Hybrid Publicly Verifiable Computation

James Alderman, Christian Janson, Carlos Cid, and Jason Crampton

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

Abstract

Publicly Verifiable Outsourced Computation (PVC) allows weak devices to delegate computations to more powerful servers, and to verify the correctness of results. Delegation and verification rely only on public parameters, and thus PVC lends itself to large multi-user systems where entities need not be registered. In such settings, individual user requirements may be diverse and cannot be realised with current PVC solutions. In this paper, we introduce *Hybrid PVC* (HPVC) which, with a single setup stage, provides a flexible solution to outsourced computation supporting multiple modes: (i) standard PVC, (ii) PVC with cryptographically enforced access control policies restricting the servers that may perform a given computation, and (iii) a reversed model of PVC which we call *Verifiable Delegable Computation* (VDC) where data is held remotely by servers. Entities may dynamically play the role of delegators or servers as required.

Keywords Publicly Verifiable Computation, Outsourced Computation, Dual-Policy Attribute-based Encryption, Revocation, Access Control

1 Introduction

The trend towards cloud computing means that there is a growing trust dependency on remote servers and the functionality they provide. *Publicly Verifiable Computation* (PVC) [23] allows *any* entity to use public information to delegate or verify computations, and lends itself to large multi-user systems that are likely to arise in practice (as delegators need not be individually registered).

However, in such a system, the individual user requirements may be diverse and require different forms of outsourced computation, whereas current PVC schemes support only a single form. Clients may wish to request computations from a particular server or to issue a request to a large pool of servers; in the latter case, they may wish to restrict the servers that can perform the computation to only those possessing certain characteristics. Moreover, the data may be provided by the client as part of the computation, or it may be stored by the server; and the role of servers and clients may be interchangeable depending on the context.

Consider the following scenarios: (i) employees with limited resources (e.g. using mobile devices when out of the office) need to delegate computations to more powerful servers. The workload of the employee may also involve responding to computation requests to perform tasks for other employees or to respond to inter-departmental queries over restricted databases. (ii) Entities that invest heavily in outsourced computations could find themselves with a valuable, processed dataset that is of interest to other parties, and hence want to to selectively share this information by allowing others to query the dataset in a verifiable fashion. (iii) database servers

that allow public queries may become overwhelmed with requests, and need to enlist additional servers to help (essentially the server acts as a delegator to outsource queries with relevant data). Finally, (iv) consider a form of peer-to-peer network for sharing computational resources – as individual resource availability varies, entities can sell spare resources to perform computations for other users or make their own data available to others, whilst making computation requests to other entities when resources run low.

Current PVC solutions do not handle these flexible requirements particularly well; although there are several different proposals in the literature that realise some of the requirements described above, each requires an independent (potentially expensive) setup stage. We introduce *Hybrid PVC* (HPVC) which is a single mechanism (with the associated costs of a single setup operation and a single set of system parameters to publish and maintain) which simultaneously satisfies all of the above requirements. Entities may play the role of both delegators and servers, in the following modes of operation, dynamically as required:

- **Revocable PVC** (RPVC) where clients with limited resources outsource computations on data of their choosing to more powerful, untrusted servers using only public information. Multiple servers can compute multiple functions. Servers may try to cheat to persuade verifiers of incorrect information or to avoid using their own resources. Misbehaving servers can be detected and revoked so that further results will be rejected and they will not be rewarded for their effort;
- **RPVC with access control** (RPVC-AC) which restricts the servers that may perform a given computation. Outsourced computations may be distributed amongst a pool of available servers that are not individually authenticated and known by the delegator. Prior work [1] used symmetric primitives and required all entities to be registered in the system (including delegators) but we achieve a fully public system where only servers need be registered (as usual in PVC);
- **Verifiable Delegable Computation** (VDC) where servers are the data owners and make a static dataset available for verifiable querying. Clients request computations on subsets of the dataset using public, descriptive labels.

We begin, in Section 2, with a summary of related work and the KP-ABE-based PVC schemes [2, 23] on which we base our HPVC construction. In Section 3, we define the generic functionality and security properties of HPVC. We then, in Section 4.1, discuss each supported mode of computation, and how it fits our generic definition. To support user revocation [2], we introduce a new cryptographic primitive called Revocable-Key Dual-policy Attribute-based Encryption (rkDPABE) in Section 4.2, and finally, in Section 4.3, we instantiate HPVC using rkDPABE. Additional details, formal security games and proofs can be found in the appendix and will be included in the full version online.

2 Background and Related Work

Verifiable computation [10, 12, 13, 16, 17, 23, 27] may be seen as a protocol between a (weak) client C and a server S, resulting in the provably correct computation of $F(x)$ by the server for the client's choice of F and x. The setup stage may be computationally expensive (amortised over multiple computations) but other operations should be efficient for the client. Some prior work used garbled circuits with fully homomorphic encryption [13, 17] or targeted specific functions [10, 12, 16]. Chung et al. [14] introduced *memory delegation* which is similar to VDC; a client uploads his memory to a server who can update and compute a function F over the entire memory. Backes et al. [8] consider a client that outsources data and requests computations on a data portion. The client can efficiently verify the correctness of the result without holding the input data. Most work requires the client to know the data in order to verify [9, 11, 18, 22].

Verifiable oblivious storage [3] ensures data confidentiality, access pattern privacy, integrity and freshness of data accesses. Work on authenticated data lends itself to verifiable outsourced computations, albeit for specific functions only. Backes et al. [7] use privacy-preserving proofs over authenticated data outsourced by a trusted client. Similar results are presented in [25] using public logs. It is notable that [7] and [11] achieve public verifiability. In independent and concurrent work, Shi et al. [24] use DP-ABE to combine keyword search on encrypted data with the enforcement of an access control policy.

Parno et al. [23] introduce *Publicly Verifiable Computation* (PVC) where multiple clients outsource computations of a single function to a single server, and verify the results. Alderman et al. [2] introduce a trusted Key Distribution Center (KDC) to handle the expensive setup for all entities, to allow multiple servers to compute multiple functions, and to revoke misbehaving servers. Informally, the KDC acts as the root of trust to generate public parameters and delegation information, and to issue secret keys and evaluation keys to servers. To outsource the evaluation of $F(x)$, a delegator sends an encoded input $\sigma_{F(x)}$ to a server S, and publishes verification tokens. S uses an evaluation key for F to produce an encoded output $\theta_{F(x)}$. *Any* entity can verify correctness of $\theta_{F(x)}$, but only entities in possession of a retrieval key created by the delegator can learn the value of $F(x)$. If S cheated they may be reported to the KDC for revocation.

The constructions of [2, 23] to outsource a Boolean function, F, are based on Key-policy Attribute-based encryption (KP-ABE), which links ciphertexts with attribute sets and decryption keys with a policy; decryption only succeeds if the attributes satisfy the policy. For PVC, two random messages are encrypted and linked to the input data X (represented as attributes) to form the encoded input. The evaluation key is a pair of decryption keys linked to F and $\overline{F}$ (the complement function of F). Exactly *one* message can be recovered, implying whether F or $\overline{F}$ was satisfied, and hence if $F(X) = 1$ or 0. Ciphertext indistinguishability ensures S cannot return the other message to imply an incorrect result.

3 Hybrid Publicly Verifiable Computation

To accommodate different modes of computation, we define HPVC generically in terms of parameters ω, $\mathbb{O}$, ψ and $\mathbb{S}$. Depending on the mode (and which party provides the input data), $\mathbb{O}$ or $\mathbb{S}$ will encode functions, while ω or ψ encode input data, as detailed in Section 4.1. We retain the single, trusted key distribution centre (KDC) from RPVC [2] who initialises the system for a function family $\mathcal{F}$ resulting in a set of public parameters PP and a master secret key. For each function $F \in \mathcal{F}$, the KDC publishes a delegation key PK_F. It also registers each entity S_i that wants to act as a server by issuing a signing key SK_{S_i}. It may also update PP during any algorithm to reflect changes in the user population.

Depending on the mode, servers either compute functions $\mathbb{O}$ on behalf of clients, or make a dataset ψ available for public querying. The Certify algorithm is run by the KDC to produce an evaluation key $EK_{(\mathbb{O},\psi),S_i}$ enabling S_i to perform these operations. S_i chooses a set of labels L_i – in RPVC or RPVC-AC modes, L_i uniquely represents the function F that S_i should be certified to compute; in VDC mode, L_i is a *set* of labels, each uniquely representing a data point contained in the dataset D_i owned by S_i.[1] In the VDC setting, the server is the data owner and so S_i also provides a list $\mathcal{F}_i$ advertising the functions that he is willing to evaluate on his data in accordance with his own data usage policies; in RPVC settings, $\mathcal{F}_i$ advertises the functions S_i is certified to compute.

[1] These descriptive labels (e.g. field names in a database) allow delegators to select data points to be used in a computation *without* knowing the data values.

To request a computation of $F(X)$ (encoded in ω or $\mathbb{S}$) from S_i, a delegator uses public information to run $\mathsf{ProbGen}$. He provides labels $L_{F,X} \subseteq L_i$ describing the computation: in RPVC or RPVC-AC modes, the delegator provides the input data X and $L_{F,X}$ labels the function F to be applied; in VDC mode, the client uses the descriptive labels to choose a subset of data points $X \subseteq D_i, X \subseteq \mathsf{Dom}(F)$ held by S_i that should be computed on. $\mathsf{ProbGen}$ generates an encoded input $\sigma_{F,X}$, a public verification key $VK_{F,X}$ and an output retrieval key $RK_{F,X}$.

A server combines $\sigma_{F,X}$ with its evaluation key to compute $\theta_{F(X)}$ encoding the result $F(X)$. Any entity can perform blind verification using $VK_{F,X}$ to verify correctness of $\theta_{F(X)}$ *without* learning the value $F(X)$. It generates an output retrieval token $RT_{F(X)}$ and a token $\tau_{F(X)}$ which is sent to the KDC if verification failed; the server is then revoked from performing further evaluations. This prevents delegators wasting their (limited) resources outsourcing to a server known to be untrustworthy, and also acts as a deterrent, especially when servers are rewarded per computation. If the result is valid, $RT_{F(X)}$ can be used in the $\mathsf{Retrieve}$ algorithm with the retrieval key $RK_{F,X}$ to reveal $y_{F(X)} = F(X)$.

Definition 1. A *Hybrid Publicly Verifiable Computation (HPVC) scheme* for a family of functions $\mathcal{F}$ comprises the following algorithms:

1. $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$: Run by the KDC to establish public parameters PP and a master secret key MK for the system. The inputs are the security parameter and the family of functions $\mathcal{F}$ that may be computed;

2. $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$: Run by the KDC to generate a public delegation key, PK_F, allowing entities to outsource, or request, computations of F;

3. $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to enrol an entity S_i within the system to act as a server. It generates a personalised signing key SK_{S_i};

4. $EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate an evaluation key $EK_{(\mathbb{O},\psi),S_i}$ enabling the server S_i to compute on the pair $(\mathbb{O}, \psi)$. The algorithm also takes as input the mode in which it should operate, a set of labels L_i and a set of functions $\mathcal{F}_i$;

5. $(\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, PK_F, \mathrm{PP})$: run by an entity to request a computation of $F(X)$ from S_i. The inputs are the mode, the pair $(\omega, \mathbb{S})$ representing the computation, a set of labels $L_{F,X} \subseteq L_i$, the delegation key for F and the public parameters. The outputs are an encoded input $\sigma_{F,X}$, a verification key $VK_{F,X}$ and an output retrieval key $RK_{F,X}$;

6. $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \mathrm{PP})$: run by an entity S_i to compute $F(X)$. The inputs are the mode, an encoded input, and an evaluation key and signing key for S_i. The output, $\theta_{F(X)}$, encodes the result;

7. $(RT_{F(X)}, \tau_{F(X)}) \leftarrow \mathsf{BVerif}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$: run by any entity. The inputs are an encoded output produced by S_i and verification key; the outputs are a retrieval token $RT_{F(X)}$ encoding the computation result, and a token $\tau_{F(X)}$ which is (accept, S_i) if $\theta_{F(X)}$ is correct, or (reject, S_i) otherwise;

8. $y \leftarrow \mathsf{Retrieve}(RT_{F(X)}, \tau_{F(X)}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$: run by any entity holding the retrieval key for $F(X)$ using the outputs from BVerif and the verification key. It outputs $y = F(X)$ if the result was computed correctly, or else $y = \perp$;

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

$(\omega^{\star}, \mathbb{O}^{\star}, \psi^{\star}, \mathbb{S}^{\star}, L_{F,X^{\star}}, \texttt{mode}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{F})$
$(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
if $(\texttt{mode} = VDC)$ **then** $(F \leftarrow \mathbb{S}^{\star}, X^{\star} \leftarrow \psi^{\star})$
else $(F \leftarrow \mathbb{O}^{\star}, X^{\star} \leftarrow \omega^{\star})$
$PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
$(\sigma^{\star}, VK^{\star}, RK^{\star}) \xleftarrow{\$} \mathsf{ProbGen}(\texttt{mode}, (\omega^{\star}, \mathbb{S}^{\star}), L_{F,X^{\star}}, PK_F, \text{PP})$
$\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^{\star}, VK^{\star}, RK^{\star}, PK_F, \text{PP})$
$(RT_{\theta^{\star}}, \tau_{\theta^{\star}}) \leftarrow \mathsf{BVerif}(\theta^{\star}, VK^{\star}, \text{PP})$
$y \leftarrow \mathsf{Retrieve}(RT_{\theta^{\star}}, \tau_{\theta^{\star}}, VK^{\star}, RK^{\star}, \text{PP})$
if $(((y, \tau_{\theta^{\star}}) \neq (\perp, (\mathsf{reject}, \cdot))))$ **and** $(y \neq F(X^{\star})))$ **then**
  return 1
else return 0

9. $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{F(X)}, \text{MK}, \text{PP})$: run by the KDC if a misbehaving server is reported. It returns $UM = \perp$ if $\tau_{F(X)} = (\mathsf{accept}, S_i)$. Otherwise, all evaluation keys $EK_{(\cdot,\cdot),S_i}$ for S_i are rendered non-functional and the update material UM is a set of updated evaluation keys $\{EK_{(\mathbb{O},\psi),S'}\}$ for all servers.

Definition 2. *A hybrid publicly verifiable outsourced computation* (HPVC) *scheme is* correct *for a family of functions $\mathcal{F}$ if, for all ω, ψ, $\mathbb{O}$, $\mathbb{S}$ defined for a computation $F(X)$, for $F \in \mathcal{F}$ and $X \in \mathsf{Dom}(F)$, as described in Table 1, if $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$ then:*

$$
\begin{aligned}
&\Pr[(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F}),\\
&\quad PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP}),\\
&\quad SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \text{MK}, \text{PP}),\\
&\quad EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP}),\\
&\quad (\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(\texttt{mode}, (\omega, \mathbb{S}), L_{F,X}, PK_F, \text{PP}),\\
&\quad \theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\texttt{mode}, \sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \text{PP}),\\
&\quad (RT_{F(X)}, \tau_{F(X)}) \leftarrow \mathsf{BVerif}(\theta_{F(X)}, VK_{F,X}, \text{PP}),\\
&\quad F(X) \leftarrow \mathsf{Retrieve}(RT_{F(X)}, \tau_{F(X)}, VK_{F,X}, RK_{F,X}, \text{PP})]\\
&\quad = 1 - \text{negl}(\ell).
\end{aligned}
$$

3.1 Security Models

We now discuss desirable security properties for HPVC; additional formal models are found in Appendix B[2]. Public verifiability, revocation and authorised computation are selective notions in line with our rkDPABE scheme in Appendix D.

3.1.1 Public Verifiability

Public Verifiability presented in Game 1, ensures that a server that returns an incorrect result is detected by the verification algorithm so that they can be reported for revocation. The adversary, $\mathcal{A}$, may corrupt other servers, generate arbitrary computations, and perform verification steps himself. $\mathcal{A}$ first selects its challenge parameters, including the mode it wishes its challenge

[2]We don't consider input privacy here, but note that a revocable dual-policy predicate encryption scheme, if found, could easily replace our ABE scheme in Section 4.3. Security against vindictive servers and managers can also be adapted from [2].

to be generated in and the labels associated to its choice of inputs. We ask $\mathcal{A}$ to choose $\mathbb{O}^\star$ and $\psi^\star$, despite the challenge inputs being only $\omega^\star$ and $\mathbb{S}^\star$. This allows us to define the challenge in terms of F and $X^\star$ on line 3; note that $\mathbb{O}^\star$ and $\psi^\star$ can also be gleaned from the mode and labels, so this does not weaken the game – the adversary has already determined these values through its choices.

The challenger runs Setup and FnInit for the chosen function F. It then runs $\mathsf{ProbGen}$ to create the challenge parameters for the adversary, which are given to $\mathcal{A}$ along with the public information. The adversary is also given oracle access to the functions $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Revoke}(\cdot, \mathrm{MK}, \mathrm{PP})$, denoted by $\mathcal{O}$. $\mathcal{A}$ wins the game if it creates an encoded output that verifies correctly yet does not encode the correct value $F(x)$.

Definition 3. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* sPubVerif *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

$\mathcal{HPVC}$ is secure with respect to selective public verifiability *if, for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}^{\text{sPubVerif}}(\mathcal{HPVC}, 1^\ell, \mathcal{F})$ is negligible in ℓ.*

3.1.2 Other Security Models

In Appendix B we discuss the following security models.

- **Blind Verification** ensures that a verifier that does not hold the output retrieval key $RK^\star$ cannot learn the value of $F(X)$ for the adversary's choice of F and a randomly chosen input X from the domain of F.
- **Revocation** ensures that a server that has been detected as misbehaving cannot produce a result (even a correct result) that is accepted by a verifier – thus, the server cannot be rewarded for future work. To reflect the revocation mechanism of the rkDPABE primitive, we include a semi-static restriction whereby a list of entities to be revoked at the time of the challenge computation must be declared before the adversary receives oracle access[3].
- **Authorised Computation** extends the model of [1] to the public-key setting to ensure that an unauthorised server cannot produce acceptable results.

4 Instantiating HPVC

We construct an HPVC scheme for the class NC^1, which includes common arithmetic and matrix operations. Let $\mathcal{F}$ be the family of Boolean formulas closed under complement using a revocable key dual-policy ABE in a black-box manner. We construct our scheme from a novel use of Dual-policy Attribute-based Encryption (DP-ABE) which combines KP-ABE and Ciphertext-policy ABE (CP-ABE). Decryption keys are linked to an "objective" policy $\mathbb{O}$ and "subjective" attribute set ψ, and ciphertexts linked to an "objective" attribute set ω and "subjective" policy $\mathbb{S}$; decryption requires both policies to be satisfied – $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$.

Following [23], we encrypt two random messages to form the encoded input, while decryption keys form evaluation keys; by linking these to F, $\overline{F}$ and X according to the mode, we ensure that exactly *one* message can be recovered, implying whether F or $\overline{F}$ was satisfied, and hence if $F(X) = 1$ or 0. DP-ABE security ensures a server cannot learn a message implying an invalid result.

[3] This restriction was also used in [4] for revocable KP-ABE, and could be removed if an adaptively, indirectly revocable ABE scheme is found.

Table 1: Parameter definitions for different modes

mode	$\mathbb{O}$	ψ	ω	$\mathbb{S}$
RPVC	F	$\{T_S\}$	X	$\{\{T_S\}\}$
RPVC-AC	F	s	X	P
VDC	$\{\{T_O\}\}$	D_i	$\{T_O\}$	F

mode	L_i	$L_{F,X}$	$\mathcal{F}_i$
RPVC	$\{l(F)\}$	$\{l(F)\}$	$\{F\}$
RPVC-AC	$\{l(F)\}$	$\{l(F)\}$	$\{F\}$
VDC	$\{l(x_{i,j})\}_{x_{i,j}\in D_i}$	$\{l(x_{i,j})\}_{x_{i,j}\in X}$	$\{(F, \{l(x_{i,j})\}_{x_{i,j}\in \mathsf{Dom}(F)})\}_{F\in\mathcal{F}}$

The values of ω, $\mathbb{O}$, ψ and $\mathbb{S}$ depend upon the mode, as detailed in Table 1. Two additional parameters T_O and T_S "disable" modes when not required. Note that, trivially, $\psi \in \mathbb{S}$ when $\psi = \{T_S\}$ and $\mathbb{S} = \{\{T_S\}\}$, and similarly for T_O.

4.1 Supporting Different Modes

4.1.1 RPVC

In this mode, a delegator owns some input data X and wants to learn $F(X)$ but lacks the computational resources to do so itself; thus, the computation is outsourced. In this setting, only the parameters $\mathbb{O}$ and ω are required, and are set to be F and X respectively. The set X comprises a single datapoint: the input data to this particular computation. The remaining parameters $\mathbb{S}$ and ψ are defined in terms of the dummy parameter T_S. The set of functions $\mathcal{F}_i$ that a server is certified for during a single $\mathsf{Certify}$ operation is simply F, and the sets of labels L_i and $L_{F,X}$ both comprise a single element $l(F)$ uniquely labelling F.

4.1.2 RPVC-AC

RPVC-AC [1] was introduced with the motivation that servers may be chosen from a pool based on resource availability, a bidding process etc. Delegators may not have previously authenticated the selected server, in contrast to prior models [23] where a client set up a PVC system with a single, known server.

The construction of [1] used a symmetric key assignment scheme allowing only authorised entities to derive the required keys. However, the KDC had to register all delegators and verifiers. This was due both to the policies being enforced (e.g. to restrict the computations delegators may outsource), and to the use of symmetric primitives – to encrypt inputsthat only authorised servers can decrypt, delegators must know the secret symmetric key. Thus, the scheme is not strictly *publicly delegable* as delegation does not depend *only* on public information, and similarly for verification.

We retain public delegability and verifiability whilst restricting the servers that may perform a given computation. In some sense, servers are already authorised for functions by being issued evaluation keys. However, we believe that access control policies in this setting must consider additional factors than just functions. The semantic meaning and sensitivity of input data may affect the policy, or servers may need to possess specific resources or characteristics, or be geographically nearby to minimise latency. E.g. a government contractor may, due to the nature of its work, require servers to be within the same country.

One solution could be for the KDC to issue signed attributes to each server who attaches the required signatures to computation results for verification. In this case, a verifier must decide if the received attributes are sufficient. We consider the delegator that runs ProbGen to "own" the computation and, as such, it should specify the authorisation policy that a server must meet. As this is a publicly verifiable setting, any entity can verify and we believe (i) verifiers should not accept a result that the delegator itself would not accept, and (ii) it may be unreasonable to expect verifiers to have sufficient knowledge to determine the authorisation policy. Of course, the delegator could attach a signed authorisation policy to the verification key, but verifiers are not obliged to adhere to this policy and doing so creates additional work for the verifier – one of the key efficiency requirements for PVC is that verification is very cheap. Using DP-ABE to instantiate HPVC allows the delegator to specify the authorisation policy during ProbGen and requires no additional work on the part of the verifier compared to standard RPVC. Furthermore, an unauthorised server cannot actually perform the computation and hence verification will always fail.

We use the objective parameters ω and $\mathbb{O}$ to compute (as for RPVC) whilst the subjective parameters ψ and $\mathbb{S}$ enforce access control on the server. Servers are assigned both an evaluation key for a function F *and* a set of descriptive attributes describing their authorisation rights, $s \subseteq \mathcal{U}_S$, where $\mathcal{U}_S$ is a universe of attributes used solely to define authorisation. ProbGen operates on both the input data X *and* an authorisation policy $P \subseteq 2^{\mathcal{U}_S} \setminus \{\emptyset\}$ which defines the permissible sets of authorisation attributes to perform this computation. Servers may produce valid, acceptable outputs only if $s \in P$ i.e. they satisfy the authorisation policy. E.g. $P = (\texttt{Country} = \texttt{UK}) \vee ((\texttt{clearance} = \texttt{Secret}) \wedge (\texttt{Country} = \texttt{USA}))$ is satisfied by $s = \{\texttt{Country} = \texttt{UK}, \texttt{Capacity} = \texttt{3TB}\}$.

4.1.3 VDC

VDC reverses the role of the data owner – a server owns a static database and enables delegators to request computations/queries over the data. Hence, the user relationship is more akin to the traditional client-server model compared to PVC. Delegators learn nothing more than the result of the computation, and do not need the input data in order to verify. The *efficiency* requirement for VDC is also very different from PVC: outsourcing a computation is not merely an attempt to gain efficiency as the delegator never possesses the input data and so cannot execute the computation himself (even with the necessary resources). Thus, VDC does not have the stringent efficiency requirement present in PVC (that outsourcing and verifying computations be more efficient than performing the computation itself, for outsourcing to be worthwhile). Our solution behaves reasonably well, achieving constant time verification; the size of the query depends on the function F, while the size of the server's response depends only on the size of the result itself and *not* on the input size which may be large, particularly when querying remote databases. Future work in this area should focus on reducing the cost of outsourcing computations.

In VDC, each entity S_i that wants to act as a server owns a dataset $D_i = \{x_{i,j}\}_{j=1}^{m_i}$ comprising m_i data points. The KDC issues a single evaluation key EK_{D_i,S_i} enabling S_i to compute on subsets of D_i. S_i publishes a list L_i comprising a unique label $l(x_{i,j}) \in L_i$ for each data point $x_{i,j} \in D_i$, and a list of functions $\mathcal{F}_i \subseteq \mathcal{F}$ that are (i) meaningful on their dataset, and (ii) permissible according to their own access control policies. Furthermore, not all data points $x_{i,j} \in D_i$ may be appropriate for each function e.g. only numeric data should be input to an averaging function. $\mathcal{F}_i$ comprises elements $(F, \bigcup_{x_{i,j} \in \mathsf{Dom}(F)} l(x_{i,j}))$ describing each function and the associated permissible inputs. Labels should *not* reveal the data values themselves to preserve the confidentiality of D_i.

Table 2: Example database

User ID	Name	Age	Height
001	Alice	26	165
002	Bob	22	172

Table 3: Example list $\mathcal{F}_i$

F	Dom(F)
Average	Age of record 1, Height of record 1, Age of record 2, Height of record 2
Most common value	Name of record 1, Age of record 1, Height of record 1, Name of record 2, Age of record 2, Height of record 2

Delegators may select servers and data using *only* these labels e.g. they may ask S_i to compute $F(X)$ for any function $F \in \mathcal{F}_i$ on a set of data points $X \subseteq \mathsf{Dom}(F)$[4] by specifying labels $\{l(x_{i,j})\}_{x_{i,j} \in X}$. Although it may be tempting to suggest that S_i simply caches the results of computing each $F \in \mathcal{F}_i$, the number of input sets $X \subseteq \mathsf{Dom}(F)$ could be large, making this an unattractive solution.

As an example, consider a server S_i that owns the database in Table 2. The dataset D_i represents this as a set of field values for each record in turn: $D_i = \{001, \text{Alice}, 26, 165, 002, \text{Bob}, 22, 172\}$. S_i publishes data labels $L_i = \{$User ID of record 1, Name of record 1, Age of record 1, Height of record 1, User ID of record 2, Name of record 2, Age of record 2, Height of record 2$\}$. $\mathcal{F}_i$, in Table 3, lists the functions and domains that S_i willing to compute. To find the average age, a delegator queries "Average" on input $X = \{$Age of record 1, Age of record 2$\}$. Example applications are found in Appendix A.

4.2 Revocable Dual-policy Attribute-based Encryption

Before instantiating HPVC, we first introduce a new cryptographic primitive which forms the basic building-block of our construction. If revocation is not required then a standard DP-ABE scheme can be used.

Definition 4. A *Revocable Key Dual-policy Attribute-based Encryption scheme* (rkDPABE) comprises five algorithms:

- $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{U})$: takes the security parameter and attribute universe and generates public parameters PP and a master secret key MK;
- $CT_{(\omega,\mathbb{S}),t} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, \text{PP})$: takes as input a message to be encrypted, an objective attribute set ω, a subjective policy $\mathbb{S}$, a time period t and the public parameters. It outputs a ciphertext that is valid for time t;
- $SK_{(\mathbb{O},\psi),\text{ID}} \xleftarrow{\$} \mathsf{KeyGen}(\text{ID}, (\mathbb{O}, \psi), \text{MK}, \text{PP})$: takes an identity ID, an objective access structure $\mathbb{O}$, a subjective attribute set ψ, the master secret key and the public parameters. It outputs a secret decryption key $SK_{(\mathbb{O},\psi),\text{ID}}$;
- $UK_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, \text{MK}, \text{PP})$: takes a revocation list R containing all revoked identities, the current time period, the master secret key and public parameters. It outputs

[4] In contrast to prior modes where X was a single data point, F now takes $|X|$ inputs.

updated key material $UK_{R,t}$ which makes the decryption keys $SK_{(\mathbb{O},\psi),\mathrm{ID}}$, for all non-revoked identities $\mathrm{ID} \notin \mathrm{R}$, functional to decrypt ciphertexts encrypted for the time t.

- $PT \leftarrow \mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega,\mathbb{S}), SK_{(\mathbb{O},\psi),\mathrm{ID}}, (\mathbb{O},\psi), UK_{R,t}, \mathrm{PP})$: takes as input a ciphertext formed for the time period t and the associated pair $(\omega,\mathbb{S})$, a decryption key for entity ID and the associated pair $(\mathbb{O},\psi)$, an update key for the time t and the public parameters. It outputs a plaintext PT which is the encrypted message m, if and only if the objective attributes ω satisfies the objective access structure $\mathbb{O}$ *and* the subjective attributes ψ satisfies the subjective policy $\mathbb{S}$ *and* the value of t in the update key matches that specified during encryption. If not, PT is set to be a failure symbol $\perp$.

Definition 4 suffices to comprehend the remainder of this paper as we shall use an rkDPABE scheme in a black-box manner. For completeness, we give correctness and security definitions, a construction and a security proof in Appendix D.

4.3 Construction

Consider a function to be delegated $F : \{0,1\}^n \rightarrow \{0,1\}$ and its complement function $\overline{F} = F(x) \oplus 1$. As mentioned, we base our construction on rkDPABE by encoding inputs as attributes in a universe $\mathcal{U}_x$, and encoding Boolean functions as access structures over $\mathcal{U}_x$. Computations with n-bit outputs can be built from n Boolean functions returning each bit in turn. Negations can be handled by building rkDPABE from non-monotonic ABE [21] or, as here, by adding negated attributes to the universe [26]. For the i^{th} bit of a binary input string $X = x_1 \ldots x_n$, define attributes $A^0_{X,i}$ and $A^1_{X,i} \in \mathcal{U}_x$[5]; X is encoded as $A_X = \{A^j_{X,i} \in \mathcal{U}_x : x_i = j\}$.

Let $\mathcal{U}_l$ be a set of attributes (disjoint from $\mathcal{U}_x$) uniquely labelling each function and data item, and let $\mathcal{U}_{ID}$ represent server identities. Let g be a one-way function and $\mathcal{DPABE} = (\mathsf{DPABE.Setup}, \mathsf{DPABE.Encrypt}, \mathsf{DPABE.KeyGen}, \mathsf{DPABE.KeyUpdate}, \mathsf{DPABE.Decrypt})$ be a revocable key DP-ABE scheme for $\mathcal{F}$ with attribute universe $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{ID}$.

We initialise two independent DP-ABE systems over $\mathcal{U}$, and define four additional "dummy" attributes to disable modes: T^0_O, T^0_S for the first system, and T^1_O, T^1_S for the second. We denote the complement functions as follows: in RPVC and RPVC-AC, recall $\mathbb{O} = F$ and $\mathbb{S} = \{\{T^0_S\}\}$; we define $\overline{\mathbb{O}} = \overline{F}$ and $\bar{\mathbb{S}} = \{\{T^1_S\}\}$. Similarly, for VDC, $\overline{\mathbb{O}} = \{\{T^1_0\}\}$ and $\bar{\mathbb{S}} = \overline{F}$.

1. Setup initialises two rkDPABE schemes over $\mathcal{U}$, an empty two-dimensional array L_{Reg} to list registered entities, a list of revoked entities L_{Rev} and a time source $\mathbb{T}$ (e.g. a networked clock or counter) to index update keys)[6].

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

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

2. FnInit sets the public delegation key PK_F (for all functions F) to be the public parameters for the system (since we use public key primitives).

[5]Either by defining a large enough $\mathcal{U}_x$ or by hashing strings to elements of the attribute group. Unlike prior schemes [2, 23], we include an identifier of the data X (based on the label $l(x_{i,j})$) in the attribute mapping to specify the data items to be used; alternatively, D_i could be a long bitstring formed by concatenating each data

Algorithm 2 $PK_F \xleftarrow{\$} \mathsf{HPVC.FnInit}(F, \mathrm{MK}, \mathrm{PP})$

1: $PK_F \leftarrow \mathrm{PP}$

3. Register runs a signature KeyGen algorithm and adds the verification key to $L_{Reg}[S_i][0]$. These prevent servers being impersonated and wrongly revoked.

Algorithm 3 $SK_{S_i} \xleftarrow{\$} \mathsf{HPVC.Register}(S_i, \mathrm{MK}, \mathrm{PP})$

1: $(SK_{\mathrm{Sig}}, VK_{\mathrm{Sig}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\ell)$
2: $SK_{S_i} \leftarrow SK_{\mathrm{Sig}}$
3: $L_{\mathrm{Reg}}[S_i][0] \leftarrow L_{\mathrm{Reg}}[S_i][0] \cup VK_{\mathrm{Sig}}$

4. Certify first adds an element $(F, \bigcup_{l \in L_i} l)$ to the list $L_{Reg}[S_i][1]$ for each $F \in \mathcal{F}_i$; this publicises the computations that S_i can perform (either functions in RPVC and RPVC-AC modes, or functions and data labels in VDC). The algorithm removes S_i from the revocation list, gets the current time from $\mathbb{T}$ and generates a decryption key for $(\mathbb{O}, \psi \cup \bigcup_{l \in L_i} l)$ in the first DP-ABE system. The additional attributes for the labels $l \in \mathcal{U}_l$ ensure that a key cannot be used to evaluate computations that do not correspond to these labels. In RPVC and RPVC-AC, this means that a key for a function G cannot evaluate a computation request for $F(X)$. In VDC, it means that an evaluation key must be issued for a dataset D_i that includes (at least) the specified input data $X^\star$. It is sufficient to include labels only on the subjective attribute set; as these labels are a security measure against a misbehaving server, we amend the servers key but need not take similar measures against the delegator. Delegators can then specify, in the subjective policy that they create, the labels that are required; these must be in the server's key for successful evaluation (decryption). The KDC should check that the label corresponds to the input to ensure that a server does not advertise data he does not own. It also generates an update key for the current time period to prove that S_i is not currently revoked. In RPVC mode, another pair of keys is generated using the second DP-ABE system for the complement inputs.

Algorithm 4 $EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{HPVC.Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

1: **for** $F \in \mathcal{F}_i$ **do**
2: $\quad L_{\mathrm{Reg}}[S_i][1] \leftarrow L_{\mathrm{Reg}}[S_i][1] \cup (F, \bigcup_{l \in L_i} l)$
3: $L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \setminus S_i$, $t \leftarrow \mathbb{T}$
4: $SK^0_{\mathrm{ABE}} \xleftarrow{\$} \mathsf{DPABE.KeyGen}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l \in L_i} l), MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
5: $UK^0_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}})$
6: **if** $(\mathtt{mode} = RPVC)$ **or** $(\mathtt{mode} = \mathrm{RPVC\text{-}AC})$ **then**
7: $\quad SK^1_{\mathrm{ABE}} \xleftarrow{\$} \mathsf{DPABE.KeyGen}(S_i, (\overline{\mathbb{O}}, A_\psi \cup \bigcup_{l \in L_i} l), MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
8: $\quad UK^1_{L_{\mathrm{Rev}},t} \xleftarrow{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, MSK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}})$
9: **else**
10: $\quad SK^1_{\mathrm{ABE}} \leftarrow \perp$, $UK^1_{L_{\mathrm{Rev}},t} \leftarrow \perp$
11: $EK_{(\mathbb{O},\psi),S_i} \leftarrow (SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$

5. ProbGen chooses two messages m_0 and m_1 randomly from the message space. A random bit b permutes the encoded input, so that a verifier without b cannot learn $F(X)$ as it cannot tell if a message was encrypted with F or $\overline{F}$. m_b is encrypted with $(A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l)$ in the first DPABE system (where A_ω is the attribute set encoding ω), whilst m_{1-b} is encrypted with the complement policy and either the first DPABE system for VDC or the second for RPVC

point, and the labels should identify the attributes corresponding to each data point.

[6]Our KDC will act as the trusted KeyGen authority already inherent in ABE schemes.

(the attributes remain the same as it is the same input data X in RPVC, or attribute T_O^0 in VDC). g is applied to each message to form the verification key (g being one-way allows the key to be published), and b forms the output retrieval key.

Algorithm 5 $(\sigma_{F,X}, VK_{F,X}, RK_{F,X}) \xleftarrow{\$} \mathsf{HPVC.ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, PK_F, \mathrm{PP})$

1: $(m_0, m_1) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
2: $b \xleftarrow{\$} \{0,1\}$, $t \leftarrow \mathbb{T}$
3: $c_b \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_b, (A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l), t, MPK^0_{\mathrm{ABE}})$
4: **if** $\mathtt{mode} = VDC$ **then** $c_{1-b} \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_{1-b}, (A_\omega, \bar{\mathbb{S}} \wedge \bigwedge_{l \in L_{F,X}} l), t, MPK^0_{\mathrm{ABE}})$
5: **else** $c_{1-b} \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_{1-b}, (A_\omega, \bar{\mathbb{S}} \wedge \bigwedge_{l \in L_{F,X}} l), t, MPK^1_{\mathrm{ABE}})$
6: **return** $\sigma_{F,X} \leftarrow (c_b, c_{1-b})$, $VK_{F,X} \leftarrow (g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $RK_{F,X} \leftarrow b$

6. Compute attempts to decrypt both ciphertexts, ensuring that different modes use the correct parameters. Decryption succeeds only if the function evaluates to 1 on the input data X i.e. the policy is satisfied. Since F and $\overline{F}$ output opposite results on X, exactly one plaintext will be a failure symbol $\perp$. The results are signed, along with the ID of the server S_i performing the computation.

Algorithm 6 $\theta_{F(X)} \xleftarrow{\$} \mathsf{HPVC.Compute}(\mathtt{mode}, \sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \mathrm{PP})$

1: Parse $EK_{(\mathbb{O},\psi),S_i}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$ and $\sigma_{F,X}$ as (c, c')
2: $d_b \leftarrow \mathsf{DPABE.Decrypt}(c, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
3: **if** $\mathtt{mode} = VDC$ **then** $d_{1-b} \leftarrow \mathsf{DPABE.Decrypt}(c', SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
4: **else** $d_{1-b} \leftarrow \mathsf{DPABE.Decrypt}(c', SK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, UK^1_{L_{\mathrm{Rev}},t})$
5: $\gamma \xleftarrow{\$} \mathsf{Sig.Sign}((d_b, d_{1-b}, S_i), SK_{S_i})$
6: $\theta_{(\omega,\mathbb{S}),(\mathbb{O},\psi)} \leftarrow (d_b, d_{1-b}, S_i, \gamma)$

7. BVerif verifies the signature using the verification key for S_i stored in L_{Reg}. If correct, it applies g to each plaintext in $\theta_{F(X)}$ and compares the results to the components of the verification key. If either comparison results in a match (i.e. the server successfully recovered a message), that plaintext is returned as the retrieval token and the output token is accept. Otherwise the result is rejected.

Algorithm 7 $(RT_{F(X)}, \tau_{F(X)}) \leftarrow \mathsf{HPVC.BVerif}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$

1: Parse $VK_{F,X}$ as $(VK, VK', L_{\mathrm{Reg}})$ and $\theta_{F(X)}$ as (d, d', S_i, γ)
2: **if** $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}((d, d', S_i), \gamma, L_{\mathrm{Reg}}[S_i][0])$ **then**
3: **if** $VK = g(d)$ **then return** $(RT_{F(X)} \leftarrow d, \tau_{F(X)} \leftarrow (\mathsf{accept}, S_i))$
4: **else if** $VK' = g(d')$ **then return** $(RT_{F(X)} \leftarrow d', \tau_{F(X)} \leftarrow (\mathsf{accept}, S_i))$
5: **else return** $(RT_{F(X)} \leftarrow \perp, \tau_{F(X)} \leftarrow (\mathsf{reject}, S_i))$
6: **return** $(RT_{F(X)} \leftarrow \perp, \tau_{F(X)} \leftarrow (\mathsf{reject}, \perp))$

8. Retrieve orders the components of the verification key according to the retrieval key $RK_{F,X} = b$ and checks which message was returned, and hence determines the value of $F(X)$. If m_0 was returned then $F(X) = 1$ as m_0 was encrypted for the non-complemented inputs; if m_1 was returned then $F(X) = 0$.

9. Revoke first checks whether a sever, S_i, should in fact be revoked. If so, it deletes the list $L_{\mathrm{Reg}}[S_i][1]$ of computations that S_i may perform. It also adds S_i to the revocation list, and refreshes the time source. It then generates new update keys for all non-revoked entities such that non-revoked keys are still functional in the new time period.

Given an IND-sHRSS secure rkDPABE scheme, a one-way function g, and an EUF-CMA signature scheme, this construction is secure in the sense of selective public verifiability, blind verification and selective semi-static revocation. A proof of security can be found in Appendix C.

Algorithm 8 $y \leftarrow \mathsf{HPVC.Retrieve}(RT_{F(X)}, \tau_{F(X)}, VK_{F,X}, RK_{F,X}, \mathrm{PP})$

Parse $VK_{F,X}$ as $(g(m_b), g(m_{1-b}), L_{\mathrm{Reg}})$, $\theta_{F(X)}$ as $(d_b, d_{1-b}, S_i, \gamma)$, $RK_{F,X}$ as b
if $(\tau_{F(X)} = (\mathsf{accept}, S_i)$ **and** $g(RT_{F(X)}) = g(m_0))$ **then return** $y \leftarrow 1$
if $(\tau_{F(X)} = (\mathsf{accept}, S_i)$ **and** $g(RT_{F(X)}) = g(m_1))$ **then return** $y \leftarrow 0$
return $y \leftarrow \perp$

Algorithm 9 $UM \xleftarrow{\$} \mathsf{HPVC.Revoke}(\tau_{F(X)}, \mathrm{MK}, \mathrm{PP})$

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

5 Conclusion

We have introduced a hybrid model of publicly verifiable outsourced computation to support flexible and dynamic interactions between entities. Entities may request computations from other users, restrict which entities can perform computations on their behalf, perform computations for other users, and make data available for queries from other users, all in a verifiable manner.

Our instantiation, built from a novel use of DP-ABE, captures prior models of PVC [2, 23], extends RPVC-AC [1] to the public key setting to allow truly public delegability and verifiability, and introduces a novel form of ABE-based verifiable computation in the form of VDC. In further work, we will investigate VDC further, particularly with regards to searching on remote databases.

ABE was developed to enforce read-only access control policies, and the use of KP-ABE in PVC was a novel and surprising result [23]. A natural question to ask is whether other forms of ABE can similarly find use in this context. Our use of all possible modes of ABE provides an affirmative answer to this question.

DP-ABE has previously attracted relatively little attentionin the literature, which we believe to be primarily due to its applications being less obvious than for the single-policy ABE schemes. Whilst KP- and CP-ABE are generally considered in the context of cryptographic access control, it is unclear that the policies enforced by DP-ABE are natural choices for access control. Thus an interesting side-effect of this work is to show that additional applications for DP-ABE do exist.

Acknowledgements

We thank Martin R. Albrecht and Naomi Farley for useful discussions and comments.
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 authorised to reproduce and distribute reprints for Government purposes notwithstanding any

copyright notation hereon.

References

[1] J. Alderman, C. Janson, C. Cid, and J. Crampton. Access control in publicly verifiable outsourced computation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 657–662, New York, NY, USA, 2015. ACM.

[2] J. Alderman, C. Janson, C. Cid, and J. Crampton. Revocation in publicly verifiable outsourced computation. In D. Lin, M. Yung, and J. Zhou, editors, *Information Security and Cryptology*, volume 8957 of *Lecture Notes in Computer Science*, pages 51–71. Springer International Publishing, 2015.

[3] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. Verifiable oblivious storage. In H. Krawczyk, editor, *Public-Key Cryptography - PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 131–148. Springer Berlin Heidelberg, 2014.

[4] N. Attrapadung and H. Imai. Attribute-based encryption supporting direct/indirect revocation modes. In M. Parker, editor, *Cryptography and Coding*, volume 5921 of *Lecture Notes in Computer Science*, pages 278–300. Springer Berlin Heidelberg, 2009.

[5] N. Attrapadung and H. Imai. Dual-policy attribute based encryption. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 168–185. Springer Berlin Heidelberg, 2009.

[6] N. Attrapadung and H. Imai. Dual-policy attribute based encryption: Simultaneous access control with ciphertext and key policies. *IEICE Transactions*, 93-A(1):116–125, 2010.

[7] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 271–286. IEEE Computer Society, 2015.

[8] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, pages 863–874, New York, NY, USA, 2013. ACM.

[9] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: Extended abstract. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 401–414, New York, NY, USA, 2013. ACM.

[10] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011.

[11] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, New York, NY, USA, 2012. ACM.

[12] D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *TCC*, pages 680–699, 2013.

[13] S. Choi, J. Katz, R. Kumaresan, and C. Cid. Multi-client non-interactive verifiable computation. In A. Sahai, editor, *Theory of Cryptography*, volume 7785 of *Lecture Notes in Computer Science*, pages 499–518. Springer Berlin Heidelberg, 2013.

[14] K.-M. Chung, Y. Kalai, F.-H. Liu, and R. Raz. Memory delegation. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer Berlin Heidelberg, 2011.

[15] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[16] D. Fiore and R. Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In T. Yu, G. Danezis, and V. D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 501–512. ACM, 2012.

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

[18] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[19] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[20] S. Micali. CS proofs. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 436–453. IEEE, 1994.

[21] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 195–203, New York, NY, USA, 2007. ACM.

[22] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In A. Sahai, editor, *Theory of Cryptography*, volume 7785 of *Lecture Notes in Computer Science*, pages 222–242. Springer Berlin Heidelberg, 2013.

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

[24] J. Shi, J. Lai, Y. Li, R. H. Deng, and J. Weng. Authorized keyword search on encrypted data. In M. Kutylowski and J. Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, volume 8712 of *Lecture Notes in Computer Science*, pages 419–435. Springer, 2014.

[25] J. van den Hooff, M. F. Kaashoek, and N. Zeldovich. Versum: Verifiable computations over large public logs. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1304–1316. ACM, 2014.

[26] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

[27] L. F. Zhang and R. Safavi-Naini. Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. In M. Abdalla, C. Nita-Rotaru, and R. Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 329–348. Springer, 2013.

A Applications for VDC

• **MapReduce** [15] (or Hadoop) is a programming model for the parallel processing of large computations using a cluster or grid of computers (nodes) which can take advantage of the locality of data to decrease transmission costs. Each worker node computes a subproblem on a portion of the data and report to a manager who combines the results. VDC enables verifiable MapReduce such that only *valid* results are combined. The manager acts as the KDC to distribute evaluation keys for partitions of the data to workers, and then requests multiple sub-problems to be solved over this partitioning.

• **Verifiable queries on remote databases.** Servers may also act as remote database providers and register with a KDC to provide a verifiable querying service. Any delegator may use public information to query *any* function allowed by the server (within the family allowed by the VDC scheme) on these databases. Data is remotely stored and delegators see nothing more than the results of queries which they are assured are correct. Alternatively, in this setting, the data owner could act as the KDC to outsource its data to an untrusted server. Due to the public delegation and verification properties, other data users can query the outsourced data and verify the correctness of the results. The data owner need not retain any knowledge of the data after it has been outsourced.

• **Three-party computation**. Backes et al. [7] consider computations over outsourced data based on privacy-preserving proofs over authenticated data outsourced by a trusted client. In this setting, a trusted source produces and authenticates some data which is given to a server. Other parties can then request computations on this data and efficiently verify the results, but learn nothing more than the computation results and their validity. The solution of Backes et al. [7] makes use of homomorphic MACs and succinct non-interactive arguments (SNARGs) [20]. Similar results are found in [11, 25].
In the context of VDC, the CP-ABE decryption mechanism achieves the same goal as SNARGs. The source can be thought of as the KDC, the service provider as the computational server and

Game 2 $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{BVERIF}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]$

1: $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
2: $(F, \mathtt{mode}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathrm{PP})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$
4: $X \xleftarrow{\$} \mathsf{Dom}(F)$
5: $S_i \xleftarrow{\$} \mathcal{U}_{\mathrm{ID}}$
6: **if** $(\mathtt{mode} = \mathrm{VDC})$ **then** $(\psi \leftarrow X, \mathbb{S} \leftarrow F, \mathbb{O} \leftarrow P, \omega \leftarrow s, L \leftarrow \{l(x_j)\}_{x_j \in X})$
7: **else** $(\omega \leftarrow X, \mathbb{O} \leftarrow F, \mathbb{S} \leftarrow P, \psi \leftarrow s, L \leftarrow \{l(F)\})$
8: $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \mathrm{MK}, \mathrm{PP})$
9: $EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L, \{F\}, \mathrm{MK}, \mathrm{PP})$
10: $(\sigma^{\star}, VK^{\star}, RK^{\star}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L, PK_F, \mathrm{PP})$
11: $\theta^{\star} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma^{\star}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \mathrm{PP})$
12: $y \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\theta^{\star}, VK^{\star}, PK_F, \mathrm{PP})$
13: **return** $(y == F(X))$

the third parties as delegators.
Backes et al. [7] considered several applications of this model. For example, trusted sensors could be placed in client premises (e.g. a smart energy meter or a sensor placed in a car to monitor driving habits). These sensors collect data which is authenticated (due to the trusted nature of the collection devices) and given to the client who acts as the service provider. Because this data could be sensitive (e.g. revealing the habits and lifestyle of the client), the service provider may be reluctant to release the data to third parties. Nevertheless, there exist legitimate business cases that require access to compute on the data (e.g. for billing purposes or to produce an insurance quote). Therefore, these third parties may request appropriate computations on the data from the service provider, and can verify that the computation is performed correctly on the correct data.
The efficiency requirement in this setting [7] is simply that verification is more efficient than computation, which our construction in Section 4.3 certainly meets, having constant time verification.

B Security Models

B.1 Blind Verification

In Game 2 we capture the notion of blind verification in the HPVC setting. We require that a verifier that does not hold the output retrieval key may not learn the result of the computation, even though the verification key is public.

On line 2, the adversary is given the public parameters for the HPVC system and oracle access (as specified for the Public Verification game), and must choose a challenge mode and function F. The challenger selects an input X at random from the domain of F; in VDC mode this will comprise k data points, whilst in the PVC modes this comprises a single input (i.e. $k = 1$). The challenger also chooses a server identity S_i at random to simulate performing a computation.

On line 6, X is embedded as the challenge input data according to the chosen mode (we use the notation s and P here to accommodate RPVC-AC; in RPVC or VDC modes these will be the dummy attribute set and policy respectively). The adversary is given the output of $\mathsf{Compute}$ on the challenge, and wins if his guess y equals $F(X)$. The advantage is defined by subtracting the most likely value of $F(X)$ for all F over all inputs to model the adversary's a

Game 3 $\mathbf{Exp}_{\mathcal{A}}^{\text{sSS-Rev}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}, q_t\right]$

1: $(\omega^{\star}, \mathbb{O}^{\star}, \psi^{\star}, \mathbb{S}^{\star}, L_{F,X^{\star}}, \mathtt{mode}^{\star}) \xleftarrow{\$} \mathcal{A}(1^{\ell}, \mathcal{F}, q_t)$
2: **if** $((\mathtt{mode}^{\star} = VDC))$ **then** $(F \leftarrow \mathbb{S}^{\star}, X^{\star} \leftarrow \psi^{\star})$
3: **else** $(F \leftarrow \mathbb{O}^{\star}, X^{\star} \leftarrow \omega^{\star})$
4: $Q_{\text{Rev}} \leftarrow \epsilon$
5: $t \leftarrow 1$
6: $(\text{PP}, \text{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$
7: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \text{MK}, \text{PP})$
8: $\overline{R} \xleftarrow{\$} \mathcal{A}(PK_F, \text{PP})$
9: $\mathcal{A}^{\mathcal{O}}(PK_F, \text{PP})$
10: **if** $(\overline{R} \not\subseteq Q_{\text{Rev}})$ **then return** 0
11: $(\sigma^{\star}, VK^{\star}, RK^{\star}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}^{\star}, (\omega^{\star}, \mathbb{S}^{\star}), L_{F,X^{\star}}, PK_F, \text{PP})$
12: $\theta^{\star} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^{\star}, VK^{\star}, RK^{\star}, PK_F, \text{PP})$
13: $(RT_{\theta^{\star}}, \tau_{\theta^{\star}}) \leftarrow \mathsf{BVerif}(\theta^{\star}, VK^{\star}, \text{PP})$
14: **if** $(\tau_{\theta^{\star}} \neq (\mathsf{reject}, \cdot))$ **and** $(S \in \overline{R}))$ **then**
15: **return** 1
16: **else return** 0

Oracle 1 $\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP})$

1: **if** $((L_{F,X^{\star}} \subseteq L_i$ **and** $S_i \notin \overline{R})$ **or** $(t = q_t$ **and** $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S_i))$ **then return** $\perp$
2: $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S$
3: **return** $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP}))$

Oracle 2 $\mathcal{O}^{\mathsf{Revoke}}(\tau_{F'(X)}, \text{MK}, \text{PP}))$

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

priori knowledge.

Definition 5. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* BVerif *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

$$Adv_{\mathcal{A}}^{\text{BVerif}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}) =$$
$$\Pr\left[1 \xleftarrow{\$} \mathbf{Exp}_{\mathcal{A}}^{\text{BVerif}}\left[\mathcal{HPVC}, 1^{\ell}, \mathcal{F}\right]\right] - \max_{F \in \mathcal{F}}\left(\max_{y \in \mathsf{Ran}(F)}\left(\Pr_{X \in \mathsf{Dom}(F)}[F(X) = y]\right)\right).$$

$\mathcal{HPVC}$ is secure with respect to blind verification *if, for all PPT adversaries $\mathcal{A}$,*

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

B.2 Selective, Semi-static Revocation

Revocation requires that, if a server is detected as misbehaving, i.e. the BVerif algorithm outputs $\tau_{F(X)} = (\mathsf{reject}, S_i)$, then any subsequent evaluations by S_i should be rejected. This notion inherits the selective, semi-static restrictions from the revocation mechanism of the underlying rkDPABE scheme in our construction. The adversary must first select its challenge parameters, which the challenger can parse to learn F and $X^{\star}$ for the challenge computation. The challenger maintains a time period t which is incremented during Revoke oracle queries, and a list Q_{Rev} of currently revoked entities. On line 4, the adversary must choose a list $\overline{R}$ of servers to be revoked during the challenge generation (which we assume will be at time q_t, given as an input to the game).

The adversary is then given oracle access to the functions $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Revoke}(\cdot, \mathrm{MK}, \mathrm{PP})$, denoted by $\mathcal{O}$. Certify and Revoke queries are handled as specified in Oracles 1 and 2. The Certify oracle returns $\perp$ if the resulting evaluation key would enable evaluation of the challenge computation. After finishing this query phase (and in particular after q_t Revoke queries), the challenge is created. The adversary wins if it outputs *any* result (even a correct encoding of $F(X^\star)$) that is accepted as a valid response from any server that was revoked at the time of the challenge.

Definition 6. *The* advantage *of a PPT adversary $\mathcal{A}$ making a polynomial number, q, of oracle queries, of which q_t are* Revoke *queries, in the* sSS-Rev *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

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

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

B.3 Selective Authorised Computation

The notion of *selective authorised computation*, presented in Game 4, ensures that only a server that satisfies the additional authorisation policy specified in the encoded input may perform a given computation and hence be rewarded for correct work. A result generated by an unauthorised server should be rejected (even if the result itself is correct). Note that this game is only meaningful when the challenge parameters are generated in RPVC-AC mode.

Game 4 $\mathbf{Exp}_{\mathcal{A}}^{\text{sAuthC}}\left[\mathcal{HPVC}, 1^\ell, \mathcal{F}\right]$

1: $(F, X^\star, P) \xleftarrow{\$} \mathcal{A}(1^\ell)$
2: $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$
3: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$
4: $(\sigma^\star, VK^\star, RK^\star) \xleftarrow{\$} \mathsf{ProbGen}(\text{RPVC-AC}, (x, P), \{l(F)\}, PK_F, \mathrm{PP})$
5: $\theta^\star \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^\star, VK^\star, RK^\star, PK_F, \mathrm{PP})$
6: $(RT^\star, \tau^\star) \leftarrow \mathsf{BVerif}(\theta^\star, VK^\star, \mathrm{PP})$
7: **if** $(\tau^\star \neq (\mathsf{reject}, \cdot))$ **then return** 1
8: **else return** 0

This is a selective notion due to the selectively secure rkDPABE primitive we use in our construction; as such, the game begins with the adversary choosing a challenge function F, a challenge input $X^\star$ and an authorisation policy P. The challenger initialises the system and generates an encoded input for the challenge computation. The adversary is given the resulting parameters and oracle access to $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Revoke}(\cdot, \mathrm{MK}, \mathrm{PP})$, denoted by $\mathcal{O}$. The Certify oracle is handled as specified in Oracle 3. It returns a failure symbol $\perp$ if the queried attributes ψ satisfies the authorisation policy P, else the adversary could trivially produce a valid response as an authorised entity. Otherwise, it simply runs the Certify algorithm. The adversary must return a result which is accepted by a verifier.

Definition 7. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* sAuthC *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

Oracle 3 $\mathcal{O}^{\mathsf{Certify}}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

1: **if** $(\psi \in P)$ **then return** $\perp$
2: **return** $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$

$\mathcal{HPVC}$ *is* secure with respect to selective authorised computation *if, for all PPT adversaries* $\mathcal{A}$*,*

$$Adv_{\mathcal{A}}^{\text{sAUTHC}}(\mathcal{HPVC}, 1^{\ell}, \mathcal{F}) \leqslant \mathrm{negl}(\ell).$$

C Proofs of Security

Theorem 1. *Given a secure* IND-sHRSS *rkDPABE scheme for a class of Boolean functions* $\mathcal{F}$ *closed under complement, a one-way function* g*, and a signature scheme secure against* EUF-CMA*, then* $\mathcal{HPVC}$*, defined by Algorithms 1 to 9, is secure in the sense of selective public verifiability, blind verification, selective semi-static revocation and authorised .*

Informally, public verifiability and revocation reduce to the indistinguishability of rkDPABE ciphertexts which allows us to replace the message for the unsatisfied function (which cannot be decrypted) with the challenge for a one-way function game. Then an adversary against these games can attack the verification token for this message. The proof of blind verification relies on a probability argument showing that adversarial inputs do not help in determining $F(X)$.

C.1 Proof of Public Verifiability

Lemma 1. $\mathcal{HPVC}$*, given in Algorithms 1–9, is secure with respect to selective public verifiability (Game 1) under the same assumptions as in Theorem 1.*

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against Game 1 when instantiated with Algorithms 1–9. We define the following three games:

- **Game 0.** This is the selective public verifiability game as in Game 1.
- **Game 1.** This differs from **Game 0** in that $\mathsf{ProbGen}$ no longer returns an encryption of m_0 and m_1. Instead, we choose a random message $m' \neq m_0, m_1$. Recall that two ciphertexts are created during $\mathsf{ProbGen}$ (encrypting m_0 and m_1) and that one is associated with the function F, and the other with the complement function $\overline{F}$. Now, only one of F and $\overline{F}$ is satisfied by the input data $X^{\star}$. We replace the plaintext associated with the *unsatisfied* function by m' which is unrelated to m_0, m_1 and the verification keys.
- **Game 2.** This is the same as **Game 1** except that we implicitly set m' to be the challenge input w in the one-way function game.

By hopping from **Game 0** to **Game 2**, we show that $\mathcal{A}_{VC}$ can be used to construct an adversary that inverts the one-way function g.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**. Suppose otherwise, that $\mathcal{A}_{VC}$ can distinguish the two games with non-negligible advantage δ. We construct an adversary $\mathcal{A}_{ABE}$ that uses $\mathcal{A}_{VC}$ as a subroutine to break the IND-sHRSS security of the rkDPABE scheme (Game 5). In this proof, we consider RPVC-AC mode to be a special case of RPVC (sincethe adversary can be authorised to evaluate the challenge computation); therefore we discuss only on RPVC and VDC modes. Let $\mathcal{C}$ a challenger playing the IND-sHRSS game with $\mathcal{A}_{ABE}$, who in turn acts as a challenger in the selective public verifiability game for $\mathcal{A}_{VC}$:

1. $\mathcal{A}_{VC}$ is given the security parameter, and declares its choice of challenge input parameters $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star,)$, a set of labels $L_{F,X^\star}$ and `mode`.

2. $\mathcal{A}_{ABE}$ must send a challenge input $(\tilde{\omega}, \tilde{\mathbb{S}})$ and a time period $\tilde{t}$ to the challenger. It first sets $\tilde{t} = 1$. Observe that in VDC mode, $\mathbb{S}^\star$ corresponds to the function F and $\psi^\star$ is the challenge input data $X^\star$. In RPVC mode, $\mathbb{O}^\star = F$ and ω corresponds to the challenge input $X^\star$. The other inputs are either dummy attributes or policies that are trivially satisfied by the dummy attribute. Now, using the relevant inputs, $\mathcal{A}_{ABE}$ computes $r = F(X^\star)$.

- If the challenge mode is RPVC, set $\tilde{\omega} = A_{\omega^\star} = A_{X^\star}$, and

$$\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X}} l_j = \{\{T_S^0\}\} \wedge \{l(F)\}.$$

- If the challenge mode is VDC, we want to set $(\tilde{\omega}, \tilde{\mathbb{S}})$ such that the pair is not satisfied by the challenge input.
  - If $r = 1$: set $\tilde{\omega} = A_{\omega^\star} = \{T_O^0\}$, and

$$\tilde{\mathbb{S}} = \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X}} l_j = \overline{F} \wedge \{l(x_{i,j})\}_{x_{i,j} \in X^\star}.$$

  - If $r = 0$: set $\tilde{\omega} = A_{\omega^\star} = \{T_O^0\}$, and

$$\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X}} l_j = F \wedge \{l(x_{i,j})\}_{x_{i,j} \in X^\star}.$$

3. $\mathcal{C}$ runs the DPABE.Setup algorithm on the security parameter to generate MPK_{ABE} and MSK_{ABE} and gives MPK_{ABE} to $\mathcal{A}_{ABE}$.

4. $\mathcal{A}_{ABE}$ sends $\overline{R} = \epsilon$ (i.e. an empty list) to $\mathcal{C}$ and simulates running HPVC.Setup such that the outcome is consistent with MPK_{ABE}. If `mode` = `VDC`, it runs lines 3 to 5 as written, sets $MPK^0_{\text{ABE}} = MPK_{\text{ABE}}$ as given by $\mathcal{C}$, and implicitly sets $MSK^0_{\text{ABE}} = MSK_{\text{ABE}}$ (any use of MSK^0_{ABE} will be simulated using oracle queries to $\mathcal{C}$). Otherwise, it sets MPK^r_{ABE} to be that issued by $\mathcal{C}$, and implicitly sets MSK^r_{ABE} to be that held by the challenger. In both cases, it also runs DPABE.Setup itself to generate a second DP-ABE system.

5. $\mathcal{A}_{ABE}$ runs HPVC.FnInit as written. To generate the challenge input, $\mathcal{A}_{ABE}$ begins by choosing two random bits, b (which it defines to be $RK_{F,X^\star}$) and s, and three random messages m_0, m_1 and m' from the message space.
$\mathcal{A}_{ABE}$ sends m_0 and m_1 to $\mathcal{C}$ as its challenge inputs. $\mathcal{C}$ chooses $b^\star \xleftarrow{\$} \{0,1\}$ and returns $CT^\star \leftarrow \mathsf{DPABE.Encrypt}(m_{b^\star}, (\tilde{\omega}, \tilde{\mathbb{S}}), \tilde{t}, MPK_{\text{ABE}})$.

- In RPVC mode, $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates

$$c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), \tilde{t}, MPK^1_{\text{ABE}})$$

  itself. It sets $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$.

- In VDC mode:
  - If $r = 1$:
    $\mathcal{A}_{ABE}$ generates $c_b \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \mathbb{S}^\star) \wedge \bigwedge_{l_j \in L_{F,X}} l_j, \tilde{t}, MPK^0_{\text{ABE}})$ itself, and sets c_{1-b} to be $CT^\star$. It sets $VK_b = g(m')$ and $VK_{1-b} = g(m_s)$.
  - If $r = 0$:
    $\mathcal{A}_{ABE}$ generates $c_{1-b} \leftarrow \mathsf{DPABE.Encrypt}(m', (\tilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X}} l_j), \tilde{t}, MPK^0_{\text{ABE}})$ itself and sets c_b to be $CT^\star$. It sets $VK_b = g(m_s)$ and $VK_{1-b} = g(m')$.

Finally, $\mathcal{A}_{ABE}$ sets $\sigma_{F,X^\star} = (c_b, c_{1-b})$, $VK_{F,X^\star} = (VK_b, VK_{1-b}, L_{Reg})$ and $RK_{F,X^\star} = b$. Note that s is essentially $\mathcal{A}_{ABE}$'s guess of $b^\star$ chosen by $\mathcal{C}$.

6. $\mathcal{A}_{VC}$ is given these outputs and oracle access which is handled as follows.

- $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$ and $\mathsf{Register}(\cdot, \mathrm{MK}, \mathrm{PP})$ can be run as written.
- $\mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, S_i, \mathrm{MK}, \mathrm{PP})$: To generate the evaluation key for the queried parameters, $\mathcal{A}_{ABE}$ uses the KeyGen oracle in the rkDPABE game. It first updates the relevant list entries as specified. Then it sets $\mathbb{O}' = \mathbb{O}$ and $\psi' = A_\psi \cup \bigcup_{l_j \in L_i} l_j$ and makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}', \psi'), \mathrm{MK}, \mathrm{PP})$ as in Oracle 4. $\mathcal{C}$ shall generate a rkDPABE decryption key $SK_{\mathbb{O}',\psi'}$ if and only if $\tilde{\omega} \notin \mathbb{O}'$ or $\psi' \notin \tilde{\mathbb{S}}$ or $S_i \in \overline{R}$.

  Observe, that $S_i \notin \overline{R}$ since we set $\overline{R}$ to be empty.

  Now, by construction (Step 2), $\psi' \in \tilde{\mathbb{S}}$ only if the labels $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$. If the labels *do not* satisfy this relation, then $\mathcal{C}$ may generate the key, which $\mathcal{A}_{ABE}$ will receive as SK^0_{ABE}.

  If, on the other hand, the labels *do* satisfy this relation, then because each label uniquely describes a single element (either a function or a data point):

  - In RPVC mode: as both L_i and $L_{F,X^\star}$ are singleton sets, and $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$, it must be that $L_i = L_{F,X^\star} = \{l(F)\}$ and hence, by uniqueness of the labels, $\mathbb{O} = \mathbb{O}^\star$ i.e. the adversary has requested an evaluation key for the challenge function F. However, in Step 4, we assigned the ABE system owned by the challenger (with master secret MSK^r_{ABE}) precisely such that $\mathbb{O}^\star$ is not satisfied by the challenge input $\tilde{\omega}$, and therefore $\mathbb{O}'$ is not satisfied either – that is, $\tilde{\omega} \notin \mathbb{O}'$ and hence $\mathcal{C}$ can generate a valid key which $\mathcal{A}_{ABE}$ will store as SK^r_{ABE}.
  - In VDC mode: $\{l_k\}_{l_k \in L_{F,X^\star}} \subseteq \{l_j\}_{l_j \in L_i} \Rightarrow \{l(x_{i,k})\}_{x_{i,k} \in X^\star} \subseteq \{l(x_{i,j})\}_{x_{i,j} \in D_i} \Rightarrow X^\star \subseteq D_i$ i.e. by uniqueness of the labels, the adversary has requested an evaluation key for a superset of the challenge input data – that is, D_i contains $X^\star$ and possibly some additional data points. Now, if $X^\star \subseteq D_i$ then, additionally D_i must satisfy either F or $\overline{F}$ to satisfy $\tilde{\mathbb{S}}$. However, note that in Step 2, $\tilde{\mathbb{S}}$ was chosen specifically such that it is not satisfied by the challenge input $X^\star$ and therefore by D_i. Hence $\mathcal{C}$ may generate a valid key which $\mathcal{A}_{ABE}$ will store as SK^0_{ABE}.

  $\mathcal{A}_{ABE}$ also must request an update key by making a $\mathsf{KeyUpdate}$ oracle query to $\mathcal{C}$. $\mathcal{C}$ will return a valid key unless the current time period is $t = \tilde{t}$ and $\overline{R} \not\subseteq Q_{Rev}$. Observe that the second clause is never satisfied since $\overline{R} = \epsilon$ and hence is a subset of any Q_{Rev}. Hence $\mathcal{C}$ may generate a valid update key.

  If in RPVC mode, $\mathcal{A}_{ABE}$ additionally generates a key SK^{1-r}_{ABE} using the second system parameters (which he owns) for $(\overline{\mathbb{O}}, \overline{\psi})$.

- $\mathsf{Revoke}(\tau_{(\mathbb{O},\psi),(\omega,\mathbb{S})}, (\mathbb{O}, \psi), (\omega, \mathbb{S}), \mathrm{MK}, \mathrm{PP})$: In response to a Revoke query, $\mathcal{A}_{ABE}$ runs Algorithm 9 as written except that it will make a $\mathsf{KeyUpdate}$ oracle query to $\mathcal{C}$ for the update key for to the ABE system owned by the challenger ($UK^r_{L_{\mathrm{Rev}},t}$ in the case of RPVC and $UK^0_{L_{\mathrm{Rev}},t}$ in the case of VDC).

  $\mathcal{C}$ will generate a valid update key unless the time period is $\tilde{t} = q_t$ and there exists a server on $\overline{R}$ that is not currently revoked. However, as $\overline{R}$ was defined to be an empty list, $\mathcal{C}$ can always return a valid key.

Eventually, $\mathcal{A}_{VC}$ outputs $\theta^\star$ which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $F(X^\star)$).

7. $\mathcal{A}_{ABE}$ parses $\theta^\star$ as $(d_b, d_{1-b}, S_{i^\star}, \gamma)$ and using the retrieval key $RK_{F,X^\star} = b$, finds d_0 and d_1. One of d_0 and d_1 will be $\perp$ (by construction) and we denote the other value by Y. If

$g(Y) = g(m_s)$, $\mathcal{A}_{ABE}$ outputs a guess $b' = s$ and otherwise guesses $b' = (1 - s)$.

If $s = b^\star$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game 0** (since the verification key comprises $g(m')$ where m' is the message a legitimate server could recover, and $g(m_s)$ where m_s is the other plaintext). Alternatively, $s = 1 - b^\star$ and the distribution of the above experiment is identical to **Game 1** (since the verification key comprises the legitimate message and a random message $m_{1-b^\star}$ that is unrelated to the ciphertext).

Now, we consider the advantage of $\mathcal{A}_{ABE}$ in the IND-sHRSS game. Recall that, by assumption, $\mathcal{A}_{VC}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1** – that is, if $\mathbf{Exp}^i_{\mathcal{A}_{VC}}\left[\mathcal{HPVC}, 1^\ell, \mathcal{F}\right]$ denotes running $\mathcal{A}_{VC}$ in **Game** i,

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

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

Hence, $Adv_{\mathcal{A}_{ABE}} \geqslant \left|\Pr[b^\star = b'] - \frac{1}{2}\right| \geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \geqslant \frac{\delta}{2}$. If $\mathcal{A}_{VC}$ has advantage δ at distinguishing these games then $\mathcal{A}_{ABE}$ can win the IND-sHRSS game with non-negligible probability. Thus since we assumed the rkDPABE scheme to be secure, we conclude that $\mathcal{A}_{VC}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** sets the value of m' to correspond to the challenge w in the one-way function inversion game (rather than be a randomly chosen message). We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system, bar the verification key $g(w)$, and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{VC}$ does not see the challenge for the one-way function).

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

- if $r = 1$, we implicitly set m_{1-b} to be w by setting the corresponding verification key component to be z. We choose m_b and the other verification key component randomly as usual.
- if $r = 0$, we implicitly set m_b to be w by setting the corresponding verification key component to be z. We choose m_{1-b} and the other verification key component randomly as usual.

Now, if $\mathcal{A}_{VC}$ is successful, it will output a forgery comprising the plaintext encrypted under the function F or $\overline{F}$ that evaluates to 0. By construction, this will be w (and the adversary's view in consistent since the verification key is simulated correctly using z). $\mathcal{A}_{ABE}$ can forward this

result to $\mathcal{C}$ to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{VC}$ has against the selective public verifiability game.

We conclude that if the rkDPABE scheme is IND-sHRSS secure and the one-way function is hard-to-invert, then the $HPVC$ as defined by Algorithms 1–9 is secure in the sense of selective public verifiability. □

C.2 Proof of Blind Verification

Lemma 2. *$\mathcal{HPVC}$, given in Algorithms 1–9, is secure with respect to blind verification (Game 2) under the same assumptions as in Theorem 1.*

Proof. We first argue that only $\theta_{F(X)}$ and $VK_{F,X}$ may be useful to the adversary, and then show that these inputs do not provide an advantage at guessing $F(X)$.

Over the course of the game, the adversary sees the following inputs: $\theta_{F(X)}$, $VK_{F,X}$, PK_F, PP and the outputs from oracle queries. By construction, $PK_F = \text{PP}$ which is defined at the beginning of the game; hence PP clearly does not reveal any information about $F(X)$. As the adversary does not see the challenge ciphertexts, the ABE public parameters are not helpful (else the ABE scheme is not IND-CPA secure), and neither is L_{Reg} which contains only function lists, data labels that do not reveal the data values, and signature verification keys.

The inputs $\theta_{F(X)}$ and $VK_{F,X}$ clearly do rely on the values of X and $F(X)$ and we will consider these shortly. We first consider oracle access:

- $\mathsf{FnInit}(\cdot, \text{MK}, \text{PP})$: FnInit queries simply return the public parameters which we considered previously as an explicit adversarial input;
- $\mathsf{Register}(\cdot, \text{MK}, \text{PP})$: queries to this oracle generate a signing key for a server S_i. However, this does not relate to the retrieval key or the choice of X;
- $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, \text{MK}, \text{PP})$: a call to this oracle will add an entry to L_{Reg} comprising a function identifier and a set of data labels (which we have assumed not to leak the data values themselves). It also creates an ABE decryption key. Again, as the adversary only sees plaintexts and does not see the ciphertexts forming the challenge encoded input, such a key is not useful.

Hence, oracle access does not help the adversary distinguish which input was selected and hence the value of $F(X)$. Thus, the only inputs that may aid the adversary are $\theta_{F(X)}$ and $VK_{F,X}$, and we restrict our attention to these.

Recall that a well-formed response by the server will be either $(m_b, \perp)$ or $(\perp, m_{1-b})$ according to $RK_{F,X}$. In detail this means, where $RK_{F,X} = b$:

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

Note that $VK_{F,X} = (g(m_b), g(m_{1-b}))$ (excluding L_{Reg} which we discussed above). We denote by $\mathcal{V}$ the adversary's view of $\theta_{F(X)}$ and $VK_{F,X}$ – that is, $\mathcal{V} = (d_b, d_{1-b}, g(m_b), g(m_{1-b}))$ if $\theta_{F(X)} = (d_b, d_{1-b})$ and $VK_{F,X} = (g(m_b), g(m_{1-b}))$.

We now show that the probability of the adversary correctly guessing the value of $F(X)$ given a particular view $\mathcal{V}$ is identical to his success at guessing without seeing $\mathcal{V}$. Thus, he has no advantage at guessing $F(X)$ over his *a priori* knowledge of the distribution of F. Let $\mathcal{V}_1 = (m', \perp, g(m'), g(m_{1-b}))$ and let $\mathcal{V}_2 = (\perp, m'', g(m_b), g(m''))$. We claim that these are the

only possible views – $\mathcal{A}$ sees one message (either m_0 or m_1, drawn uniformly from the same distribution) along with g applied to that message and to a different (unseen) message.

Note that: (i) the value of $F(X)$ and $b \xleftarrow{\$} \{0,1\}$ are independent; (ii) $\Pr[b=1] = \frac{1}{2}$; and (iii) $\Pr[F(X)=0] + \Pr[F(X)=1] = 1$ since F is Boolean. Now,

$$\begin{aligned}
\Pr[\mathcal{V}=\mathcal{V}_1] &= \Pr[(F(X)=1 \wedge b=0) \vee (F(X)=0 \wedge b=1)] \\
&= \Pr[F(X)=1 \wedge b=0] + \Pr[F(X)=0 \wedge b=1] \\
&= \Pr[F(X)=1]\Pr[b=0] + \Pr[F(X)=0]\Pr[b=1] \text{ by (i)} \\
&= \frac{1}{2}\Pr[F(X)=1] + \frac{1}{2}\Pr[F(X)=0] \\
&= \frac{1}{2}(\Pr[F(X)=0] + \Pr[F(X)=1]) \\
&= \frac{1}{2} \qquad (1)
\end{aligned}$$

$$\begin{aligned}
\Pr[F(X)=0|\mathcal{V}=\mathcal{V}_1] &= \frac{\Pr[F(X)=0 \wedge \mathcal{V}=\mathcal{V}_1]}{\Pr[\mathcal{V}=\mathcal{V}_1]} \\
&= \frac{\Pr[F(X)=0 \wedge b=1]}{\Pr[\mathcal{V}=\mathcal{V}_1]} \\
&= \frac{\Pr[F(X)=0]\Pr[b=1]}{\Pr[\mathcal{V}=\mathcal{V}_1]} \text{ by (i)} \\
&= \frac{\frac{1}{2}\Pr[F(X)=0]}{\frac{1}{2}} \text{ by (1)} \\
&= \Pr[F(X)=0]
\end{aligned}$$

$$\begin{aligned}
\Pr[\mathcal{V}=\mathcal{V}_2] &= \Pr[(F(X)=1 \wedge b=1) \vee (F(X)=0 \wedge b=0)] \\
&= \Pr[F(X)=1 \wedge b=1] + \Pr[F(X)=0 \wedge b=0] \\
&= \Pr[F(X)=1]\Pr[b=1] + \Pr[F(X)=0]\Pr[b=0] \text{ by (i)} \\
&= \frac{1}{2}\Pr[F(X)=1] + \frac{1}{2}\Pr[F(X)=0] \\
&= \frac{1}{2}(\Pr[F(X)=0] + \Pr[F(X)=1]) \\
&= \frac{1}{2} \qquad (2)
\end{aligned}$$

$$\begin{aligned}
\Pr[F(X)=0|\mathcal{V}=\mathcal{V}_2] &= \frac{\Pr[F(X)=0 \wedge \mathcal{V}=\mathcal{V}_2]}{\Pr[\mathcal{V}=\mathcal{V}_2]} \\
&= \frac{\Pr[F(X)=0 \wedge b=0]}{\Pr[\mathcal{V}=\mathcal{V}_2]} \\
&= \frac{\Pr[F(X)=0]\Pr[b=0]}{\Pr[\mathcal{V}=\mathcal{V}_2]} \text{ by (i)} \\
&= \frac{\frac{1}{2}\Pr[F(X)=0]}{\frac{1}{2}} \text{ by (2)} \\
&= \Pr[F(X)=0]
\end{aligned}$$

A symmetric argument holds for $F(X)=1$. We conclude that knowledge of the adversarial inputs provides no advantage in guessing $F(X)$ other than that which could already be guessed

(i.e. inputs leak no information about $F(X)$). □

C.3 Proof of Revocation

Lemma 3. *$\mathcal{HPVC}$, given in Algorithms 1–9, is secure in the sense of selective, semi-static revocation (Game 3) under the same assumptions as in Theorem 1.*

Proof. We reduce the security of the selective, semi-static revocation game to the IND-sHRSS security of the rkDPABE scheme (Game 5). To achieve a contradiction, let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the selective, semi-static revocation game (Game 3) when instantiated with Algorithms 1–9 and making q_t Revoke queries. We show that, if such an $\mathcal{A}_{VC}$ exists, then it can be used to construct an adversary $\mathcal{A}_{ABE}$ that can break the IND-sHRSS security of the revocable-key DPABE scheme. Again, we consider only RPVC and VDC modes here (and view RPVC-AC as a special case of RPVC) as the adversary should be authorised (in terms of the authorisation policy, if not in terms of revocation) for the challenge computation. Let $\mathcal{C}$ be a challenger for the IND-sHRSS game playing with $\mathcal{A}_{ABE}$, who in turn acts as the challenger in the selective, semi-static revocation game with $\mathcal{A}_{VC}$.

1. $\mathcal{A}_{VC}$ selects its challenge inputs $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode}^\star)$ for a challenge computation of $F(X^\star)$.

2. $\mathcal{A}_{ABE}$ initialises the list $Q_{\text{Rev}} = \epsilon$ and time parameter $t = 1$. It then forms its own challenge input as follows. It sets $t^\star = q_t$. Then, it sets $\tilde{\omega} = A_{\omega^\star}$ and sets $\tilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j$. It sends $t^\star, \tilde{\omega}$ and $\tilde{\mathbb{S}}$ to $\mathcal{C}$.

3. $\mathcal{C}$ runs DPABE.Setup and returns the public parameters MPK_{ABE} to $\mathcal{A}_{ABE}$ who stores them as MPK^0_{ABE}.

4. $\mathcal{A}_{ABE}$ now simulates the HPVC.Setup algorithm such that the output is consistent with the public parameters generated by $\mathcal{C}$. It runs Algorithm 1 as written, with the exception of line 1, since MSK^0_{ABE} and MPK^0_{ABE} was already generated by $\mathcal{C}$. $\mathcal{A}_{ABE}$ also runs HPVC.FnInit as written, and gives the public parameters and public delegation key to $\mathcal{A}_{VC}$.

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

6. $\mathcal{A}_{VC}$ is now given oracle access to which $\mathcal{A}_{ABE}$ can respond as follows:

- Queries to HPVC.FnInit and HPVC.Register can be run as written.
- Queries of the form HPVC.Certify$(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Oracle 1. To simulate running the HPVC.Certify algorithm, $\mathcal{A}_{ABE}$ runs Algorithm 4 as written with the exception of lines 4 and 5, as these depend on MSK^0_{ABE} held by $\mathcal{C}$.

  To simulate line 4, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_k \in L_i} l_k), MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ responds by running Oracle 4 which will return a valid key unless $(\tilde{\omega} \in \mathbb{O})$ *and* $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \tilde{\mathbb{S}})$ *and* $(S_i \notin \overline{R})$.

  If $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \tilde{\mathbb{S}})$ then $\mathcal{C}$ can return a valid decryption key. Otherwise, we observe that $(A_\psi \cup \bigcup_{l_k \in L_i} l_k) \in (\mathbb{S} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j)$ only if $\{l_k\}_{l_k \in L_i} \supseteq \{l_j\}_{l_j \in L_{F,X^\star}}$. As labels uniquely label the objects they relate to (either functions or data points), this implies $L_{F,X^\star} \subseteq L_i$. However, in this case, by the first check in Oracle 1, $\mathcal{A}_{ABE}$ would have returned $\perp$ without querying KeyGen if $S_i \notin \overline{R}$, to avoid certifying $\mathcal{A}_{VC}$ for the challenge computation.

  Thus, at the point of making a KeyGen query, if $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \tilde{\mathbb{S}})$ then $S_i \in \overline{R}$. Therefore, $\mathcal{C}$ can respond to all queries made to it during this phase with a valid key which $\mathcal{A}_{ABE}$ can use to simulate line 4 correctly.

  To simulate line 5 of HPVC.Certify, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MP$
  $\mathcal{C}$ responds as written in Oracle 5 – that is, it returns a valid update key *unless* $t = t^\star$ *and* $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, note that $\mathcal{A}_{ABE}$ chose $t^\star = q_t$, and at the point of calling the

KeyUpdate oracle, the list $Q_{\text{Rev}} = Q_{\text{Rev}} \setminus S_i$. Therefore, if $\mathcal{C}$ would return $\perp$ in response to this query, then $\mathcal{A}_{ABE}$ would already have returned $\perp$ as a result of the checks performed in Oracle 1. Hence, for all queries made to $\mathcal{C}$, a valid update key is returned and line 4 is simulated correctly.

- Queries of the form $\mathsf{HPVC.Revoke}(\tau_{F(X)}, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Oracle 2. To simulate running the HPVC.Revoke algorithm, $\mathcal{A}_{ABE}$ runs Algorithm 9 as written with the exception of line 4. Instead, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. Note that, according to Oracle 2, $\mathcal{A}_{ABE}$ would have returned $\perp$ if $t = q_t$ (where, recall, $q_t = t^\star$) and $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S_i$. This corresponds directly to the conditions wherein $\mathcal{C}$ cannot form a valid update key according to Oracle 5 (since, if HPVC.Revoke is called, S_i was already removed from the list Q_{Rev}). Hence, for all queries, $\mathcal{C}$ can form a valid update key and $\mathcal{A}_{ABE}$ can simulate the expected behaviour.

7. Eventually (after q_t Revoke queries), $\mathcal{A}_{VC}$ finishes this query phase. $\mathcal{A}_{ABE}$ checks whether the list of revoked entities is compatible with the challenge list $\overline{R}$ and returns 0 if not.

8. $\mathcal{A}_{ABE}$ must now generate the challenge input for $\mathcal{A}_{VC}$. To do so, it chooses three distinct messages, m_0, m_1 and m', uniformly at random from the messagespace. It also chooses a random bit b which it defines to be $RK_{F,X^\star}$. It sends m_0 and m_1 to $\mathcal{C}$ as its challenge inputs in the IND-sHRSS game. $\mathcal{C}$ chooses a bit $b^\star$ uniformly at random and returns $CT^\star \xleftarrow{\$} \mathsf{Encrypt}(m_{b^\star}, \tilde{\omega}, \tilde{\mathbb{S}}, t^\star, MPK^0_{\text{ABE}})$. $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates c_{1-b} himself by encrypting m' as specified on lines 4 and 5 of Algorithm 5.
Finally, $\mathcal{A}_{ABE}$ selects another bit s uniformly at random and, if $b = 0$, it sets $VK_{F,X^\star} = (g(m_s), g(m'), L_{\text{Reg}})$, or otherwise it sets $VK_{F,X^\star} = (g(m'), g(m_s), L_{\text{Reg}})$. Note that s can be thought of as $\mathcal{A}_{ABE}$'s guess as to the value of $b^\star$.

9. $\mathcal{A}_{VC}$ is given the resulting parameters and again given oracle access which $\mathcal{A}_{ABE}$ responds to as in Step 6.

10. Eventually, $\mathcal{A}_{VC}$ outputs its result $\theta^\star$ which, in order to appear valid, should contain exactly one non-$\perp$ plaintext; we denote this plaintext by y. If $g(y) = g(m_s)$, $\mathcal{A}_{ABE}$ guesses $b' = s$. If $g(y) = g(m')$, $\mathcal{A}_{ABE}$ guesses randomly $b' \leftarrow \{0, 1\}$ (as $\mathcal{A}_{VC}$ did not respond for either m_0 or m_1, it reveals no information to aid $\mathcal{A}_{ABE}$). Otherwise, $\mathcal{A}_{ABE}$ aborts as $\mathcal{A}_{VC}$ was not successful (either for legitimate reasons or because $\mathcal{A}_{ABE}$ chose s incorrectly and issued a malformed challenge).

By assumption, $\mathcal{A}_{VC}$ has non-negligible advantage δ against the selective, semi-static revocation game – that is, $\Pr[g(y) = g(m_s)] + \Pr[g(y) = g(m')] = \delta$. Therefore,

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

Hence, $Adv_{\mathcal{A}_{ABE}} \geq \left|\Pr[b^\star = b'] - \frac{1}{2}\right| \geq \left|\frac{\delta}{2} - \frac{1}{2}\right| \geq \frac{1}{2}(\delta - 1)$, which is non-negligible. However, the DPABE scheme was assumed to be secure in the sense of IND-sHRSS and hence such an adversary as $\mathcal{A}_{ABE}$ may not exist. Therefore, our assumption on $\mathcal{A}_{VC}$ must be wrong, and no adversary with non-negligible advantage against the selective, semi-static revocation game can exist. □

C.4 Proof of Authorised Computation

Lemma 4. *$\mathcal{HPVC}$, given in Algorithms 1–9, is secure with respect to selective authorised computation (Game 4) under the assumptions in Theorem 1.*

Proof. We reduce the security of the selective authorised computation game to the IND-sHRSS security of the underlying revocable-key DPABE scheme (Game 5). To achieve a contradiction, let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage against the selective authorised computation game (Game 4) when instantiated with Algorithms 1–9. We show that, if such an $\mathcal{A}_{VC}$ exists, then it can be used to construct an adversary $\mathcal{A}_{ABE}$ that can break the IND-sHRSS security of the revocable-key DPABE scheme. Note that this notion is only meaningful in RPVC-AC mode. Let $\mathcal{C}$ be a challenger for the IND-sHRSS game playing with $\mathcal{A}_{ABE}$, who in turn acts as the challenger in the selective authorised computation game with $\mathcal{A}_{VC}$.

1. $\mathcal{A}_{VC}$ first selects its challenge inputs $F, X^\star$ and authorisation policy P.

2. $\mathcal{A}_{ABE}$ defines its own challenge inputs for the IND-sHRSS game by setting $t^\star = 1$, $\omega^\star = A_{X^\star}$ and $\mathbb{S}^\star = P \wedge \{l(F)\}$, and sends these to $\mathcal{C}$.

3. $\mathcal{C}$ runs the Setup algorithm for the DPABE scheme and returns the public parameters MPK_{ABE} to $\mathcal{A}_{ABE}$ who stores them as MPK^0_{ABE}.

4. $\mathcal{A}_{ABE}$ now simulates the $\mathsf{HPVC.Setup}$ algorithm such that the output is consistent with MPK^0_{ABE}. It runs Algorithm 1 as written, with the exception of line 1, since MSK^0_{ABE} and MPK^0_{ABE} are defined to be those generated by $\mathcal{C}$. $\mathcal{A}_{ABE}$ also runs $\mathsf{HPVC.FnInit}$ and sends an empty revocation list $\overline{R} = \epsilon$ to $\mathcal{C}$.

5. $\mathcal{A}_{ABE}$ must next create the challenge input for $\mathcal{A}_{VC}$. To do so, it chooses three distinct messages, m_0, m_1 and m', uniformly at random from the messagespace. It also chooses a random bit b which it defines to be $RK_{F,X^\star}$. It sends m_0 and m_1 to $\mathcal{C}$ as its challenge inputs in the IND-sHRSS game. $\mathcal{C}$ chooses a bit $b^\star$ uniformly at random and returns $CT^\star \xleftarrow{\$} \mathsf{Encrypt}(m_{b^\star}, \omega^\star, \mathbb{S}^\star, t^\star, MPK^0_{\text{ABE}})$. $\mathcal{A}_{ABE}$ sets c_b to be $CT^\star$ and generates c_{1-b} himself by encrypting m' as specified on lines 4 and 5 of Algorithm 5. Finally, $\mathcal{A}_{ABE}$ chooses another bit s uniformly at random and, if $b = 0$, it sets $VK_{F,X^\star} = (g(m_s), g(m'), L_{\text{Reg}})$, or otherwise sets $VK_{F,X^\star} = (g(m'), g(m_s), L_{\text{Reg}})$.

6. $\mathcal{A}_{ABE}$gives the resulting parameters to $\mathcal{A}_{VC}$ along with oracle access which $\mathcal{A}_{ABE}$ can handle as follows:

- Queries to $\mathsf{HPVC.FnInit}$ and $\mathsf{HPVC.Register}$ can be run as written.
- Queries of the form $\mathsf{HPVC.Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Oracle 3. If the queried ψ satisfies the challenge authorisation policy P then $\mathcal{A}_{ABE}$ returns $\perp$. Otherwise it simulates running $\mathsf{HPVC.Certify}$ by running Algorithm 4 as written with the exception of lines 4 and 5, as these depend on MSK^0_{ABE} held by $\mathcal{C}$. To simulate line 4, $\mathcal{A}_{ABE}$ queries $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_k \in L_i} l_k), MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ will return $\perp$ if $(\omega^\star \in \mathbb{O})$ *and* $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \mathbb{S}^\star)$ *and* $(S_i \notin \overline{R})$. However, for the query to be made, $\mathcal{A}_{ABE}$ must not have returned $\perp$ in Oracle 3, and therefore $\psi \notin P$, and so $\psi \notin \mathbb{S}^\star$. Hence, $\mathcal{C}$ can return a valid decryption key SK^0_{ABE}.
  To simulate line 5, $\mathcal{A}_{ABE}$ queries $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. $\mathcal{C}$ responds as in Oracle 5 and returns a valid update key *unless* $t = t^\star$ *and* $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, as $\overline{R}$ was chosen to be empty, $\overline{R} \subseteq Q_{\text{Rev}}$ for any Q_{Rev}, and hence $\mathcal{C}$ can create a valid update key.
- Queries of the form $\mathsf{HPVC.Revoke}(\tau_{F(X)}, \text{MK}, \text{PP})$: $\mathcal{A}_{ABE}$ runs Algorithm 9 as written with the exception of line 4. To simulate this line, $\mathcal{A}_{ABE}$ makes a query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(L_{\text{Rev}}, t, MSK^0_{\text{ABE}}, MPK^0_{\text{ABE}})$. As specified in Oracle 5, $\mathcal{C}$ will return a valid key unless $t = t^\star$ and $\overline{R} \not\subseteq R$). However, $\mathcal{A}_{ABE}$ chose $\overline{R}$ to be empty and so it is certainly a

subset of any R, and in particular L_{Rev}. Hence, for all queries, $\mathcal{C}$ can form a valid update key and $\mathcal{A}_{ABE}$ can simulate the expected behaviour.

7. Eventually, $\mathcal{A}_{VC}$ finishes this query phase and outputs a result $\theta^\star$ corresponding to $F(X^\star)$, where $\mathcal{A}_{VC}$ never received a key for a set of authorisation attributes $s \in P$. As $\theta^\star$ should appear valid, it should comprise exactly one non-$\perp$ element which we denote by y.

8. If $g(y) = g(m_s)$, $\mathcal{A}_{ABE}$ guesses $b' = s$. If $g(y) = g(m')$, $\mathcal{A}_{ABE}$ randomly guesses $b' \xleftarrow{\$} \{0, 1\}$ (as $\mathcal{A}_{VC}$ did not respond for either m_0 or m_1, it reveals no information to aid $\mathcal{A}_{ABE}$ in its IND-sHRSS game). Otherwise, $\mathcal{A}_{ABE}$ aborts as $\mathcal{A}_{VC}$ was not successful (either for legitimate reasons or because $\mathcal{A}_{ABE}$ chose s incorrectly and issued a malformed challenge).

By assumption, $\mathcal{A}_{VC}$ has non-negligible advantage δ in the selective authorised computation game – $\Pr[g(y) = g(m_s)] + \Pr[g(y) = g(m')] = \delta$. Therefore,

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

Hence, $Adv_{\mathcal{A}_{ABE}} \geq \left|\Pr[b^\star = b'] - \frac{1}{2}\right| \geq \left|\frac{\delta}{2} - \frac{1}{2}\right| \geq \frac{1}{2}(\delta - 1)$, which is non-negligible. However, as the DPABE scheme was assumed to be IND-sHRSS secure, such an adversary can not exist. Therefore, our assumption on $\mathcal{A}_{VC}$ must be incorrect, and no adversary with non-negligible advantage against the selective authorised computation game can exist. □

D Additional Details for Revocable Dual-policy Attribute-based Encryption

Revocation in ABE schemes was introduced by Attrapadung and Imai [4], and supports two different modes: *direct revocation* and *indirect revocation*. Direct revocation allows users to specify a revocation list at the point of encryption such that periodic re-keying is not required, but encryptors must have knowledge of the current revocation list. In contrast, indirect revocation requires a time period to be specified at the point of encryption and an authority to issue updated key material at each time period to enable non-revoked entities to update their key to be functional during that time period. With the HPVC setting in mind, we choose to focus on *indirect* revocation to minimise the workload of the client devices in terms of maintaining synchronised revocation lists.

To implement a revocation mechanism in the KP-ABE setting, the policy is amended to include an identifier of the entity owning the key, and the current time period is embedded into the ciphertext. Update keys are issued for all non-revoked identities at each time period which are used in combination with the decryption key to decrypt ciphertexts formed for particular time periods – only if the entity was issued an update key for time t (i.e. was not revoked) can they decrypt ciphertexts formed using t. We observe that, to define a *revocable DP-ABE* scheme, the revocation mechanism can be embedded either, as above, in the KP-ABE functionality *or* in the CP-ABE functionality. In more detail, decryption in DP-ABE is successful if and only if *both* attribute sets satisfy their corresponding access structure. To prevent decryption, therefore,

Game 5 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{RKDPABE}, 1^{\ell}, \mathcal{U}\right]$

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

Oracle 4 $\mathcal{O}^{\mathsf{KeyGen}}(\text{ID}, (\mathbb{O}, \psi), \text{MK}, \text{PP})$:

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

Oracle 5 $\mathcal{O}^{\mathsf{KeyUpdate}}(R, t, \text{MK}, \text{PP})$:

1: **if** $((t = t^{\star})$ **and** $(\tilde{R} \sqsubseteq R))$ **then return** $\perp$
2: **return** $\mathsf{KeyUpdate}(R, t, \text{MK}, \text{PP})$

at least one attribute set should not satisfy the corresponding access structure. In this work, we consider revocable DP-ABE using indirect revocation in the key-policy. Future work will compare the efficiency of the two approaches.

Definition 8. *An rkDPABE scheme is* correct *if for all messages $m \in \mathcal{M}$, for all access structures $\mathbb{O}, \mathbb{S} \subseteq 2^{\mathcal{U}} \setminus \{\emptyset\}$, and for all attribute sets $\omega, \psi \subseteq \mathcal{U}$ where $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$,*

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

The security model for rkDPABE is a natural extension of the IND-sHRSS game for an indirectly revocable KP-ABE scheme and is presented in Game 5 and Oracles 4 and 5.

Definition 9. *The* advantage *of a PPT adversary $\mathcal{A}$ in the* IND-sHRSS *game for an rkDPABE construction $\mathcal{DPABE}$ is defined as:*

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

An rkDPABE scheme is indistinguishable against selective-target with semi-static query attack (IND-sHRSS) *if for all PPT adversaries $\mathcal{A}$,*

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

D.1 Preliminaries

In this section we introduce some preliminary notions we require in order to construct a Revocable Dual-policy Attribute-based Encryption scheme.

Access Structures and Linear Secret Sharing

Here we define access structures and linear secret sharing schemes recapped from [26]. These are fundamental building blocks for ABE schemes.

Definition 10 (Access Structure)**.** *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of parties (or attributes). A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if for all B, C we have that if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (resp., monotonic access structure) is a collection (resp., monotone collection) $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the* authorised sets *and the sets not in $\mathbb{A}$ are called* unauthorised sets.

Definition 11 (Linear Secret Sharing Schemes (LSSS))**.** *Let $\mathcal{P}$ be a set of parties. Let M be a matrix of size $l \times k$. Let $\pi\colon \{1, \ldots, l\} \to \mathcal{P}$ be a function that maps a row to a party for labeling. A secret sharing scheme Π for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$ is a linear secret-sharing scheme in $\mathbb{Z}_p$ and is represented by (M, π) if it consists of two polynomial-time algorithms:*

- $\text{Share}_{(M,\pi)}$*: The algorithm takes as input $s \in \mathbb{Z}_p$ which is to be shared. It randomly chooses $y_2, \ldots, y_k \in \mathbb{Z}_p$ and let $\boldsymbol{v} = (s, y_2, \ldots, y_k)$. It outputs $M\boldsymbol{v}$ as a vector of l shares. The share $\lambda_{\pi(i)} := \boldsymbol{M_i} \cdot \boldsymbol{v}$ belongs to party $\pi(i)$, where we denote $\boldsymbol{M_i}$ as the ith row in M.*
- $\text{Recon}_{(M,\pi)}$*: The algorithm takes as input an authorised set $S \in \mathbb{A}$. Let $I = \{i : \pi(i) \in S\}$. It outputs reconstruction constants $\{(i, \mu_i)\}_{i \in I}$ such that the secret can be reconstructed as $s = \sum_{i \in I} \mu_i \cdot \lambda_{\pi(i)}$.*

We require the following important fact [26] in the proof of our construction (cf. Appendix D.3).

Proposition 1. *Let (M, π) be a LSSS for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$, where M is a matrix of size $l \times k$. For all unauthorised sets $S \notin \mathbb{A}$, there exists a polynomial time algorithm that outputs a vector $\boldsymbol{w} = (w_1, \ldots, w_k) \in \mathbb{Z}_p^k$ such that $w_1 = -1$ and for all $i \in I$ it holds that $\boldsymbol{M_i} \cdot \boldsymbol{w} = 0$.*

In Appendix D.2 we make use of Lagrange interpolation as the reconstruction algorithm for LSSSs.

Definition 12 (Lagrange Interpolation)**.** *For $i \in \mathbb{Z}$ and $S \subseteq \mathbb{Z}$, the Lagrange basis polynomial is defined as $\Delta_{i,S}(z) = \prod_{j \in S, j \neq i} \frac{z-j}{i-j}$. Let $f(z) \in \mathbb{Z}[z]$ be a d-th degree polynomial. If $|S| = d+1$, from a set of $d+1$ points $\{(i, f(i))\}_{i \in S}$, one can reconstruct $f(z)$ as*

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

In our scheme in Appendix D.2, we especially use the interpolation for a first degree polynomial. In particular, let $f(z)$ be a first degree polynomial, one can obtain $f(0)$ from two points $(i_1, f(i_1)), (i_2, f(i_2))$ where $i_1 \neq i_2$ by computing

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

Bilinear Maps and Hardness Assumptions

Here we review the notions of bilinear maps and the hardness assumptions on which we base the security of our scheme. We follow the formlisation in [4, 5].

Definition 13 (Bilinear Map)**.** *Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative groups of order p, with g a generator of $\mathbb{G}$. A* bilinear map *is a map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where:*

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

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

Definition 14. *Let $\mathbb{G}$ be a bilinear group of prime order p. The* Decisional q Bilinear Diffie-Hellman Exponent problem *(q-BDHE) in $\mathbb{G}$ is as follows. Given a vector*

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

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

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

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

Terminology for Binary Trees

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

D.2 Construction

Our revocable DP-ABE scheme will be based on a combination of DP-ABE [6], which itself is a combination of CP-ABE [26] and KP-ABE [19], and an ABE scheme supporting revocation [4]. We represent a subjective access structure $\mathbb{S}$ by a linear secret sharing scheme (LSSS) which we denote by (M, ρ) and represent an objective access structure $\mathbb{O}$ as an LSSS denoted by (N, π).

Let $\mathcal{U}_s$ and $\mathcal{U}_o$ be the universe of subjective and objective attributes respectively. The objective attribute universe comprises disjoint sub-universes $\mathcal{N}, \mathcal{T}, \mathcal{M}$ and $\mathcal{U}_{\mathrm{ID}}$ referring to standard ABE attributes, time periods, messages and user identities respectively. $\mathcal{U}_{\mathrm{ID}}$ is set to be the set of leaves in a complete binary tree $\mathcal{X} = \{1, \ldots, n\}$. Without loss of generality, we assume that $\mathcal{T} \cap \mathcal{X} = \emptyset$ (e.g. by using a collision resistant hash function and using distinct prefixes to map elements from $\mathcal{T}$ and $\mathcal{X}$). The attribute set for the DP-ABE scheme is defined to be $\mathcal{U} = \mathcal{U}_s \cup \mathcal{U}_o$. Let us define m to be the maximum size of a subjective attribute set assigned to a key, i.e. we restrict $|\psi| \leqslant m$, and similarly define n to be the maximum size of an objective attribute set associated with a ciphertext, i.e. $|\omega| \leqslant n$. Furthermore we denote the maximum number of rows of a subjective access structure matrix M to be $l_{s,\mathbf{max}}$. Now let $m' = m + l_{s,\mathbf{max}} - 1$ and $n' = n - 1$. Finally, let d be the maximum of $|\mathsf{Cover}(R)|$ for all $R \subseteq \mathcal{U}_{\mathrm{ID}}$, where $\mathsf{Cover}(R)$ is defined as in Appendix D.1.

- $\mathsf{Setup}(1^\ell, \mathcal{U})$: The algorithm picks random exponents $\gamma, \alpha \in \mathbb{Z}_p$ and a generator $g \in \mathbb{G}$. It defines three functions $F_s\colon \mathbb{Z}_p \to \mathbb{G}$, $F_o\colon \mathbb{Z}_p \to \mathbb{G}$ and $P\colon \mathbb{Z}_p \to \mathbb{G}$ by randomly choosing $h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d$ and setting

$$F_s(x) = \prod_{j=0}^{m'} h_j^{x^j}, \quad F_o(x) = \prod_{j=0}^{n'} q_j^{x^j}, \quad P(x) = \prod_{j=0}^{d} u_j^{x^j}. \tag{3}$$

The public parameters are defined as

$$\text{PP} = (g, e(g,g)^{\gamma}, g^{\alpha}, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d).$$

For each node label $x \in \mathcal{X}$ in the tree, it randomly chooses $a_x \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ to define a first degree polynomial $f_x(z) = a_x z + \alpha r_x + \gamma$. The master key is $\text{MK} = (\gamma, \alpha, \{a_x, r_x\}_{x \in \mathcal{X}})$.

- $\mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, \text{PP})$: The encryption algorithm takes as input a LSSS access structure (M, ρ) for the subjective policy $\mathbb{S}$ and an objective attribute set $\omega \subset \mathcal{U}_o$. Denote the dimensions of M as $l_s \times k_s$ matrix. The algorithm randomly chooses values $s, y_2, \ldots, y_{k_s} \in \mathbb{Z}_p$ and sets $\boldsymbol{u} = (s, y_2, \ldots, y_{k_s})$. It computes $\lambda_i = \boldsymbol{M_i} \cdot \boldsymbol{u}$ (for $i = 1, \ldots, l_s$), where $\boldsymbol{M_i}$ is the vector corresponding to the ith row of M. The ciphertext is then computed as $CT = (C, C^{(1)}, \{C_k^{(2)}\}_{k \in \omega}, \{C_i^{(3)}\}_{i=1,\ldots,l_s}, C^{(4)})$, where

$$\begin{aligned} C &= m \cdot (e(g,g)^{\gamma})^s, & C^{(1)} &= g^s, \\ C_k^{(2)} &= F_o(k)^s, & C_i^{(3)} &= g^{\alpha\lambda_i} F_s(\rho(i))^{-s}, \\ C^{(4)} &= P(t)^s. \end{aligned}$$

  Intuitively, C masks the message by a group element in the target group of the bilinear map formed from the master secret γ and an encryption secret s (to randomise the encryption procedure). Decryption will have to compute this mask to recover the message.

  $C^{(1)}$ provides the encryption secret s. $C_k^{(2)}$ embeds each attribute in the objective set ω into the ciphertext, incorporating the encryption secret s such that attributes from prior ciphertexts cannot be combined with this encryption. Similarly, $C_i^{(3)}$ embeds the subjective policy $\mathbb{S}$ into the ciphertext using the shares of s divided according to $\mathbb{S}$ – that is, s is shared over the set of attributes such that any set of attributes that satisfies $\mathbb{S}$ can reconstruct the encryption secret s. Finally, $C^{(4)}$ links the encryption secret (and hence this particular ciphertext) to the specified time period t such that an update key for t is required to decrypt the ciphertext; this enables the revocation mechanism.

- $\mathsf{KeyGen}(\text{ID}, (\mathbb{O}, \psi), \text{MK}, \text{PP})$: The key generation algorithm takes as input a LSSS access structure (N, π) for the objective policy $\mathbb{O}$ and a subjective attribute set $\psi \subset \mathcal{U}_s$. Let the dimensions of N be denoted $l_o \times k_o$. The algorithm also takes an identity $\text{ID} \in \mathcal{U}$ which is a leaf in the binary tree.

  For all $x \in \mathsf{Path}(\text{ID})$, the algorithm shares $f_x(1)$ using the LSSS (N, π). To do so, it randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and sets $\boldsymbol{v_x} = (f_x(1), z_{x,2}, \ldots, z_{x,k_o})$. For $i = 1, \ldots, l_o$, it calculates the share $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$, where $\boldsymbol{N_i}$ is the vector corresponding to the ith row of N.

  The algorithm then randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ for all $x \in \mathsf{Path}(\text{ID})$, and outputs the private key

$$SK_{(N,\pi),\text{ID}} = ((D_{x,i}^{(1)}, D_{x,i}^{(2)})_{x \in \mathsf{Path}(\text{ID}), \text{i}=1,\ldots,\text{l}_\text{o}}, (D_x, \{D_k^{(3)}\}_{k \in \psi})_{x \in \mathsf{Path}(\text{ID})}),$$

  where

$$\begin{aligned} D_x &= g^{r_x}, & D_{x,i}^{(1)} &= g^{r_{x,i}}, \\ D_{x,i}^{(2)} &= g^{\sigma_{x,i}} F_o(\pi(i))^{r_{x,i}}, & D_k^{(3)} &= F_s(k)^{r_x}. \end{aligned}$$

  Intuitively, r_x and $r_{x,i}$ for each $x \in \mathsf{Path}(\text{ID})$ randomises the key for the user ID (so that users may not collude). D_x and $D_{x,i}^{(1)}$ allow use of these random key values during decryption.

$D_{x,i}^{(2)}$ embeds the shares of $f_x(1) = a_x + \alpha r_x + \gamma$ such that only the authorised sets according to $\mathbb{O}$ may reconstruct $f_x(1)$. Finally, $D_k^{(3)}$ embeds the attributes in ψ with the randomness chosen for this particular key. By linking these parameters to the path in a tree, only users for whom a valid update key has been issued (i.e. the non-revoked users) will be able to make use of these parameters to compute $f_x(1)$ for a node x; $f_x(1)$ is required as it contains the master secret γ which is used to cancel with the ciphertext component C to recover the message.

- $\mathsf{KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP})$: The algorithm first computes $\mathsf{Cover}(R)$ to find a minimal node set that covers $\mathcal{U} \setminus R$. For each $x \in \mathsf{Cover}(R)$, it randomly chooses $r_x \in \mathbb{Z}_p$ and sets the update key as $UK(R,t) = \left\{U_x^{(1)}, U_x^{(2)}\right\}_{x \in \mathsf{Cover}(R)}$, where

$$U_x^{(1)} = g^{f_x(t)} P(t)^{r_x}, \quad U_x^{(2)} = g^{r_x}.$$

  Intuitively, each update key component is randomised by r_x and linked to a particular node x in the tree (covering only non-revoked users). $P(t)$ embeds the current time period which will match with the ciphertext component $C^{(4)}$. We also embed a point of the polynomial $f_x(t)$; given this point, and the point $f_x(1)$ (which can be recovered from the decryption key components $D_{x,i}^{(2)}$ given a satisfying set of objective attributes ω), one can perform Lagrange interpolation to recover the point $f_x(0)$ which will yield use of the master secret γ to cancel with the ciphertext component C.

- $\mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), SK_{(\mathbb{O},\psi),\mathrm{ID}}, (\mathbb{O}, \psi), UK_{R,t}, \mathrm{PP})$: The decryption algorithm takes as an input the ciphertext CT which contains a subjective access structure (M, ρ) for $\mathbb{S}$ and a set of objective attributes ω, and a decryption key $SK_{(N,\pi),\mathrm{ID}}$ which contains a set of subjective attributes ψ and an objective access structure (N, π) for $\mathbb{O}$. Suppose that ψ satisfies (M, ρ), the set ω satisfies (N, π), and that $\mathrm{ID} \notin \mathrm{R}$ (so that decryption is possible).

  Let $I_s = \{i : \rho(i) \in \psi\}$ and $I_o = \{i : \pi(i) \in \omega\}$. The algorithm computes sets of reconstruction constants $\{(i, \mu_i)\}_{i \in I_s}$ and $\{(i, \nu_i)\}_{i \in I_o}$ using the LSSS reconstruction algorithm. Since $\mathrm{ID} \notin \mathrm{R}$, the algorithm also finds a node x such that $x \in \mathsf{Path}(\mathrm{ID}) \cap \mathsf{Cover}(\mathrm{R})$. Finally, it computes the following

$$C \cdot \frac{\prod_{i \in I_s} \left(e(C_i^{(3)}, D_x) \cdot e(C^{(1)}, D_{\rho(i)}^{(3)}) \right)^{\mu_i}}{\left(\prod_{j \in I_o} \left(\frac{e(D_{x,j}^{(2)}, C^{(1)})}{e(C_{\pi(j)}^{(2)}, D_{x,j}^{(1)})} \right)^{\nu_j} \right)^{\frac{t}{t-1}} \left(\frac{e(U_x^{(1)}, C^{(1)})}{e(C^{(4)}, U_x^{(2)})} \right)^{\frac{1}{1-t}}} = m.$$

We verify the correctness of the decryption as follows. Let us write the decryption computation as $C \cdot \frac{C'}{K}$, where $K = (K')^{\frac{t}{t-1}} (K'')^{\frac{1}{1-t}}$, and then consider each part in turn. Intuitively, C' is similar to a standard ABE decryption operation to match attributes to policies, whilst K' and K'' combine the two components of a functional decryption key (namely, a secret key and an update key) and perform a Lagrange interpolation to form a group element $e(g,g)^{s(\gamma + \alpha r_x)} = e(g,g)^{s\gamma} \cdot e(g,g)^{s\alpha r_x}$. The second part of this product will be the result of

computing C' whilst the first will cancel with C to leave only m.

$$\begin{aligned}
C' &= \prod_{i \in I_s} \left(e(C_i^{(3)}, D_x) \cdot e(C^{(1)}, D_{\rho(i)}^{(3)}) \right)^{\mu_i} \\
&= \prod_{i \in I_s} \left(e(g^{\alpha \lambda_i} F_s(\rho(i))^{-s}, g^{r_x}) \cdot e(g^s, F_s(\rho(i))^{r_x}) \right)^{\mu_i} \\
&= \prod_{i \in I_s} \left(e(g,g)^{\alpha \lambda_i r_x} \cdot e(g, F_s(\rho(i)))^{-r_x s} \cdot e(g, F_s(\rho(i)))^{r_x s} \right)^{\mu_i} \\
&= e(g,g)^{\alpha r_x \sum_{i \in I_s} \mu_i \lambda_i} \\
&= e(g,g)^{\alpha r_x s}.
\end{aligned}$$

The second equality follows by substituting the values from the construction; the third equality follows from the properties of bilinear maps; the fourth equality simply moves the product into the exponent; and the final equality follows from the reconstruction constants of the LSSS, namely that $\sum_{i \in I_s} \mu_i \lambda_i = s$.

$$\begin{aligned}
K' &= \prod_{j \in I_o} \left(\frac{e(D_{x,j}^{(2)}, C^{(1)})}{e(C_{x,\pi(j)}^{(2)}, D_{x,j}^{(1)})} \right)^{\nu_j} = \prod_{j \in I_o} \left(\frac{e(g^{\sigma_{x,j}} F_o(\pi(j))^{r_{x,j}}, g^s)}{e(F_o(\pi(j))^s, g^{r_{x,j}})} \right)^{\nu_j} \\
&= \prod_{j \in I_o} \left(\frac{e(g,g)^{\sigma_{x,j} s} \cdot e(g, F_o(\pi(j)))^{r_{x,j}, s}}{e(g, F_o(\pi(j)))^{r_{x,j}, s}} \right)^{\nu_j} \\
&= e(g,g)^{s \sum_{j \in I_o} \nu_j \sigma_{x,j}} = e(g,g)^{s f_x(1)}.
\end{aligned}$$

The second equality follows directly from the construction; the third equality follows from the properties of bilinear maps; the fourth equality stems from moving the product into the exponent; and the last equality follows from the set of LSSS reconstruction constants with $\sum_{j \in I_o} \nu_j \sigma_{x,j} = f_x(1) = a_x + \alpha r_x + \gamma$.

$$\begin{aligned}
K'' &= \frac{e(U_x^{(1)}, C^{(1)})}{e(C^{(4)}, U_x^{(2)})} = \frac{e(g^{f_x(t)} P(t)^{r_x}, g^s)}{e(P(t)^s, g^{r_x})} = \frac{e(g,g)^{f_x(t) s} \cdot e(g, P(t)^{r_x s})}{e(g, P(t)^{r_x s})} \\
&= e(g,g)^{f_x(t) s}
\end{aligned}$$

Then,

$$\begin{aligned}
K &= (K')^{\frac{t}{t-1}} (K'')^{\frac{1}{1-t}} = (e(g,g)^{s f_x(1)})^{\frac{t}{t-1}} (e(g,g)^{f_x(t) s})^{\frac{1}{1-t}} \\
&= (e(g,g)^s)^{f_x(1) \frac{t}{t-1} + f_x(t) \frac{1}{1-t}}
\end{aligned}$$

Notice that $f_x(1)\frac{t}{t-1} + f_x(t)\frac{1}{1-t}$ is in fact a Lagrange interpolation for the two points $(1, f_x(1)), (1, f_x(t))$ for the first degree polynomial f_x. Thus, $f_x(1)\frac{t}{t-1} + f_x(t)\frac{1}{1-t} = f_x(0) = \alpha r_x + \gamma$. Hence, $K = e(g,g)^{s(\alpha r_x + \gamma)}$. Combining all of these results, we obtain the result of the decryption operation

$$C \cdot \frac{C'}{K} = m \cdot e(g,g)^{s\gamma} \cdot \frac{e(g,g)^{\alpha s r_x}}{e(g,g)^{s(\alpha r_x + \gamma)}} = m \cdot e(g,g)^{s\gamma} \cdot \frac{e(g,g)^{\alpha s r_x}}{e(g,g)^{s\gamma} \cdot e(g,g)^{\alpha s r_x}} = m.$$

D.3 Proof of Security

Theorem 2. *The rkDPABE construction presented in Appendix D.2 is secure with respect to Indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*) assuming that the Decisional q-BDHE problem is hard.*

The proof follows a combination of [4] and [5] with some adjustment in the simulation of the private keys. We show that if an adversary can win the IND-sHRSS game with advantage ϵ with a challenge subjective access structure matrix of size $l_s^\star \times k_s^\star$, then a simulator with advantage ϵ in solving the Decisional q-BDHE problem can be constructed, where $m + k_s^\star \leqslant q$.

Proof. Suppose, to achieve a contradiction with Theorem 2, that there exists an adversary $\mathcal{A}$ that has an advantage ϵ in attacking the rkDPABE scheme. We build a simulator $\mathcal{B}$ that solves the Decisional q-BDHE problem (see Definition 14) in $\mathbb{G}$. Recall that we denote g^{a^j} by g_j. The simulator $\mathcal{B}$ is given a random q-BDHE challenge $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ where $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ where Z is either $e(g_{q+1}, h)$ or a random element in $\mathbb{G}_1$. $\mathcal{B}$ acts as the challenger for $\mathcal{A}$ in the IND-sHRSS game (Game 5) as follows.

1. $\mathcal{A}$ begins by selecting its challenge parameters $(t^\star, \omega^\star, \mathbb{S}^\star)$ where $\mathbb{S}^\star$ is represented by an LSSS $(M^\star, \rho^\star)$. Let the matrix $M^\star$ be of size $l_s^\star \times k_s^\star$, where $m + k_s^\star \leq q$ and let $l_s^\star = l_{s,\mathbf{max}}$ and $|\omega^\star| = n$.

2. $\mathcal{B}$ now simulates running Setup for the rkDPABE scheme, and embeds the challenge policy into the public parameters. It first chooses $\gamma' \xleftarrow{\$} \mathbb{Z}_p$, sets $g^\alpha = g_1 = g^a$, and implicitly defines $\gamma = \gamma' + a^{q+1}$ by defining

$$\begin{aligned} e(g,g)^\gamma &= e(g_1, g_q) \cdot e(g,g)^{\gamma'} = e(g^a, g^{a^q}) \cdot e(g,g)^{\gamma'} \\ &= e(g,g)^{\gamma' + a^{q+1}}. \end{aligned}$$

It then must define the polynomials F_s, F_o and P (as in [4] and [5]). To define F_s, $\mathcal{B}$ begins by defining $F_s(x) = g^{p(x)}$, where p is a polynomial in $\mathbb{Z}_p[x]$ of degree $m + l_s^\star - 1$ which is implicitly defined in the following manner. It chooses $k_s^\star + m + 1$ polynomials $p_0, \ldots, p_{k_s^\star + m}$ in $\mathbb{Z}_p[x]$, each of degree $m + l_s^\star - 1$, such that for all $x = \rho^\star(i)$ for some i (i.e. all x in the image of $\rho^\star$, of which there are exactly $l_s^\star$ since $\rho^\star$ is an injective mapping):

$$p_j(x) = \begin{cases} M^\star_{i,j} & \text{for } j \in [1, k_s^\star] \\ 0 & \text{for } j \in [k_s^\star + 1, k_s^\star + m] \end{cases} \tag{4}$$

The polynomial p_0 is chosen randomly, and for all other x (not in the image of $\rho^\star$), p_j is defined randomly by randomly choosing values at m other points). By writing the coefficients of each polynomial as $p_j(x) = \sum_{i=0}^{m+l_s^\star - 1} p_{j,i} \cdot x^i$, one can define the polynomial $p(x)$ to be

$$p(x) = \sum_{j=0}^{k_s^\star + m} p_j(x) a^j. \tag{5}$$

Then, $\mathcal{B}$ sets $h_i = \prod_{j=0}^{k_s^\star + m} g_j^{p_{j,i}}$ for $i \in [0, m + l_s^\star - 1]$ Finally, as we assumed $l_s^\star = l_{s,\mathbf{max}}$, note

that $m' = m + l_{s,\mathbf{max}} - 1 = m + l_s^\star - 1$,

$$\begin{aligned}
F_s(x) &= \prod_{i=0}^{m'} h_i^{x^i} && \text{(by 3)} \\
&= \prod_{i=0}^{m'} \left(\prod_{j=0}^{k_s^\star + m} g_j^{p_{j,i}} \right)^{x^i} && \text{(by definition of } h_i^{x^i}\text{)} \\
&= \prod_{i=0}^{m'} \left(\prod_{j=0}^{k_s^\star + m} g^{p_{j,i} a^j} \right)^{x^i} && \text{(by definition of } g_j = g^{a^j}\text{)} \\
&= g^{\sum_{j=0}^{k_s^\star+m} \sum_{i=0}^{m'} p_{j,i} x^i a^j} = g^{\sum_{j=0}^{k_s^\star+m} p_j(x) a^j} \\
&= g^{p(x)} && \text{(by 5)}
\end{aligned}$$

To define F_o, $\mathcal{B}$ randomly picks a polynomial $f'(x) = \sum_{j=0}^{n-1} f'_j x^j$ in $\mathbb{Z}_p[x]$ of degree $n-1$. It then defines $f(x) = \prod_{k \in \omega^\star}(x-k) = \sum_{j=0}^{n-1} f_j x^j$ (which can be computed entirely from $\omega^\star$); note that $f(x) = 0$ if and only if $x \in \omega^\star$. It defines $q_j = g_q^{f_j} g^{f'_j}$ for $j = [0, n-1]$. Finally,

$$F_o(x) = \prod_{j=0}^{n-1} q_j^{(x^j)} = g_q^{f(x)} g^{f'(x)}.$$

To define P, $\mathcal{B}$ defines

$$\hat{p}(y) = y^{d-1} \cdot (y - t^\star) = \sum_{j=0}^{d} \hat{p}_j y^j.$$

This ensures $\hat{p}(t) = 0$ if and only if $t = t^\star$ for $t \in \mathcal{T}$, and that for $x \in \mathcal{X}$, $\hat{p}(x) \neq 0$ since we assumed $\mathcal{T} \cap \mathcal{X} = \emptyset$.
$\mathcal{B}$ then randomly picks a degree d polynomial $\rho(y) = \sum_{j=0}^{d} \rho_j y^j$ in $\mathbb{Z}_p[x]$ and lets $u_j = (g^a)^{\hat{p}_j} g^{\rho_j}$ for $j = 0, \ldots, d$. Thus,

$$P(y) = \prod_{j=0}^{d} u_j^{y^j} = (g^a)^{\hat{p}(y)} g^{\rho(y)}. \tag{6}$$

The public key PK for the DPABE scheme is defined to be

$$\text{PK} = (g, e(g,g)^\gamma, g^\alpha, h_0, \ldots, h_{m'}, q_1, \ldots, q_{n'}, u_1, \ldots, u_d),$$

which is given to $\mathcal{A}$. Note that the randomness of the q-BDHE challenge $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ and the independently chosen randomness used in the construction of the polynomials p_j, f', and ρ ensure the public parameters are distributed as expected.

3. $\mathcal{A}$ declares its list $\overline{R}$ and is then given oracle access to the KeyGen and KeyUpdate functions. Let $\mathcal{X}_{\overline{R}} = \{x \in \mathsf{Path}(\mathrm{ID}) : \mathrm{ID} \in \overline{\mathrm{R}}\}$. For each node label $x \in \mathcal{X}$ in the tree, $\mathcal{B}$ randomly chooses $a'_x \in \mathbb{Z}_p$ and implicitly defines

$$a_x = \begin{cases} a'_x - \alpha r_x - \gamma & \text{if } x \in \mathcal{X}_{\overline{R}} \\ a'_x - \frac{\alpha r_x - \gamma}{t^\star} & \text{if } x \notin \mathcal{X}_{\overline{R}} \end{cases} \tag{7}$$

Hence,

$$f_x(1) = a_x + \alpha r_x + \gamma = a'_x - \alpha r_x - \gamma + \alpha r_x + \gamma = a'_x \qquad \text{if } x \in \mathcal{X}_{\overline{R}} \qquad (8)$$

$$f_x(t^\star) = a_x t^\star + \alpha r_x + \gamma = (a'_x - \frac{\alpha r_x - \gamma}{t^\star})t^\star + \alpha r_x + \gamma = a'_x t^\star \qquad \text{if } x \notin \mathcal{X}_{\overline{R}} \qquad (9)$$

To simulate KeyGen queries for an objective access structure (N, π), a subjective attribute set ψ and an identity ID, we consider the following cases:

- $(\omega^\star \in \mathbb{O})$ and $(\mathrm{ID} \in \overline{\mathrm{R}})$:
  For each $x \in \mathsf{Path}(\mathrm{ID})$, note that since $\mathrm{ID} \in \overline{\mathrm{R}}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (8), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathrm{ID})$. $\mathcal{B}$ can therefore compute the key components precisely as in the construction by sharing the value of $f_x(1)$.

- $(\omega^\star \notin \mathbb{O})$ and $(\mathrm{ID} \in \overline{\mathrm{R}})$:
  For each $x \in \mathsf{Path}(\mathrm{ID})$,note that, since $\mathrm{ID} \in \overline{\mathrm{R}}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (8), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathrm{ID})$.
  $\mathcal{B}$ randomly choses $r_x \in \mathbb{Z}_p$. It then lets $D_x = g^{r_x}$, and for all $k \in \psi$ lets $D_k^{(3)} = F_s(k)^{r_x}$ as in the construction. Recall that the dimensions of N are $l_0 \times k_0$. Since $\omega^\star$ does not satisfy N for this case of the query, and by Proposition 1, there exists a vector $\boldsymbol{a_x} = (a_1, \ldots, a_{k_o}) \in \mathbb{Z}_p^{k_o}$ such that $a_1 = -1$ and $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$ for all i where $\pi(i) \in \omega^\star$.
  $\mathcal{B}$ randomly chooses $z'_{x,2}, \ldots, z'_{x,k_o} \in \mathbb{Z}_p$ and defines $\boldsymbol{v'_x} = (0, z'_{x,2}, \ldots, z'_{x,k_o})$. It then implicitly defines a vector $\boldsymbol{v_x} = -(a'_x)\boldsymbol{a_x} + \boldsymbol{v'_x}$ (by using 4) which will be used for creating the share of $f_x(1) = \gamma + \alpha r_x + a_x$ (note that the first element of $\boldsymbol{v_x}$ is indeed $f_x(1)$ by (8)), as in our construction.
  Now, for all i such that $\pi(i) \in \omega^\star$, $\mathcal{B}$ randomly chooses $r_{x,i} \in \mathbb{Z}_p$ and computes $D_{x,i}^{(1)} = g^{r_{x,i}}$ and
  $$D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}},$$
  where the last equality holds because $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$. Note that $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$ in our construction and hence $D_{x,i}^{(2)}$ is of the valid form.
  For all other i, where $\pi(i) \notin \omega^\star$, $\mathcal{B}$ randomly chooses $r'_{x,i} \in \mathbb{Z}_p$. Observe that

  $$\begin{aligned} \boldsymbol{N_i} \cdot \boldsymbol{v_x} &= \boldsymbol{N_i} \cdot (-(a'_x)\boldsymbol{a_x} + \boldsymbol{v'_x}) \\ &= \boldsymbol{N_i} \cdot (\boldsymbol{v'_x} - (a'_x)\boldsymbol{a_x}) \end{aligned}$$

  Note that, unlike [5], due to our definition of a_x, we do not have a term in a^{q+1} here, and $\mathcal{B}$ can generate $D_{x,i}^{(2)} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}}$ and $D_{x,i}^{(1)} = g^{r_{x,i}}$.

- $(\psi \notin \mathbb{S}^\star)$ and $(\mathrm{ID} \notin \overline{\mathrm{R}})$:
  For each $x \in \mathsf{Path}(\mathrm{ID})$, $\mathcal{B}$ does the following. Since ψ does not satisfy $M^\star$, by Proposition 1, there exists a vector $\boldsymbol{w_x} = (w_1, \ldots, w_{k_s^\star}) \in \mathbb{Z}_p^{k_s^\star}$ such that $w_1 = -1$ and $M_i \cdot \boldsymbol{w_x} = 0$ for all i where $\rho(i) \in \psi^\star$. Now, by our definition of $p_j(x)$ in (4), we have that $(p_1(x), \ldots, p_{k_s^\star}(x)) \cdot (w_1, \ldots, w_{k_s^\star}) = 0$.
  $\mathcal{B}$ then computes one possible solution of variables $w_{k_s^\star+1}, \ldots, w_{k_s^\star+m}$ for the system of $|\psi|$ equations: for all $x \in \psi$

  $$(p_1(x), \ldots, p_{k_s^\star+m}(x)) \cdot (w_1, \ldots, w_{k_s^\star+m}) = 0,$$

  which is possible as $|\psi| \leqslant m$.

$\mathcal{B}$ then randomly chooses $r'_x \in \mathbb{Z}_p$ and implicitly defines

$$r_x = r'_x + w_1(\frac{t^\star}{t^\star - 1}) \cdot \alpha^q + w_2(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-1} + \cdots + w_{k_s^\star+m}(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-(k_s^\star+m)+1}$$

by setting the key $D_x = g^{r'_x} \prod_{k=1}^{k_s^\star+m} (g_{q+1-k})^{w_k(\frac{t^\star}{t^\star-1})} = g^{r_x}$. Then, since $\gamma = \gamma' + \alpha^{q+1}$ and as $x \notin \mathcal{X}_{\overline{\mathcal{R}}}$, we have

$$\begin{aligned}
f_x(1) &= \gamma + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a'_x - \frac{\alpha r_x - \gamma}{t^\star} \quad \text{by (7)} \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + (\alpha(\frac{t^\star - 1}{t^\star}))r_x \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + (\alpha(\frac{t^\star - 1}{t^\star})(r'_x + w_1(\frac{t^\star}{t^\star - 1}) \cdot \alpha^q \\
&\quad + w_2(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-1} + \cdots + w_{k_s^\star+m}(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-(k_s^\star+m)+1}) \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + (\alpha r'_x + w_2(\frac{t^\star}{t^\star - 1}) \cdot \alpha^q + \cdots + w_{k_s^\star+m}(\frac{t^\star}{t^\star - 1}) \cdot \alpha^{q-(k_s^\star+m)+2})
\end{aligned}$$

where the α^{q+1} term in γ has canceled out. The simulator now randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and implicitly lets $\boldsymbol{v_x} = (\gamma + \alpha r_x + a_x, z_{x,2}, \ldots, z_{x,k_o})$ as in the construction.
$\mathcal{B}$ also randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and computes for $i = 1$ to l_o the key $D^{(1)}_{x,1} = g^{r_{x,i}}$. The other keys are computed in the following way. We have

$$D^{(2)}_{x,i} = \left(g^{\gamma' + a'_x + \frac{\gamma}{t^\star}} \cdot g_1^{r'_x} \prod_{k=2}^{k_s^\star+m} (g_{q-k+2})^{w_k} \right)^{N_{i,1}} \cdot \prod_{j=2}^{k_o} g^{N_{i,j} z_j} F_o(\pi(i))^{r_{x,i}}$$

which can be computed since g_{q+1} is not required and, by collecting the exponents, it can be verified that $D^{(2)}_{x,i} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} \cdot F_o(\pi(i))^{r_i}$.
Recall that $(p_1(k), \ldots, p_{k_s^\star+m}(k)) \cdot (w_1, \ldots, w_{k_s^\star+m}) = 0$ for all $k \in \psi$.

$$\begin{aligned}
D^{(3)}_k &= D_x^{p_0(k)} \prod_{j=1}^{k_s^\star+m} \left(g_j^{r'_x} \prod_{k \in [1, k_s^\star+m], k \neq j} (g_{q+1-k+j})^{w_k} \right)^{p_j(k)} \\
&= (g^{r_x})^{p_0(k)} \prod_{j=1}^{k_s^\star+m} (g^{r_x})^{\alpha^j p_j(k)} = (g^{r_x})^{p(k)} \\
&= F_s(k)^{r_x},
\end{aligned}$$

where the second equality holds by observing that

$$D^{(3)}_k = D^{(3)}_k (g_{q+1})^{(p_1(k), \ldots, p_{k_s^\star+m}(k)) \cdot (w_1, \ldots, w_{k_s^\star+m})}$$

since $(g_{q+1})^{(p_1(k), \ldots, p_{k_s^\star+m}(k)) \cdot (w_1, \ldots, w_{k_s^\star+m})} = (g_{q+1})^0 = 1$ (see [5]).

- $(\omega^\star \notin \mathbb{O})$ and $(\psi \in \mathbb{S}^\star)$ and $(\mathrm{ID} \notin \overline{\mathrm{R}})$:
  For each $x \in \mathsf{Path}(\mathrm{ID})$, $\mathcal{B}$ randomly choses $r_x \in \mathbb{Z}_p$. It then lets $D_x = g^{r_x}$, and for all $k \in \psi$ lets $D^{(3)}_k = F_s(k)^{r_x}$ as in the construction. Recall that the dimensions of N are

$l_0 \times k_0$. Since $\omega^\star$ does not satisfy N for this case of the query, and by Proposition 1, there exists a vector $\boldsymbol{a_x} = (a_1, \ldots, a_{k_o}) \in \mathbb{Z}_p^{k_o}$ such that $a_1 = -1$ and $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$ for all i where $\pi(i) \in \omega^\star$.

$\mathcal{B}$ randomly chooses $z'_{x,2}, \ldots, z'_{x,k_o} \in \mathbb{Z}_p$ and defines $\boldsymbol{v'_x} = (0, z'_{x,2}, \ldots, z'_{x,k_o})$. It then implicitly defines a vector $\boldsymbol{v_x} = -(a'_x - \frac{\alpha r_x - \gamma}{t^\star} + \alpha r_x + \gamma)\boldsymbol{a_x} + \boldsymbol{v'_x}$ which will be used to create the share of $f_x(1) = \gamma + \alpha r_x + a_x$ (note that the first element of $\boldsymbol{v_x}$ is indeed $f_x(1)$ by (7)), as in our construction.

Now, for all i such that $\pi(i) \in \omega^\star$, $\mathcal{B}$ randomly chooses $r_{x,i} \in \mathbb{Z}_p$ and computes $D^{(1)}_{x,i} = g^{r_{x,i}}$ and

$$D^{(2)}_{x,i} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} = g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}},$$

where the last equality holds because $\boldsymbol{N_i} \cdot \boldsymbol{a_x} = 0$. Note that $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$ in our construction and hence $D^{(2)}_{x,i}$ is of the valid form.

For all other i, where $\pi(i) \notin \omega^\star$, $\mathcal{B}$ randomly chooses $r'_{x,i} \in \mathbb{Z}_p$. Observe that

$$\begin{aligned}
\boldsymbol{N_i} \cdot \boldsymbol{v_x} &= \boldsymbol{N_i} \cdot (-(a'_x - \frac{\alpha r_x - \gamma}{t^\star} + \alpha r_x + \gamma)\boldsymbol{a_x} + \boldsymbol{v'_x}) \\
&= \boldsymbol{N_i} \cdot (-(a'_x - \frac{\alpha r_x - (\gamma' + a^{q+1})}{t^\star} + \alpha r_x + (\gamma' + a^{q+1}))\boldsymbol{a_x} + \boldsymbol{v'_x}) \\
&= \boldsymbol{N_i} \cdot (\boldsymbol{v'_x} - (a'_x + \gamma'(\frac{1}{t^\star} + 1))\boldsymbol{a_x}) + (r_x(\frac{1}{t^\star} - 1)\boldsymbol{N_i} \cdot \boldsymbol{a_x})\alpha \\
&\quad - ((\frac{1}{t^\star} + 1)\boldsymbol{N_i} \cdot \boldsymbol{a_x})a^{q+1}
\end{aligned}$$

contains a term in a^{q+1} and hence we cannot compute this value (as a^{q+1} is the gap in the q-BDHE game). Instead, we will use the r_i term in $F_o(\pi(i))^{r_{x,i}}$ to cancel the unknown value a^{q+1}. $\mathcal{B}$ implicitly defines $r_{x,i} = r'_{x,i} - \frac{a(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}}{f(\pi(i))}$. To do so, it defines

$$D^{(2)}_{x,i} = g_1^{\left(r_x(\frac{1}{t^\star}-1)\boldsymbol{N_i} \cdot \boldsymbol{a_x} - (\frac{1}{t^\star}+1)\frac{\boldsymbol{N_i} \cdot \boldsymbol{a_x} f'(\pi(i))}{f(\pi(i))}\right)} \cdot g^{\boldsymbol{N_i} \cdot (\boldsymbol{v'_x} - (a'_x + \gamma'(\frac{1}{t^\star}+1)))\boldsymbol{a_x}} F_o(\pi(i))^{r'_{x,i}}$$

To see that $D^{(2)}_{x,i}$ is valid, we observe

$$\begin{aligned}
D^{(2)}_{x,i} &= g_{q+1}^{(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}} \cdot D^{(2)}_{x,i} \cdot g_{q+1}^{-(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}} \\
&= g_{q+1}^{(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}} \cdot g_1^{r_x(\frac{1}{t^\star}-1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}} \cdot g^{\boldsymbol{N_i} \cdot (\boldsymbol{v'_x} - (a'_x + \gamma'(\frac{1}{t^\star}+1)))\boldsymbol{a_x}} \\
&\quad \cdot \left(g_{q+1}^{-(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}} g_1^{-(\frac{1}{t^\star}+1)\frac{\boldsymbol{N_i} \cdot \boldsymbol{a_x} f'(\pi(i))}{f(\pi(i))}} \right) \cdot F_o(\pi(i))^{r'_{x,i}} \\
&= g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} \left(g_q^{f(\pi(i))} g^{f'(\pi(i))} \right)^{\frac{-a(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}}{f(\pi(i))}} \cdot F_o(\pi(i))^{r'_{x,i}} \\
&= g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} \cdot F_o(\pi(i))^{\frac{-a(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}}{f(\pi(i))}} \cdot F_o(\pi(i))^{r'_{x,i}} \text{ by (6)} \\
&= g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} \cdot F_o(\pi(i))^{r_{x,i}}
\end{aligned}$$

$\mathcal{B}$ also defines

$$D^{(1)}_{x,i} = g^{r'_{x,i}} g_1^{-\frac{(\frac{1}{t^\star}+1)\boldsymbol{N_i} \cdot \boldsymbol{a_x}}{f(\pi(i))}} = g^{r_{x,i}}$$

Note that $f(\pi(i)) \neq 0$ since $\pi(i) \notin \omega^\star$, and so $D^{(1)}_{x,i}$ and $D^{(2)}_{x,i}$ are well defined.

To simulate $\mathsf{KeyUpdate}$ queries for time period t and revocation list R, we consider the following cases:

- $t = t^\star$ and $\overline{R} \subseteq R$:
  For each $x \in \mathsf{Cover}(R)$, $\mathcal{B}$ chooses a random $r_x \in \mathbb{Z}_p$ and computes $U_x^{(1)} = (g^{a'_x t^\star}) P(t^\star)^{r_x}$ and $U_x^{(2)} = g^{r_x}$. Both keys are valid because since $\overline{R} \subseteq R$ and thus for all $x \in \mathsf{Cover}(R)$ we have $x \notin \mathcal{X}_{\overline{R}}$. Hence, by (9), $f_x(t^\star) = a'_x t^\star$.

- $t \neq t^\star$:
  For each $x \in \mathsf{Cover}(R)$, $\mathcal{B}$ chooses a random $r'_x \in \mathbb{Z}_p$
  - If $x \in \mathsf{Cover}(R) \cap \mathcal{X}_{\overline{R}}$, it defines

$$U_x^{(1)} = (g^{a'_x})^t (g^{\gamma'})^{(1-t)} (g_1^{r'_x})^{(1-t)} g_q^{-\frac{\rho(t)(1-t)}{\hat{p}(t)+1-t}} P(t)^{r'_x}$$

$$U_x^{(2)} = (g^{r'_x})(g_q)^{-\frac{1-t}{\hat{p}(t)+1-t}}$$

    Note that $\hat{p}(t) \neq 0$ for $t \neq t^\star$ so this is well defined. We claim that these keys look valid according to the construction with implicit randomness $r_x = r'_x - \frac{a^q(1-t)}{\hat{p}(t)+1-t}$. Note that, in this case, $x \in \mathcal{X}_{\overline{\mathcal{R}}}$ and hence by (7)

$$\begin{aligned} f_x(t) &= a_x t + \alpha r_x + \gamma = (a'_x - \alpha r_x - \gamma)t + \alpha r_x + \gamma \\ &= a'_x t + \alpha r_x (1-t) + \gamma'(1-t) + a^{q+1}(1-t) \end{aligned}$$

    Then,

$$\begin{aligned} U_x^{(1)} &= g^{f_x(t)} P(t)^{r_x} \text{ by the construction} \\ &= g^{a'_x t + a r_x (1-t) + \gamma'(1-t) + a^{q+1}(1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \text{ by } f_x(t) \text{ and (6)} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a r_x (1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{a\hat{p}(t) r'_x} g^{-a\hat{p}(t)B} g^{\rho(t) r'_x} g^{-\rho(t)B} \\ &\qquad \text{by } r_x = r'_x - B \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{-a\hat{p}(t)B} g^{-\rho(t)B} \text{ by (6)} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)B} g^{-B^{a(1-t) + a\hat{p_t}}} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} (g^a)^{-(\frac{a^q(1-t)}{\hat{p}(t)+1-t})^{(1-t)+\hat{p_t}}} \\ &\qquad \text{by } B = \frac{a^q(1-t)}{\hat{p}(t)+1-t} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} (g^a)^{-(a^q(1-t))} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} g^{-a^{q+1}(1-t)} \\ &= g^{a'_x t} g^{\gamma'(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)(\frac{a^q(1-t)}{\hat{p}(t)+1-t})} \\ &= (g^{a'_x})^t (g^{\gamma'})^{(1-t)} (g_1^{r'_x})^{(1-t)} g_q^{-\frac{\rho(t)(1-t)}{\hat{p}(t)+1-t}} P(t)^{r'_x} \text{ as we defined} \end{aligned}$$

  - If $x \in \mathsf{Cover}(R) \setminus \mathcal{X}_{\overline{R}}$, it defines

$$U_x^{(1)} = (g^{a'_x})^t (g^{\gamma'})^{(\frac{t}{t^\star}+1)} (g_1^{r'_x})^{(1-\frac{t}{t^\star})} g_q^{-\frac{\rho(t)(1+\frac{t}{t^\star})}{\hat{p}(t)+1-\frac{t}{t^\star}}} P(t)^{r'_x}$$

$$U_x^{(2)} = (g^{r'_x})(g_q)^{-\frac{1+\frac{t}{t^\star}}{\hat{p}(t)+1-\frac{t}{t^\star}}}$$

In this case, by (7), $a_x = a'_x - \frac{\alpha r_x - \gamma}{t^\star}$. By a similar argument as above, these keys look valid according to the construction with implicit randomness $r_x = r'_x - -\frac{a^q(1+\frac{t}{t^\star})}{\hat{p}(t)+1-\frac{t}{t^\star}}$.

4. $\mathcal{A}$ selects two messages m_0 and m_1. $\mathcal{B}$ chooses $b \xleftarrow{\$} \{0,1\}$ and creates a ciphertext $C = m_b \cdot Z \cdot e(h, g^{\gamma'}), C^{(1)} = h$, and for $k \in \omega^\star$ we write $C_k^{(2)} = h^{f'(x)}$. We write $h = g^s$ for some unknown s. The simulator then chooses random elements $y'_2, \ldots, y'_{k_s^\star} \in \mathbb{Z}_p$ and lets $\boldsymbol{y'} = (0, y'_2, \ldots, y'_{k_s^\star})$. It defines $C_i^{(3)} = (g_1)^{\boldsymbol{M_i^\star} \cdot \boldsymbol{y'}} \cdot (g^s)^{-p_0(\rho^\star(i))}$ for $i = 1, \ldots, l'_s$ and $C^{(4)} = (g^s)^{\rho(t^\star)}$, to implicitly share the secret s via the vector

$$\boldsymbol{v_x} = (s, s\alpha + y'_2, s\alpha^2 + y'_3, \ldots, s\alpha^{k'_s - 1} + y'_{k'_s}).$$

We claim that if $Z = e(g_{q+1}, h)$ then the created ciphertext is a valid challenge. The validity of $C^{(1)} = h = g^s$ comes from the implicit definition of h. To see that C is valid, recall that $\gamma = \gamma' + a^{q+1}$. Then,

$$\begin{aligned} C &= m_b \cdot Z \cdot e(h, g^{\gamma'}) = m_b \cdot e(g_{q+1}, h) \cdot e(h, g^{\gamma'}) = m_b \cdot e(g,g)^{s a^{q+1}} \cdot e(g,g)^{s\gamma'} \\ &= m_b \cdot e(g,g)^{s(\gamma' + a^{q+1})} = m_b \cdot e(g,g)^{s\gamma}. \end{aligned}$$

For all $k \in \omega^\star$, we defined $f(k)$ such that $f(k) = 0$, and hence

$$C_k^{(2)} = h^{f'(k)} = (g^s)^{f'(k)} = (g_q^{f(k)} g^{f'(k)})^s = F_o(k)^s.$$

For $i = 1, \ldots, l'_s$, we have

$$\begin{aligned} C_i^{(3)} &= (g_1)^{\boldsymbol{M_i^\star} \cdot \boldsymbol{y'}} \cdot (g^s)^{-p_0(\rho^\star(i))} \\ &= (g^\alpha)^{\boldsymbol{M_i^\star} \cdot \boldsymbol{y'}} \prod_{j=1}^{k_s^\star} g^{M^\star_{i,j} s\alpha^j} \cdot (g^s)^{-p_0(\rho^\star(i))} \prod_{j=1}^{k_s^\star} (g^s)^{-M^\star_{i,j}\alpha^j} \\ &= g^{\alpha \boldsymbol{M_i^\star} \cdot \boldsymbol{v_x}} \cdot (g^s)^{-p(\rho^\star(i))} = g^{\alpha \boldsymbol{M_i^\star} \cdot \boldsymbol{v_x}} \cdot F_s(\rho^\star(i))^{-s}, \end{aligned}$$

Finally, since $\hat{p}(t^\star) = 0$, $C^{(4)} = (g^s)^{\rho(t^\star)} = ((g^a)^{\hat{p}(t^\star)} g^{\rho(t^\star)})^s = P(t^\star)^s$.

5. The challenge ciphertext is given to $\mathcal{A}$ along with oracle access which is handled as in Step 3.

6. $\mathcal{A}$ eventually outputs $b' \in \{0,1\}$ as its guess of b. If $b = b'$ then $\mathcal{B}$ outputs 1 to guess that $Z = e(g_{q+1}, h)$. Otherwise, $\mathcal{B}$ outputs 0 to guess that Z is random.

If $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{R}_{BDHE}$ then $\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) = 0] = \frac{1}{2}$ since $\mathcal{A}$ was given a malformed challenge and hence can only guess the value of b. On the other hand if $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ is sampled from $\mathcal{P}_{BDHE}$ then we formed a valid challenge ciphertext and, as $\mathcal{A}$ is assumed to have non-negligible advantage ϵ in the IND-sHRSS game, $|\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,a,q}, Z) = 0] - \frac{1}{2}| \geq \epsilon$. It follows that $\mathcal{B}$ has advantage at least ϵ in solving q-BDHE problem in $\mathbb{G}$. However, we assumed that this problem is hard, so an adversary with non-negligible advantage in the IND-sHRSS game cannot exist.

□