Revocation in Publicly Verifiable Outsourced Computation

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

Information Security Group, Royal Holloway, University of London

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 an increasing trend towards cloud computing and enormous volumes of data ("big data") which mean that computations may require considerable computing resources. In short, there is, increasingly, a 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 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

[*]James.Alderman.2011@live.rhul.ac.uk

[†]Carlos.Cid@rhul.ac.uk

[‡]Jason.Crampton@rhul.ac.uk

[§]Christian.Janson.2012@live.rhul.ac.uk

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 the time or resources to perform the computation. Thus, the client wishes to have some assurance that the result y returned by the server is, in fact, $F(x)$.

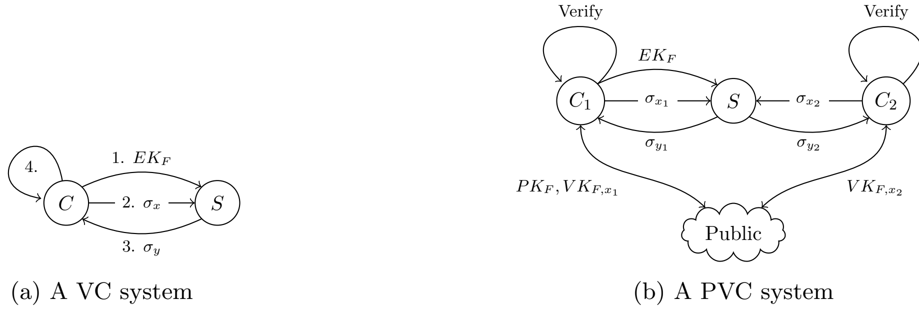
This problem, known as *Verifiable Outsourced Computation* (VC), has attracted a lot of attention in the community recently. In practical scenarios, it may well be desirable that cheating servers are prevented from performing future computations, as they are deemed completely untrustworthy. Thus, future 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 also that the functions of interest to each of the clients will substantially overlap, as in the "bring your own device" scenario discussed above. It is also conceivable that the number of computation 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 the computation of the same (potentially large) set of functions to a set of servers that are not fully trusted. 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 Centre (KDC) to perform the computationally intensive parts of VC and manage keys for all clients, and also 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 computations of F 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 corresponds to real environments than previously considered in the literature. 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. The paper will finish in section 5 with a conclusion. The paper is self-contained: more details on the background can be found in the Appendix.

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



(a) A VC system

(b) A PVC system

Figure 1: The operation of verifiable computation schemes

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 σ_x to S (input preparation)
3. S computes $y = F(x)$ using EK_F and σ_x and returns an encoding of the output σ_y to C (output computation)
4. C checks whether σ_y 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 are now required to 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.

2.2 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. 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 $\mathcal{F}$. 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 $\{1, \ldots, n\}$.

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. Different servers may compute the same function F and servers are "certified" to compute F by the KDC. 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.

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, we may wish, in real-world system architectures, to outsource the setup phase to a trusted third party. In this setting, the third party would operate in a way rather similar 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, in other real-world system architectures, to limit the functions that some clients can outsource. In other words, we wish to enforce some kind of access control policy. In this setting, an internal trusted entity will operate both as a facilitator of outsourced computation and as the policy enforcement point. (We hope to examine the integration of RPVC and access control in future work.)

Notice that the KDC could in fact be a distinguished client device (which has the resources to perform more expensive setup operations), but in this work we consider it to be a separate entity to illustrate separation of duty between the clients that request computations, and the KDC that is an authority on the system and users. Additionally, the KDC could be authoritative over many sets of clients (e.g. at an organisational level as opposed to a work group level), and we minimise its workload to key generation and revocation only. It may be tempting to suggest that the KDC, as a trusted entity, performs all computations itself. However we believe that this is not a practical solution in many real world scenarios, e.g. the KDC could be an authority within the organisation responsible for user authorisation that wishes to enable workers to securely use cloud-based software-as-a-service. As an entity within organisation 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 initialises the system and generates keys to enable verifiable computation. Keys to delegate computations are published for the clients, whilst keys to evaluate specific functions are given to individual servers. Clients submit computation requests, for a given input, to a particular server and publish some verification information. 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 of this function.

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.

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 Output	–	–	–	$C_1, C_2, \ldots$

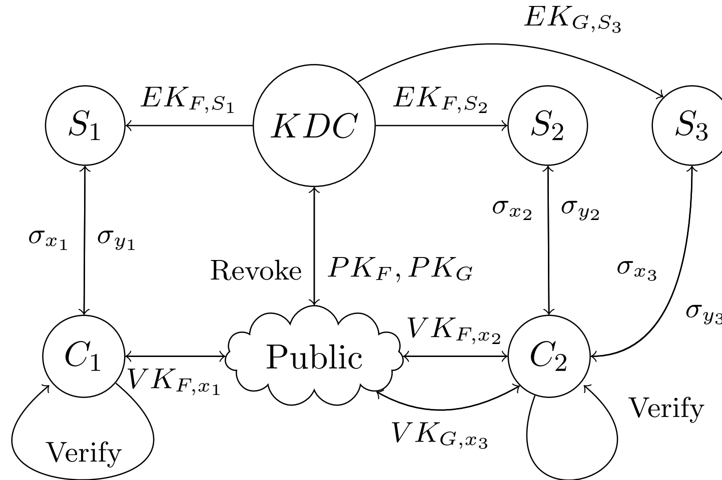


Figure 2: The operation of RPVC in the Standard Model

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 utilise 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 3. The manager and computational servers are shown within a dashed region to illustrate the boundaries of internal and external entities – that is, the entities not within the dashed region could all be within an organisation that wishes to utilise the external resources provided by the manager to outsource computational work. Notice that the manager performs a blind verification operation (denoted BVerify) but only entities within the organisation may run the output retrieval algorithm (de-

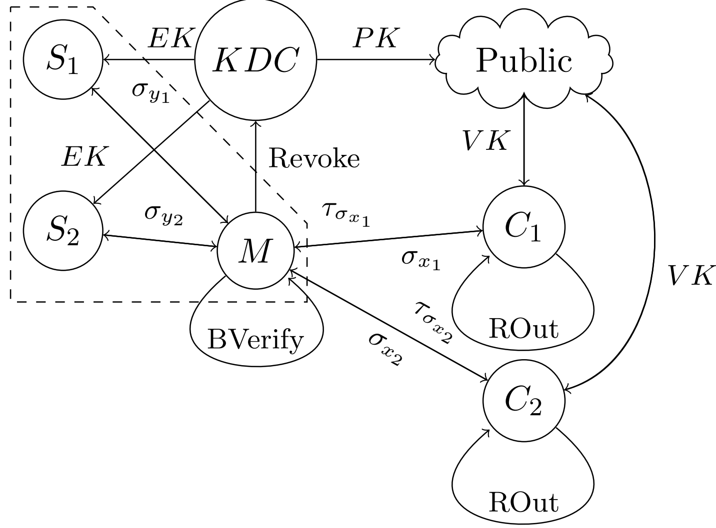


Figure 3: The operation of RPVC in the Manager model

noted here as Rout) 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:

- $\mathsf{Setup}(1^\lambda) \rightarrow (\mathrm{PP}, \mathrm{MK})$: Run by the KDC to establish public parameters PP and a master secret key MK.
- $\mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, F) \rightarrow (PK_F, L_F)$: Run by the KDC to generate a public delegation key, PK_F, for a function F as well as a list L_F of available servers for evaluating F, which is initially empty.
- $\mathsf{Register}(\mathrm{PP}, \mathrm{MK}, S) \rightarrow SK_S$: Run by the KDC to generate a personalised key SK_S for a computation server S.
- $\mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, S) \rightarrow (EK_{F,S}, L_F)$: 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. S is added to the list, L_F, of available servers for evaluating F.
- $\mathsf{ProbGen}(x, PK_F) \rightarrow (\sigma_x, VK_{F,x}, RK_{F,x})$: The $\mathsf{ProbGen}$ algorithm is run by a client to delegate the computation of $F(x)$ to a server. The output value $RK_{F,x}$ is used to enable output retrieval after the blind verification step.
- $\mathsf{Compute}(\sigma_x, EK_{F,S}, SK_S) \rightarrow \sigma_y$: Run by a server S in possession of an evaluation key $EK_{F,S}$, SK_S and an encoded input σ_x of x to evaluate $F(x)$ and output an encoding, σ_y, of the result, which includes an identifier of S.
- $\mathsf{Verify}(\mathrm{PP}, \sigma_y, VK_{F,x}, RK_{F,x}, L_F) \rightarrow (\tilde{y}, \tau_{\sigma_y})$: Verification consists of two steps.
  - $\mathsf{BlindVerify}(\mathrm{PP}, \sigma_y, VK_{F,x}, L_F) \rightarrow (\mu, \tau_{\sigma_y})$: Run by any verifying party party (standard model), or run by the manager (manager model), in possession of $VK_{F,x}$ and encoded output, σ_y. This outputs a token $\tau_{\sigma_y} = (\mathsf{accept}, S)$ if the output is valid, or $\tau_{\sigma_y} = (\mathsf{reject}, S)$ if S misbehaved. It also outputs μ which is an encoding of the actual output value.
  - $\mathsf{RetrieveOutput}(\mu, \tau_{\sigma_y}, VK_{F,x}, RK_{F,x}) \rightarrow \tilde{y}$: Run by a verifier in possession of $RK_{F,x}$

to retrieve the actual result $\tilde{y}$ which is either $F(x)$ or $\perp$[1].

- $\mathsf{Revoke}(\mathrm{MK}, \tau_{\sigma_y}, F, L_F) \rightarrow (\{EK_{F,S'}\}, L_F)$ or $\perp$: Run by the KDC if a misbehaving server is reported i.e. that Verify returned $\tau_{\sigma_y} = (\mathsf{reject}, S)$ (if $\tau_{\sigma_y} = (\mathsf{accept}, S)$ then this algorithm should output $\perp$). It revokes the evaluation key $EK_{F,S}$ of the server S thereby preventing any further evaluations of F. This is achieved by removing S from L_F (the list of servers for F) and issuing updated evaluation keys $EK_{F,S'}$ to all servers $S' \neq S$.

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

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

3.5 Security Models

We now introduce several security models capturing different requirements of a RPVC scheme. We will formalise 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 probabilistic polynomial time algorithm $\mathcal{A}$ run by a challengerduring the game with input parameters chosen to represent the knowledge of a real attacker as well the security parameter λ. 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 $\mathcal{A}^{\mathcal{O}}$ is used to denote the adversary $\mathcal{A}$ being provided with oracle access to the following functions: $\mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, \cdot)$, $\mathsf{Register}(\mathrm{PP}, \mathrm{MK}, \cdot)$, $\mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, \cdot, \cdot, \cdot)$ and $\mathsf{Revoke}(\mathrm{MK}, \cdot, \cdot, \cdot)$[2]. 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.

Due to the use of a revocable KP-ABE scheme, we require that inputs have a notion of time attached to them which is changed every time a server is revoked. Alternatively, the time period could be regularly updated but the Revoke algorithm must be run at each interval even if the revocation list has not changed. As such we identify the input x in the formal definitions with the input (t, x) in these games where t is drawn from some time source τ (e.g. a counter maintained in the public parameters or a networked clock).

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

[2]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).

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

1: $\{t_i^\star, x_i^\star\}_{i\in[n]} \leftarrow \mathcal{A}(1^\lambda)$;
2: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
3: $(PK_F, L_F) \leftarrow \mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, F)$;
4: $SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathrm{PP}, \mathrm{MK}, \mathcal{A})$;
5: $L_F \leftarrow \mathcal{A}(PK_F, \mathrm{PP}, L_F, SK_{\mathcal{A}})$;
6: $(EK_{F,\mathcal{A}}, L_F) \leftarrow \mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, \mathcal{A})$;
7: **for** $i = 1$ **to** n **do**
8: $\quad (\sigma_{x_i^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}) \leftarrow \mathsf{ProbGen}(\{t_i^\star, x_i^\star\}, PK_F)$;
9: $\sigma_{y^\star} \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, \mathrm{PP}, L_F, \{\sigma_{x_i^\star}, VK_{F,x_i^\star}\}, EK_{F,\mathcal{A}}, SK_{\mathcal{A}})$;
10: **if** $\exists i \in [n]$ s.t. $(((\tilde{y}, \tau_{\sigma_{y^\star}}) \leftarrow \mathsf{Verify}(\mathrm{PP}, \sigma_{y^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}, L_F))$ and $((\tilde{y}, \tau_{\sigma_{y^\star}}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ and $(\tilde{y} \neq F(x_i^\star)))$ **then**
11: $\quad$ **return** 1
12: **else**
13: $\quad$ **return** 0

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.
- We must ensure that neither an uncertified nor a de-certified server can convince a client to accept an output.
- We must ensure that a malicious server S cannot convince a client to believe an honest server has produced an incorrect output.
- We must ensure that, in the manager model, a malicious manager cannot convince a client of an incorrect result.
- We must ensure, in the manager model, that the manager performing the BlindVerify algorithm learns nothing of the actual output value other than its correctness.

3.5.1 Public Verifiability

In Game 1 we wish 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 a σ_y for $y \neq F(x)$). The game begins (line 1) with the adversary selecting a (polynomially sized) set of n input values that he would like to see the problem encoding of. The challenger runs Setup, FnInit and Register to initialise the system and create a public delegation key for a function F given as a parameter to the game (lines 2 to 4). The adversary is given the delegation key, his private key and the public parameters (i.e. all values known to a server in the real setting), and must output a list of servers that should be certified to compute F (line 6)[3].

The challenger then runs ProbGen for each challenge input and gives the encoded inputs to the adversary. The adversary also has oracle access to the FnInit, Register, Certify and Revoke algorithms to model the corruption of other servers (line 10), and aims to create an encoded output that is accepted by the challenger yet is not valid for any challenge input.

[3] This corresponds to the revocation list in the model of [1] except that we consider a certification list of servers that should receive the update keys rather than a revocation list of servers that should not receive these keys. The requirement to output this list here is due to the selective IND-sHRSS game that we base the construction upon. Since this is used in a black-box manner however, a stronger primitive may allow this game to be improved accordingly.

Game 2 $\mathbf{Exp}_{\mathcal{A}}^{Revocation}\left[\mathcal{RPVC}, F, 1^{\lambda}\right]$:

1: $\{t_i^\star, x_i^\star\}_{i\in[n]} \leftarrow \mathcal{A}(1^\lambda)$;
2: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
3: $(PK_F, L_F) \leftarrow \mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, F)$;
4: $SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathrm{PP}, \mathrm{MK}, \mathcal{A})$;
5: $L_F \leftarrow \mathcal{A}(PK_F, \mathrm{PP}, L_F, SK_{\mathcal{A}})$;
6: $(EK_{F,\mathcal{A}}, L_F) \leftarrow \mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, \mathcal{A})$;
7: $\tau^\star = (\mathsf{reject}, \mathcal{A}) \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, \mathrm{PP}, L_F, SK_{\mathcal{A}})$;
8: $(\{EK_{F,S}\}, L_F) \leftarrow \mathsf{Revoke}(\mathrm{MK}, \tau^\star, F, L_F)$;
9: **for** $i = 1$ **to** n **do**
10: $\quad(\sigma_{x_i^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}) \leftarrow \mathsf{ProbGen}(\{t_i^\star, x_i^\star\}, PK_F)$;
11: $\sigma_{y^\star} \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, \mathrm{PP}, L_F, \{\sigma_{x_i^\star}, VK_{F,x_i^\star}\}, \{EK_{F,S}\}, SK_{\mathcal{A}})$;
12: **if** $\exists i \in [n]$ s.t. $(((\tilde{y}, \tau_{\sigma_{y^\star}}) \leftarrow \mathsf{Verify}(\mathrm{PP}, \sigma_{y^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}, L_F))$ and $((\tilde{y}, \tau_{\sigma_{y^\star}}) \neq (\perp, (\mathsf{reject}, \mathcal{A}))))$ **then**
13: $\quad$**return** 1
14: **else**
15: $\quad$**return** 0

Definition 3. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT), making a polynomial number of queries q in the Public Verifiability Experiment is defined as:*

$$Adv_{\mathcal{A}}^{PubVerif}(\mathcal{RPVC}, F, 1^\lambda, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{PubVerif}[\mathcal{RPVC}, F, 1^\lambda] = 1].$$

A RPVC is secure in the sense of public verifiability *for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{PubVerif}(\mathcal{RPVC}, F, 1^\lambda, q) \leq \mathrm{negl}(\lambda)$.*

Note that this game is a generalisation 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 (represented by choosing the set of target inputs) 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).

3.5.2 Revocation

In Game 2 we require that if a server is detected as misbehaving (i.e. a result for $F(x)$ causes the Verify algorithm to output $(\perp, (\mathsf{reject}, S)))$ then any subsequent evaluations of F by S should be rejected. The motivation here is that even though we have outsourced the costly computation and pre-processing stages to the server and KDC respectively, there is still a cost involved in delegating and verifying a computation. If a server is known not to be trustworthy then we remove any incentive for it to attempt to provide an outsourcing service for this function (since it knows the result will not be accepted). In addition, we may like to punish and further disincentivize malicious servers by removing their ability to perform work (and earn rewards) for a period of time. Finally, from a privacy perspective, we may not wish to supply input data to a server that is known not to be trustworthy. In this game the adversary chooses the target input values as before (line 1) but now the evaluation key $EK_{F,\mathcal{A}}$ that it had access to when selecting x is revoked (line 8) before the computation is run. The remainder of the game proceeds as in the Public Verifiability game but we require that the adversary is no longer able to provide *any* result that verifies correctly (even $F(x)$).

Game 3 $\mathbf{Exp}_{\mathcal{A}}^{VindictiveS}\left[\mathcal{RPVC}, F, 1^{\lambda}\right]$:

1: $\{t_i^\star, x_i^\star\}_{i\in[n]} \leftarrow \mathcal{A}(1^\lambda)$;
2: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
3: $(PK_F, L_F) \leftarrow \mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, F)$;
4: $SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathrm{PP}, \mathrm{MK}, \mathcal{A})$;
5: $L_F \leftarrow \mathcal{A}(PK_F, \mathrm{PP}, L_F, SK_{\mathcal{A}})$;
6: $(EK_{F,\mathcal{A}}, L_F) \leftarrow \mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, \mathcal{A})$;
7: **for** $i = 1$ **to** n **do**
8: $\quad(\sigma_{x_i^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}) \leftarrow \mathsf{ProbGen}(\{t_i^\star, x_i^\star\}, PK_F)$;
9: $\tilde{S} \leftarrow \mathcal{A}^{\mathcal{O}}(PK_F, \mathrm{PP}, L_F, \{(\sigma_{x_i^\star}, VK_{F,x_i^\star})\}, EK_{F,\mathcal{A}}, SK_{\mathcal{A}})$ subject to Condition 1;
10: $\sigma_{y^\star} \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{Compute}}(PK_F, \mathrm{PP}, L_F, \{(\sigma_{x_i^\star}, VK_{F,x_i^\star})\}, EK_{F,\mathcal{A}}, SK_{\mathcal{A}})$ subject to Condition 2;
11: **if** $\exists i \in [n]$ s.t. $((\tilde{y}, \tau_{\sigma_{y^\star}}) \leftarrow \mathsf{Verify}(\mathrm{PP}, \sigma_{y^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}, L_F))$ and $((\tilde{y}, \tau_{\sigma_{y^\star}}) = (\perp, (\mathsf{reject}, \tilde{S})))$ and $(\perp \not\leftarrow \mathsf{Revoke}(\mathrm{MK}, \tau_{\sigma_{y^\star}}, F, L_F)))$ **then**
12: $\quad$**return** 1
13: **else**
14: $\quad$**return** 0

Definition 4. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q in the Revocation Experiment is defined as:*

$$Adv_{\mathcal{A}}^{Revocation}(\mathcal{RPVC}, F, 1^\lambda, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{Revocation}[\mathcal{RPVC}, F, 1^\lambda] = 1].$$

A RPVC is secure against revoked servers *for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{Revocation}(\mathcal{RPVC}, F, 1^\lambda, q) \leq \mathrm{negl}(\lambda)$.*

3.5.3 Vindictive Server

The motivation for this notion of security is 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 or care about the individual server identities. Now, since an invalid result can lead to revocation, this leads to a new threat model (particularly in systems where servers gain rewards per computation performed) in which a malicious server may return incorrect results but attribute them to an alternate server ID such that the (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 3 the adversary must (on lines 9 and 10) output an invalid result $\sigma_{y^\star}$ and the ID of a server $\tilde{S}$ that it aims to cause to be revoked. It is provided with the standard oracle access on line 9 and on line 10 additionally with oracle access to $\mathsf{Compute}$ such that he can see outputs returned by honest servers (i.e. modelling the adversary submitting computation requests to the system himself), subject to the following constraints:

1. No query was made of the form $\mathcal{O}^{\mathsf{Register}}(\mathrm{PP}, \mathrm{MK}, \tilde{S})$;
2. As above but also no query was made of the form $\mathcal{O}^{\mathsf{Compute}}(\sigma_{x_i^\star}, EK_{F,\tilde{S}}, SK_{\tilde{S}})$;

The adversary wins if the KDC believes that $\tilde{S}$ 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:*

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

A RPVC is secure against vindictive servers *for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{VindictiveS}(\mathcal{RPVC}, F, 1^\lambda, q) \leq \mathrm{negl}(\lambda)$.*

3.5.4 Vindictive Manager

In Game 4 we capture security against vindictive managers attempting to provide the client with an incorrect answer. This is a natural extension of the Public Verifiability notion (Game 1) to the manager model. The adversary, on line 5, chooses a challenge input value x, and the server computes an encoded output of $F(x)$. The adversary is then provided the encoded output and verification key and must output an encoded output μ and an acceptance token. The challenger runs RetrieveOutput on μ to get an output value $\tilde{y}$, and the adversary wins if the challenger accepts this output and $\tilde{y} \neq F(x)$. We remark that manager model 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 a completely untrusted manager may have to provide the full output from the server and the client performs the 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.

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

1: $\{t_i^\star, x_i^\star\}_{i\in[n]} \leftarrow \mathcal{A}(1^\lambda)$;
2: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
3: $(PK_F, L_F) \leftarrow \mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, F)$;
4: $SK_S \leftarrow \mathsf{Register}(\mathrm{PP}, \mathrm{MK}, S)$;
5: $(EK_{F,S}, L_F) \leftarrow \mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, S)$;
6: **for** $i = 1$ **to** n **do**
7: $\quad (\sigma_{x_i^\star}, VK_{F,x_i^\star}, RK_{F,x_i^\star}) \leftarrow \mathsf{ProbGen}(\{t_i^\star, x_i^\star\}, PK_F)$;
8: $\quad \sigma_{y_i^\star} \leftarrow \mathsf{Compute}(\sigma_{x_i^\star}, EK_{F,S}, SK_S)$;
9: $(\mu, \tau_{\sigma_y}) \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{RetrieveOutput}}(\mathrm{PP}, \{\sigma_{y_i^\star}\}, \{VK_{F,x_i^\star}\}, PK_F, L_F)$;
10: **if** $\exists i \in [n]$ s.t. $(\tilde{y} \leftarrow \mathsf{RetrieveOutput}(\mu, \tau_{\sigma_y}, VK_{F,x_i^\star}, RK_{F,x_i^\star}))$ and $(\tilde{y} \neq F(x_i^\star))$ and $(\tilde{y} \neq \perp)$ **then**
11: $\quad$ **return** 1
12: **else**
13: $\quad$ **return** 0

Definition 6. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number of queries q in the Vindictive Manager Experiment is defined as:*

$$Adv_{\mathcal{A}}^{VindictiveM}(\mathcal{RPVC}, F, 1^\lambda, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{VindictiveM}[\mathcal{RPVC}, F, 1^\lambda] = 1].$$

A RPVC is secure against vindictive servers *for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{VindictiveM}(\mathcal{RPVC}, F, 1^\lambda, q) \leq \mathrm{negl}(\lambda)$.*

3.5.5 Blind Verificaton

With Game 5, we aim to show that a verifier that does not know the value of b chosen in ProbGen cannot learn the value of $F(x)$ given the encoded output. The challenger chooses an input value, x, at random from the domain of F and a time period, and uses these to generate an encoded input. He runs Compute on this input and gives the encoded output and the verification key to the adversary who must output a guess for the value of $F(x)$. We require that $\mathcal{A}$ does not make a query to the RetrieveOutput oracle for $RK_{F,x}$ as this would constitute a trivial win.

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. Alternatively, if one considers that client devices and the KDC are within organisational boundaries then one could publish the attribute keys (in PP) only to these entities, and therefore the external servers and manager will not understand the semantic meaning of attribute sets given as input.

Game 5 $\mathbf{Exp}_{\mathcal{A}}^{BVerif}\left[\mathcal{RPVC}, F, 1^{\lambda}\right]$:

1: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^{\lambda})$;
2: $(PK_F, L_F) \leftarrow \mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, F)$;
3: $SK_S \leftarrow \mathsf{Register}(\mathrm{PP}, \mathrm{MK}, S)$;
4: $(EK_{F,S}, L_F) \leftarrow \mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, S)$;
5: $t \xleftarrow{\$} \tau$;
6: $x \xleftarrow{\$} \mathsf{Dom}(F)$;
7: $(\sigma_x, VK_{F,x}, RK_{F,x}) \leftarrow \mathsf{ProbGen}(t, x, PK_F)$;
8: $\sigma_y \leftarrow \mathsf{Compute}(\sigma_x, EK_{F,S}, SK_S)$;
9: $\hat{y} \leftarrow \mathcal{A}^{\mathcal{O}, \mathsf{RetrieveOutput}}(\sigma_y, VK_{F,x}, \mathrm{PP}, PK_F, L_F)$;
10: **if** $(\hat{y} = F(x))$ **then**
11: **return** 1
12: **else**
13: **return** 0

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:*

$$Adv_{\mathcal{A}}^{BVerif}(\mathcal{RPVC}, F, 1^{\lambda}, q) = \Pr[\mathbf{Exp}_{\mathcal{A}}^{BVerif}[\mathcal{RPVC}, F, 1^{\lambda}] = 1].$$

A RPVC is secure against vindictive servers *for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{BVerif}(\mathcal{RPVC}, F, 1^{\lambda}, q) \leq \mathrm{negl}(\lambda)$.*

4 Construction

4.1 Introduction

We now provide an instantiation of a RPVC scheme. Our construction is based on that used by Parno et al. [14] (summarised in App. A) which uses Key-Policy Attribute-based Encryption (KP-ABE) in a black-box manner to outsource the computation of a Boolean function. Following Parno et al. 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 follow Parno *et al.* and 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 satisfies the following, where $RK_{F,x} = b$:

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

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

4.2 Technical Details

We assume the existence of a *revocable KP-ABE scheme* for a class of functions $\mathcal{F}$ that is closed under complement. Such a scheme defines the algorithms ABE.Setup, ABE.KeyGen, ABE.KeyUpdate, ABE.Encrypt and ABE.Decrypt. We also make use of a signature scheme with algorithms Sig.KeyGen, Sig.Sign and Sig.Verify and a one-way function g. Let the following be universes of attributes acceptable by the ABE scheme: $\mathcal{U}_{\text{attr}}$ form characteristic tuples for input values to outsourced computations, as detailed in Sect. A; $\mathcal{U}_{\text{ID}}$ comprises attributes representing entity identifiers; $\mathcal{U}_{\text{time}}$ comprises attributes representing time periods issued by the time source $\mathcal{T}$. Finally, for notational convenience, let $\mathcal{U}_{\mathcal{F}}$ be a universe of attribute labels representing functions, thus $f \in \mathcal{U}_{\mathcal{F}}$ represents the function F. Then $F \wedge f$ 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.

Then we construct a publicly verifiable computation scheme for the same class of functions comprising the algorithms RPVC.Setup, RPVC.FnInit, RPVC.Register, RPVC.Certify, RPVC.ProbGen, RPVC.Compute, RPVC.Verify and RPVC.Revoke.

Note that in the original scheme by Parno et al. [14] the required security property of the underlying KP-ABE scheme was a one-key IND-CPA notion. This is a more relaxed notion that considered in the vast majority of the ABE literature (where usually the adversary is provided with 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 certifies for only a single function per set of public parameters (that is, the client must set up a new ABE environment per function). In our setting with a trusted third party however, we are interested in a more decentralised (and more efficient) environment where the KDC can issue keys for multiple functions within a single system. Thus we require the more standard, multi-key notion of security usually considered for ABE schemes.

On a similar note, we again mention that the security games presented in this paper are in the selective ABE model and are written in a format that allows consistency with the IND-sHRSS game for revocable KP-ABE [1]. However, since the ABE algorithms are used in a black box manner, we believe that choosing a instantiation with stronger security properties (for example, a fully secure ABE [12] scheme supporting indirect user revocation) should easily allow for a correspondingly more secure VC construction than presented here.

Finally, we remark that whereas in the revocable ABE scheme of Attrapadung et al. [1] the update keys were generated for the set of nodes in $Cover(R)$ where R is the list of revoked users (as discussed in Section A), in our definitions we have a list L_F of certified servers for a particular function. Thus L_F is the inverse of R i.e. $L_F = \mathcal{U}_{\text{id}} \setminus R$. To construct the set of nodes for which key update material should be generated, therefore, we can either compute R by taking the complement as above, or by altering the ABE scheme to use the following

algorithm instead. First mark all leaves in L_F. Then, working from the leaves upwards in a breadth-first manner, mark all nodes that have both children marked and subsequently unmark the child nodes. For ease of notation, in the following algorithms we simply associate the roles of R and L_F and pass L_F directly to the revocable KP-ABE algorithms instead of R, with the assumption that one of the above transformations has been performed.

4.3 Construction

Informally, the scheme operates in the following way.

1. RPVC.Setup establishes public parameters and a master secret key by calling the ABE.Setup algorithm twice. This algorithm also initializes a list of registered servers L_{Reg} and a time source τ[4].
2. RPVC.FnInit initializes a list of servers L_F authorized to compute function F.
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 include 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 $\overline{F}$. The algorithm also updates L_F to include S.
5. RPVC.ProbGen creates a problem instance $\sigma_x = (c_0, c_1)$ by encrypting two randomly chosen messages, 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 for $\overline{F}$.
6. RPVC.Compute is run by a server S and computes $F(x)$. Given a problem instance $\sigma_x = (c_0, c_1)$ it returns $(m_0, \perp)$ if $F(x) = 1$ or $(\perp, m_1)$ if $F(x) = 0$, ordered according to b chosen in RPVC.ProbGen, together with a digital signature computed over the output.
7. RPVC.Verify either accepts the output $\sigma_y = (d_0, d_1)$ or rejects it. This algorithm verifies the signature on the output and then confirms the output is correct by applying g and comparing with $VK_{F,x}$. In RPVC.BlindVerify the verifier can compare pairwise between the components of σ_y 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 therefore do not learn 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 satisfying (non-$\perp$) output value $\mu \in \{d_b, d_{1-b}\}$ where $RK_{F,x} = b$. Parno et al. [14] gave a one-line remark that permuting the key pairs and ciphertexts being given out in the ProbGen stage could allow for 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 a different set of 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.RetrieveOutput a verifier that has knowledge of $RK_{F,x}$ can check whether the output from BlindVerify 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 updates L_F 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,

[4] τ could be a counter that is maintained in the public parameters or a networked clock.

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.

Algorithm 1 RPVC.Setup

1: Let $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup \mathcal{U}_{\mathcal{F}}$
2: $(MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, \mathcal{U})$
3: $(MPK^1_{\text{ABE}}, MPK^1_{\text{ABE}}) \leftarrow \mathsf{ABE.Setup}(1^\lambda, \mathcal{U})$
4: $L_{\text{Reg}} = \epsilon$ (i.e. an empty list is created)
5: Initialize τ
6: $\text{PP} = (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{\text{Reg}}, \tau)$
7: $\text{MK} = (MSK^0_{\text{ABE}}, MSK^1_{\text{ABE}})$

Algorithm 2 RPVC.FnInit

1: Set $PK_F = \text{PP}$
2: Set $L_{\text{F}} = \epsilon$ (i.e. an empty list is created)

Algorithm 3 RPVC.Register

1: $(SK_{\text{Sig}}, VK_{\text{Sig}}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
2: $SK_S = SK_{\text{Sig}}$
3: $L_{\text{Reg}} = L_{\text{Reg}} \cup (S, VK_{\text{Sig}})$

Algorithm 4 RPVC.Certify

1: $t \leftarrow \tau$
2: $SK^0_{\text{ABE}} \leftarrow \mathsf{ABE.KeyGen}(S, F \wedge f, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$
3: $SK^1_{\text{ABE}} \leftarrow \mathsf{ABE.KeyGen}(S, \overline{F} \wedge f, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$
4: $UK^0_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_{\text{F}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$
5: $UK^1_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_{\text{F}}, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$
6: Output: $EK_{\text{F,S}} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$ and $L_{\text{F}} = L_{\text{F}} \cup S$

Algorithm 5 RPVC.ProbGen

1: $t \leftarrow \tau$
2: $(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
3: $b \xleftarrow{\$} \{0, 1\}$
4: $c_b \leftarrow \mathsf{ABE.Encrypt}(t, (x \cup f), m_b, MPK^0_{\text{ABE}})$
5: $c_{1-b} \leftarrow \mathsf{ABE.Encrypt}(t, (x \cup f), m_{1-b}, MPK^1_{\text{ABE}})$
6: Output: $\sigma_x = (c_b, c_{1-b})$, $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\text{Reg}})$ and $RK_{F,x} = b$

Algorithm 6 RPVC.Compute

1: Input: $EK_{\text{F,S}} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$ and $\sigma_x = (c_b, c_{1-b})$
2: Parse σ_x as (c, c')
3: $d_0 \leftarrow \mathsf{ABE.Decrypt}(c, SK^0_{\text{ABE}}, MPK^0_{\text{ABE}}, UK^0_{L_F,t})$
4: $d_1 \leftarrow \mathsf{ABE.Decrypt}(c', SK^1_{\text{ABE}}, MPK^1_{\text{ABE}}, UK^1_{L_F,t})$
5: $\gamma \leftarrow \mathsf{Sig.Sign}((d_b, d_{1-b}, S), SK_S)$
6: Output: $\sigma_y = (d_b, d_{1-b}, S, \gamma)$

Algorithm 7 RPVC.BlindVerify

1: Input: $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\text{Reg}})$ and $\sigma_y = (d_b, d_{1-b}, S, \gamma)$
2: **if** $S \in L_F$ and $(S, VK_{\text{Sig}}) \in L_{\text{Reg}}$ **then**
3: **if** Sig.Verify$((d_b, d_{1-b}, S), \gamma, VK_{\text{Sig}}) \to$ accept **then**
4: **if** $g(m_b) = g(d_b)$ **then**
5: Output $(\mu = d_b, \tau_{\sigma_y} = (\mathsf{accept}, S))$
6: **else if** $g(m_{1-b}) = g(d_{1-b})$ **then**
7: Output $(\mu = d_{1-b}, \tau_{\sigma_y} = (\mathsf{accept}, S))$
8: **else**
9: Output $(\mu = \perp, \tau_{\sigma_y} = (\mathsf{reject}, S))$
10: Output $(\mu = \perp, \tau_{\sigma_y} = (\mathsf{reject}, \perp))$

Algorithm 8 RPVC.RetrieveOutput

1: Input: $VK_{F,x} = (g(m_b), g(m_{1-b}), L_{\text{Reg}})$, $\sigma_y = (d_b, d_{1-b}, S, \gamma)$, $RK_{F,x} = b$, and (μ, τ_{σ_y}) where $\mu \in \{d_b, d_{1-b}, \perp\}$
2: **if** $\tau_{\sigma_y} = (\mathsf{accept}, S)$ and $g(\mu) = g(m_0)$ **then**
3: Output $\tilde{y} = 1$
4: **else if** $\tau_{\sigma_y} = (\mathsf{accept}, S)$ and $g(\mu) = g(m_1)$ **then**
5: Output $\tilde{y} = 0$
6: **else**
7: Output $\tilde{y} = \perp$

Algorithm 9 RPVC.Revoke

1: **if** $\tau_{\sigma_y} = (\mathsf{reject}, S)$ **then**
2: $L_{\text{F}} = L_{\text{F}} \setminus S$
3: Refresh[5] τ
4: $t \leftarrow \tau$
5: $UK^0_{L_F,t} \leftarrow$ ABE.KeyUpdate$(L_{\text{F}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$
6: $UK^1_{L_F,t} \leftarrow$ ABE.KeyUpdate$(L_{\text{F}}, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$
7: **for** all $S' \in L_{\text{F}}, S' \neq S$ **do**
8: Parse $EK_{F,S'}$ as $(SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t-1}, UK^1_{L_F,t-1})$
9: Update and send $EK_{F,S'} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$.
10: **else**
11: output $\perp$

4.4 Proof of Security

We now give a theorem and proof that the construction presented above is secure against the games presented in Section 3.5.

Theorem 1. *Given a secure revocable KP-ABE scheme in the sense of indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*) [1] for a class of functions $\mathcal{F}$ closed under complement, a signature scheme secure against* EUF-CMA *and a one-way function g, let $\mathcal{VC}$ be the verifiable computation scheme defined in Algorithms 1–9. Then $\mathcal{VC}$ is secure against Public Verifiability, Revocation and Vindictive Servers.*

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). We now present a more formal proof for Theorem 1. The proofs partially follow in the spirit of [14]. In all the following proofs, for ease

of notation, we denote the function F by f^0 and the complement function $\overline{F}$ by f^1. First we prove the following Lemma.

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

Proof. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the Public Verifiability game (Game 1) when instantiated by Algorithms 1–9. We begin by arguing that if the revocable ABE scheme is IND-sHRSS secure then $\mathcal{A}_{VC}$ cannot distinguish between the following two games:

- **Game 0:** The real Public Verifiability game.
- **Game 1:** The Public Verifiability game is modified such that in the challenge ProbGen stage, rather than encrypting m_0 and m_1 to create challenge ciphertexts c_0 and c_1, the challenger chooses a random message $m_2 \neq m_0, m_1$ and chooses r such that $f^r(x) = 0$.[6] The challenger then replaces c_r with the encryption of m_2.

If $\mathcal{A}_{VC}$ succeeds in distinguishing between these two games with non-negligible probability δ then we are able to construct an adversary $\mathcal{A}_{ABE}$ against the ABE security. $\mathcal{A}_{ABE}$ interacts with the challenger $\mathcal{C}$ in the ABE security game and acts as the challenger for $\mathcal{A}_{VC}$ in the security game for Public Verifiability for a function F as follows.

1. *(Initialise phase.)* $\mathcal{A}_{VC}$ declares a set of n challenge input sets with corresponding time periods $\{t_i^\star, x_i^\star\}_{i\in[n]}$. $\mathcal{A}_{ABE}$ chooses one of these n input attribute sets at random that he will issue as his challenge input set to $\mathcal{C}$ for the ABE game. Denote this attribute set by $\overline{(t,x)}$.
2. *(Setup phase.)* $\mathcal{C}$ runs $\mathsf{ABE.Setup}(1^\lambda) \to (MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}})$ and sends MPK^0_{ABE} to $\mathcal{A}_{ABE}$.
3. $\mathcal{A}_{ABE}$ runs $\mathsf{ABE.Setup}(1^\lambda) \to (MPK^1_{\text{ABE}}, MSK^1_{\text{ABE}})$. He creates an empty list $L_{Reg} = \epsilon$ and initializes τ. He also creates the public parameters $PP = (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{Reg}, \tau)$.
4. *(FnInit phase.)* $\mathcal{A}_{ABE}$ sets $PK_F = PP$ and creates an empty list $L_F = \epsilon$.
5. *(Register phase.)* $\mathcal{A}_{ABE}$ runs the signature key generation algorithm $(SK_{Sig}, VK_{Sig}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ to register $\mathcal{A}_{VC}$. He sets $SK_{\mathcal{A}_{VC}} = SK_{Sig}$ and adds $\mathcal{A}_{VC}$ to the registered entities list $L_{Reg} = L_{Reg} \cup (\mathcal{A}_{VC}, VK_{Sig})$.
6. *(Query phase.)* $\mathcal{A}_{ABE}$ sends $(PK_F, PP, L_F, SK_{\mathcal{A}_{VC}})$ to $\mathcal{A}_{VC}$ who outputs an updated list L_F.
7. $\mathcal{A}_{ABE}$ forwards to $\mathcal{C}$ the list $\tilde{R} = L_F$ received from $\mathcal{A}_{VC}$. Furthermore, draw a $t \leftarrow \tau$.
8. $\mathcal{A}_{ABE}$ certifies $\mathcal{A}_{VC}$ for the function F as follows. Choose $r \leftarrow \{0,1\}$ such that $f^r(\{\overline{t,x}\}) = 0$. He makes a KeyGen request to $\mathcal{C}$ for $(\mathsf{id}, \mathbb{A})$ pair $(\mathcal{A}_{VC}, f^r)$ (where for simplicity we assume some notational equivalence between access structures and functions).
   $\mathcal{C}$ checks if $\overline{(t,x)} \in \mathbb{A}$, if so it checks whether $\mathsf{id} \in \tilde{\mathsf{R}}$ and if so returns $\perp$. Otherwise $\mathcal{C}$ runs $SK_{(\mathsf{id},\mathbb{A})} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A} \wedge \mathsf{f}, \mathsf{MSK}^0_{\text{ABE}}, \mathsf{MPK}^0_{\text{ABE}})$ and returns this to $\mathcal{A}_{ABE}$ who parses this as SK^r_{ABE}.
   $\mathcal{A}_{ABE}$ also makes a KeyUpdate oracle request to $\mathcal{C}$ for the (R,t) pair (L_F, t). $\mathcal{C}$ responds by checking if $t = \bar{t}$ then if $\tilde{R} \not\subseteq R$ return $\perp$. Otherwise, $\mathcal{C}$ runs $UK^r_{R,t} \leftarrow \mathsf{ABE.KeyUpdate}(R, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$ and returns this to $\mathcal{A}_{ABE}$.
   $\mathcal{A}_{ABE}$ then creates a key for f^{1-r} by running
   $$SK^{1-r}_{\text{ABE}} \leftarrow \mathsf{ABE.KeyGen}(\mathcal{A}_{VC}, f^{1-r} \wedge f, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$$
   and an update key $UK^{1-r}_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_F, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$.
   $\mathcal{A}_{ABE}$ finally sets $EK_{F,\mathcal{A}_{VC}} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$ and updates $L_F = L_F \cup \mathcal{A}_{VC}$.

[6] Recall that $f^0(x) = F(x)$ and $f^1(x) = \overline{F}(x) = \neg F(x)$

9. *(Probgen stage.)* Here $\mathcal{A}_{ABE}$ must run $\mathsf{ProbGen}$ for all challenge inputs issued by $\mathcal{A}_{VC}$ in the Initialise phase. Note that $\mathcal{A}_{ABE}$ chose one input $\overline{(t,x)}$ to be its challenge input for $\mathcal{C}$. Thus, for $\overline{(t,x)}$, $\mathcal{A}_{ABE}$ should make a challenge query to $\mathcal{C}$ but for all other inputs, he can use the public parameters MPK^0_{ABE} to create ciphertexts himself.
   - For $\{t^\star_i, x^\star_i\}_{i\in[n]} \setminus \overline{(t,x)}$: Choose $m^i_0, m^i_1 \xleftarrow{\$} \mathcal{M}\times\mathcal{M}$ and $b^i \xleftarrow{\$} \{0,1\}$. Compute

     $$c^i_{b^i} = \mathsf{ABE.Encrypt}(t^\star_i, (x^\star_i \cup f), m^i_{b^i}, MPK^0_{\text{ABE}})$$

     and

     $$c^i_{1-b^i} = \mathsf{ABE.Encrypt}(t^\star_i, (x^\star_i \cup f), m^i_{1-b^i}, MPK^1_{\text{ABE}})$$

     Set $\sigma_{x^\star_i} = (c^i_{b^i}, c^i_{1-b^i})$, $VK_{F,x^\star_i} = (g(m^i_{b^i}), g(m^i_{1-b^i}), L_{Reg})$ and $RK_{F,x^\star_i} = b^i$.
   - For $\overline{(t,x)}$: Choose $\overline{m_0}, \overline{m_1}, \overline{m_2} \xleftarrow{\$} \mathcal{M}\times\mathcal{M}\times\mathcal{M}$. Send $\overline{m_0}, \overline{m_1}$ to $\mathcal{C}$ to receive back an encryption c of one of these messages $\overline{m_b}$ where $b \xleftarrow{\$} \{0,1\}$ under the attribute set $\overline{(t,x)}$.

     $\mathcal{A}_{ABE}$ also computes the encryption of $\overline{m_2}$ himself and sets

     $$\sigma_{\bar{x}} = (c, \mathsf{ABE.Encrypt}(\bar{t}, (\bar{x} \cup f), \overline{m_2}, MPK^1_{\text{ABE}}))$$

     He chooses a random $s \xleftarrow{\$} \{0,1\}$, sets $VK_{F,\bar{x}} = (g(\overline{m_s}), g(\overline{m_2}), L_{Reg})$ and $RK_{F,\bar{x}} \xleftarrow{\$} \{0,1\}$.
10. *(Query phase.)* $\mathcal{A}_{VC}$ is now given the values $(PK_F, \text{PP}, L_F, EK_{F,\mathcal{A}_{VC}}, SK_{\mathcal{A}_{VC}})$ as well as all $(\sigma_x, VK_{F,x})$ pairs generated in the previous step. It is also given oracle access to the following functions:
    - $\mathsf{FnInit}(\text{PP}, \text{MK}, \cdot)$: Let G be the function queried for. $\mathcal{A}_{ABE}$ sets $PK_G = \text{PP}$ and creates an empty list $L_G = \epsilon$.
    - $\mathsf{Register}(\text{PP}, \text{MK}, \cdot)$: Let S be the server which should be registered. $\mathcal{A}_{ABE}$ runs the signature key generation algorithm $(SK_{Sig}, VK_{Sig}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ to register S. He sets $SK_S = SK_{Sig}$ and adds S to the registered entities list $L_{Reg} = L_{Reg} \cup (S, VK_{Sig})$.
    - $\mathsf{Certify}(\text{PP}, \text{MK}, \cdot, \cdot, \cdot)$: Let G be the queried function, L_G be the list of certified servers for this function, and S be the server identity. $\mathcal{A}_{ABE}$ certifies S for G as stated in the $\mathsf{Certify}$ phase above.
    - $\mathsf{Revoke}(\text{MK}, \cdot, \cdot, \cdot)$: Let τ_{σ_y}, G, L_G be the queried parameters. If $\tau = (\mathsf{accept}, S, \sigma_y)$ then return $\perp$. Otherwise set $L_G = L_G \setminus S$ and $t = t+1$. Then query $\mathcal{C}$ for key update $UK^0_{L_G,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_G, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. Compute $UK^1_{L_G,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_G, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$. Then, for all $S' \in L_G, S' \neq S$, parse $EK_{G,S'}$ as $(SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_G,t-1}, UK^1_{L_G,t-1})$ and update and send $EK_{G,S'} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_G,t}, UK^1_{L_G,t})$.
    - *(Guess phase.)* $\mathcal{A}_{VC}$ outputs a result $\sigma_{y^\star}$.
    - If $g(\sigma_{y^\star}) = g(m_s)$ then $\mathcal{A}_{ABE}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{ABE}$ guesses $b' = 1-s$.

Note that if $\mathcal{A}_{ABE}$ correctly guesses a input attribute set $\overline{(t,x)}$ that $\mathcal{A}_{VC}$ wins on (which happens with probability at least $\frac{1}{n}$), then if $s = b$ then the distribution of the above coincides

with Game 0. Otherwise, if $s = 1 - b$ the distribution coincides with Game 1. Thus,

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

Hence,

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

Since n is polynomial in the security parameter and δ is non-negligible, $\frac{\delta}{2n}$ is also non-neglible. If $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the ABE 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.

We now show that using $\mathcal{A}_{VC}$ in Game 1, we can construct an adversary that inverts the one-way function g – that is, given a challenge $g(z)$ we construct an adversary that can recover z. Specifically, we implicitly choose the challenge message $\overline{m_r} = z$ and then set the verification key for the successful forgery to be $g(z)$. Since we choose r such that $f^{1-r}(x) = 1$, an honest server would return $\overline{m_{1-r}}$. A cheating server (i.e. $\mathcal{A}_{VC}$) will return $\sigma_{y^\star} = \overline{m_r}$ such that $g(\overline{m_r}) = g(z)$, and hence our adversary can output $z = \sigma_{y^\star}$ to invert the one-way function with non-negligible probability.

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

Lemma 2. *The $\mathcal{RPVC}$ scheme is secure against Revocation (Game 2) from $\mathcal{ABE}$ defined by Algorithms 1–9 under the same assumptions as in Theorem 1.*

Proof. The proof of this lemma is similar to the one of Lemma 1. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the Revocation game (Game 2) when instantiated by Algorithms 1–9. We begin by arguing that if the revocable ABE scheme is IND-sHRSS secure then $\mathcal{A}_{VC}$ cannot distinguish between the following two games:

- **Game 0:** The real Revocation game.
- **Game 1:** The Revocation game is modified in the following way. In the challenge ProbGen stage, instead of returning an encryption of m_0 and an encryption of m_1 to create challenge ciphertexts c_0 and c_1, the challenger chooses a random message $m_2 \neq m_0, m_1$ and chooses r such that $f^r(x) = 0$. The challenger then replaces c_r with the encryption of m_2.

In case $\mathcal{A}_{VC}$ succeeds in distinguishing both games with non-negligible probability δ then we are able to construct a second adversary $\mathcal{A}_{ABE}$ against the ABE security. We let $\mathcal{A}_{ABE}$ interact with the challenger $\mathcal{C}$ in the ABE security game and let $\mathcal{A}_{ABE}$ act as the challenger for $\mathcal{A}_{VC}$ in the security game for Revocation for a function F as follows.

1. *(Initialise phase.)* First, $\mathcal{A}_{VC}$ announces a set of n challenge input sets with corresponding time periods $\{t_i^\star, x_i^\star\}_{i\in[n]}$. Next, $\mathcal{A}_{ABE}$ chooses one of these n input attribute pairs at random that he will use as his challenge input and issue it to $\mathcal{C}$ for the ABE game. Denote this attribute set by $\overline{(t,x)}$.
2. *(Setup phase.)* $\mathcal{C}$ runs $\mathsf{ABE.Setup}(1^\lambda) \rightarrow (MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}})$ and sends MPK^0_{ABE} to $\mathcal{A}_{ABE}$. Then $\mathcal{A}_{ABE}$ runs $\mathsf{ABE.Setup}(1^\lambda) \rightarrow (MPK^1_{\text{ABE}}, MSK^1_{\text{ABE}})$ and creates an empty list $L_{Reg} = \epsilon$ and initializes τ. $\mathcal{A}_{ABE}$ then publishes the public parameters $PP = (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{Reg}, \tau)$.
3. *(FnInit phase.)* $\mathcal{A}_{ABE}$ sets $PK_F = PP$ and creates an empty list $L_F = \epsilon$.
4. *(Register phase.)* The adversary $\mathcal{A}_{ABE}$ registers $\mathcal{A}_{VC}$ by running the signature key generation algorithm $(SK_{Sig}, VK_{Sig}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$. $\mathcal{A}_{ABE}$ sets $SK_{\mathcal{A}_{VC}} = SK_{Sig}$ and addputs $\mathcal{A}_{VC}$ to the list with registered entities $L_{Reg} = L_{Reg} \cup (\mathcal{A}_{VC}, VK_{Sig})$ and finally draw a $t \leftarrow \tau$.
5. *(Query phase.)* $\mathcal{A}_{ABE}$ sends $(PK_F, PP, L_F, SK_{\mathcal{A}_{VC}})$ to $\mathcal{A}_{VC}$ who outputs an updated list $L_F = L_F \cup \mathcal{A}_{VC}$.
6. $\mathcal{A}_{ABE}$ forwards to $\mathcal{C}$ the list $\tilde{R} = L_F \setminus \mathcal{A}_{VC}$ (if $\mathcal{A}_{VC}$ is in the list, otherwise he just directly forwards the list) received from $\mathcal{A}_{VC}$. Note that we require $\mathcal{A}_{VC} \notin \tilde{R}$ since this is a list of revoked entities at the challenge time $\bar{t}$ received from $\mathcal{A}_{VC}$.
7. $\mathcal{A}_{ABE}$ has to certify $\mathcal{A}_{VC}$ for the function F by choosing a $r \leftarrow \{0,1\}$ such that $f^r(\overline{(t,x)}) = 0$. Then $\mathcal{A}_{ABE}$ makes a KeyGen request to $\mathcal{C}$ for $(\mathsf{id}, \mathbb{A})$ which is notational equivalent to the pair $(\mathcal{A}_{VC}, f^r)$.
   Then the challenger $\mathcal{C}$ checks if $\overline{(t,x)} \in \mathbb{A}$, if so $\mathcal{C}$ checks whether $\mathsf{id} \in \tilde{\mathsf{R}}$ and if so returns $\perp$. Otherwise $\mathcal{C}$ runs $SK_{(\mathsf{id},\mathbb{A})} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A} \wedge \mathsf{f}, \mathsf{MSK}^0_{\text{ABE}}, \mathsf{MPK}^0_{\text{ABE}})$ and returns this to $\mathcal{A}_{ABE}$ who parses this as SK^r_{ABE}.
   $\mathcal{A}_{ABE}$ also makes a $\mathsf{KeyUpdate}$ oracle request to $\mathcal{C}$ for the (R,t) pair (L_F, t). $\mathcal{C}$ responds by checking if $t = \bar{t}$ then if $\tilde{R} \not\subseteq R$ return $\perp$. Otherwise, $\mathcal{C}$ runs $UK^r_{R,t} \leftarrow \mathsf{ABE.KeyUpdate}(R, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$ and returns this to $\mathcal{A}_{ABE}$.
   $\mathcal{A}_{ABE}$ then creates a key for f^{1-r} by running
   $$SK^{1-r}_{\text{ABE}} \leftarrow \mathsf{ABE.KeyGen}(\mathcal{A}_{VC}, f^{1-r} \wedge f, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$$
   and an update key $UK^{1-r}_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_F, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$.
   Finally $\mathcal{A}_{ABE}$ sets $EK_{F,\mathcal{A}_{VC}} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$ and updates $L_F = L_F \cup \mathcal{A}_{VC}$.
8. *(Revocation phase.)* $\mathcal{A}_{VC}$ is now given $(PK_F, PP, L_F, SK_{\mathcal{A}_{VC}})$ and oracle access to $\mathcal{O}$ and outputs a rejection token $\tau^\star = (\mathsf{reject}, \mathcal{A}_{VC})$ for himself.
9. $\mathcal{A}_{ABE}$ now revokes $\mathcal{A}_{VC}$ for the function F as follows. First, $\mathcal{A}_{VC}$ will be removed from $L_{\text{F}} = L_{\text{F}} \setminus S$ and the clock is updated to $t = t + 1$. $\mathcal{A}_{ABE}$ makes a $\mathsf{KeyUpdate}$ oracle request to $\mathcal{C}$ to obtain $UK^0_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_F, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$ and updates $UK^1_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_F, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$ himself. Now for all $S \in L_{\text{F}}$ where $S \neq \mathcal{A}_{VC}$, we parse $EK_{F,S} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t-1}, UK^1_{L_F,t-1})$ and update and send $EK_{F,S} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$ around.
10. *(Probgen stage.)* In this stage $\mathcal{A}_{ABE}$ must run $\mathsf{ProbGen}$ for all $(i \in [n])$ challenge inputs issued by $\mathcal{A}_{VC}$ in the Initialise phase. Notice that $\mathcal{A}_{ABE}$ choses one input $\overline{(t,x)}$ to be his challenge input for the challenger $\mathcal{C}$. Thus, for $\overline{(t,x)}$, $\mathcal{A}_{ABE}$ makes a challenge query to $\mathcal{C}$

but for all other inputs, he is able to use the public parameters MPK^0_{ABE} (created by $\mathcal{C}$) to create ciphertexts himself.

- For $\{t^\star_i, x^\star_i\}_{i\in[n]} \setminus \overline{(t,x)}$: Choose $m^i_0, m^i_1 \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$ and $b^i \xleftarrow{\$} \{0,1\}$. Compute

$$c^i_{b^i} = \mathsf{ABE.Encrypt}(t^\star_i, (x^\star_i \cup f), m^i_{b^i}, MPK^0_{\text{ABE}})$$

  and

$$c^i_{1-b^i} = \mathsf{ABE.Encrypt}(t^\star_i, (x^\star_i \cup f), m^i_{1-b^i}, MPK^1_{\text{ABE}})$$

  Set $\sigma_{x^\star_i} = (c^i_{b^i}, c^i_{1-b^i})$, $VK_{F,x^\star_i} = (g(m^i_{b^i}), g(m^i_{1-b^i}), L_{Reg})$ and $RK_{F,x^\star_i} = b^i$.
- For $\overline{(t,x)}$: Choose $\overline{m_0}, \overline{m_1}, \overline{m_2} \xleftarrow{\$} \mathcal{M} \times \mathcal{M} \times \mathcal{M}$. Send $\overline{m_0}, \overline{m_1}$ to $\mathcal{C}$ to receive back an encryption c of one of these messages $\overline{m_b}$ where $b \xleftarrow{\$} \{0,1\}$ under the attribute set $\overline{(t,x)}$.

Finally $\mathcal{A}_{ABE}$ computes the encryption of $\overline{m_2}$ himself and sets

$$\sigma_{\bar{x}} = (c, \mathsf{ABE.Encrypt}(\bar{t}, (\bar{x} \cup f), \overline{m_2}, MPK^1_{\text{ABE}}))$$

He chooses a random $s \xleftarrow{\$} \{0,1\}$, sets $VK_{F,\bar{x}} = (g(\overline{m_s}), g(\overline{m_2}), L_{Reg})$and $RK_{F,\bar{x}} \xleftarrow{\$} \{0,1\}$.

11. *(Query phase.)* $\mathcal{A}_{VC}$ is now given the values $(PK_F, \text{PP}, L_F, \{EK_{F,S}\}, SK_{\mathcal{A}_{VC}})$ as well as all $(\sigma_x, VK_{F,x})$ pairs generated in the previous step (here the set $\{EK_{F,S}\}$ can consist of "old" keys with the former time stamp $t-1$ too). Furthermore, $\mathcal{A}_{VC}$ is also given oracle access to the following functions:
    - $\mathsf{KeyGen}(\text{PP}, \text{MK}, \cdot)$: Let G be the function queried for. $\mathcal{A}_{ABE}$ sets $PK_G = \text{PP}$ and creates an empty list $L_G = \epsilon$.
    - $\mathsf{Register}(\text{PP}, \text{MK}, \cdot)$: Let $\tilde{S}$ be the server which should be registered. $\mathcal{A}_{ABE}$ runs the signature key generation algorithm $(SK_{Sig}, VK_{Sig}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ to register $\tilde{S}$. He sets $SK_{\tilde{S}} = SK_{Sig}$ and adds $\tilde{S}$ to the registered entities list $L_{Reg} = L_{Reg} \cup (\tilde{S}, VK_{Sig})$.
    - $\mathsf{Certify}(\text{PP}, \text{MK}, \cdot, \cdot, \cdot)$: Let G be the queried function, L_G be the list of certified servers for this function, and $\tilde{S}$ be the server identity. $\mathcal{A}_{ABE}$ certifies $\tilde{S}$ for G as stated in the $\mathsf{Certify}$ phase above.
    - $\mathsf{Revoke}(\text{MK}, \cdot, \cdot, \cdot)$: Let τ_{σ_y}, G, L_G be the queried parameters. If $\tau = (\mathsf{accept}, \tilde{S}, \sigma_y)$ then return $\perp$. Otherwise set $L_G = L_G \setminus \tilde{S}$ and $t = t+1$. Then query $\mathcal{C}$ for key update $UK^0_{L_G,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_G, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. Compute $UK^1_{L_G,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_G, t, MSK^1_{\text{ABE}}, MPK^1_{\text{ABE}})$. Then, for all $S' \in L_G, S' \neq \tilde{S}$, parse $EK_{G,S'}$ as $(SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_G,t-1}, UK^1_{L_G,t-1})$ and update and send $EK_{G,S'} = (SK^0_{\text{ABE}}, SK^1_{\text{ABE}}, UK^0_{L_G,t}, UK^1_{L_G,t})$.
    - *(Guess phase.)* $\mathcal{A}_{VC}$ outputs a result $\sigma_{y^\star}$.
    - If $g(\sigma_{y^\star}) = g(m_s)$ then $\mathcal{A}_{ABE}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{ABE}$ guesses $b' = 1-s$.

Notice that in the case $\mathcal{A}_{ABE}$ guesses an input attribute set $\overline{(t,x)}$ correctly where $\mathcal{A}_{VC}$ wins on (which happens again with a probability of at least $\frac{1}{n}$), then in case that $s = b$ the distribution of the above coincides with Game 0. Otherwise, we have the case that $s = 1-b$

and the distribution coincides with Game 1. Thus, we can claim that

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

Hence,

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

Since n is polynomial in the security parameter and δ is non-negligible, $\frac{\delta}{2n}$ is also non-neglible. If $\mathcal{A}_{VC}$ has an advantage of δ at distinguishing between the games then $\mathcal{A}_{ABE}$ can win the ABE IND-sHRSS game with non-negligible probability. Based on the assumption that the ABE scheme is IND-sHRSS secure we conclude that $\mathcal{A}_{VC}$ is not able to distinguish between Game 0 from Game 1 with non-negligible probability.

Next we argue that using $\mathcal{A}_{VC}$ in Game 1, we construct an adversary that is able to invert the one-way function g – that is, given a challenge $g(z)$ we construct an adversary that can recover z. In particular, we choose the challenge message $\overline{m_r} = z$ and set the verification key for the successful forgery to be $g(z)$. Since we choose r such that $f^{1-r}(x) = 1$, an honest server would return $\overline{m_{1-r}}$. A cheating server (i.e. $\mathcal{A}_{VC}$) will return $\sigma_{y^\star} = \overline{m_r}$ such that $g(\overline{m_r}) = g(z)$, and hence our adversary can output $z = \sigma_{y^\star}$ to invert the one-way function with non-negligible probability.

We conclude that if the ABE scheme is IND-sHRSS secure and the one-way function is secure, then the $\mathcal{VC}$ scheme defined by Algorithms 1–9 is secure in the sense of Revocation. □

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

Proof. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the Vindictive Servers game (Game 3) when instantiated by Algorithms 1–9. We show that an adversary $\mathcal{A}_{Sig}$ with non-negligble advantage δ in the EUF-CMA signatures game (Game 7) 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. $\mathcal{C}$ initialises $Q = \epsilon$ to be an empty list of messages queried to the $\mathsf{Sig.Sign}$ oracle and runs $\mathsf{Sig.KeyGen}(1^\lambda)$ to generate a challenge signing key $\overline{SK}$ and verification key $\overline{VK}$. $\mathcal{C}$ sends $\overline{VK}$ to $\mathcal{A}_{Sig}$.
2. $\mathcal{A}_{Sig}$ chooses a function F on which to instantiate $\mathcal{A}_{VC}$.
3. $\mathcal{A}_{VC}$ selects a set of n challenge input attribute sets and time periods $\{t_i^\star, x_i^\star\}_{i \in [n]}$ and sends these to $\mathcal{A}_{Sig}$.
4. $\mathcal{A}_{Sig}$ chooses a server identity from $\mathcal{U}_{\mathsf{ID}} \setminus \mathcal{A}_{VC}$ which will be denoted by $\overline{S}$.
5. $\mathcal{A}_{Sig}$ runs $\mathsf{RPVC.Setup}(1^\lambda)$, $\mathsf{RPVC.FnInit}(\mathrm{PP}, \mathrm{MK}, F)$, $\mathsf{RPVC.Register}(\mathrm{PP}, \mathrm{MK}, \mathcal{A}_{VC})$ as specified in Algorithms 1, 2 and 3 respectively and passes the resulting values PK_F, PP, L_F and $SK_{\mathcal{A}_{VC}}$ to the VC adversary.
6. $\mathcal{A}_{VC}$ outputs an updated list of authorised servers L_F.
7. $\mathcal{A}_{Sig}$ runs $\mathsf{RPVC.Certify}(\mathrm{PP}, \mathrm{MK}, F, L_F, \mathcal{A}_{VC})$ as per Algorithm 4 and for all challenge input sets chosen by $\mathcal{A}_{VC}$ in Step 3 runs $\mathsf{RPVC.ProbGen}(\{t_i^\star, x_i^\star\}, PK_F)$ as in Algorithm 5.
8. $\mathcal{A}_{VC}$ is given the values of $PK_F, \mathrm{PP}, L_F, \{\sigma_{x_i^\star}, VK_{F,x_i^\star}\}, EK_{F,\mathcal{A}_{VC}}$ and $SK_{\mathcal{A}_{VC}}$. 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.
   - $\mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, \cdot)$: Let G be the function queried for. $\mathcal{A}_{Sig}$ sets $PK_G = \mathrm{PP}$ and creates an empty list $L_G = \epsilon$ as per Algorithm 2.
   - $\mathsf{Register}(\mathrm{PP}, \mathrm{MK}, \cdot)$: Let S be the server which should be registered. If $S \neq \overline{S}$ then $\mathcal{A}_{Sig}$ runs Algorithm 3.
     Otherwise, he implicitly sets $SK_{\overline{S}} = \overline{SK}$ and updates the list $L_{Reg} = L_{Reg} \cup (\overline{S}, \overline{VK})$.
   - $\mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, \cdot, \cdot, \cdot)$: Let G be the queried function, L_G be the list of certified servers for this function, and S be the server identity. $\mathcal{A}_{Sig}$ certifies S for G as in Algorithm 4.
   - $\mathsf{Revoke}(\mathrm{MK}, \cdot, \cdot, \cdot)$: Let τ_{σ_y}, G, L_G be the queried parameters. $\mathcal{A}_{Sig}$ operates as in Algorithm 9.

   $\mathcal{A}_{VC}$ outputs a target server identity $\tilde{S}$ which has not been queried to the $\mathsf{Register}$ oracle.
9. If $\tilde{S} \neq \overline{S}$ then $\mathcal{A}_{Sig}$ outputs $\perp$ and stops. Otherwise, $\mathcal{A}_{VC}$ is allowed to continue with access to the oracles as described in Step 8 as well as a $\mathsf{Compute}$ oracle. $\mathcal{A}_{VC}$ submits queries of the form $\mathsf{Compute}(\sigma_x, EK_{F,S}, SK_S)$ for its choice of server S and σ_x (note that he may generate a valid σ_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, the query is for the challenge server identity and $\mathcal{A}_{Sig}$ does not have access to the signing key $SK_{\overline{S}}$. Thus, he runs the $\mathsf{ABE.Decrypt}$ operations correctly to generate plaintexts d_0 and d_1, and submits $m = (d_0, d_1, \overline{S})$ as a $\mathsf{Sig.Sign}$ oracle query to $\mathcal{C}$. $\mathcal{C}$ adds m to the list Q and returns $\gamma \leftarrow \mathsf{Sig.Sign}(m, \overline{SK})$, which $\mathcal{A}_{Sig}$ uses to return $\sigma_y = (d_0, d_1, \overline{S}, \gamma)$ to $\mathcal{A}_{VC}$.
10. $\mathcal{A}_{VC}$ finally outputs a result $\sigma_{y^\star}$ which he claims to be identical to an invalid result computed by $\tilde{S}$. Thus, in particular, $(\perp, (\mathsf{reject}, \tilde{S})) \leftarrow \mathsf{RPVC.Verify}(\mathrm{PP}, \sigma_{y^\star}, VK_{Fx_i^\star}, L_F)$ and $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}((d_0, d_1, \tilde{S}), \gamma, \overline{VK})$. Thus, γ is a valid signature under signing 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 3, $\mathcal{A}_{VC}$ is not allowed to have made a query for $\mathcal{O}^{\mathsf{Compute}}(\sigma_{x_i^\star}, EK_{F,\tilde{S}}, SK_{\tilde{S}})$ and thus the forgery $(m^\star, \gamma^\star)$ output by $\mathcal{A}_{Sig}$ will satisfy the requirement in Game 7 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 the size of $\mathcal{U}_{\mathsf{id}}$ is polynomial (else the KDC would not be able to operate efficiently when searching the list L_{Reg} for example and as a consequence the EUF-CMA adversary $\mathcal{A}_{Sig}$ who simulates the

KDC in the above game will also not be polynomial time). Let $n = |\mathcal{U}_{\mathsf{id}}|$, then the probability that $\mathcal{A}_{Sig}$ correctly guesses $\overline{S} = \tilde{S}$ is $\frac{1}{n}$ and

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

Thus we conclude that $\mathcal{A}_{Sig}$ has a non-negligible advantage against the EUF-CMA game if $\mathcal{A}_{VC}$ has a non-negligible advantage in the Vindictive Servers game, but since we assume the signature scheme in our construction to be EUF-CMA secure, such as adversary 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 against vindictive managers (Game 4) under the same assumptions as in Theorem 1.*

Proof. Let us assume that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the Vindictive Manager game (Game 4) when instantiated by algorithms 1–9. We start by arguing that if the revocable ABE scheme is IND-sHRSS secure then $\mathcal{A}_{VC}$ cannot distinguish between the following two games:

- **Game 0:** The real Vindictive Manager game.
- **Game 1:** The Vindictive Manager game is modified such that in the challenge ProbGen stage, rather than encrypting m_0 and m_1 to create challenge ciphertexts c_0 and c_1, the challenger chooses a random message $m_2 \neq m_0, m_1$ and chooses r such that $f^r(x) = 0$. The challenger then replaces c_r with the encryption of m_2.

If $\mathcal{A}_{VC}$ succeeds in distinguishing between these two games with non-negligible probability δ then we are able to construct an adversary $\mathcal{A}_{ABE}$ against the ABE security. $\mathcal{A}_{ABE}$ interacts with the challenger $\mathcal{C}$ in the ABE security game and acts as the challenger for $\mathcal{A}_{VC}$ in the security game for Public Verifiability for a function F as follows.

1. *(Initialise phase.)* $\mathcal{A}_{VC}$ declares a set of n challenge input sets with corresponding time periods $\{t_i^\star, x_i^\star\}_{i\in[n]}$. $\mathcal{A}_{ABE}$ chooses one of these n input attribute sets at random that he will issue as his challenge input set to $\mathcal{C}$ for the ABE game. Denote this attribute set by $\overline{(t, x)}$.
2. *(Setup phase.)* $\mathcal{C}$ runs $\mathsf{ABE.Setup}(1^\lambda) \to (MPK^0_{\text{ABE}}, MSK^0_{\text{ABE}})$ and sends MPK^0_{ABE} to $\mathcal{A}_{ABE}$.
3. $\mathcal{A}_{ABE}$ runs $\mathsf{ABE.Setup}(1^\lambda) \to (MPK^1_{\text{ABE}}, MSK^1_{\text{ABE}})$. He creates an empty list $L_{Reg} = \epsilon$ and initializes τ. He also creates the public parameters $PP = (MPK^0_{\text{ABE}}, MPK^1_{\text{ABE}}, L_{Reg}, \tau)$.
4. *(FnInit phase.)* $\mathcal{A}_{ABE}$ sets $PK_F = PP$ and creates an empty list $L_F = \epsilon$.
5. *(Register phase.)* $\mathcal{A}_{ABE}$ runs the signature key generation algorithm $(SK_{Sig}, VK_{Sig}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ to register S. He sets $SK_S = SK_{Sig}$ and adds S to the registered entities list $L_{Reg} = L_{Reg} \cup (S, VK_{Sig})$.
6. $\mathcal{A}_{ABE}$ forwards to $\mathcal{C}$ the list $\tilde{R} = L_F$ and draw $t \leftarrow \tau$.

7. *(Certify phase.)* $\mathcal{A}_{ABE}$ certifies S for the function F as follows. Choose $r \leftarrow \{0,1\}$ such that $f^r(\{\overline{t,x}\}) = 0$. He makes a KeyGen request to $\mathcal{C}$ for $(\mathsf{id}, \mathbb{A})$ pair (S, f^r) (where for simplicity we assume some notational equivalence between access structures and functions). $\mathcal{C}$ checks if $\overline{(t,x)} \in \mathbb{A}$, if so it checks whether $\mathsf{id} \in \tilde{\mathsf{R}}$ and if so returns $\perp$. Otherwise $\mathcal{C}$ runs $SK_{(\mathsf{id},\mathbb{A})} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A} \wedge \mathsf{f}, \mathsf{MSK}^0_{\mathrm{ABE}}, \mathsf{MPK}^0_{\mathrm{ABE}})$ and returns this to $\mathcal{A}_{ABE}$ who parses this as SK^r_{ABE}.
   $\mathcal{A}_{ABE}$ also makes a $\mathsf{KeyUpdate}$ oracle request to $\mathcal{C}$ for the (R,t) pair (L_F, t). $\mathcal{C}$ responds by checking if $t = \bar{t}$ then if $\tilde{R} \not\subseteq R$ return $\perp$. Otherwise, $\mathcal{C}$ runs $UK^r_{R,t} \leftarrow \mathsf{ABE.KeyUpdate}(R, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$ and returns this to $\mathcal{A}_{ABE}$.
   $\mathcal{A}_{ABE}$ then creates a key for f^{1-r} by running
   $$SK^{1-r}_{\mathrm{ABE}} \leftarrow \mathsf{ABE.KeyGen}(\mathcal{A}_{VC}, f^{1-r} \wedge, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$$
   and an update key $UK^{1-r}_{L_F,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_F, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$.
   $\mathcal{A}_{ABE}$ finally sets $EK_{F,S} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_F,t}, UK^1_{L_F,t})$ and updates $L_F = L_F \cup S$.
8. *(Probgen stage.)* Here $\mathcal{A}_{ABE}$ must run $\mathsf{ProbGen}$ for all challenge inputs issued by $\mathcal{A}_{VC}$ in the Initialise phase. Note that $\mathcal{A}_{ABE}$ chose one input $\overline{(t,x)}$ to be its challenge input for $\mathcal{C}$. Thus, for $\overline{(t,x)}$, $\mathcal{A}_{ABE}$ should make a challenge query to $\mathcal{C}$ but for all other inputs, he can use the public parameters MPK^0_{ABE} to create ciphertexts himself.
   - For $\{t^\star_i, x^\star_i\}_{i\in[n]} \setminus \overline{(t,x)}$: Choose $m^i_0, m^i_1 \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$ and $b^i \xleftarrow{\$} \{0,1\}$. Compute
     $$c^i_{b^i} = \mathsf{ABE.Encrypt}(t^\star_i, (x^\star_i \cup f), m^i_{b^i}, MPK^0_{\mathrm{ABE}})$$
     and
     $$c^i_{1-b^i} = \mathsf{ABE.Encrypt}(t^\star_i, (x^\star_i \cup f), m^i_{1-b^i}, MPK^1_{\mathrm{ABE}})$$
     Set $\sigma_{x^\star_i} = (c^i_{b^i}, c^i_{1-b^i})$, $VK_{F,x^\star_i} = (g(m^i_{b^i}), g(m^i_{1-b^i}), L_{Reg})$ and $RK_{F,x^\star_i} = b^i$.
   - For $\overline{(t,x)}$: Choose $\overline{m_0}, \overline{m_1}, \overline{m_2} \xleftarrow{\$} \mathcal{M} \times \mathcal{M} \times \mathcal{M}$. Send $\overline{m_0}, \overline{m_1}$ to $\mathcal{C}$ to receive back an encryption c of one of these messages $\overline{m_b}$ where $b \xleftarrow{\$} \{0,1\}$ under the attribute set $\overline{(t,x)}$.
     $\mathcal{A}_{ABE}$ also computes the encryption of $\overline{m_2}$ himself and sets
     $$\sigma_{\bar{x}} = (c, \mathsf{ABE.Encrypt}(\bar{t}, (\bar{x} \cup f), \overline{m_2}, MPK^1_{\mathrm{ABE}}))$$
     He chooses a random $s \xleftarrow{\$} \{0,1\}$, sets $VK_{F,\bar{x}} = (g(\overline{m_s}), g(\overline{m_2}), L_{Reg})$ and $RK_{F,\bar{x}} \xleftarrow{\$} \{0,1\}$.
9. *(Compute stage.)* $\mathcal{A}_{ABE}$ runs the $\mathsf{Compute}$ algorithm on all encoded inputs $\{\sigma_{x^\star_i}\}_{i\in[n]}$ and outputs for each query an encoded output $\sigma_{y^\star_i}$ to $\mathcal{A}_{VC}$. $\mathcal{A}_{ABE}$ parses each $\sigma_{x^\star_i}$ as (c, c') and runs $\mathsf{ABE.Decrypt}$ on c and c' with the appropriate ordered keys. If the decryption fails then $\mathcal{A}_{ABE}$ runs $\mathsf{ABE.Decrypt}$ on the same c and c' but with the reversed order of the keys. $\mathcal{A}_{ABE}$ runs $\mathsf{Sig.Sign}$ on the decrypted outputs using the earlier created signing key SK_S and finally sends $\{\sigma_{y^\star_i}\}$ to $\mathcal{A}_{VC}$.
10. *(Query phase.)* $\mathcal{A}_{VC}$ is now given the values $(\{\sigma_{y^\star_i}\}, PK_F, \mathrm{PP}, L_F, \{VK_{F,x^\star_i}\})_{i\in[n]}$. It is also given oracle access to the following functions:
    - $\mathsf{FnInit}(\mathrm{PP}, \mathrm{MK}, \cdot)$: Let G be the function queried for. $\mathcal{A}_{ABE}$ sets $PK_G = \mathrm{PP}$ and creates an empty list $L_G = \epsilon$.
    - $\mathsf{Register}(\mathrm{PP}, \mathrm{MK}, \cdot)$: Let $\tilde{S}$ be the server which should be registered. $\mathcal{A}_{ABE}$ runs the signature key generation algorithm $(SK_{Sig}, VK_{Sig}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ to register $\tilde{S}$. He sets $SK_{\tilde{S}} = SK_{Sig}$ and adds $\tilde{S}$ to the registered entities list $L_{Reg} = L_{Reg} \cup (\tilde{S}, VK_{Sig})$.

- $\mathsf{Certify}(\mathrm{PP}, \mathrm{MK}, \cdot, \cdot, \cdot)$: Let G be the queried function, L_G be the list of certified servers for this function, and $\tilde{S}$ be the server identity. $\mathcal{A}_{ABE}$ certifies $\tilde{S}$ for G as stated in the Certify phase above.
- $\mathsf{RetrieveOutput}(\cdot, \cdot, \cdot, \cdot)$: Let $\mu, \tau_{\sigma_y}, VK_{G,x}, RK_{G,x}$ be the queried parameters. If $\tau_{\sigma_y} = (\mathsf{accept}, \tilde{S})$ and $g(\mu) = g(m_0)$ then $\tilde{y} = 1$, else if $\tau_{\sigma_y} = (\mathsf{accept}, \tilde{S})$ and $g(\mu) = g(m_1)$ then $\tilde{y} = 0$. In case that τ_{σ_y} corresponds a reject token then the ouput corresponds to $\perp$.
- $\mathsf{Revoke}(\mathrm{MK}, \cdot, \cdot, \cdot)$: Let τ_{σ_y}, G, L_G be the queried parameters. If $\tau = (\mathsf{accept}, \tilde{S}, \sigma_y)$ then return $\perp$. Otherwise set $L_G = L_G \setminus \tilde{S}$ and $t = t + 1$. Then query $\mathcal{C}$ for key update $UK^0_{L_G,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_G, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$. Compute $UK^1_{L_G,t} \leftarrow \mathsf{ABE.KeyUpdate}(L_G, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$. Then, for all $S' \in L_G, S' \neq \tilde{S}$, parse $EK_{G,S'}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_G,t-1}, UK^1_{L_G,t-1})$ and update and send $EK_{G,S'} = (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_G,t}, UK^1_{L_G,t})$.

11. *(Guess phase.)* $\mathcal{A}_{VC}$ outputs a pair (μ, τ_{σ_y}).
12. If $\tau_{\sigma_y} = (\mathsf{accept}, S)$, then if $g(\mu) = g(m_s)$ then $\mathcal{A}_{ABE}$ outputs a guess $b' = s$. Else, $\mathcal{A}_{ABE}$ guesses $b' = 1 - s$.

Note that if $\mathcal{A}_{ABE}$ correctly guesses a input attribute set $\overline{(t, x)}$ that $\mathcal{A}_{VC}$ wins on (which happens with probability at least $\frac{1}{n}$), then if $s = b$ then the distribution of the above coincides with Game 0. Otherwise, if $s = 1 - b$ the distribution coincides with Game 1. Thus,

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

Hence,

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

Since n is polynomial in the security parameter and δ is non-negligible, $\frac{\delta}{2n}$ is also non-neglible. If $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the ABE 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.

We now show that using $\mathcal{A}_{VC}$ in Game 1, we can construct an adversary that inverts the one-way function g – that is, given a challenge $g(z)$ we construct an adversary that can recover z. Specifically, we implicitly choose the challenge message $\overline{m_r} = z$ and then set the verification key for the successful forgery to be $g(z)$. Since we choose r such that $f^{1-r}(x) = 1$, an honest server

would return $\overline{m_{1-r}}$. A cheating server (i.e. $\mathcal{A}_{VC}$) will return $\mu = \overline{m_r}$ such that $g(\overline{m_r}) = g(z)$, and hence our adversary can output $z = \mu$ to invert the one-way function with non-negligible probability.

We conclude that if the ABE scheme is IND-sHRSS secure and the one-way function is secure, then the $\mathcal{VC}$ scheme defined by Algorithms 1–9 is secure in the sense of Vindictive Manager. □

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

Proof. The proof follows from a standard probability argument. The inputs to the adversary that depend on x or $F(x)$ are σ_x, σ_y and $VK_{F,x}$, and thus we restrict out attention to these. First note that σ_x comprises two ABE encryptions of random messages m_0 and m_1. Since $\mathcal{A}$ does not hold a valid decryption key for these ciphertexts (since by definition it does not possess EK_F), the IND-sHRSS property of the revocable ABE scheme ensures that no information about m_0 or m_1 is learnt by the adversary. As observed in (1) in Appendix 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 $\sigma_y = \begin{cases} (m_0, \perp), & \text{if } b = 0 \\ (\perp, m_0), & \text{if } b = 1 \end{cases}$
- if $F(x) = 0$, then $\sigma_y = \begin{cases} (\perp, m_1), & \text{if } b = 0 \\ (m_1, \perp), & \text{if } b = 1 \end{cases}$

Finally note also that $VK_{F,x} = (g(m_b), g(m_{1-b}))$ by definition. We introduce the notation $\mathcal{V}$ to denote the adversary's view of σ_y and $VK_{F,x}$ (we omit σ_x since we showed that this does not reveal information related to the messages) – that is, $\mathcal{V} = (d_b, d_{1-b}, g(m_b), g(m_{1-b}))$ would imply that $\sigma_y = (d_b, d_{1-b})$ 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

In this paper we have introduced the new notion of Revocable Publicly Verifiable Outsourced Computation and provided a rigorous new framework that we believe to be more realistic than the purely theory oriented models of prior work, especially when considering the KDC to be an entity within a organisation that is responsible for user authorisation. Compared to prior

models, we believe ours to more accurately reflect practical environments and 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 the second allowed for multiple functions but required a more exotic type of ABE scheme. This allowed a single $\mathsf{ProbGen}$ 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 $\mathsf{ProbGen}$ 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 the use of 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 allow the revocation of misbehaving servers such that they receive a penalty for cheating, and that by permuting elements within messages we can achieve output privacy (as hinted at by Parno et al. although seemingly with two fewer decryptions than their brief description implies), which gives the new functionality of blind verification. We have shown that this could be used when a manager runs a pool of servers and rewards correct work – thus he should be able to verify but is not entitled to learn the result. We have extended previous notions of security to fit our new definitional framework as well as introducing models to capture threats arising from this new functionality (e.g. vindictive servers using revocation to remove competing servers), and provided a provably secure construction to meet these notions.

We believe that this work is a useful step towards making PVC practical in real environments and also 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 into these definitions (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.

We note that the construction presented in this paper (and the security notions that we can achieve) has been geared towards using existing primitives from the literature to illustrate that an instantiation of RPVC is practical. However, using stronger primitives will enable correspondingly stronger security reductions, and thus finding such constructions (such as a fully secure revocable KP-ABE scheme with multiple challenge messages) is useful future work.

Acknowledgements

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 Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

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. Cryptology ePrint Archive, Report 2014/224, 2014. `http://eprint.iacr.org/`.

[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, V. Goyal, A. Jain, and A. Sahai. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727, 2013. `http://eprint.iacr.org/`.

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

[12] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

[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

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.

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:

- $\mathsf{ABE.Setup}(1^\lambda) \rightarrow (\mathrm{PP}, \mathrm{MK})$: a randomized algorithm that takes a security parameter as input and outputs a master key MK and public parameters PP
- $\mathsf{ABE.Encrypt}(m, A, \mathrm{PP}) \rightarrow CT$: 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
- $\mathsf{ABE.KeyGen}(\mathbb{A}, \mathrm{MK}, \mathrm{PP}) \rightarrow SK_{\mathbb{A}}$: 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 $SK_{\mathbb{A}}$
- $\mathsf{ABE.Decrypt}(CT, SK, \mathrm{PP}) \rightarrow m$ or $\perp$: 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.

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:

- $\mathsf{ABE.Setup}(1^\lambda, \mathcal{U}) \to (\mathrm{PP}, \mathrm{MK})$: This randomised algorithm takes the security parameter and the universe of attributes as input and outputs public parameters PP and master secret key MK.
- $\mathsf{ABE.Encrypt}(t, A, m, \mathrm{PP}) \to CT$: 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.
- $\mathsf{ABE.KeyGen}(\mathsf{id}, \mathbb{A}, \mathrm{MK}, \mathrm{PP}) \to \mathsf{SK}_{\mathsf{id},\mathbb{A}}$: 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.
- $\mathsf{ABE.KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP}) \to UK_{R,t}$: 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 $UK_{R,t}$.

- $\mathsf{ABE.Decrypt}(CT, SK_{\mathsf{id},\mathbb{A}}, \mathrm{PP}, UK_{R,t}) \rightarrow m$ or $\perp$: 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*

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

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 4.

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 τ from which the current time period t (be that a literal time value or counter etc.) may be sampled as $t \leftarrow \tau$.

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 6 [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 $\tilde{R}$ 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 1 and 2. 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 $\tilde{R}$. 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:*

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

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

Game 6 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}[\mathcal{ABE}, 1^\lambda]$:

1: $(t^\star, x^\star) \leftarrow \mathcal{A}(1^\lambda)$;
2: $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\lambda)$;
3: $\tilde{R} \leftarrow \mathcal{A}(\mathrm{PP})$;
4: $(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,\mathrm{MK},\mathrm{PP}),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,\mathrm{MK},\mathrm{PP}))}(\tilde{R}, \mathrm{PP})$;
5: $b \xleftarrow{\$} \{0,1\}$;
6: $CT^\star \leftarrow \mathsf{Encrypt}(t^\star, x^\star, m_b, \mathrm{PP})$;
7: $b^\star \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}(\cdot,\cdot,\mathrm{MK},\mathrm{PP}),\mathcal{O}^{\mathsf{KeyUpdate}}(\cdot,\cdot,\mathrm{MK},\mathrm{PP}))}(\tilde{R}, \mathrm{PP})$;
8: If $b' = b$
9: Return 1
10: Else Return 0

Oracle Query 1 $\mathcal{O}^{\mathsf{KeyGen}}(\mathsf{id}, \mathbb{A}, \mathrm{MK}, \mathrm{PP})$:

1: **if** $x^\star \in \mathbb{A}$ **then**
2: **if** $\mathsf{id} \notin \tilde{\mathsf{R}}$ **then**
3: **return** $\perp$
4: $SK_{\mathsf{id},\mathbb{A}} \leftarrow \mathsf{KeyGen}(\mathsf{id}, \mathbb{A}, \mathrm{MK}, \mathrm{PP})$
5: **return** $SK_{\mathsf{id},\mathbb{A}}$

Oracle Query 2 $\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, \mathrm{MK}, \mathrm{PP})$:

1: **if** $t = t^\star$ **then**
2: **if** $\tilde{R} \not\subseteq R$ **then**
3: **return** $\perp$
4: $UK_{R,t} \leftarrow \mathsf{KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP})$
5: **return** $UK_{R,t}$

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 $\mathsf{Sig.KeyGen}$, $\mathsf{Sig.Sign}$ and $\mathsf{Sig.Verify}$ defined as follows [11]:

- $\mathsf{Sig.KeyGen}(1^\lambda) \rightarrow (SK, VK)$: The probabilistic KeyGen algorithm takes as input the security parameter and generates a signing key SK and a verification key VK.
- $\mathsf{Sig.Sign}(m, SK) \rightarrow \gamma$: The probabilistic Sign algorithm takes as input a message to be signed and the signing key, and outputs a signature γ of m.
- $\mathsf{Sig.Verify}(m, \gamma, VK) \rightarrow \mathsf{accept}$ or reject: 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* $\mathsf{Sig.KeyGen}(1^\lambda)$ *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

Game 7 $\mathbf{Exp}_{\mathcal{A}}^{\text{EUF-CMA}}\left[\mathsf{Sig}, 1^\lambda\right]$:

1: Initialise $Q = \epsilon$ to be an empty list
2: $(SK, VK) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
3: $(m^\star, \gamma^\star) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sig.Sign}}(\cdot, SK)}(VK)$
4: If $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}(m^\star, \gamma^\star, VK)$ and $m^\star \notin Q$
5: Return 1
6: Else Return 0

Oracle Query 3 $\mathcal{O}^{\mathsf{Sig.Sign}}(m, SK)$:

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

Game 8 $\mathbf{Exp}_{\mathcal{A}}^{Invert}\left[g, 1^\lambda\right]$:

1: $w \leftarrow \{0,1\}^\lambda$
2: $z = g(w)$
3: $w' \leftarrow \mathcal{A}(1^\lambda, z)$
4: If $g(w') = z$
5: Return 1
6: Else Return 0

of its choice, cannot create a message $m^\star$ with a valid signature where $m^\star$ was not one of the messages that it saw a signature for. More formally, this is defined in Game 7 where $\mathcal{A}$ has access to a Sig.Sign oracle which is handled by the algorithm given in Oracle Query 3.

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

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

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

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 8) where we consider the experiment for any algorithm $\mathcal{A}$, any value λ for the security parameter, and the function $g\colon \{0,1\}^\star \to \{0,1\}^\star$. 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 [11].

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*

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

Table 1: PVC using KP-ABE

Abstract PVC parameter	**Parameter in KP-ABE instantiation**
EK_F	$SK_{\mathbb{A}_F}$
PK_F	Master public key PP
σ_x	Encryption of m using PP and A_x
σ_y	m or $\perp$
$VK_{F,x}$	$g(m)$

PVC using Key-Policy Attribute-based Encryption.

Parno et al. [14] provide a concrete instantiation of PVC using KP-ABE[7] 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 in which 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 function $F : \{0,1\}^n \to \{0,1\}$, the set $\{x \in \{0,1\}^n : F(x) = 1\}$ defines a monotonic access structure which we denote $\mathbb{A}_F$.

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.

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