputs and verify outputs. Parno et al. gave an instantiation of PVC using Key-Policy Attribute-based Encryption (KP-ABE) for a class of Boolean functions. Further details are available in

Appendix A.

2.1 Other Related Work

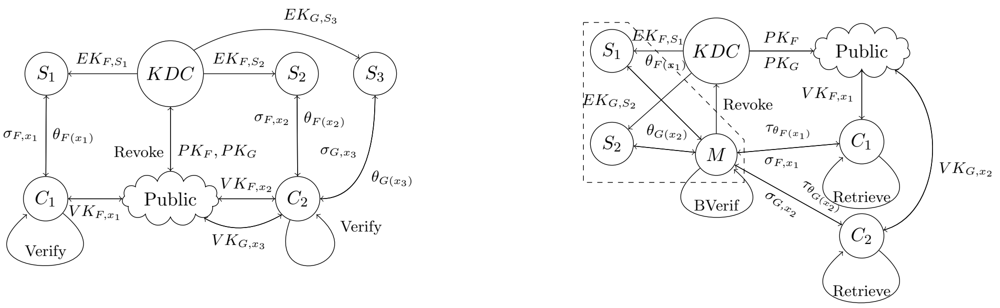
Gennaro et al. [7] formalized the problem of *non-interactive* verifiable computation in which there is only one round of interaction between the client and the server each time a computation is performed and introduced a construction based on Yao's Garbled Circuits [15] which provides a "one-time" Verifiable Outsourced Computation allowing a client to outsource the evaluation of a function on a single input. However it is insecure if the circuit is reused on a different input and thus this cost cannot be amortized, and the cost of generating a new garbled circuit is approximately equal to the cost of evaluating the function itself. To overcome this, the authors additionally use a fully homomorphic encryption scheme [8] to re-randomize the garbled circuit for multiple executions on different inputs. In independent and concurrent work, Carter et al. [5] introduce a third party to generate garbled circuits for such schemes but require this entity to be online throughout the computations and models the system as a secure multi-party computation between the client, server and third-party. We do not believe this solution is practical in all situations since it is conceivable that a trusted entity is not always available to take part in computations, for example in the battlefield scenario discussed in Section 1. Here, the KDC could be physically located within a high security base or governmental building and field agents may receive relevant keys before being deployed, but actual computations are performed using more local available servers and communications links. It may not be feasible, or desirable, for a remote agent to contact the headquarters and maintain a communications link with them for the duration of the computation. In addition, the KDC could easily become a bottleneck in the system and limit the number of computations that can take place at any one time, since we assume there are many servers but only a single (or small number of) trusted third parties.

Some works have also considered the multi-client case in which the input data to be sent to the server is shared between multiple clients, and notions such as input privacy become more important. Choi et al. [6] extended the garbled circuit approach [7] using a proxy-oblivious transfer primitive to achieve input privacy in a non-interactive scheme. Recent work of Goldwasser et al. [9] extended the construction of Parno et al. [14] to allow multiple clients to provide input to a functional encryption algorithm.

3 Revocable Publicly Verifiable Computation

We now describe our new notion of PVC, which we call *Revocable Publicly Verifiable Computation* (RPVC). We assume there is a Key Distribution Center (KDC) and many clients which make use of multiple untrusted or semi-trusted servers to perform complex computations. Multiple servers may be certified, by the KDC, to compute the same function F. As we briefly explained in the introduction, there appear to be good reasons for adopting an architecture of this nature and several scenarios in which such an architecture would be appropriate. The increasing popularity of relatively lightweight mobile computing devices in the workplace means that complex computations may best be performed by more powerful servers run by the organization. One can also imagine clients delegating computation to servers in the cloud and would wish to have some guarantee that those servers are certified to perform certain functions. It is essential that we can verify the results of the computation. If cloud services are competing on price to provide "computation-as-a-service" then it is important that a server cannot obtain an unfair advantage by simply not bothering to compute $F(x)$ and returning garbage instead. It is also important that a server who is not certified cannot return a result without being detected.

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 2: The operation of RPVC

3.1 Key Distribution Center

Existing frameworks assume that a client or clients run the expensive phases of a VC scheme and that a single server performs the outsourced computation. 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 organization (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 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 future work.)

We consider the KDC to be a separate entity to illustrate separation of duty between the clients that request computations, and the KDC that is authoritative on the system and users. The KDC could be authoritative over many sets of clients (e.g. at an organizational 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 organization responsible for user authorization that wishes to enable workers to securely use cloud-based software-as-a-service. As an entity within organization boundaries, performing all computations would negate the benefits gained from outsourcing computations to externally available servers. We examine the possible security concerns arising from RPVC in Sect. 3.5.

The basic idea of our scheme is to have the KDC perform the expensive setup operation. The KDC provides each server with a distinct key to compute F. A client may request the computation of $F(x)$ from any server that is certified to compute F. As mentioned in the introduction, in this paper we focus on two example system architectures, which we call the

Standard Model and the Manager Model.

3.2 Standard Model

The *standard model* is a natural extension of the PVC architecture with the addition of a KDC. The entities comprise a set of clients, a set of servers and a KDC. The KDC initializes 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. The server receives the encoded input values and performs the computation to generate a result. Any party can verify the correctness of the 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.

Note that the expensive KeyGen operation is now run by the more capable KDC, and many servers are able to use the generated keys to evaluate the same function, whereas previously each client would have run KeyGen to set up a system with its choice of server.

Figure 2 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 PVC detailed in this section, and finally PVC in the Manager model which we will discuss in the next section. The figure also includes a illustration of how the entities interact in the standard model.

3.3 Manager Model

The *manager model*, in contrast, employs an additional Manager entity who "owns" a pool of computation servers. Clients submit jobs to the manager, who will select a server from the pool based on workload scheduling, available resources or as a result of some bidding process if servers are to be rewarded per computation. A plausible scenario is that servers enlist with a manager to "sell" the use of spare resources, whilst clients subscribe to utilizee these 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 whilst a misbehaving server may be reported to the KDC for revocation, and the job assigned to another server. Due to public verifiability, any party with access to the output and the verification token can also verify the result. However, in many situations we may not desire external entities to access the result, yet there remains legitimate reasons for the manager to perform verification. Thus we introduce "blind verification" such that the manager (or other entity) may verify the validity of the computation without learning the output, but the delegating client holds an extra piece of information that enables the output to be retrieved.

The interaction between entities in this model is illustrated in Figure 2b. 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 organization that wishes to utilizee the external resources provided by the manager to outsource computational work. Notice that the manager performs a blind verification operation (denoted BVerif) but only entities within the organization may run the output retrieval algorithm (denoted here as $\mathsf{Retrieve}$) to learn the actual result of the computation.

3.4 Formal Definition

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

Definition 1. A *Revocable Publicly Verifiable Outsourced Computation Scheme (RPVC)* comprises the following algorithms:

- $(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$: Run by the KDC to establish public parameters PP and a master secret key MK.
- $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$: Run by the KDC to generate a public delegation key, PK_F, for a function F.
- $SK_S \leftarrow \mathsf{Register}(S, MK, PP)$: Run by the KDC to generate a personalised signing key SK_S for a computation server S.
- $EK_{F,S} \leftarrow \mathsf{Certify}(S, F, MK, PP)$: Run by the KDC to generate a certificate in the form of an evaluation key $EK_{F,S}$ for a function F and server S.
- $(\sigma_{F,x}, VK_{F,x}, RK_{F,x}) \leftarrow \mathsf{ProbGen}(x, PK_F, PP)$: $\mathsf{ProbGen}$ is run by a client to delegate the computation of $F(x)$ to a server. The output values are: the encoded input of x, $\sigma_{F,x}$; a verification key, $VK_{F,x}$ that will verify the result; and a retrieval key $RK_{F,x}$ which will enable the output to be read.
- $\theta_{F(x)} \leftarrow \mathsf{Compute}(\sigma_{F,x}, EK_{F,S}, SK_S, PP)$: Run by a server S holding an evaluation key $EK_{F,S}$, SK_S and an encoded input $\sigma_{F,x}$ of x, to output an encoding, $\theta_{F(x)}$, of $F(x)$, including an identifier of S.
- $(\tilde{y}, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{Verify}(\theta_{F(x)}, VK_{F,x}, RK_{F,x}, PP)$: Verification comprises two steps. These two steps could be run together if the Blind Verification property is not required.
  - $(RT_{F,x}, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{BVerif}(\theta_{F(x)}, VK_{F,x}, PP)$: Run by any verifying party (standard model), or by the manager (manager model), in possession of $VK_{F,x}$ and an encoded output, $\theta_{F(x)}$. This outputs a token $\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ if the output is valid, or $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ if S misbehaved. It also outputs a retrieval token $RT_{F,x}$ which is an encoding of the actual output value.
  - $\tilde{y} \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x)}}, RT_{F,x}, VK_{F,x}, RK_{F,x}, PP)$: Run by a verifier holding $RK_{F,x}$ to retrieve the actual result $\tilde{y}$ which is either $F(x)$ or $\perp$.[1]
- $\{EK_{F,S'}\}$ or $\perp \leftarrow \mathsf{Revoke}(\tau_{\theta_{F(x)}}, MK, PP)$: Run by the KDC if a misbehaving server is reported i.e. that Verify returned $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ (if $\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ then this algorithm outputs $\perp$). It revokes all evaluation keys $EK_{\cdot,S}$ of the server S thereby preventing S from performing any further evaluations. Updated evaluation keys $EK_{\cdot,S'}$ are issued to all servers.[2]

Although not stated, the KDC may update the public parameters PP during any algorithm. We say that a RPVC scheme is *correct* if the verification algorithm almost certainly outputs accept when run on a valid verification key and an encoded output honestly produced by a computation server given a validly generated encoded input and evaluation key. That is, if all algorithms are run honestly then the verifying party should almost certainly accept the returned result. A more formal definition follows:

Definition 2 (Correctness). *A Publicly Verifiable Computation Scheme with a Key Distribution Center (RPVC) is* correct *for a family of functions $\mathcal{F}$ if for all functions $F \in \mathcal{F}$ and inputs x,*

[1] Note that if a server is not given $RK_{F,x}$ then it too cannot learn the outputand we gain output privacy.

[2] In some instantiations, it may not be necessary to issue entirely new evaluation keys to each entity. In Sect. 4, we only need to issue a partially updated key for example.

where $\mathrm{negl}(\cdot)$ *is a negligible function of its input:*

$$\begin{aligned}
&\Pr[(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa), PK_F \leftarrow \mathsf{FnInit}(F, MK, PP),\\
&\quad SK_S \leftarrow \mathsf{Register}(S, MK, PP), EK_{F,S} \leftarrow \mathsf{Certify}(S, F, MK, PP),\\
&\quad (\sigma_{F,x}, VK_{F,x}, RK_{F,x}) \leftarrow \mathsf{ProbGen}(x, PK_F, PP),\\
&\quad (F(x), (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\mathsf{Compute}(\sigma_{F,x}, EK_{F,S}, SK_S, PP), VK_{F,x}, RK_{F,x}, PP)]\\
&\quad = 1 - \mathrm{negl}(\kappa).
\end{aligned}$$

3.5 Security Models

We now introduce several security models capturing different requirements of a RPVC scheme. We will formalize these notions of security as a series of cryptographic games run by a challenger. The adversary against a particular function F is modelled as a PPT algorithm $\mathcal{A}$ run by a challenger with input parameters chosen to represent the knowledge of a real attacker as well as the security parameter κ and a parameter $q_t > 1$ denoting the number of queries the adversary makes to the Revoke oracle before the challenge is generated. The adversary algorithm may maintain state and be multi-stage (i.e. be called several times by the challenger, with different input parameters) and we overload the notation by calling each of these adversary algorithms $\mathcal{A}$. This represents the adversary performing tasks at different points during the execution of the system, and we assume that the adversary may maintain a state storing any knowledge it gains during each phase (we do not provide the state as an input or output of the adversary for ease of notation). The notation $\mathcal{A}^{\mathcal{O}}$ denotes the adversary $\mathcal{A}$ being provided with oracle access to the following functions: $\mathsf{FnInit}(\cdot, MK, PP)$, $\mathsf{Register}(\cdot, MK, PP)$, $\mathsf{Certify}(\cdot, \cdot, \cdot, MK, PP)$ and $\mathsf{Revoke}(\cdot, \cdot, \cdot, MK, PP)$.[3] This means that the adversary can query (multiple times) the challenger for any of these functions with the adversary's choice of values for parameters represented with a dot above. This models information the adversary could learn from observing a functioning system or by acting like a legitimate client (or corrupting one) to request some functionality.

The introduction of the KDC and subsequent changes in operation give rise to new security concerns:

- Since two (or more) servers may be able to compute the same function, it is important to ensure that servers cannot collude in order to convince a client to accept an incorrect output as correct (Public Verifiability).
- We must ensure that neither an uncertified nor a de-certified server can convince a client to accept an output (Revocation).
- We must ensure that a malicious server S cannot convince a client to believe an honest server has produced an incorrect output (Vindictive Servers).
- We must ensure that, in the manager model, a malicious manager cannot convince a client of an incorrect result (Vindictive Manager).
- We must ensure, in the manager model, that the manager performing the BVerif algorithm learns nothing of the actual output value other than its correctness (Blind Verification).

3.6 Discussion of Games

As mentioned above, we define five notions of security for RPVC. We model each notion as a cryptographic game. However, in the cases of Public Verifiability, Revocation and Vindictive

[3] We do not need to provide a Verify oracle since this is a publicly verifiable scheme and $\mathcal{A}$ is given verification keys (thus we also avoid the rejection problem).

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 introduces similar restrictions. Thus, with our current primitives we cannot achieve full security for these notions, but can achieve the slightly weaker variants presented here. In this section we will discuss the restrictions we must impose and how they could be removed in the future. 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 4.

These variants require two additional restrictions on the adversary. Firstly, the adversary must declare upfront (before seeing the public parameters) the set of input values to be used in the challenge stage. This is in contrast to the full game where the inputs are chosen after the adversary has oracle access to the system. Secondly, the adversary must (e.g. on line 6 in Game 3), declare a list $\overline{R}$ of servers that must be revoked when the challenge encoded inputs are generated from $\mathsf{ProbGen}$. The adversary must do this before receiving oracle access.

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.[4] At present, instantiating such a primitive is an open problem.

To implement the semi-static restriction, we must alter the games somewhat from their full versions. The challenger must now define two additional parameters: t and Q_{Rev}. The variable t models system time and is initialized to 1. It is incremented each time a revoke query is made to illustrate that keys generated at prior time periods may no longer function. In the IND-sHRSS game, 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 and as such time must increase linearly. 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 would allow a trivial win against the challenge ciphertext. 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 of revoke queries he is allowed to make in its first query phase to be q_t (and define security over all choices of q_t). Since t is incremented only when a Revoke query is made, the challenge will occur at time $t^\star = q_t$ (or $q_t + 1$ in the case of the Revocation game), 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 (initialized 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 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.

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. Thus, as shown in Oracle Query 2, a Revoke query will increment the time parameter t and return $\perp$ if the queried token is $(\mathsf{accept}, \cdot)$ i.e. there is no server to revoke. Since t is still incremented, the adversary may

[4] Attrapadung et al. [1] defined a notion with adaptive queries but did not provide an instantiation.

query acceptance tokens to Revoke in order to progress the system time if desired. If the 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. If $\perp$ was not returned in this case, the adversary would receive a valid update key, which combined with a decryption key would form a valid evaluation key for a revoked server.

Similarly, as specified in Oracle Query 1, a query to the Certify oracle will result in $\perp$ if made during the time period q_t and if, excluding S as it is about to be revoked, there is a server that should be revoked according to $\overline{R}$ but is not actually revoked. The challenger also returns $\perp$ if the query is made for the challenge function F (given as a parameter to the adversary) *and* the queried identity is not on the list of servers to be revoked at challenge time, $\overline{R}$. Otherwise, an evaluation key would be issued for F and for a server that will not be revoked at the challenge time, and hence the server will have a valid update key and so a fully functional evaluation key that can decrypt the challenge ciphertexts.

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.

3.6.1 Public Verifiability

We extend the Public Verifiability game of Parno et al. [14] to formalize that multiple servers should not be able to collude to gain an advantage in convincing *any* verifying party of an incorrect output (i.e. that Verify returns accept on an encoded output $\theta^\star$ not corresponding to the true output of the computation). Note that this game is a generalization of the Public Verifiability game of Parno et al. [14] since they consider the case where the adversary is limited to learning only one evaluation key and one encoded input. The motivation for this updated game is that there is a now a trusted party issuing keys to multiple servers who may collude, as opposed to the traditional model in which the system comprises a single client choosing a single server to whom to outsource a computation. Thus we allow the adversary to collect multiple inputs from clients (though oracle access) and to learn multiple evaluation keys for different functions and associated with different servers (since evaluation keys are server-specific in our setting to enable per-server revocation).

Full Public Verifiability. This is captured in the full game presented in Game 2. The game begins with the challenger setting up the system and running FnInit to initialize the challenge function F. The adversary, $\mathcal{A}$, is given the resulting public parameters and given oracle access to $\mathsf{FnInit}(\cdot, MK, PP)$, $\mathsf{Register}(\cdot, MK, PP)$, $\mathsf{Certify}(\cdot,\cdot,\cdot, MK, PP)$ and $\mathsf{Revoke}(\cdot,\cdot,\cdot, MK, PP)$ as mentioned previously. All oracles simply run the relevant algorithm.

Eventually, the adversary will finish this query phase and output a challenge input $x^\star$. The challenger will then generate a challenge by running 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 $F(x^\star)$.

Selective, semi-static Public Verifiability. As mentioned in Section 3.6, we also define a selective, semi-static notion of Public Verifiability in Game 3. The adversary first selects an input value to be outsourced. The challenger initializes a list of currently revoked entities

Game 1 $\mathbf{Exp}_{\mathcal{A}}^{mPubVerif}[\mathcal{RPVC}, F, 1^\kappa]$:

```
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$\{x_i^\star\}_{i \in [n]} \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
for $i = 1$ to $n$ do
    $(\sigma_{F,x_i^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}) \leftarrow \mathsf{ProbGen}(x_i^\star, PK_F, PP)$;
$\theta^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\{\sigma_{F,x_i^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}\}, PK_F, PP)$;
if $(\exists i \in [n]$ s.t. $(((\tilde{y}, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, VK_{F,x_i^\star}, RK_{F,x_i^\star}, PP))$
and $((\tilde{y}, \tau_{\theta^\star}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ and $(\tilde{y} \neq F(x_i^\star))))$
    return 1;
else return 0;
```

Game 2 $\mathbf{Exp}_{\mathcal{A}}^{PubVerif}[\mathcal{RPVC}, F, 1^\kappa]$:

```
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$x^\star \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
$(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}) \leftarrow \mathsf{ProbGen}(x^\star, PK_F, PP)$;
$\theta^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}, PK_F, PP)$;
if $((((\tilde{y}, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, VK_{F,x^\star}, RK_{F,x^\star}, PP))$
and $((\tilde{y}, \tau_{\theta^\star}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ and $(\tilde{y} \neq F(x^\star))))$
    return 1;
else return 0;
```

Q_{Rev} and a time parameter t before running Setup and FnInit to create a public delegation key for the function F given as a parameter to the game(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 given oracle access to the above functions which simulate all values known to a real server as well as those learnt through corrupting entities. The challenger responds to $\mathsf{Certify}$ and Revoke queries as detailed in Oracle Queries 1 and 2 respectively. It 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.

Once the adversary has finished this query phase (and in particular, due to the parameterisation of the adversary, after exactly q_t Revoke queries), the challenger must check that the queries made by the adversary has indeed left the list of revoked entities to be at least that selected beforehand by the adversary. If there is a server that the adversary included on $\overline{R}$ but is not currently revoked, then the adversary loses the game. Otherwise, the challenger generates the challenge by running $\mathsf{ProbGen}$ on $x^\star$. The adversary is given the resulting encoded input and oracle access again, and wins the game if it creates an encoded output that verifies correctly yet does not encode the correct value $F(x^\star)$.

Definition 3. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q (including q_t* Revoke *queries), where $\mathbf{X} \in \{mPubVerif, PubVerif\}$, is defined as:*

- $Adv_{\mathcal{A}}^{\mathbf{X}}(\mathcal{RPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathbf{X}}[\mathcal{RPVC}, F, 1^\kappa] = 1]$
- $Adv_{\mathcal{A}}^{sSS\text{-}PubVerif}(\mathcal{RPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}PubVerif}[\mathcal{RPVC}, F, q_t, 1^\kappa] = 1]$

A RPVC is secure against Game $\mathbf{X}$ or sSS-PubVerif *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{\mathbf{X}, sSS\text{-}PubVerif}(\mathcal{RPVC},\ F,\ 1^\kappa, q) \leq \operatorname{negl}(\kappa).$$

Game 3 $\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}PubVerif}[\mathcal{RPVC}, F, q_t, 1^\kappa]$:

```
$x^\star \leftarrow \mathcal{A}(1^\kappa)$;
$Q_{\text{Rev}} = \epsilon$;
$t = 1$;
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$\overline{R} \leftarrow \mathcal{A}(PK_F, PP)$;
$\mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
if $(\overline{R} \not\subseteq Q_{\text{Rev}})$ return $0$;
$(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}) \leftarrow \mathsf{ProbGen}(x^\star, PK_F, PP)$;
$\theta^\star \leftarrow \mathcal{A}^{\mathcal{O}}(\{\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}\}, EK_{F,\mathcal{A}}, SK_{\mathcal{A}}, PK_F, PP)$;
if $((((\tilde{y}, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, VK_{F,x^\star}, RK_{F,x^\star}, PP))$
and $((\tilde{y}, \tau_{\theta^\star}) \neq (\perp, (\mathsf{reject}, \cdot)))$ and $(\tilde{y} \neq F(x^\star))))$
    return $1$;
else return $0$;
```

Oracle Query 1 $\mathcal{O}^{\mathsf{Certify}}(S, F', MK, PP)$:

```
if $((F' = F$ and $S \notin \overline{R})$ or $(t = q_t$ and $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S))$ return $\perp$;
$Q_{\text{Rev}} = Q_{\text{Rev}} \setminus S$;
return $\mathsf{Certify}(S, F', MK, PP)$;
```

Oracle Query 2 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(x)}}, MK, PP)$:

```
$t = t + 1$;
if $(\tau_{\theta_{F'(x)}} = (\mathsf{accept}, \cdot))$ return $\perp$;
if $(t = q_t$ and $\overline{R} \not\subseteq Q_{\text{Rev}} \cup S)$ return $\perp$;
$Q_{\text{Rev}} = Q_{\text{Rev}} \cup S$;
return $\mathsf{Revoke}(\tau_{\theta_{F'(x)}}, MK, PP)$;
```

3.6.2 Revocation

The notion of revocation requires that any subsequent computations by a server detected as misbehaving (i.e. a result for $F(x)$ causes the Verify algorithm to output $(\perp, (\mathsf{reject}, S)))$ should be rejected (even if the result is correct). The motivation here is that even though the costly computation and pre-processing stages have been outsourced to the server and KDC respectively, there is still a cost to delegating and verifying a computation. We remove any incentive for an untrustworthy server to attempt to provide an outsourcing service (since it knows the result will not be accepted). In addition, we may punish and further disincentivise 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.

Full Revocation. As before, we define both a full game and a weaker variant that we can achieve with current primitives. The full notion, in Game 4, begins by declaring a Boolean flag `chall` which in initially set to `false` and a list Q_{Rev} which servers will be added to when revoked and removed from when 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 and FnInit to initialize the system and providing the public parameters to the adversary along with oracle access. All oracles simply run the relevant algorithms except for Certify and Revoke which additionally maintain the list of revoked entities as mentioned above and specified

Game 4 $\mathbf{Exp}_{\mathcal{A}}^{Revocation}\left[\mathcal{RPVC}, F, 1^{\kappa}\right]$:

```
$\mathtt{chall} = \mathtt{false}$;
$Q_{\mathrm{Rev}} = \epsilon$;
$(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$x^{\star} \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
$\mathtt{chall} = \mathtt{true}$;
$(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \leftarrow \mathsf{ProbGen}(x^{\star}, PK_F, PP)$;
$\theta^{\star} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, PP)$;
if $(((\tilde{y}, (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PP))$
and $(S \in Q_{\mathrm{Rev}})$ then
    return 1
else
    return 0
```

Oracle Query 3 $\mathcal{O}^{\mathsf{Certify}}(S, F', MK, PP)$:

```
if $(\mathtt{chall} = \mathtt{false})$ $Q_{\mathrm{Rev}} = Q_{\mathrm{Rev}} \setminus S$;
return $\mathsf{Certify}(S, F', MK, PP)$;
```

Oracle Query 4 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{\theta_{F'(x)}}, F', MK, PP)$:

```
$r \leftarrow \mathsf{Revoke}(\tau_{\theta_{F'(x)}}, F', MK, PP)$;
if $(r \neq \perp$ and $\mathtt{chall} = \mathtt{false})$ $Q_{\mathrm{Rev}} = Q_{\mathrm{Rev}} \cup S$;
return $r$;
```

Game 5 $\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}Revocation}\left[\mathcal{RPVC}, F, q_t, 1^{\kappa}\right]$:

```
$x^{\star} \leftarrow \mathcal{A}(1^{\kappa})$;
$Q_{\mathrm{Rev}} = \epsilon$;
$t = 1$;
$(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$\overline{R} \leftarrow \mathcal{A}(PK_F, PP)$;
$\mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
if $(\overline{R} \not\subseteq Q_{\mathrm{Rev}})$ return 0;
$(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \leftarrow \mathsf{ProbGen}(x^{\star}, PK_F, PP)$;
$\theta^{\star} \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, PP)$;
if $(((\tilde{y}, (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PP))$
and $(S \in \overline{R})$ then
    return 1
else
    return 0
```

in Oracle Queries 3 and 4 respectively. After the adversary has finished this query phase, it outputs a challenge input $x^{\star}$, and the challenger sets the chall flag to true. It then generates the challenge by running ProbGen on $x^{\star}$ and gives the resulting parameters to the adversary along with oracle access again (however, since chall is set, 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 the challenge was generated (even a correct result).

Selective, semi-static Revocation. On the other hand, the selective, semi-static notion of Revocation given in Game 5 proceeds exactly as the sSS-PubVerif game for Public Verifiability except for the winning condition. Here, 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 which the adversary chose to be (at least) those servers on $\overline{R}$. This game also uses the Certify and Revoke oracles specified in Oracle Queries 1 and 2 respectively.

Game 6 $\mathbf{Exp}_{\mathcal{A}}^{\mathit{VindictiveS}}[\mathcal{RPVC}, F, 1^{\kappa}]$:

```
1: $Q_{\text{Reg}} = \epsilon$;
2: $(PP, MK) \leftarrow \mathsf{Setup}(1^{\kappa})$;
3: $PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
4: $x^{\star} \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
5: $(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}) \leftarrow \mathsf{ProbGen}(x^{\star}, PK_F, PP)$;
6: $\tilde{S} \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{Register2}}(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, PP)$ subject to (1);
7: $\theta^{\star} \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{Compute},\mathsf{Register2}}(\sigma_{F,x^{\star}}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PK_F, PP)$ subject to (2);
8: if $((\tilde{y}, \tau_{\theta^{\star}}) \leftarrow \mathsf{Verify}(\theta^{\star}, VK_{F,x^{\star}}, RK_{F,x^{\star}}, PP))$
   and $((\tilde{y}, \tau_{\theta^{\star}}) = (\perp, (\mathsf{reject}, \tilde{S})))$ and $(\perp \not\leftarrow \mathsf{Revoke}(\tau_{\theta^{\star}}, MK, PP)))$ then
9:     return 1
10: else
11:     return 0
```

Definition 4. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q against Revocation or sSS-Revocation is defined as:*

- $\mathit{Adv}_{\mathcal{A}}^{\mathit{Revocation}}(\mathcal{RPVC}, F, 1^{\kappa}, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathit{Revocation}}[\mathcal{RPVC}, F, 1^{\kappa}] = 1]$
- $\mathit{Adv}_{\mathcal{A}}^{\mathit{sSS\text{-}Revocation}}(\mathcal{RPVC}, F, 1^{\kappa}, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathit{sSS\text{-}Revocation}}[\mathcal{RPVC}, F, q_t, 1^{\kappa}] = 1]$

A RPVC is secure against Revocation or sSS-Revocation *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$\mathit{Adv}_{\mathcal{A}}^{\mathit{Revocation,sSS\text{-}Revocation}}(\mathcal{RPVC},\ F,\ 1^{\kappa}, q) \leq \operatorname{negl}(\kappa).$$

3.6.3 Vindictive Server

This notion is motivated by the manager model where a pool of computational servers is available to accept a 'job' but they are abstracted by the manager such that the client does not know the individual server identities. 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 an alternate server ID such that an (honest) server is revoked and the pool of available servers for future computations is reduced in size, leading to a likely increase in reward for the malicious server.

In Game 6, the challenger maintains a list of registered entities Q_{Reg}. The game proceeds similarly to the previous notions, except that, on lines 6 and 7, the adversary selects a target server ID, $\tilde{S}$, he wishes to be revoked and generates an encoded output that will cause this. He is given oracle access subject to the following constraints to avoid trivial wins:

(1) No query of the form $\mathcal{O}^{\mathsf{Register}}(\tilde{S}, MK, PP)$ was made;

(2) As above and no query $\mathcal{O}^{\mathsf{Compute}}(\sigma_{F,x_i^{\star}}, EK_{F,\tilde{S}}, SK_{\tilde{S}}, PP)$ was made.

In addition, he is provided with an oracle, Register2, which performs the 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 Register2 (including $\tilde{S}$). We also modify the standard Register oracle such that if an identity has been previously queried to the Register2 oracle, it generates the same parameters (and vice versa). These oracles are shown in Oracle Queries 5 and 6. All other oracles simply run the relevant algorithm.The adversary wins if the KDC believes $\tilde{S}$ returned $\tilde{y}$ and revokes $\tilde{S}$.

Definition 5. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q in the Vindictive Server Experiment is defined as:*

$$\mathit{Adv}_{\mathcal{A}}^{\mathit{VindictiveS}}(\mathcal{RPVC}, F, 1^{\kappa}, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathit{VindictiveS}}[\mathcal{RPVC}, F, 1^{\kappa}] = 1].$$

A RPVC is secure against vindictive servers *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$\mathit{Adv}_{\mathcal{A}}^{\mathit{VindictiveS}}(\mathcal{RPVC},\ F,\ 1^{\kappa}, q) \leq \operatorname{negl}(\kappa).$$

Oracle Query 5 $\mathcal{O}^{\mathsf{Register}}(S, MK, PP)$:

```
if $(S, \cdot) \notin Q_{\mathrm{Reg}}$ then
    $SK_S \leftarrow \mathsf{Register}(S, MK, PP)$;
    $Q_{\mathrm{Reg}} = Q_{\mathrm{Reg}} \cup (S, SK_S)$;
return $SK_S$
```

Oracle Query 6 $\mathcal{O}^{\mathsf{Register2}}(S, MK, PP)$:

```
if $(S, \cdot) \notin Q_{\mathrm{Reg}}$ then
    $SK_S \leftarrow \mathsf{Register}(S, MK, PP)$;
    $Q_{\mathrm{Reg}} = Q_{\mathrm{Reg}} \cup (S, SK_S)$;
return $\perp$
```

3.6.4 Vindictive Manager

This is a natural extension of the Public Verifiability notion to the manager model where a vindictive manager may attempt to provide a client with an incorrect answer. 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 output from the server so that the client can performsthe full Verify step as well (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 the clients would still like a final, efficient check.

Full Vindictive Managers. Game 7 begins with the challenger initializing the system as usual. The adversary is given oracle access (each oracle runs the relevant algorithm) and outputs a challenge input $x^\star$. The challenger now selects a random server identity from the space of all identities $\mathcal{U}_{\mathrm{ID}}$ that it will use to generate the challenge. It runs Register and Certify for this server (if not already done so), 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 and the output from Compute, as well as oracle access, and 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 $\tilde{y}$, and the adversary wins if the challenger accepts this output and $\tilde{y} \neq F(x^\star)$.

Selective, semi-static Vindictive Managers. The weaker variant, in Game 8, performs similarly. First, the adversary selects its challenge input $x^\star$, and the challenger initializes a list of revoked entities Q_{Rev} and a time paramter t. It also sets up the system and gives the public parameters to the adversary, who must select a list $\overline{R}$ of servers to be revoked at the challenge time. We require that $\overline{R}$ is not the full set of all servers in the system, as one non-revoked identity is required to generate the challenge. The adversary then gets oracle access (using the Certify and Revoke oracles specified in Oracle Queries 1 and 2 respectively). If, after finishing this query phase (and in particular after q_t Revoke queries), the list of revoked entities does not include $\overline{R}$ then the adversary loses the game. Otherwise, a server S is chosen at random from the set of all server identities $\mathcal{U}_{\mathrm{ID}}$ excluding $\overline{R}$ (as these must be revoked at the challenge time). This server is used to generate the challenge. If not already done, the challenger registers and certifies S for F, and runs ProbGen on the challenge input, before finally running 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, and must output a retrieval token $RT_{F,x^\star}$ and an

Game 7 $\mathbf{Exp}_{\mathcal{A}}^{VindictiveM}[\mathcal{RPVC}, F, 1^\kappa]$:

```
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$x^\star \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
$S \xleftarrow{\$} \mathcal{U}_{\mathrm{ID}}$;
$SK_S \leftarrow \mathsf{Register}(S, MK, PP)$;
$EK_{F,S} \leftarrow \mathsf{Certify}(S, F, MK, PP)$;
$(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}) \leftarrow \mathsf{ProbGen}(x^\star, PK_F, PP)$;
$\theta_{F(x^\star)} \leftarrow \mathsf{Compute}(\sigma_{F,x^\star}, EK_{F,S}, SK_S, PP)$;
$(RT_{F,x^\star}, \tau_{\theta_{F(x^\star)}}) \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, \theta_{F(x^\star)}, VK_{F,x^\star}, PK_F, PP)$;
if ($\tilde{y} \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x^\star)}}, RT_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}, PP)$)
 and ($\tilde{y} \neq F(x^\star)$) and ($\tilde{y} \neq \perp$) then
    return 1
else
    return 0
```

Game 8 $\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}VindictiveM}[\mathcal{RPVC}, F, q_t, 1^\kappa]$:

```
$x^\star \leftarrow \mathcal{A}(1^\kappa)$;
$Q_{\mathrm{Rev}} = \epsilon$;
$t = 1$
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$\overline{R} \leftarrow \mathcal{A}(PK_F, PP)$
$\mathcal{A}^{\mathcal{O}}(PK_F, PP)$;
if (($\overline{R} \not\subseteq Q_{\mathrm{Rev}}$) or ($\overline{R} = \mathcal{U}_{\mathrm{ID}}$)) return 0;
$S \xleftarrow{\$} \mathcal{U}_{\mathrm{ID}} \setminus \overline{R}$;
$SK_S \leftarrow \mathsf{Register}(S, MK, PP)$;
$EK_{F,S} \leftarrow \mathsf{Certify}(S, F, MK, PP)$;
$(\sigma_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}) \leftarrow \mathsf{ProbGen}(x^\star, PK_F, PP)$;
$\theta_{F(x^\star)} \leftarrow \mathsf{Compute}(\sigma_{F,x^\star}, EK_{F,S}, SK_S, PP)$;
$(RT_{F,x^\star}, \tau_{\theta_{F(x^\star)}}) \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{F,x^\star}, \theta_{F(x^\star)}, VK_{F,x^\star}, PK_F, PP)$;
if ($\tilde{y} \leftarrow \mathsf{Retrieve}(\tau_{\theta_{F(x^\star)}}, RT_{F,x^\star}, VK_{F,x^\star}, RK_{F,x^\star}, PP)$)
 and ($\tilde{y} \neq F(x^\star)$) and ($\tilde{y} \neq \perp$) then
    return 1
else
    return 0
```

acceptance token $\tau_{\theta_{F(x^\star)}}$. The challenger runs Retrieve on $RT_{F,x}$ to get an output value $\tilde{y}$, and the adversary wins if the challenger accepts this output and $\tilde{y} \neq F(x^\star)$.

Definition 6. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q against VindictiveM or sSS-VindictiveM is defined as:*

- $Adv_{\mathcal{A}}^{VindictiveM}(\mathcal{RPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{Revocation}[\mathcal{RPVC}, F, 1^\kappa] = 1]$
- $Adv_{\mathcal{A}}^{sSS\text{-}VindictiveM}(\mathcal{RPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{sSS\text{-}Revocation}[\mathcal{RPVC}, F, q_t, 1^\kappa] = 1]$

A RPVC is secure against VindictiveM or sSS-VindictiveM *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{VindictiveM, sSS\text{-}VindictiveM}(\mathcal{RPVC},\ F,\ 1^\kappa, q) \leq \mathrm{negl}(\kappa).$$

3.6.5 Blind Verification

With this notion we aim to show that a verifier that does not hold the retrieval token $RT_{F,x}$ chosen in ProbGen cannot learn the value of $F(x)$ given the encoded output. This property was hinted at by Parno et al. [14] but was not formalized. The game begins as usual with the challenger initializing the system. The challenger then selects an input at random from the domain of F, and a random server S. It registers and certifies S, runs ProbGen for the chosen

Game 9 $\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{RPVC}, F, 1^\kappa]$:

```
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$PK_F \leftarrow \mathsf{FnInit}(F, MK, PP)$;
$x \xleftarrow{\$} \mathsf{Dom}(F)$;
$S \xleftarrow{\$} \mathcal{U}_{\mathrm{ID}}$;
$SK_S \leftarrow \mathsf{Register}(S, MK, PP)$;
$EK_{F,S} \leftarrow \mathsf{Certify}(S, F, MK, PP)$;
$(\sigma_{F,x}, VK_{F,x}, RK_{F,x}) \leftarrow \mathsf{ProbGen}(x, PK_F, PP)$;
$\theta_{F(x)} \leftarrow \mathsf{Compute}(\sigma_{F,x}, EK_{F,S}, SK_S, PP)$;
$\hat{y} \leftarrow \mathcal{A}^{\mathcal{O}}(\theta_{F(x)}, VK_{F,x}, PK_F, PP)$;
if $(\hat{y} = F(x))$ then
    return 1
else
    return 0
```

input and runs Compute to generate an output $\theta_{F(x)}$. This is given to the adversary along with the verification key and oracle access, and the adversary wins if it can guess the value of $F(x)$ without seeing the retrieval key. Clearly, the adversary can trivially make a guess for $F(x)$ based on a priori knowledge of the distribution of F over all possible inputs. Unless F is balanced (i.e. outputs 1 exactly half the time), the adversary could gain an advantage. Thus, we define security by subtracting the most likely guess for $F(x)$.

Note that in this game we do not provide the adversary with access to the encoded inputs. In KP-ABE, the ciphertext reveals the set of attributes it was encrypted under (and hence the input values) and the adversary may simply compute F on this input to learn the output independently of the encoded output computed by the server. In practice, it may be desirable to give access to the ciphertexts such that a manager may distribute the input to a chosen server. In this case, one should replace the KP-ABE scheme with a predicate encryption scheme which provides input privacy and then our blind verification technique will apply straightforwardly even with access to the encoded input. Finding an indirectly revocable predicate encryption scheme will be the subject of future work.

Definition 7. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q in the Blind Verification Experiment is defined as:*

$$Adv_{\mathcal{A}}^{BVerif}(\mathcal{RPVC}, F, 1^\kappa, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{RPVC}, F, 1^\kappa] = 1] - \max_{y \in \mathsf{Ran}(F)} \Big(\Pr_{x \in \mathsf{Dom}(F)}[F(x) = y]\Big).$$

A RPVC is secure against vindictive servers *for a function F, if for all PPT adversaries $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{BVerif}(\mathcal{RPVC},\ F,\ 1^\kappa, q) \leq \mathrm{negl}(\kappa).$$

4 Construction

We now provide an instantiation of a RPVC scheme. Our construction is based on that used by Parno et al. [14] (summarized in App. A.2) which uses Key-Policy Attribute-based Encryption (KP-ABE) in a black-box manner to outsource the computation of a Boolean function. We restrict our attention to Boolean functions, and in particular the complexity class NC^1 which includes all circuits of depth $\mathcal{O}(\log n)$. Thus functions we can outsource can be built from common operations such as AND, OR, NOT, equality and comparison operators, arithmetic operators and regular expressions. Notice that to achieve the outsourced evaluation of functions with n bit outputs, it is possible to evaluate n different functions, each of which applies a mask to output the single bit in position i.

Notice also that 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 NP-complete problems 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 return that a solution cannot be found for the given problem instance, and the restricted client could not verify the correctness of this statement.

Recall that if $\perp$ is returned by the server 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 $\mathcal{F}$. Then, the client encrypts two random messages m_0 and m_1. The server is required to return the decryption of those ciphertexts. Thus, a well-formed response $\theta_{F(x)}$, comprising recovered plaintexts (d_b, d_{1-b}), satisfies the following, where $RK_{F,x} = b$:

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

Hence, the client will be able to detect whether the server has misbehaved.

4.1 Technical Details

We require an *indirectly revocable KP-ABE scheme* comprising the algorithms $\mathsf{ABE.Setup}$, $\mathsf{ABE.KeyGen}$, $\mathsf{ABE.KeyUpdate}$, $\mathsf{ABE.Encrypt}$ and $\mathsf{ABE.Decrypt}$. We also use a signature scheme with algorithms $\mathsf{Sig.KeyGen}$, $\mathsf{Sig.Sign}$ and $\mathsf{Sig.Verify}$, and a one-way function g. Let $\mathcal{U}$ be the universe of attributes acceptable by the ABE scheme, and let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\mathcal{F}}$ where: attributes in $\mathcal{U}_{\text{attr}}$ form characteristic tuples for input data, as detailed in Appendix A.2; $\mathcal{U}_{\text{ID}}$ comprises attributes representing entity identifiers; $\mathcal{U}_{\text{time}}$ comprises attributes representing time periods issued by the time source $\mathbb{T}$; and finally $\mathcal{U}_{\mathcal{F}}$ comprises attributes that represent functions in $\mathcal{F}$. Define a bijective mapping between functions $F \in \mathcal{F}$ and attributes $f \in \mathcal{U}_{\mathcal{F}}$. Then the policy $F \wedge f$ denotes adding a conjunctive clause requiring the presence of the label f to the expression of the function F, and $(x \cup f)$ denotes adding the function attribute to the attribute set representing the input data x. This will prevent servers using alternate evaluation keys for a given input and hence we are able to certify servers to compute multiple functions.

Parno et al. [14] considered two models of publicly verifiable computation, namely single function and multi-function. In single function PVC, the function to be computed is embedded in the public parameters, whilst in multi-function PVC delegation keys for multiple functions can be generated and a single encoded input can be used to input the same data to multiple functions. To achieve this latter notion, Parno et al. required the somewhat complex primitive of KP-ABE with Outsourcing [11]. 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, and we also believe that it is a reasonable cost expectation to prepare an encoded input per computation, and 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 use the simple trick of adding a conjunctive clause to the functions requiring the presence of the appropriate function label in the input data – that is, the function F is encoded in a decryption key for the policy $F \wedge f$ where f is the attribute representation of F in $\mathcal{U}_{\mathcal{F}}$; the complement function $\overline{F}$ is encoded as a key for $\overline{F} \wedge f$; and we encode the input data x to the function F as $x \cup f$. Thus, the client must perform the $\mathsf{ProbGen}$ stage per computation as the function label in the input data will differ, but servers can be certified for multiple functions and may not use a key for one function to compute on data intended for another (since the function label required by the conjunctive clause in the key will not be present in the input data). 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.

The scheme of Parno et al. 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 (where the adversary is given a KeyGen oracle and the scheme must prevent collusion between holders of different decryption keys). Parno et al. could use this property due to their restricted system model where the client is certified for only a single function per set of public parameters (so the client must set up a new ABE environment per function). In our setting, we must be able to certify servers for multiple functions and hence the KDC must be able to issue multiple keys and we require the more standard, multi-key notion of security usually considered for ABE schemes.

4.2 Instantiation

Informally the scheme operates as follows.

1. RPVC.Setup establishes public parameters and a master secret key by calling the ABE.Setup algorithm twice. This algorithm also initializes a time source[5] $\mathbb{T}$, a list of revoked servers, and a two-dimensional array of registered servers L_{Reg} – the array is indexed in the first dimension by server identities and the first dimension will store signature verification keys while the second will store a list of functions that server is authorized to compute.
2. RPVC.FnInit simply outputs the public parameters. This step is not required in our particular construction, but we retain the algorithm for generality and to enable further computations.
3. RPVC.Register creates a public-private key pair by calling the signature KeyGen algorithm. This is run by the KDC (or the manager in the manager model) and updates L_{Reg} to store the verification key for S.
4. RPVC.Certify creates the key $EK_{F,S}$ that will be used by a server S to compute F by calling the ABE.KeyGen and ABE.KeyUpdate algorithms twice – once with a "policy" for F and once with the complement $\overline{F}$. It also updates L_{Reg} to include F. Note that since we have a form of multi-function PVC, we must 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 claiming it as a result for F. To prevent this, 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 F attribute) will only satisfy a key issued for F (comprising the F conjunctive clause), and a key for G will not be satisfied as G is not in the input set. The algorithm also updates the lists to remove the server from the revocation list and to add F to the list of functions it is authorized for.
5. RPVC.ProbGen creates a problem instance $\sigma_{F,x} = (c_b, c_{1-b})$ by encrypting two randomly chosen messages under an attribute set corresponding to x, and a verification key $VK_{F,x}$ by applying a one-way function g (such as a pre-image resistant hash function) to the messages. The ciphertexts and verification tokens are ordered randomly according to $RK_{F,x} = b$ for a random bit b, such that the positioning of an element does not imply whether it relates to F or to $\overline{F}$. The output also includes a copy of L_{Reg} from the public parameters incase the list changes between now and verification time, e.g. a server is revoked. This copy may be removed if verification is likely to be imminent or if results computed *before* a server was revoked should be rejected.

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

6. RPVC.Compute is run by a server S. Given an input $\sigma_{F,x} = (c_b, c_{1-b})$ it returns $(m_0, \perp)$ if $F(x) = 1$ or $(\perp, m_1)$ if $F(x) = 0$ (ordered according to $RK_{F,x}$ chosen in RPVC.ProbGen) and a signature on the output.
7. RPVC.Verify either accepts the output $\theta_{F(x)} = (d_b, d_{1-b})$ or rejects it. This algorithm verifies the signature on the output and confirms the output is correct by applying g and comparing with $VK_{F,x}$. In RPVC.BVerif the verifier can compare pairwise between the components of $\theta_{F(x)}$ and $VK_{F,x}$ to determine correctness but as they are unaware of the value of $RK_{F,x}$, they do not know the order of these elements and hence whether the correct output corresponds to F or $\overline{F}$ being satisfied i.e. if $F(x) = 1$ or 0 respectively. The verifier outputs an accept or reject token as well as the output value $RT_{F,x} \in \{d_b, d_{1-b}, \perp\}$ where $RK_{F,x} = b$. Parno et al. [14] 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 RPVC.Retrieve a verifier that has knowledge of $RK_{F,x}$ can check whether the output from BVerif matches m_0 or m_1.
8. RPVC.Revoke is run by the KDC and redistributes fresh keys to all non-revoked servers. This algorithm first refreshes the time source $\mathbb{T}$ (e.g. increments $\mathbb{T}$ if it is a counter). It then updates L_{Reg} and L_{Rev}, and updates $EK_{F,S}$ using the results of two calls to the ABE.KeyUpdate algorithm.

We require two distinct sets of system parameters in Step 1 for the security proof to work. In Step 4 we have to run the ABE.KeyGen algorithm twice – once for F and once for $\overline{F}$. However, to prevent a trivial win in the IND-sHRSS game, the adversary is not allowed to query for a key with a policy that is satisfied by the challenge input attributes. By definition, either $F(x)$ or $\overline{F}(x)$ will output 1 and hence one of these will not be able to be queried to the Challenger. Thus we use the two separate parameters such that the non-satisfied function can be queried to the Challenger and the adversary can use the other set of parameters to generate a key himself.

More formally, our scheme is defined by Algorithms 1–9.

Alg. 1 $(PP, MK) \leftarrow$ RPVC.Setup(1^κ)

```
Let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\mathcal{F}}$
$(MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}}) \leftarrow$ ABE.Setup$(1^\kappa, \mathcal{U})$
$(MPK^1_{\text{ABE}}, MPK^1_{\text{ABE}}) \leftarrow$ ABE.Setup$(1^\kappa, \mathcal{U})$
for $S \in \mathcal{U}_{\text{ID}}$ do
    $L_{\text{Reg}}[S][0] = \epsilon$
    $L_{\text{Reg}}[S][1] = \{\epsilon\}$
$L_{\text{Rev}} = \epsilon$
Initialise $\mathbb{T}$
$PP = (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{\text{Reg}}, \mathbb{T})$
$MK = (MSK^0_{\text{ABE}}, MSK^1_{\text{ABE}}, L_{\text{Rev}})$
```

Alg. 2 $PK_F \leftarrow$ RPVC.FnInit(F, MK, PP)

```
Set $PK_F = PP$
```

Alg. 3 $SK_S \leftarrow$ RPVC.Register(S, MK, PP)

```
$(SK_{\text{Sig}}, VK_{\text{Sig}}) \leftarrow$ Sig.KeyGen$(1^\kappa)$
$SK_S = SK_{\text{Sig}}$
$L_{\text{Reg}}[S][0] = VK_{\text{Sig}}$
```

Alg. 4 $EK_{F,S} \leftarrow \mathsf{RPVC.Certify}(S, F, MK, PP)$

```
$L_{\mathrm{Reg}}[S][1] = L_{\mathrm{Reg}}[S][1] \cup F$
$L_{\mathrm{Rev}} = L_{\mathrm{Rev}} \setminus S$
$t \leftarrow \mathbb{T}$
$SK^0_{\mathrm{ABE}} \leftarrow \mathsf{ABE.KeyGen}(S, F \wedge f, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
$SK^1_{\mathrm{ABE}} \leftarrow \mathsf{ABE.KeyGen}(S, \overline{F} \wedge f, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
$UK^0_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
$UK^1_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
$EK_{F,S} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$
```

Alg. 5 $(\sigma_{F,x}, VK_{F,x}, RK_{F,x}) \leftarrow \mathsf{RPVC.ProbGen}(x, PK_F, PP)$

```
$t \leftarrow \mathbb{T}$
$(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
$b \xleftarrow{\$} \{0, 1\}$
$c_b \leftarrow \mathsf{ABE.Encrypt}(m_b, (x \cup f), t, MPK^0_{\mathrm{ABE}})$
$c_{1-b} \leftarrow \mathsf{ABE.Encrypt}(m_{1-b}, (x \cup f), t, MPK^1_{\mathrm{ABE}})$
Output: $\sigma_{F,x} = (c_b, c_{1-b})$, $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$ and $RK_{F,x} = b$
```

Alg. 6 $\theta_{F(x)} \leftarrow \mathsf{RPVC.Compute}(\sigma_{F,x}, EK_{F,S}, SK_S, PP)$

```
Input: $EK_{F,S} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$ and $\sigma_{F,x} = (c_b, c_{1-b})$
Parse $\sigma_{F,x}$ as $(c, c')$
$d_b \leftarrow \mathsf{ABE.Decrypt}(c, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
$d_{1-b} \leftarrow \mathsf{ABE.Decrypt}(c', SK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, UK^1_{L_{\mathrm{Rev}},t})$
$\gamma \leftarrow \mathsf{Sig.Sign}((d_b, d_{1-b}, S), SK_S)$
Output: $\theta_{F(x)} = (d_b, d_{1-b}, S, \gamma)$
```

Alg. 7 $(RT_{F,x}, \tau_{\theta_{F(x)}}) \leftarrow \mathsf{RPVC.BVerif}(\theta_{F(x)}, VK_{F,x}, PP)$

```
Input: $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$ and $\theta_{F(x)} = (d_b, d_{1-b}, S, \gamma)$
if $F \in L_{\mathrm{Reg}}[S][1]$ then
    if $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}((d_b, d_{1-b}, S), \gamma, L_{\mathrm{Reg}}[S][0])$ then
        if $g(m_b) = g(d_b)$ then Output $(RT_{F,x} = d_b, \tau_{\theta_{F(x)}} = (\mathsf{accept}, S))$
        else if $g(m_{1-b}) = g(d_{1-b})$ then Output $(RT_{F,x} = d_{1-b}, \tau_{\theta_{F(x)}} = (\mathsf{accept}, S))$
        else Output $(RT_{F,x} = \perp, \tau_{\theta_{F(x)}} = (\mathsf{reject}, S))$
Output $(RT_{F,x} = \perp, \tau_{\theta_{F(x)}} = (\mathsf{reject}, \perp))$
```

Alg. 8 $\tilde{y} \leftarrow \mathsf{RPVC.Retrieve}(\tau_{\theta_{F(x)}}, RT_{F,x}, VK_{F,x}, RK_{F,x}, PP)$

```
Input: $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $\theta_{F(x)} = (d_b, d_{1-b}, S, \gamma)$, $RK_{F,x} = b$, and $(RT_{F,x}, \tau_{\theta_{F(x)}})$ where $RT_{F,x} \in$
    $\{d_b, d_{1-b}, \perp\}$
if $(\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ and $g(RT_{F,x}) = g(m_0))$ then Output $\tilde{y} = 1$
else if $(\tau_{\theta_{F(x)}} = (\mathsf{accept}, S)$ and $g(RT_{F,x}) = g(m_1))$ then Output $\tilde{y} = 0$
else Output $\tilde{y} = \perp$
```

Alg. 9 $\{EK_{F,S'}\}$ or $\perp \leftarrow \mathsf{RPVC.Revoke}(\tau_{\theta_{F(x)}}, MK, PP)$

```
if $\tau_{\theta_{F(x)}} = (\mathsf{reject}, S)$ then
    $L_{\mathrm{Reg}}[S][1] = \{\epsilon\}$
    $L_{\mathrm{Rev}} = L_{\mathrm{Rev}} \cup S$
    Refresh $\mathbb{T}$
    $t \leftarrow \mathbb{T}$
    $UK^0_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
    $UK^1_{L_{\mathrm{Rev}},t} \leftarrow \mathsf{ABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
    for all $S \in \mathcal{U}_{\mathrm{ID}}$ do
        Parse $EK_{F,S}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t-1}, UK^1_{L_{\mathrm{Rev}},t-1})$
        Update and send $EK_{F,S} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$
else
    output $\perp$
```

Theorem 1. *Given a revocable KP-ABE scheme secure in the sense of indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*)* [1] *for a class of 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 1–9. Then $\mathcal{RPVC}$ is secure in the sense of selective semi-static Public Verifiability, selective semi-static Revocation, Vindictive Servers, Blind Verification and selective semi-static Vindictive Managers.*

Informally, the proof of Public Verifiability relies on the IND-CPA security of the underlying revocable KP-ABE scheme and the one-wayness of the function g. Revocation relies on the IND-sHRSS security of the revocable KP-ABE scheme. Finally, 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).

The proofs partially follow in the spirit of [14]. First we prove the following Lemma.

Lemma 1. *The $\mathcal{RPVC}$ construction defined by Algorithms 1–9 is secure in the sense of selective, semi-static Public Verifiability (Game 3) under the same assumptions as in Theorem 1.*

Proof. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the selective, semi-static Public Verifiability game (Game 3) when instantiated by Algorithms 1–9. We begin by defining the following three games:

- **Game 0.** This is the selective, semi-static Public Verifiability game as defined in Game 3.
- **Game 1.** This is the same as **Game 0** with the modification that in $\mathsf{ProbGen}$, we no longer return an encryption of m_0 and m_1. Instead, we choose another random message $m' \neq m_0, m_1$ and, if $F(x^\star) = 1$, we replace c_1 by the encryption of m', and otherwise we replace c_0. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.
- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

Partially in the fashion of Parno et al. [14], we aim to show that an adversary with non-negligible advantage distinguishing **Game 0** and **Game 1** 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**, both with parameters $(\mathcal{RPVC}, F, q_t, 1^\kappa)$. Suppose otherwise, that $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We then 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 10) with $\mathcal{A}_{ABE}$, who in turn acts as a challenger for $\mathcal{A}_{VC}$:

1. $\mathcal{A}_{VC}$ declares its choice of challenge input $x^\star$.
2. $\mathcal{A}_{ABE}$ transforms this into its own challenge input $\overline{x^\star} = x^\star \cup f$ where $f \in \mathcal{U}_{\mathcal{F}}$ is the attribute representing the challenge function F. It then sends this choice to $\mathcal{C}$ along with a challenge time period $t^\star = q_t$.
3. $\mathcal{C}$ runs the ABE.Setup algorithm to generate $MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}}$ and sends MPK^0_{ABE} to $\mathcal{A}_{ABE}$.
4. $\mathcal{A}_{ABE}$ initializes $Q_{\text{Rev}} = \epsilon$ and $t = 1$. It then simulates running RPVC.Setup by running Algorithm 1 as written, with the exception of Line 2 where it sets MPK^0_{ABE} to be that provided by $\mathcal{C}$, and implicitly sets MSK^0_{ABE} to be that held by the challenger.
5. $\mathcal{A}_{ABE}$ runs RPVC.FnInit as written and gives PK_F and PP to $\mathcal{A}_{VC}$, who returns a revocation list $\overline{R}$ comprising servers that must be revoked at challenge time. $\mathcal{A}_{ABE}$ forwards this list to $\mathcal{C}$.
6. $\mathcal{A}_{VC}$ is now provided with oracle access, to which $\mathcal{A}_{ABE}$ can respond as follows:
   - Queries to RPVC.FnInit and RPVC.Register are performed as in Algorithms 2 and 3.
   - Queries of the form RPVC.Certify(S, F', MK, PP): If $\mathcal{A}_{VC}$ has queried for the challenge function F and for an identity that is not to be revoked at the challenge time i.e. an identity $S \notin \overline{R}$, then $\mathcal{A}_{ABE}$ will return $\perp$ (since generating an evaluation key that will not be revoked for the challenge would be a trivial win). Similarly, $\perp$ is returned if the current time period is the challenge time q_t and there is a server (other than the queried S) that is not currently revoked but should be in accordance with $\mathcal{A}_{VC}$'s challenge revocation list $\overline{R}$ (i.e. $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S$).

     If $\perp$ has not already been returned, $\mathcal{A}_{ABE}$ removes S from the list, Q_{Rev}, of currently revoked servers (if present) and then simulates running RPVC.Certify as follows. Algorithm 4 is run as written with the exception that Lines 4 and 6 are simulated using oracle queries to $\mathcal{C}$. To simulate Line 4, $\mathcal{A}_{ABE}$ will make an oracle query to the ABE.KeyGen oracle. $\mathcal{C}$ will return the decryption key *unless* $\overline{x^\star} \in F'$ (i.e. $F'(\overline{x^\star}) = 1$) *and* $S \notin \overline{R}$. The KeyGen query will be of the form $\mathcal{O}^{\mathsf{KeyGen}}(S, F' \wedge f', MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. Observe that $\overline{x^\star} \notin F' \wedge f'$ unless $F' = F$. Hence, $\mathcal{C}$ will always return a valid key if $F' \neq F$. If $\mathcal{A}_{VC}$ has queried for the challenge function F for a server S then $\mathcal{C}$ will return $\perp$ if S is supposed to be revoked at the challenge time, or return the correct key otherwise. This is in-keeping with the expected behaviour of the Certify oracle by the first clause of Line 1 of Oracle Query 1.

     To simulate Line 6, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* the current time is the challenge time (which $\mathcal{A}_{ABE}$ chose to be q_t) *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. Now, by the second clause of Line 1 in Oracle Query 1, this precise occurrence will return $\perp$, so to the view of $\mathcal{A}_{VC}$ this is consistent behaviour.
   - Queries of the form RPVC.Revoke$(\tau_{\theta_{F(x)}}, MK, PP)$: $\mathcal{A}_{ABE}$ first increments its time counter, and outputs $\perp$ if the token is acceptance, as no server should be revoked (and in particular, Revoke would similarly return $\perp$ and no server should be added to the revocation list). Otherwise, S is added to Q_{Rev}. Then, if the current time is the challenge time, then $\mathcal{A}_{ABE}$ returns $\perp$ if Q_{Rev} does not contain all servers listed on the challenge revocation list $\overline{R}$. It then simulates running the RPVC.Revoke algorithm by running Algorithm 9 as written with the exception of Line 6. To simulate this

line, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* $t = q_t$ *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. However, by the above check, $\mathcal{A}_{VC}$ will expect to receive $\perp$ in this case. Otherwise, a valid update key is returned.

7. Eventually (and in particular after q_t Revoke queries), $\mathcal{A}_{VC}$ finishes this query phase. $\mathcal{A}_{ABE}$ then checks whether the queries made by $\mathcal{A}_{VC}$ are consistent with the challenge revocation list $\overline{R}$ chosen beforehand. If there is an entity listed within $\overline{R}$ that is not currently revoked (i.e. listed in Q_{Rev}) then $\mathcal{A}_{VC}$ loses the game.
8. $\mathcal{A}_{ABE}$ must now generate a challenge for either **Game 0** or **Game 1**.
   To do so, it samples three distinct messages m_0, m_1 and m_2 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 as its choice of challenge to $\mathcal{C}$, and receives back the encryption, $CT^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$), under attributes $\overline{x^\star}$ and time $t^\star = q_t$. $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$. It generates c_{1-b} itself by running $\mathsf{ABE.Encrypt}(m_2, \overline{x^\star} = (x^\star \cup f), q_t, MPK^1_{\text{ABE}})$.
   $\mathcal{A}_{ABE}$ chooses a random bit $s \xleftarrow{\$} \{0,1\}$. If $b = 0$, it sets $VK_{F,x^\star} = (g(m_s), g(m_2), L_{Reg})$. Otherwise, $VK = F, x^\star = (g(m_2), g(m_s), L_{Reg})$. Note that s is essentially $\mathcal{A}_{ABE}$'s guess of the bit $b^\star$ chosen by $\mathcal{C}$.
9. The resulting values are sent to $\mathcal{A}_{VC}$ who is provided with oracle access. These queries are handled in the same way as before, and eventually $\mathcal{A}_{VC}$ outputs its guess $\theta^\star$.
10. 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 $g(m')$ 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 ciphertext).

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

$$|\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa])| \geqslant \delta$$

where $\mathbf{Exp}^i_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa]$ denotes the output of running $\mathcal{A}_{VC}$ in Game i.

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

Hence,

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

Since δ is assumed 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 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 1**, $\mathcal{A}_{ABE}$ 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, since $\mathcal{A}_{VC}$ is assumed to be successful, it will output a forgery comprising the plaintext that was encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ 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 public verifiability game.

We conclude that if the ABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $\mathcal{RPVC}$ as defined by Algorithms 1–9 is secure in the sense of selective, semi-static Public Verifiability. □

Lemma 2. *The $\mathcal{RPVC}$ construction defined by Algorithms 1–9 is secure in the sense of selective, semi-static Revocation (Game 5) under the same assumptions as in Theorem 1.*

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the selective, semi-static Revocation game (Game 5) when instantiated by Algorithms 1–9. We define the following three games:

- **Game 0.** This is the selective, semi-static Revocation game as defined in Game 5.
- **Game 1.** This is identical to **Game 0** with the modification that ProbGen no longer returns an encryption of m_0 and m_1. Instead, another random message $m' \neq m_0, m_1$ is chosen. If $F(x^\star) = 1$, ciphertext c_1 is replaced by the encryption of m', and otherwise c_0 is set to be the encryption of m'.

- **Game 2.** This is as in **Game 1** but m' is set to be the challenge input w in the one-way function game.

We show that an adversary that can distinguish **Game 0** from **Game 1** with non-negligible advantage 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**, both instantiated with parameters $(\mathcal{RPVC}, F, q_t, 1^\kappa)$. For contradiction, let $\mathcal{A}_{VC}$ have non-negligible distinguishing 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. Let $\mathcal{C}$ be a challenger playing the IND-sHRSS game (Game 10) with $\mathcal{A}_{ABE}$, who in turn acts as a challenger for $\mathcal{A}_{VC}$:

1. $\mathcal{A}_{VC}$ selects a challenge input $x^\star$.
2. $\mathcal{A}_{ABE}$ chooses its own challenge input to be $\overline{x^\star} = x^\star \cup f$ where $f \in \mathcal{U}_\mathcal{F}$ represents the parameterised challenge function F. It also chooses the challenge time period to be $t^\star = q_t$, where $\mathcal{A}_{VC}$ is parameterised by making exactly q_t revoke queries.
3. $\mathcal{C}$ runs $\mathsf{ABE.Setup}$ to create $MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}}$ and sends MPK^0_{ABE} to $\mathcal{A}_{ABE}$.
4. $\mathcal{A}_{ABE}$ initializes the revocation list $Q_{\text{Rev}} = \epsilon$ and sets $t = 1$. It simulates $\mathsf{RPVC.Setup}$ by running Algorithm 1 as written, except for Line 2. Instead of initializing the first ABE system itself, it sets MPK^0_{ABE} to be that given by $\mathcal{C}$. As it does not possess MSK^0_{ABE}, it will make use of oracle queries to $\mathcal{C}$ wherever this is needed.
5. $\mathcal{A}_{ABE}$ runs $\mathsf{RPVC.FnInit}$ as written. It sends PK_F and PP to $\mathcal{A}_{VC}$, who declares a revocation list $\overline{R}$ listing all servers that should be revoked when the challenge is created in Line 9. $\mathcal{A}_{ABE}$ forwards $\overline{R}$ to $\mathcal{C}$.
6. $\mathcal{A}_{VC}$ may now perform oracle queries which $\mathcal{A}_{ABE}$ handles as follows:
   - Queries to $\mathsf{RPVC.FnInit}$ and $\mathsf{RPVC.Register}$ are run as written in Algorithms 2 and 3.
   - Queries of the form $\mathsf{RPVC.Certify}(S, F', MK, PP)$:
     $\mathcal{A}_{ABE}$ checks whether the queried function F' is the challenge function F *and* if the queried identity is not to be revoked at the challenge time i.e. $S \notin \overline{R}$. If both conditions hold, $\mathcal{A}_{ABE}$ will return $\perp$ (as issuing an evaluation key that will not be revoked at the time of the challenge would be a trivial win). Similarly, $\mathcal{A}_{ABE}$ returns $\perp$ if the current time period t is equal to the challenge time q_t and if there is a server (other than S) that is not currently revoked but should be in accordance with $\mathcal{A}_{VC}$'s challenge revocation list $\overline{R}$ (i.e. $\overline{R} \nsubseteq Q_{\text{Rev}} \setminus S$). If $\perp$ has not been returned, $\mathcal{A}_{ABE}$ checks whether the queried identity S is on the challenge revocation list $\overline{R}$, and returns $\perp$ if so. The execution of $\mathsf{RPVC.Certify}$ is simulated as follows.
     $\mathcal{A}_{ABE}$ runs Algorithm 4 as written with the exception that Lines 4 and 6 are simulated using oracle queries to $\mathcal{C}$. To simulate Line 4, $\mathcal{A}_{ABE}$ queries $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S, F' \wedge f', MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ returns the decryption key *unless* $\overline{x^\star} \in F' \wedge f'$ (i.e. $F' \wedge f'(\overline{x^\star}) = 1$) *and* $S \notin \overline{R}'$. Observe that $\overline{x^\star} \notin F' \wedge f'$ unless $F' = F$ (since there is a bijection between attributes $f \in \mathcal{U}_\mathcal{F}$ and functions $F \in \mathcal{F}$). 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 the query, 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 $\mathsf{Certify}$ oracle.
     To simulate Line 6, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* the current time 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 $\perp$. Therefore, $\mathcal{C}$ shall always return an update key which $\mathcal{A}_{ABE}$ can use in the running of $\mathsf{Certify}$.

- Queries of the form $\mathsf{RPVC.Revoke}(\tau_{\theta_{F(x)}}, MK, PP)$: $\mathcal{A}_{ABE}$ first increments t. If the token does not identify a server to revoke, it outputs $\perp$ (as would the Revoke algorithm). Otherwise, S is added to Q_{Rev}. 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 $\overline{R}$. $\mathcal{A}_{ABE}$ now simulates running the $\mathsf{RPVC.Revoke}$ algorithm by running Algorithm 9 as written with the exception of Line 6. To simulate this line, $\mathcal{A}_{ABE}$ makes a query of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key *unless* $t = q_t$ *and* the queried revocation list does not contain the challenge revocation list $\overline{R}$. However, if this was so, $\mathcal{A}_{ABE}$ would have returned $\perp$ above, and so a valid update key is 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.
8. $\mathcal{A}_{ABE}$ must now generate a challenge for either **Game 0** or **Game 1**.
   It chooses three distinct random messages m_0, m_1 and m_2, 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 $CT^\star \leftarrow \mathsf{ABE.Encrypt}(m_{b^\star}, \overline{x^\star} = (x^\star \cup f), q_t, MPK^0_{\text{ABE}})$. $\mathcal{A}_{ABE}$ sets $c_b = CT^\star$ and generates $c_{1-b} \leftarrow \mathsf{ABE.Encrypt}(m_2, \overline{x^\star} = (x^\star \cup f), q_t, MPK^1_{\text{ABE}})$. $\mathcal{A}_{ABE}$ selects another bit $s \xleftarrow{\$} \{0,1\}$ and, if $b = 0$, sets $VK_{F,x^\star} = (g(m_s), g(m_2), L_{Reg})$. Otherwise, $VK_{F,x^\star} = (g(m_2), g(m_s), L_{Reg})$. Note that s is $\mathcal{A}_{ABE}$'s guess for $b^\star$.
9. The resulting encoded input is 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 $b' = s$. Else, $\mathcal{A}_{ABE}$ guesses $b' = 1 - s$.

Observe 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 ciphertext).

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

$$|\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa])| \geqslant \delta$$

where $\mathbf{Exp}^i_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa]$ denotes the output of running $\mathcal{A}_{VC}$ in Game i.

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

Hence,

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

Since δ is 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. However, since the ABE scheme was assumed IND-sHRSS secure, such an $\mathcal{A}_{VC}$ cannot exist, and therefore it is not possible to distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** straightforwardly sets the new message m' to be the challenge w in the one-way function inversion game (Game 12). An adversary has no distinguishing advantage between these games as the new value is independent of anything else in the system except the verification key $g(w)$ (as was the case with m' in **Game 1**, 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 show that $\mathcal{A}_{ABE}$ can use $\mathcal{A}_{VC}$ (running against **Game 1** as a sub-routine to 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 $F(x^\star) = 1$, implicitly set $m_{1-b} = w$ and set the corresponding verification key component to be z. As usual, m_b is randomly chosen and the remainder of the verification key is computed as usual.
- if $F(x^\star) = 0$, 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, since $\mathcal{A}_{VC}$ is assumed to be successful, it will output a forgery comprising the plaintext that was encrypted under the unsatisfied function (F or $\overline{F}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ 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 revocation game. We conclude that if the ABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $\mathcal{RPVC}$ as defined by Algorithms 1–9 is secure in the sense of selective, semi-static Public Verifiability. □

Lemma 3. *The $\mathcal{RPVC}$ construction defined by Algorithms 1–9 is secure against Vindictive Servers (Game 6) under the same assumptions as in Theorem 1.*

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the Vindictive Servers game (Game 6) when instantiated by Algorithms 1–9. We show that an adversary $\mathcal{A}_{Sig}$ with non-negligble advantage δ in the EUF-CMA signature game (Game 11)can be constructed using $\mathcal{A}_{VC}$. $\mathcal{A}_{Sig}$ 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 a VC instance and play the Vindictive Servers game with $\mathcal{A}_{VC}$ by executing Algorithms 1–9 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 Register algorithm for this server will be implicitly set to be the signing key in the

EUF-CMA game and any Compute oracle queries for this identity will be forwarded to the challenger to compute. Then, assuming that $\mathcal{A}_{Sig}$ 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. $\mathcal{C}$ initializes $Q = \epsilon$ to be an empty list of messages queried to the Sig.Sign oracle and runs Sig.KeyGen(1^κ) to generate a challenge signing key $\overline{SK}$ and verification key $\overline{VK}$. $\mathcal{C}$ sends $\overline{VK}$ to $\mathcal{A}_{Sig}$.
2. $\mathcal{A}_{Sig}$ chooses a function F on which to instantiate $\mathcal{A}_{VC}$.
3. $\mathcal{A}_{Sig}$ initializes the revocation list $Q_{\text{Reg}} = \epsilon$. Furthermore, it chooses a server identity from $\mathcal{U}_{\mathsf{ID}} \setminus \mathcal{A}_{VC}$ which will be denoted by $\overline{S}$.
4. $\mathcal{A}_{Sig}$ runs RPVC.Setup(1^κ) and RPVC.FnInit(F, MK, PP), as specified in Algorithms 1 and 2 and passes PK_F and PP to the VC adversary $\mathcal{A}_{VC}$.
5. $\mathcal{A}_{VC}$ may now perform oracle queries to RPVC.FnInit RPVC.Register RPVC.Certify and RPVC.Revoke which $\mathcal{A}_{Sig}$ handles by running Algorithms 2, 3, 4 and 9 respectively.
6. Eventually, $\mathcal{A}_{VC}$ finishes querying and declares the challenge input $x^\star$.
7. $\mathcal{A}_{Sig}$ runs RPVC.ProbGen on the challenge $x^\star$ as specified in Algorithm 5.
8. $\mathcal{A}_{VC}$ is given the values of $PK_F, PP, \sigma_{F,x^\star}, VK_{F,x^\star}$ and $RK_{F,x^\star}$. It is also given oracle access to the following functions. $\mathcal{A}_{Sig}$ simulates these oracles and maintains a state of the generated parameters for each query.
   - FnInit$(\cdot, MK, PP)$: $\mathcal{A}_{Sig}$ runs this step as per Algorithm 2.
   - Register$(\cdot, MK, PP)$: If, for a queried server S, $S = \overline{S}$ then return $\perp$. Otherwise, $\mathcal{A}_{Sig}$ makes queries to $\mathcal{O}^{\mathsf{Register}}(S, MK, PP)$. If S has not been registered before and therefore does not appear on the registration list Q_{Reg} then the oracle returns a signing key SK_S for S and adds the pair (S, SK_S) to Q_{Reg}. Otherwise, the stored signing key is returned.
   - Certify$(\cdot, \cdot, MK, PP)$: $\mathcal{A}_{Sig}$ honestly runs Algorithm 4.
   - Revoke$(\cdot, MK, PP)$: $\mathcal{A}_{Sig}$ operates as in Algorithm 9.
   - Register2$(\cdot, MK, PP)$: $\mathcal{A}_{Sig}$ responds in the same way as for standard Register queries above, but *always* returns $\perp$ and not a signing key.

   $\mathcal{A}_{VC}$ eventually outputs a target server identity $\tilde{S}$.
9. If $\tilde{S} \neq \overline{S}$ then $\mathcal{A}_{Sig}$ outputs $\perp$ and stops. Else, $\mathcal{A}_{VC}$ continues with oracle access as in Step 8 as well as a Compute oracle. $\mathcal{A}_{VC}$ submits queries $\mathcal{O}^{\mathsf{Compute}}(\sigma_{F,x}, EK_{F,S}, SK_S, PP)$ for its choice of server S and $\sigma_{F,x}$(note that he may generate a valid $\sigma_{F,x}$ using the public delegation key). If $S \neq \overline{S}$ then $\mathcal{A}_{Sig}$ simply follows Algorithm 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_0 and d_1, and submits $m = (d_0, d_1, \overline{S})$ as a Sig.Sign oracle query to $\mathcal{C}$. $\mathcal{C}$ adds m to the list Q and returns $\gamma \leftarrow$ Sig.Sign$(m, \overline{SK})$, which $\mathcal{A}_{Sig}$ uses to return $\theta_{F(x)} = (d_0, d_1, \overline{S}, \gamma)$.
10. $\mathcal{A}_{VC}$ finally outputs $\theta^\star$ which appears to be an invalid result computed by $\tilde{S}$. Thus, Verify will output a reject token for $\tilde{S}$ and accept $\leftarrow$ Sig.Verify$((d_0, d_1, \tilde{S}), \gamma, \overline{VK})$. Thus, γ is a valid signature under key $\overline{SK}$.
11. $\mathcal{A}_{Sig}$ outputs $m^\star = (d_0, d_1, \tilde{S})$ and $\gamma^\star = \gamma$ to $\mathcal{C}$.

Note that due to Constraint 2 in Game 6, $\mathcal{A}_{VC}$ is not allowed to have made a query for $\mathcal{O}^{\mathsf{Compute}}(\sigma_{x^\star}, EK_{F,\tilde{S}}, SK_{\tilde{S}}, PP)$ and thus the forgery $(m^\star, \gamma^\star)$ output by $\mathcal{A}_{Sig}$ will satisfy the requirement in Game 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

$$\begin{aligned} Adv_{\mathcal{A}_{Sig}} &\geq \frac{1}{n} Adv_{\mathcal{A}_{VC}} \\ &\geq \frac{\delta}{n} \\ &\geq negl(\kappa) \end{aligned}$$

We conclude that if $\mathcal{A}_{VC}$ has a non-negligible advantage in the Vindictive Servers game then $\mathcal{A}_{Sig}$ has the same 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 4. *The $\mathcal{RPVC}$ construction defined by Algorithms 1–9 is secure in the sense of selective, semi-static Vindictive Managers (Game 8) under the same assumptions as in Theorem 1.*

Proof. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the selective, semi-static Vindictive Managers game (Game 8) when instantiated by Algorithms 1–9. We begin by defining the following three games:

- **Game 0.** This is the selective, semi-static Vindictive Managers game as defined in Game 8.
- **Game 1.** This is the same as **Game 0** with the modification that in ProbGen, we no longer return an encryption of m_0 and m_1. Instead, we choose another random message $m' \neq m_0, m_1$ and, if $F(x^\star) = 1$, we replace c_1 by the encryption of m', and otherwise we replace c_0. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random message unrelated to the other system parameters, and in particular to the verification keys.
- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random message m', we implicitly set m' to be the challenge input w in the one-way function game.

We aim to show that an adversary with non-negligible advantage distinguishing **Game 0** and **Game 1** 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**, both with parameters $(\mathcal{RPVC}, F, q_t, 1^\kappa)$. Suppose otherwise, that $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We then 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 10) with $\mathcal{A}_{ABE}$, who in turn acts as a challenger for $\mathcal{A}_{VC}$:

1. $\mathcal{A}_{VC}$ declares its choice of challenge input $x^\star$.
2. $\mathcal{A}_{ABE}$ transforms this into its own challenge input $\overline{x^\star} = x^\star \cup f$ where $f \in \mathcal{U}_\mathcal{F}$ is the attribute representing the challenge function F. It then sends this choice to $\mathcal{C}$ along with a challenge time period $t^\star = q_t$. 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, let us use the notation F^r as follows:
- If $r = 0$ then $F_r = F$ and $F_{1-r} = \overline{F}$
- If $r = 1$ then $F_r = \overline{F}$ and $F_{1-r} = F$.

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

3. $\mathcal{C}$ runs the ABE.Setup algorithm to generate $MPK_{\text{ABE}}, MSK_{\text{ABE}}$ and sends MPK_{ABE} to $\mathcal{A}_{ABE}$.
4. $\mathcal{A}_{ABE}$ initializes $Q_{\text{Rev}} = \epsilon$ and $t = 1$. It then simulates running RPVC.Setup by running Algorithm 1 as written, with the exception that one of the sets of ABE system parameters is assigned to be those generated by the challenger. Recall that $r = F(x)$. $\mathcal{A}_{ABE}$ 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 $\mathcal{C}$. It runs ABE.Setup to generate $MPK^{1-r}_{\text{ABE}}, MSK^{1-r}_{\text{ABE}}$ as usual.
5. $\mathcal{A}_{ABE}$ runs RPVC.FnInit as written and gives PK_F and PP to $\mathcal{A}_{VC}$, who returns a revocation list $\overline{R}$ comprising servers that must be revoked at the challenge time. $\mathcal{A}_{ABE}$ forwards this list to $\mathcal{C}$.
6. $\mathcal{A}_{ABE}$ now provides oracle access to $\mathcal{A}_{VC}$ as follows:
   - Queries to RPVC.FnInit and RPVC.Register are performed as in Algorithms 2 and 3.
   - Queries of the form RPVC.Certify(S, F', MK, PP): As specified in Oracle Query 1, $\mathcal{A}_{ABE}$ will return $\perp$ if the query is for the challenge function F and for a server S that is not necessarily to be revoked at the challenge time (i.e. $S \notin \overline{R}$) to avoid trivial wins. Similarly, $\perp$ is returned if the current time period t is the challenge time q_t and there is a server (other than the queried S) that is not currently revoked but should be in accordance with $\mathcal{A}_{VC}$'s challenge revocation list $\overline{R}$ (i.e. $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S$). If $\perp$ has not already been returned, $\mathcal{A}_{ABE}$ removes S from the list, Q_{Rev}, of currently revoked servers (if present) and then simulates running RPVC.Certify as follows.
     - SK^r_{ABE} is generated by issuing an oracle query to the ABE.KeyGen oracle on parameters $(S, F'_r \wedge f', MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. Now, $\mathcal{C}$ will return a valid decryption key unless $\overline{x^\star} \in F'_r \wedge f'$ *and* $S \notin \overline{R}$. Recall that $\overline{x^\star} = x^\star \cup f$ and hence if $F'_r \neq F_r$ then $f' \neq f$ and $\overline{x^\star} \notin F'_r \wedge f'$. On the other hand, if $F'_r = F_r$ then necessarily $S \in R$ else $\mathcal{A}_{ABE}$ would have returned $\perp$ previously. Hence, $\mathcal{C}$ will always be able to return a valid decryption key SK^r_{ABE}.
     - SK^{1-r}_{ABE} is generated by $\mathcal{A}_{ABE}$ running ABE.KeyGen using MSK^{1-r}_{ABE} for the function $F_{1-r} \wedge f$ as usual.
     - $UK^r_{L_{\text{Rev}},t}$ is generated by making a query to the ABE.KeyUpdate oracle for parameters $(Q_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ will return a valid update key unless the current time period t is the challenge time q_t and $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, if this was the case then $\mathcal{A}_{ABE}$ would already have returned $\perp$, and hence a valid update key is always returned.
     - $UK^{1-r}_{L_{\text{Rev}},t}$ is generated by $\mathcal{A}_{ABE}$ running ABE.KeyUpdate using MSK^{1-r}_{ABE} as usual.
     - The remainder of the Certify algorithm is run as written.
   - Queries of the form RPVC.Revoke($\tau_{\theta_{F(x)}}, MK, PP$): As stated in Oracle Query 2, $\mathcal{A}_{ABE}$ will first increment the current time t and will return $\perp$ if the query does not request a revocation (as would the Revoke algorithm). It will also return $\perp$ if the time period is now the challenge time q_t (i.e. this is the q_t^{th} Revoke query made by $\mathcal{A}_{VC}$) *and* $\overline{R} \not\subseteq Q_{\text{Rev}} \cup S$ (i.e. there exists a server other than S that is listed on $\overline{R}$ but is not currently revoked). Otherwise, S is added to the list Q_{Rev} and $\mathcal{A}_{ABE}$ simulates running Revoke as written with the following exception:
     - $UK^r_{L_{\text{Rev}},t}$ is generated by making a query to the ABE.KeyUpdate oracle for pa-

rameters $(Q_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ will return a valid update key unless the current time period t is the challenge time q_t and $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, if this was the case then $\mathcal{A}_{ABE}$ would already have returned $\perp$, and hence a valid update key is always returned.

7. Eventually (and in particular after q_t Revoke queries), $\mathcal{A}_{VC}$ finishes this query phase. $\mathcal{A}_{ABE}$ then checks whether the queries made by $\mathcal{A}_{VC}$ are consistent with the challenge revocation list $\overline{R}$ chosen beforehand. If there is an entity listed within $\overline{R}$ that is not currently revoked (i.e. listed in Q_{Rev}) then $\mathcal{A}_{VC}$ loses the game.
8. Otherwise, $\mathcal{A}_{ABE}$ must now generate a challenge encoded output and to do so it must simulate a computation server. It first picks a server S which is not an identity on the challenge revocation list $\overline{R}$ i.e. $S \notin \overline{R}$. It then runs Algorithm 3 as written to register S and then it must then simulate running the $\mathsf{Certify}$ 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 will run Algorithm 4 as written with the following exceptions:
   - SK^r_{ABE} is generated querying the $\mathsf{ABE.KeyGen}$ oracle on $(S, F_r \wedge f, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ will return a valid decryption key unless $\overline{x^\star} \in F_r \wedge f$ *and* $S \notin \overline{R}$. Recall that $\overline{x^\star} = x^\star \cup f$ so $\overline{x^\star} \in F_r \wedge f$ if and only if $x^\star \in F_r$. However, r was chosen such that $F_r(x^\star) = 0$ and hence $x^\star \notin F_r$ and so $\overline{x^\star} \notin F_r \wedge f$. Thus, $\mathcal{C}$ will return a valid decryption key.
   - SK^{1-r}_{ABE} is generated by $\mathcal{A}_{ABE}$ running $\mathsf{ABE.KeyGen}$ using MSK^{1-r}_{ABE} for the function $F_{1-r} \wedge f$ as usual.
   - $UK^r_{L_{\text{Rev}},t}$ is generated by making a query to the $\mathsf{ABE.KeyUpdate}$ oracle for parameters $(Q_{\text{Rev}}, t, MSK^r_{\text{ABE}}, MPK^r_{\text{ABE}})$. $\mathcal{C}$ returns a valid update key unless the current time period t is the challenge time q_t and $\overline{R} \not\subseteq Q_{\text{Rev}}$. Now, by virtue of the fact that $\mathcal{A}_{VC}$ has finished its query phase and no further revocations have occurred, the time is indeed q_t. However, $\mathcal{A}_{ABE}$ ended the game in Step 7 if $\overline{R} \not\subseteq Q_{\text{Rev}}$ and hence a this point, $\mathcal{C}$ will certainly return a valid update key.
   - $UK^{1-r}_{L_{\text{Rev}},t}$ is generated by $\mathcal{A}_{ABE}$ running $\mathsf{ABE.KeyUpdate}$ using MSK^{1-r}_{ABE} as usual.
9. $\mathcal{A}_{ABE}$ must now run $\mathsf{ProbGen}$ 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_2 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 as its choice of challenge to $\mathcal{C}$, and receives back the encryption, $CT^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$), under attributes $\overline{x^\star}$, time $t^\star = q_t$ and public parameters MPK^r_{ABE}. $\mathcal{A}_{ABE}$ sets $c_{r\oplus b} \leftarrow CT^\star$ and $c_{1-(r\oplus b)} \leftarrow \mathsf{ABE.Encrypt}(m_2, \overline{x^\star}, t^\star, MPK^{1-r}_{\text{ABE}})$.

   $\mathcal{A}_{ABE}$ then selects a random bit $s \xleftarrow{\$} \{0,1\}$. If $b = 0$, it sets $VK_{F,x^\star} = (g(m_s), g(m_2), L_{Reg})$. Otherwise, $VK = F, x^\star = (g(m_2), g(m_s), L_{Reg})$. Note that s is essentially $\mathcal{A}_{ABE}$'s guess of the bit $b^\star$ chosen by $\mathcal{C}$.
10. $\mathcal{A}_{ABE}$ now simulates the server S performing the computation to output $\theta_{F(x^\star)}$ by running Algorithm 6 as written, since valid keys have been generated for S in the preceding steps.
11. The resulting values $\sigma_{F,x^\star}$, $\theta_{F(x^\star)}$ and $VK_{F,x^\star}$ are sent to $\mathcal{A}_{VC}$ who is provided with oracle access. These queries are handled in the same way as before. and eventually $\mathcal{A}_{VC}$ outputs its guess $RT_{F,x}$ and $\tau_{\theta_{F(x)}}$.
12. 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 ciphertext).

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

$$|\Pr(\mathbf{Exp}^0_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa]) - \Pr(\mathbf{Exp}^1_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa])| \geqslant \delta$$

where $\mathbf{Exp}^i_{\mathcal{A}_{VC}}[\mathcal{RPVC}, F, q_t, 1^\kappa]$ denotes the output of running $\mathcal{A}_{VC}$ in Game i.

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

Hence,

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

Since δ is assumed 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 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 1**, $\mathcal{A}_{ABE}$ can invert the one-way function g – that is, given a challenge $z = g(w)$ $\mathcal{A}_{ABE}$ can recover w. Specifically, during $\mathsf{ProbGen}$, $\mathcal{A}_{ABE}$ chooses the message $m_{1-r\oplus b}$ to be w and the corresponding verification key component to be z. We randomly choose $m_{r\oplus b}$ and compute the remainder of the verification key as usual. Now, since $\mathcal{A}_{VC}$ is assumed to be 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, semi-static 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 1–9 is secure in the sense of selective, semi-static Vindictive Manager. □

Lemma 5. *The $\mathcal{RPVC}$ construction defined by Algorithms 1–9 is secure against Blind Verification (Game 9) under the same assumptions as in Theorem 1.*

Proof. The proof follows from a standard probability argument. We first argue that the only inputs that may reveal useful information to the adversary are $\theta_{F(x)}$ and $VK_{F,x}$. We then show that the adversarial view of these inputs does not provide an advantage at guessing the result.

The adversary is provided with the following inputs over the course of the game: PK_F, PP, $\theta_{F(x)}$, $VK_{F,x}$ and the outputs from oracle queries. Now, in our construction, $PK_F = PP$ and as this is just public parameters that is constant over all computations, this clearly does not reveal any information about $F(x)$. In particular, since the adversary does not see the encoded input (ciphertexts) from the challenge computation, the ABE public parameters in PP are not helpful (else the ABE scheme would not be IND-sHRSS secure), and neither is the time parameter which just comprises a counter or the list L_{Reg} that contains only function lists and signature verification keys.

The inputs $\theta_{F(x)}$ and $VK_{F,x}$ clearly do rely on the choice of x and hence of $F(x)$ and we will consider these shortly. We first consider the oracle access given to the following functions:

- $\mathsf{FnInit}(\cdot, MK, PP)$: Queries of this form simply output the public parameters PP which is already considered as an explicit input to the adversary.
- $\mathsf{Register}(\cdot, MK, PP)$: Queries to this oracle will result in the output of a signing key for a server S. However, this does not relate to the retrieval key and would be the same for any choice of x.
- $\mathsf{Certify}(\cdot, \cdot, \cdot, MK, PP)$: A call to this oracle will cause an additional label to be added to the list L_{Reg} in PP, which as a simple function identifier does not yield useful information about the challenge computation. It will also output two decryption keys and two update keys from the underlying ABE systems. Again, as the adversary only sees plaintexts and does not see the ciphertexts forming the challenge encoded input, such a key is not useful.
- $\mathsf{Revoke}(\cdot, \cdot, \cdot, MK, PP)$: As with $\mathsf{Certify}$ queries, this results in update keys and decryption keys for the ABE systems which do not help the adversary in this game.

Hence, providing oracle access to these functions does not help the adversary to distinguish which input was selected and hence the value of $F(x)$. Thus, the only inputs to the adversary that depend on the choice of challenge input are $\theta_{F(x)}$ and $VK_{F,x}$, and so we restrict our attention to these. As observed in (1) in Section 4, a well-formed response by the server will be either $(m_b, \perp)$ or $(\perp, m_{1-b})$ according $RK_{F,x}$. In detail this means, where $RK_{F,x} = b$:

- if $F(x) = 1$, then $\theta_{F(x)} = \begin{cases} (m_0, \perp), & \text{if } b = 0 \\ (\perp, m_0), & \text{if } b = 1 \end{cases}$
- if $F(x) = 0$, then $\theta_{F(x)} = \begin{cases} (\perp, m_1), & \text{if } b = 0 \\ (m_1, \perp), & \text{if } b = 1 \end{cases}$

Finally note also that $VK_{F,x} = (g(m_b), g(m_{1-b}))$ by definition (excluding L_{Reg} as this is covered by the consideration of PP above). We introduce the notation $\mathcal{V}$ to denote the adversary's view of $\theta_{F(x)}$ and $VK_{F,x}$ – that is, $\mathcal{V} = (d_b, d_{1-b}, g(m_b), g(m_{1-b}))$ would imply that $\theta_{F(x)} = (d_b, d_{1-b})$ and that $VK_{F,x} = (g(m_b), g(m_{1-b}))$.

We show that the probability that the adversary outputs a correct guess of $F(x)$ given a particular set of inputs $\mathcal{V}$ is the same as his chance of guessing without seeing $\mathcal{V}$. Thus, he cannot guess $F(x)$ with any advantage over what he knows about the distribution of F a priori. The argument proceeds as follows. Let $\mathcal{V}_1 = (m', \perp, g(m'), g(m_{1-b}))$ and let $\mathcal{V}_2 = (\perp, m'', g(m_b), g(m''))$. Note that these are the two possible views – $\mathcal{A}$ sees one message (either m_0 or m_1, both of which are uniformly drawn from the same distribution) and the one-way function applied to that message and the one way function applied to a different (unseen) message.

First observe that the value of $F(x)$ and the value of $b \xleftarrow{\$} \{0,1\}$ are independent events, $\Pr[b=1] = \frac{1}{2}$, and that $\Pr[F(x)=0] + \Pr[F(x)=1] = 1$ since F is a Boolean function and must result in either 1 or 0. Then,

$$\begin{aligned}
\Pr[\mathcal{V} = \mathcal{V}_1] &= \Pr[(F(x)=1 \wedge b=0) \vee (F(x)=0 \wedge b=1)] \\
&= \Pr[F(x)=1 \wedge b=0] + \Pr[F(x)=0 \wedge b=1] \\
&= \Pr[F(x)=1]\Pr[b=0] + \Pr[F(x)=0]\Pr[b=1] \text{ since } F(x) \text{ and } b \text{ are independent} \\
&= \frac{1}{2}\Pr[F(x)=1] + \frac{1}{2}\Pr[F(x)=0] \\
&= \frac{1}{2}(\Pr[F(x)=0] + \Pr[F(x)=1]) \\
&= \frac{1}{2} \qquad (2)
\end{aligned}$$

Now,

$$\begin{aligned}
\Pr[F(x)=0 | \mathcal{V} = \mathcal{V}_1] &= \frac{\Pr[F(x)=0 \wedge \mathcal{V} = \mathcal{V}_1]}{\Pr[\mathcal{V} = \mathcal{V}_1]} \\
&= \frac{\Pr[F(x)=0 \wedge b=1]}{\Pr[\mathcal{V} = \mathcal{V}_1]} \\
&= \frac{\Pr[F(x)=0]\Pr[b=1]}{\Pr[\mathcal{V} = \mathcal{V}_1]} \text{ since } F(x) \text{ and } b \text{ are independent} \\
&= \frac{\frac{1}{2}\Pr[F(x)=0]}{\frac{1}{2}} \text{ by (2)} \\
&= \Pr[F(x)=0]
\end{aligned}$$

Similarly,

$$\begin{aligned}
\Pr[\mathcal{V} = \mathcal{V}_2] &= \Pr[(F(x)=1 \wedge b=1) \vee (F(x)=0 \wedge b=0)] \\
&= \Pr[F(x)=1 \wedge b=1] + \Pr[F(x)=0 \wedge b=0] \\
&= \Pr[F(x)=1]\Pr[b=1] + \Pr[F(x)=0]\Pr[b=0] \text{ since } F(x) \text{ and } b \text{ are independent} \\
&= \frac{1}{2}\Pr[F(x)=1] + \frac{1}{2}\Pr[F(x)=0] \\
&= \frac{1}{2}(\Pr[F(x)=0] + \Pr[F(x)=1]) \\
&= \frac{1}{2} \qquad (3)
\end{aligned}$$

Now,

$$\begin{aligned}\Pr[F(x)=0|\mathcal{V}=\mathcal{V}_2] &= \frac{\Pr[F(x)=0 \wedge \mathcal{V}=\mathcal{V}_2]}{\Pr[\mathcal{V}=\mathcal{V}_2]}\\ &= \frac{\Pr[F(x)=0 \wedge b=0]}{\Pr[\mathcal{V}=\mathcal{V}_2]}\\ &= \frac{\Pr[F(x)=0]\Pr[b=0]}{\Pr[\mathcal{V}=\mathcal{V}_2]} \text{ since } F(x) \text{ and } b \text{ are independent}\\ &= \frac{\frac{1}{2}\Pr[F(x)=0]}{\frac{1}{2}} \text{ by (3)}\\ &= \Pr[F(x)=0]\end{aligned}$$

A symmetric argument holds for $F(x) = 1$, and hence we can conclude that knowledge of the adversarial inputs does not provide any advantage in determining $F(x)$ other than that which could be guessed without that knowledge (i.e. the inputs leak no information about $F(x)$). □

We conclude that combining the results of Lemmas 1–5 gives a proof of Theorem 1.

5 Conclusion

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 authorization within a organization. We believe our model more accurately reflects practical environments and the necessary interaction between entities for PVC. Each server may provide services for many different functions and for many different clients. The first model of Parno et al. [14] considered evaluations of a single function, while their second allowed for multiple functions but required a more exotic type of ABE scheme. This allowed a single ProbGen stage to encode input for any function, whilst in our model, we also allow multiple functions but use a simpler ABE scheme that also permits the revocation functionality. We require ProbGen to be run for each unique $F(x)$ to be outsourced which we believe to be reasonable. Additionally, in our model, any clients may submit multiple requests to any available servers, whereas prior work considered just one server.

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 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 future work 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.

References

[1] N. Attrapadung and H. Imai. Attribute-based encryption supporting direct/indirect revocation modes. In M. G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 278–300. Springer, 2009.

[2] N. Attrapadung and H. Imai. Dual-policy attribute based encryption. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 168–185, 2009.

[3] 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.

[4] A. Boldyreva, V. Goyal, and V. Kumar. Identity-based encryption with efficient revocation. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM Conference on Computer and Communications Security*, pages 417–426. ACM, 2008.

[5] H. Carter, C. Lever, and P. Traynor. Whitewash: outsourcing garbled circuit generation for mobile devices. In C. N. P. Jr., A. Hahn, K. R. B. Butler, and M. Sherr, editors, *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014*, pages 266–275. ACM, 2014.

[6] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In *TCC*, pages 499–518, 2013.

[7] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[8] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[9] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

[10] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.

[11] M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of ABE ciphertexts. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

[12] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[13] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 195–203. ACM, 2007.

[14] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In R. Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[15] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.

A Background

A.1 Cryptographic Primitives

In this section we introduce some cryptographic primitives that will be required in our construction of a VC scheme, namely Key-policy Attribute-based Encryption (KP-ABE), a revocable extension of KP-ABE, digital signatures and one-way functions. For each, we also give a brief insight into the intended purpose of these primitives in the VC construction to follow. These remarks will become clearer in the remainder of the paper. We begin by providing an overview of the notation used throughout the remainder of the paper.

A.1.1 Key-policy Attribute-based Encryption

Attribute-based encryption (ABE) is a public key, functional encryption primitive that allows the decryption of a ciphertext if and only if some policy formula formed over the data and decrypting entity is satisfied. More specifically, 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 $\mathbb{A} \in 2^{2^{\mathcal{U}}}$. Then decryption may succeed if and only if $A \in \mathbb{A}$. Variants of ABE include *Key-policy ABE* (KP-ABE) [10] where the policy is associated with the decryption key and a set of attributes is associated with each ciphertext; *Ciphertext-policy ABE* (CP-ABE) [3] where the policy is attached to a ciphertext and decryption keys are associated with sets of attributes; and *Dual-policy ABE* (DP-ABE) [2] in which 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. In this paper we will focus only on KP-ABE.

More concretely, in KP-ABE, each private key is associated with some family of attribute sets $\mathbb{A} = \{A_1, \ldots, A_m\}$, 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 private key includes the attribute set under which the message was encrypted: that is $A_i = A$ for some $i \in [m]$. The set of attribute sets defining a private key is usually called an *access structure* and, in most schemes, is *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. [13].

A KP-ABE scheme comprises the following algorithms:

- $(PP, MK) \leftarrow \mathsf{ABE.Setup}(1^\kappa)$: a randomized algorithm that takes a security parameter as input and outputs a master key MK and public parameters PP
- $CT \leftarrow \mathsf{ABE.Encrypt}(m, A, PP)$: a randomized algorithm that takes as input a message m, a set of attributes A and the public parameters PK, and outputs a ciphertext CT
- $SK_{\mathbb{A}} \leftarrow \mathsf{ABE.KeyGen}(\mathbb{A}, MK, PP)$: a randomized algorithm that takes as input an access structure $\mathbb{A}$, the master key MK and the public parameters PP, and outputs a private decryption key $SK_{\mathbb{A}}$
- m or $\perp \leftarrow \mathsf{ABE.Decrypt}(CT, SK, PP)$: takes as input a ciphertext CT of a message m associated with a set of attributes A, a decryption key SK with embedded access structure $\mathbb{A}$, and the public parameters. It outputs the message m if $A \in \mathbb{A}$, and $\perp$ otherwise.

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. The reader is referred to the cited prior literature for more details. ABE has previously been used primally as a means of cryptographically enforcing access control – for example, with KP-ABE objects are encrypted and a descriptive set of attributes attached, while entities are certified and issued a key containing a policy defining the types of objects they may access; decryption of an object succeeds if and only if the access control policy is satisfied by the requested object's attributes. In this work, we use KP-ABE in a different setting as a proof that a policy has been satisfied by a set of input values.

A.1.2 Revocable KP-ABE

To enable the revocation of malicious computation servers, we require a KP-ABE scheme that supports entity revocation (as opposed to attribute revocation). Revocable ABE schemes can support two different modes [1]:

- *Direct revocation* allows users to specify a revocation list at the point of encryption. This means that periodic rekeying is not required but the encryptors must have knowledge of, or be able to choose, the current revocation list.
- *Indirect revocation* requires ciphertexts to be associated with a time period (as an additional attribute) and for a key authority to issue key update material at each time period which enables non-revoked users to update their key to be functional during that time period. A revoked user will not be able to use the update material and thus their key will not succeed at decrypting ciphertexts associated with the current time period attribute. With indirect revocation, users 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 paper we use the indirect revocable KP-ABE scheme given by Attrapadung et al. [1], itself a more formal definition of that given by Boldyreva et al. [4]. This choice is primarily due to our assumption that the KDC should be the authority on trusted servers (since it is the KDC that certifies them in the first place) and that client devices should have the least amount of work to do and therefore shouldn't be required to maintain the revocation list, and to synchronise it with that held by other clients. However, due to the largely black-box use of this primitive, it should be easy to change to an alternate revocation scheme.

These schemes work by defining the universe of attributes to be $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}}$ where $\mathcal{U}_{\text{attr}}$ is the normal attribute universe for describing ciphertexts and forming access control policies, $\mathcal{U}_{\text{time}}$ comprises attributes for time periods, and $\mathcal{U}_{\text{ID}}$ contains an attribute per server identity. They then use the following algorithms:

- $(PP, MK) \leftarrow \mathsf{ABE.Setup}(1^{\kappa}, \mathcal{U})$: This randomised algorithm takes the security parameter and the universe of attributes as input and outputs public parameters PP and master secret key MK.
- $CT \leftarrow \mathsf{ABE.Encrypt}(t, A, m, PP)$: The randomised encryption algorithm takes the current time period $t \in \mathcal{U}_{\text{time}}$, an attribute set $A \subset \mathcal{U}_{\text{attr}}$, a message m and the public parameters, and outputs a ciphertext that is valid for time t.
- $SK_{\mathsf{id},\mathbb{A}} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A}, \mathsf{MK}, \mathsf{PP})$: The randomised key generation algorithm takes as input an identity $\mathsf{id} \in \mathcal{U}_{\text{ID}}$ for a user, an access structure encoding a policy, as well as the master secret key and public parameters. It outputs a decryption key for the user id.
- $UK_{R,t} \leftarrow \mathsf{ABE.KeyUpdate}(R, t, MK, PP)$: This randomised algorithm takes a revocation list $R \subseteq \mathcal{U}_{\text{ID}}$ containing the identities of revoked entities, the current time period, as well as the master secret key and public parameters. It outputs updated key material $UK_{R,t}$.

- m or $\perp \leftarrow \mathsf{ABE.Decrypt}(CT, SK_{\mathsf{id},\mathbb{A}}, PP, UK_{R,t})$: The decryption algorithm takes a ciphertext, a decryption key, the public parameters and an update key as input. It outputs the plaintext 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, and outputs $\perp$ otherwise.

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

Definition 8. *A revocable KP-ABE scheme is* correct *if for all* $m \in \mathcal{M}, \mathsf{id} \in \mathcal{U}_{\mathrm{id}}, \mathsf{R} \subseteq \mathcal{U}_{\mathrm{id}}, \mathbb{A} \in 2^{\mathcal{U}}_{attr}, \mathsf{A} \subset \mathcal{U}_{attr}, \mathsf{t} \in \mathcal{U}_{time}$*, if* $A \in \mathbb{A}$ *and* $\mathsf{id} \notin \mathsf{R}$*, then*

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

The schemes cited above use the Complete-subtree method to arrange users as the leaves of a binary tree such that the required key-update material can be reduced from the naive method of $\mathcal{O}(n-r)$ where n is the number of users and r is the number of revoked users, to $\mathcal{O}(r \log(\frac{n}{2}))$. This approach works as follows for a revocation list R. For a leaf node $l \in \mathcal{U}_{\mathrm{ID}}$, let $Path(l)$ be the set of nodes on the path between the root node and l inclusively. Then, for each $l \in R$, mark all nodes in $Path(l)$. Define $Cover(R)$ to be the set of all unmarked children of marked nodes, and generate update keys for these nodes. In this paper we use a permitted list rather than a revocation list and thus this algorithm will be adjusted accordingly, as discussed in Section **??**.

Note that the time parameter in the above algorithms could be a literal clock value where all entities have access to some synchronised 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, in our instantiation we will assume a time source $\mathbb{T}$ from which the current time period t (be that a literal time value or counter etc.) may be efficiently sampled as $t \leftarrow \mathbb{T}$.

The security property we consider in this paper for revocable KP-ABE is indistinguishability against selective-target with semi-static query attack (IND-sHRSS), presented in Game 10 [1]. This is a selective notion where the adversary must declare at the beginning of the game the set of attributes $(t^\star, x^\star)$ to be challenged upon. He is then given access to the public parameters and must choose a target revocation set $\overline{R}$ which is the set of entities that should be in a revoked state at time $t^\star$. The adversary is then given oracle access to the $\mathsf{ABE.KeyGen}$ and $\mathsf{ABE.KeyUpdate}$ functions as specified in Oracle Queries 7 and 8. 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 $x^\star$ satisfies $\mathbb{A}$ *and* the identity is not revoked at time $t^\star$. 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 $\overline{R}$. As in a standard IND-CPA notion, the adversary outputs two messages and the challenger chooses one of them at random to encrypt and passes the resulting ciphertext to the adversary. The adversary then guesses which message was encrypted. The advantage of the adversary is given in Definition 9.

Definition 9. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT) is defined as:*

$$Adv_{\mathcal{A}}^{\text{IND-sHRSS}}(\mathcal{ABE}, 1^{\kappa}) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}[\mathcal{ABE}, 1^{\kappa}] = 1] - \tfrac{1}{2}.$$

A 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* $\mathcal{A}$*,* $Adv_{\mathcal{A}}^{\text{IND-sHRSS}}(\mathcal{ABE}, 1^{\kappa}) \leq \mathrm{negl}(\kappa)$.

Game 10 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}[\mathcal{ABE}, 1^{\kappa}]$:

```
$(t^\star, x^\star) \leftarrow \mathcal{A}(1^\kappa)$;
$(PP, MK) \leftarrow \mathsf{Setup}(1^\kappa)$;
$\overline{R} \leftarrow \mathcal{A}(PP)$;
$(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,MK,PP),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,MK,PP))}(\overline{R}, PP)$;
$b \xleftarrow{\$} \{0,1\}$;
$CT^\star \leftarrow \mathsf{Encrypt}(t^\star, x^\star, m_b, PP)$;
$b^\star \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,MK,PP),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,MK,PP))}(\overline{R}, PP)$;
if $(b' = b)$ return 1;
else return 0;
```

Oracle Query 7 $\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{id}, \mathbb{A}, \mathsf{MK}, \mathsf{PP})$:

```
if $x^\star \in \mathbb{A}$ then
    if $\mathsf{id} \notin \overline{\mathsf{R}}$ then
        return $\perp$
$SK_{\mathsf{id},\mathbb{A}} \leftarrow \mathsf{KeyGen}(\mathsf{id}, \mathbb{A}, \mathsf{MK}, \mathsf{PP})$
return $SK_{\mathsf{id},\mathbb{A}}$
```

Oracle Query 8 $\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, MK, PP)$:

```
if $t = t^\star$ then
    if $\overline{R} \nsubseteq R$ then
        return $\perp$
$UK_{R,t} \leftarrow \mathsf{KeyUpdate}(R, t, MK, PP)$
return $UK_{R,t}$
```

A.1.3 Digital Signatures

Digital signatures provide a proof message integrity, as well as data origin authentication (since keys can be associated to particular users). We require a message to be signed using a private signing key owned by a particular entity, and using a public verification key we can verify that the signature was actually generated using the given signing key and that the contents of the message has not changed since the signature was computed. We will use this primitive to provide a means of validating that the result of a computation was computed by the claimed server and that it has not been maliciously altered.

A digital signature scheme Sig comprises three polynomial-time algorithms Sig.KeyGen, Sig.Sign and Sig.Verify defined as follows [12]:

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

Definition 10. *A signature scheme is* correct *if for all (SK, VK) pairs generated by* $\mathsf{Sig.KeyGen}(1^\kappa)$ *and every message m in the message space,* $\mathsf{Sig.Verify}(m, \mathsf{Sig.Sign}(m, SK), VK) = 1$.

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

Game 11 $\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}[\mathsf{Sig}, 1^\kappa]$:

```
Initialise $Q = \epsilon$ to be an empty list
$(SK, VK) \leftarrow \mathsf{Sig.KeyGen}(1^\kappa)$
$(m^\star, \gamma^\star) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sig.Sign}}(\cdot, SK)}(VK)$
if $(\mathsf{accept} \leftarrow \mathsf{Sig.Verify}(m^\star, \gamma^\star, VK)$ and $m^\star \notin Q)$ return 1;
else return 0;
```

Oracle Query 9 $\mathcal{O}^{\mathsf{Sig.Sign}}(m, SK)$:

```
$Q = Q \cup m$
return $\mathsf{Sig.Sign}(m, SK)$
```

Game 12 $\mathbf{Exp}_{\mathcal{A}}^{Invert}[g, 1^\kappa]$:

```
$w \leftarrow \{0,1\}^\kappa$
$z = g(w)$
$w' \leftarrow \mathcal{A}(1^\kappa, z)$
if $(g(w') = z)$ return 1;
else return 0;
```

messages that it saw a signature for. More formally, this is defined in Game 11 where $\mathcal{A}$ has access to a Sig.Sign oracle which is handled by the algorithm given in Oracle Query 9.

Definition 11. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT) is defined as:*

$$Adv_{\mathcal{A}}^{\text{EUF-CMA}}(\mathsf{Sig}, 1^\kappa) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}[\mathsf{Sig}, 1^\kappa] = 1].$$

A digital signature scheme Sig *is* existentially unforgeable under an adaptive chosen message attack (EUF-CMA) *if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{\text{EUF-CMA}}(\mathsf{Sig}, 1^\kappa) \leq \text{negl}(\kappa)$.*

A.1.4 One-way Functions

A *one-way function* g is characterized by having the properties of being easy to compute, but hard to invert. The first condition is given by the requirement that g is computable in polynomial time. The second condition is formalized by requiring that it is infeasible for any probabilistic polynomial-time algorithm to invert g (that is, to find a pre-image of a given value y) except with negligible probability. This requirement will be captured in the *inverting experiment* (Game 12) where we consider the experiment for any algorithm $\mathcal{A}$, any value κ for the security parameter, and the function $g\colon \{0,1\}^\star \to \{0,1\}^\star$. Note that it suffices for $\mathcal{A}$ to find any value of x' for which $g(x') = y = g(x)$ in the experiment.

Here we give a definition what it means for a function g to be one-way [12].

Definition 12. *A function $g\colon \{0,1\}^\star \to \{0,1\}^\star$ is* one-way *if the following two conditions hold.*

1. (Easy to compute.) *There exists a polynomial-time algorithm M_g computing g; i.e. $M_g(w) = g(w)$ for all w.*
2. (Hard to invert.) *For every PPT algorithm $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{Invert}[g, 1^\kappa] = 1] \leq \text{negl}(\kappa).$$

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 1: PVC using KP-ABE

A.2 PVC using KP-ABE

Parno et al. [14] provide a instantiation of PVC using KP-ABE[6] for the case when F is a Boolean function [14]. Define a universe $\mathcal{U}$ of n attributes and associate $V \subseteq \mathcal{U}$ with a binary n-tuple where the ith place is 1 if and only if the ith 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 set associated with x. An alternative way to view this is to let $\mathcal{U} = \{A_1, A_2, \ldots, A_n\}$. Then, a bit string $\bar{v}$ of length n is the characteristic tuple of the set $V \subseteq \mathcal{U}$ if $V = \{A_i : \overline{v_i} = 1\}$. A function $F : \{0,1\}^n \to \{0,1\}$ is monotonic if $x \leqslant y$ implies $F(x) \leqslant F(y)$, where $x = (x_1, \ldots, x_n)$ is less than or equal to $y = (y_1, \ldots, y_n)$ if and only if $x_i \leqslant y_i$ for all i. For a monotonic $F : \{0,1\}^n \to \{0,1\}$, the set $\mathbb{A}_F = \{x \in \{0,1\}^n : F(x) = 1\}$ defines a monotonic access structure. The mapping between PVC and KP-ABE parameters is shown in Table 1.Informally, for a Boolean function F, the client generates a private key $SK_{\mathbb{A}_F}$ using the KeyGen algorithm. Given an input x, a client encrypts a random message m "with" A_x using the Encrypt algorithm and publishes $VK_{F,x} = g(m)$ where g is a suitable one-way function (e.g. a pre-image resistant hash function). The server decrypts the message using the Decrypt algorithm, which will either return m (when $F(x) = 1$) or $\perp$. The server returns m to the client. Any client can test whether the value returned by the server is equal to $g(m)$. Note, however, that a "rational" malicious server will always return $\perp$, since returning any other value will (with high probability) result in the verification algorithm returning a reject decision. Thus, it is necessary to have the server compute both F and its "complement" (and for both outputs to be verified). We revisit this point in Sect. 4. The interested reader may also consult the original paper for further details [14]. Note that, to compute the private key $SK_{\mathbb{A}_F}$, it is necessary to identify all minimal elements x of $\{0,1\}^n$ such that $F(x) = 1$. There may be exponentially many such x. Thus, the initial phase is indeed computationally expensive for the client. Note also that the client may generate different private keys to enable the evaluation of different functions.

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