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 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

* A preliminary version of this paper appears in the proceedings of *Topics in Cryptology - CT-RSA 2016*, DOI:http://dx.doi.org/10.1007/978-3-319-29485-8_9. This is the full version.

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 [23, 2] 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 provide a detailed construction and proof in Appendix B. In Section 4.3, we instantiate HPVC using rkDPABEand in Section 5 we present the security proofs for our scheme. Finally, in Section 6 we conclude the paper.

2 Background and Related Work

Verifiable computation [17, 13, 27, 16, 12, 10, 23] 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 [17, 13] or targeted specific functions [10, 16, 12]. 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 [18, 9, 11, 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 Centre (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 $\theta_{F(x)}$ using a verification key and learn the value of $F(x)$. If S cheated they may be reported to the KDC for revocation.

The constructions of [23, 2] 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

3.1 Formal Definition

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 mk. 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 $\mathsf{Certify}$ 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.

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 $\sigma_{(\omega,\mathbb{S})}$ and a public verification key $vk_{(\omega,\mathbb{S})}$.

A server combines $\sigma_{(\omega,\mathbb{S})}$ with its evaluation key to compute $\theta_{F(X)}$ encoding the result $F(X)$. Any entity can verify the correctness of $\theta_{F(X)}$ using $vk_{(\omega,\mathbb{S})}$. Verification outputs the result $y = F(X)$ of the computation (if correct) and generates a token $\tau_{\theta_{F(X)}}$ which is sent to the KDC; if the token signifies that the result was incorrectly formed then the server is 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.

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

Definition 1. *An* hybrid publicly verifiable outsourced computation (HPVC) scheme *for a family of functions $\mathcal{F}$ comprises the following algorithms:*

1. $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{F})$*: this randomised algorithm is run by the KDC to establish public parameters pp and a master secret key mk for the system. The inputs are the security parameter λ, and the family of functions $\mathcal{F}$ that may be computed;*
2. $pk_F \xleftarrow{\$} \mathsf{FnInit}(F, mk, pp)$*: this randomised algorithm is run by the KDC to generate a public delegation key, pk_F, allowing entities to outsource, or request, computations of F;*
3. $sk_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, mk, pp)$*: this randomised algorithm is run by the KDC to enrol an entity S_i within the system to act as a server. It generates a personalised signing key sk_{S_i};*
4. $ek_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$*: this randomised algorithm is run by the KDC to generate an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ enabling the entity S_i to compute on the pair $(\mathbb{O}, \psi)$. The algorithm also takes as input the mode in which it should operate, a set of labels L_i, a set of functions $\mathcal{F}_i$, the master secret key as well as the public parameters;*

[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.

5. $(\sigma_{(\omega,\mathbb{S})}, vk_{(\omega,\mathbb{S})}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega,\mathbb{S}), L_{F,X}, pk_F, pp)$*: this randomised algorithm is run by an entity to request a computation of $F(X)$ from S_i. The inputs are the mode, the pair $(\omega,\mathbb{S})$ representing the computation request, a set of labels $L_{F,X} \subseteq L_i$, the delegation key for F and the public parameters. The algorithm outputs an encoded input $\sigma_{(\omega,\mathbb{S})}$ and a verification key $vk_{(\omega,\mathbb{S})}$;*
6. $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, ek_{(\mathbb{O},\psi),S_i}, sk_{S_i}, pp)$*: this randomised algorithm is run by an entity S_i to compute $F(X)$. The inputs are the mode, an encoded input $\sigma_{(\omega,\mathbb{S})}$, an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ and a signing key for S_i. The algorithm outputs an encoded output $\theta_{F(X)}$ representing $F(X)$;*
7. $(y, \tau_{\theta_{F(X)}}) \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{(\omega,\mathbb{S})}, pp)$*: this algorithm is run by any entity that wants to verify whether the result was computed correctly or not. The inputs are the encoded output $\theta_{F(X)}$, the verification key $vk_{(\omega,\mathbb{S})}$ and the public parameters. The algorithm outputs the actual result y. If the result y corresponds to $F(x)$ it additionally creates a token $\tau_{\theta_{F(x)}} = (\mathtt{accept}, S_i)$ indicating that the result was correctly computed. Otherwise, the result y corresponds to $\perp$ and it creates a token $\tau_{\theta_{F(x)}} = (\mathtt{reject}, S_i)$ indicating that the result is malformed and S_i misbehaved;*
8. $um \xleftarrow{\$} \mathsf{Revoke}(\tau_{\theta_{F(X)}}, mk, pp)$*: this randomised algorithm is run by the KDC inputting the token from the verification process, the master secret key and public parameters. If $\tau_{\theta_{F(X)}} = (\mathtt{reject}, S_i)$, the algorithm revokes all evaluation keys $ek_{(\cdot,\cdot),S_i}$ of the server S_i by rendering them non-functional and thereby preventing S_i from performing any further evaluations within the current system. The update material um consists of a set of updated evaluation keys $\{ek_{(\mathbb{O},\psi),S'}\}$ which are issued to all servers. Otherwise, in case $\tau_{\theta_{F(X)}} = (\mathtt{accept}, S_i)$ then the algorithm outputs $\perp$ indicating that no update was necessary.*

Although not explicitly stated, the KDC may update the public parameters pp during any algorithm in order to address any changes in the entity population.

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

The adversary is given access to a set of oracles; for each algorithm in Definition 1, we define a corresponding oracle which executes the corresponding algorithm on arguments provided by the adversary, and returns the output of the algorithm to the adversary. The adversary may query the Setup oracle only once (before making any other oracle queries), but can thereon call the remaining oracles any number of times and in any order.

The challenger maintains two lists, L_{Reg} and L_F. L_{Reg} is a list of tuples comprising server identities, S_i, and the resulting signing keys, sk_{S_i}, that have been queried to the Register oracle. L_F comprises tuples of the form $(S_i, \psi, L_i, \mathcal{F}_i, ek_{(\mathbb{O},\psi),S_i})$ denoting that the server S_i has been queried to the Certify oracle for the set of functions $\mathcal{F}_i$ and that $ek_{(\mathbb{O},\psi),S_i}$ was generated. When the adversary makes a Revoke query with a revocation token that identifies a server S_i to be revoked (that is, if $\tau_{\theta_{F(X)}} = (\mathtt{reject}, S_i)$ is given as input to the Revoke oracle), the challenger removes all entries of the form $(S_i, \cdot, \cdot, \cdot, \cdot)$ (i.e. all entries for S_i for any function) from L_F.

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

- ProbGen: the challenger creates a new row in T comprising 7 components, all of which are initialised to be empty; it then assigns X (which is either given explicitly in ω or can be found by searching L_F for the labels $L_{F,X}$), F, the result $F(X)$ (computed by the challenger itself), $\sigma_{(\omega,\mathbb{S})}$ and $vk_{(\omega,\mathbb{S})}$ to the first 5 components;
- Compute: the challenger first searches T for all rows that contain the queried $\sigma_{(\omega,\mathbb{S})}$ in the 4^{th} component and where the 6^{th} component is empty (i.e. those rows relating to computations on this encoded input that have not yet been performed). For each such row, r, the challenger takes the second component (the function identifier, $\tilde{F}$), and checks that there exists a server identity $\tilde{S}_i$ such that the tuple $(\tilde{S}_i, sk_{\tilde{S}_i}) \in L_{\text{Reg}}$ (where $sk_{\tilde{S}_i}$ is that given as input to the Compute oracle) *and* such that the tuple $(\tilde{S}_i, \cdot, \cdot, \mathcal{F}_i, ek_{(\mathbb{O},\psi),S_i}) \in L_F$ (where $ek_{(\mathbb{O},\psi),S_i}$ is also that given as input to the Compute oracle) and where $\tilde{F} \in \mathcal{F}_i$. This check ensures that there is a currently un-revoked server (as the entries of L_F for $\tilde{S}_i$ have not been removed) that holds the signing key and evaluation key

being used to perform the computation and which is certified for a function $\tilde{F}$ for which the encoded input $\sigma_{F,X}$ was generated.
The challenger then performs the Compute algorithm on the queried $\sigma_{(\omega,\mathbb{S})}, ek_{(\mathbb{O},\psi),S_i}$ and SK_{S_i} to produce an output $\theta_{F(X)}$. For each of the rows r of T found above, the challenger writes $\theta_{F(X)}$ and $\tilde{S}_i$ to the 6^{th} and 7^{th} components of r respectively. Thus, a row of T will only have a (non-empty) value in the 6^{th} component if there exists a non-revoked, certified server to perform the computation for which $\sigma_{(\omega,\mathbb{S})}$ was generated.

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

$$(X, F, F(X), \sigma_{(\omega,\mathbb{S})}, vk_{(\omega,\mathbb{S})}, \theta_{F(X)}, S).$$

After a polynomial number of queries, the adversary will return a value $\theta^{\star}_{F(X)}$ which it believes either encodes an incorrect computational result or which encodes a correct computational result yet which the Verify algorithm will reject (that is, an output for which the protocol execution will not be correct). The challenger first performs a look up in T for all entries containing $\theta^{\star}_{F(X)}$ in the 6^{th} position of the tuple, and stores any such entries as another table $\tilde{T}$. Note that this means that $\theta^{\star}_{F(X)}$ must have been honestly generated by the Compute oracle (else it would not be in T).

For each such row, the challenger uses the 5^{th} component (the verification key) to run Verify on $\theta^{\star}_{F(X)}$ to generate the output y.

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

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

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

An HPVC scheme is *correct* if, for all PPT adversaries, the probability that the adversary wins the game described above is 0.

3.2 Security Models

In this section we discuss the security notions we wish to achieve in our HPVC framework. In more detail, we can achieve security in the sense of *public verifiability*, *revocation* and *authorised computation*.[2] We require to include some additional restrictions on the games that are placed from our current rkDPABE primitive (which we introduce in Section B) which acts as our main building block for our HPVC construction.

Selective Public Verifiability. In Figure 1, we define a *selective notion of public verifiability*. This notion captures that no server is able to return a malformed result for a computation without being detected.

The game begins with the adversary first selecting its challenge parameters. Note that the adversary chooses the mode it wishes the challenge to be generated in and the respective labels necessary for this mode. Furthermore, the adversary outputs choices for $\omega^{\star}$, $\mathbb{O}^{\star}$, $\psi^{\star}$ and $\mathbb{S}^{\star}$, despite only $\omega^{\star}$ and $\mathbb{S}^{\star}$ are used to form the challenge input. This notation was used mainly for notational convenience to allow us to define the challenge computation in terms of F and $X^{\star}$ in line 3 or 4 depending on the mode. However, we want to stress that this information can also be learnt from the set of labels $L_{F,X^{\star}}$ and the chosen mode of computation. Thus, this notational convenience does not weaken the game since the information

[2] We do not 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].

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

1 : $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode}) \leftarrow_\$ \mathcal{A}(1^\lambda, \mathcal{F})$
2 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mathcal{F})$
3 : **if** $\texttt{mode} = \text{VDC}$ **then** $(F \leftarrow \mathbb{S}^\star, X^\star \leftarrow \psi^\star)$
4 : **else** $(F \leftarrow \mathbb{O}^\star, X^\star \leftarrow \omega^\star)$
5 : $pk_F \leftarrow_\$ \mathsf{FnInit}(F, mk, pp)$
6 : $(\sigma^\star, vk^\star) \leftarrow_\$ \mathsf{ProbGen}(\texttt{mode}, (\omega^\star, \mathbb{S}^\star), L_{F,X^\star}, pk_F, pp)$
7 : $\theta^\star \leftarrow_\$ \mathcal{A}^{\mathcal{O}}(\sigma^\star, vk^\star, pk_F, pp)$
8 : $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, vk^\star, pp)$
9 : **if** $(y, \tau_{\theta^\star}) \neq (\perp, (\texttt{reject}, S))$ **and** $(y \neq F(X^\star))$ **then**
10 : **return** 1
11 : **else return** 0

Fig. 1. The selective public verifiability experiment $\mathbf{Exp}^{\text{sPubVerif}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right]$

has been already determined by the adversary's choices.

After the adversary has chosen the parameters, the game proceeds with the challenger running Setup to initialise the system and FnInit to return the public delegation key pk_F for the chosen challenge function. The challenger continues with running $\mathsf{ProbGen}$ on the challenge inputs to output a challenge for the adversary. The adversary receives the challenge and public information and is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm. Finally, the adversary wins the game if it is able to create an encoded output that verifies correctly but does not encode the correct value $F(X)$.

Definition 2. *The* advantage *of a PPT adversary in the* sPubVerif *game for an hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

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

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

3.3 Selective, Semi-static Revocation.

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

In Figure 2, we define a *selective, semi-static notion of revocation*. This notion starts with the adversary choosing its challenge parameters which the challenger can parse to determine F and $X^\star$. The challenger maintains a (initially empty) list Q_{Rev} of currently revoked entities as well as the current time period t which can be incremented during Revoke oracle queries. The game proceeds with the challenger running Setup to initialise the system and FnInit to return the public delegation key pk_F for the chosen challenge function. After this, on line 8, the adversary needs to declare (before receiving oracle access) a list $\overline{R}$ of servers to be revoked at the time period where the challenge will be generated which we assume will be at time period q_t. The adversary is then provided with oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$,

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

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

$\mathcal{O}^{\mathsf{Certify}}(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$

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

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

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

Fig. 2. The selective, semi-static revocation experiment $\mathbf{Exp}^{\text{sssRevoc}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}, q_t\right]$

$\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. Certify and Revoke oracle queries are handled as specified in Figure 2. The adversary finishes its oracles query phase (line 9) after making a polynomial number of queries q, including q_t many Revoke queries, and does not return a value other than signalling to the challenger that it may proceed with the remainder of the game. The challenger checks that all queries made by the adversary have indeed generated a list of currently revoked servers that is a superset of the challenge revocation list $\overline{R}$. If this is not true, the challenger aborts the game and the adversary loses as it was not able to choose its queries or the list $\overline{R}$ appropriately. Otherwise, the challenger continues with the game and generates the challenge by running ProbGen. The adversary wins the game if it outputs any result, i.e. a correct or malformed response, as a valid result from any server that was revoked at the time of the challenge.

Definition 3. *The* advantage *of a PPT adversary in the* sssRevoc *game for an hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

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

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

3.4 Selective Authorised Computation.

In Figure 3, we define a *selective notion of authorised computation*. This notion ensures that only a server that satisfies an additional authorisation policy in the encoded input should be able to perform a given computation on this encoded input. Thus, in contrast, a result generated by an unauthorised server should always be rejected even if the result is correct.

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

1 : $\texttt{mode} = \text{RPVC-AC}$
2 : $(F, X^\star, P, \{l(F)\}) \leftarrow_\$ \mathcal{A}(1^\lambda, \mathcal{F})$
3 : $(pp, mk) \leftarrow_\$ \mathsf{Setup}(1^\lambda, \mathcal{F})$
4 : $pk_F \leftarrow_\$ \mathsf{FnInit}(F, mk, pp)$
5 : $(\sigma^\star, vk^\star) \leftarrow_\$ \mathsf{ProbGen}(\text{RPVC-AC}, (X^\star, P), \{l(F)\}, pk_F, pp)$
6 : $\theta^\star \leftarrow_\$ \mathcal{A}^{\mathcal{O}}(\sigma^\star, vk^\star, pk_F, pp)$
7 : $(y, \tau^\star) \leftarrow \mathsf{Verify}(\theta^\star, vk^\star, pp)$
8 : **if** $\tau^\star \neq (\texttt{reject}, \cdot)$ **then**
9 : **return** 1
10 : **else return** 0

$\mathcal{O}^{\mathsf{Certify}}(\text{RPVC-AC}, S_i, (F, \psi), \{l(F)\}, \mathcal{F}_i, mk, pp)$

1 : **if** $(\psi \in P)$ **then return** $\perp$
2 : **return** $\mathsf{Certify}(\text{RPVC-AC}, S_i, (F, \psi), \{l(F)\}, \mathcal{F}_i, mk, pp)$

Fig. 3. The selective authorised computation experiment $\mathbf{Exp}^{\text{sAuthComp}}_{\mathcal{A}}\left[\mathcal{HPVC}, 1^\lambda, \mathcal{F}\right]$

The game begins with explicitly setting the mode of computation to RPVC-AC and the adversary chooses the parameters for the game accordingly as otherwise the parameters would not be meaningful. Those parameters consist of a challenge function F, challenge input $X^\star$, the authorisation policy P and the respective function labels for this mode. The game proceeds with the challenger running Setup to initialise the system and FnInit to return the public delegation key pk_F for the chosen challenge function. It continues with the challenger running $\mathsf{ProbGen}$ to create the challenge for the adversary. The adversary receives the challenge and public information and is given oracle access to $\mathsf{FnInit}(\cdot, mk, pp)$, $\mathsf{Register}(\cdot, mk, pp)$, $\mathsf{Certify}(\cdot, \cdot, (\cdot, \cdot), \cdot, \cdot, mk, pp)$ and $\mathsf{Revoke}(\cdot, mk, pp)$ which we denote by $\mathcal{O}$. All oracles simply run the relevant algorithm with the exception of $\mathsf{Certify}$ queries which are separately specified in Figure 3. The $\mathsf{Certify}$ oracle returns $\perp$ if the queried attribute set ψ satisfies the authorisation policy P, as otherwise the adversary would be able to trivially produce a valid response as an authorised entity. The adversary wins the game if it outputs a result and token that is accepted by a verifier.

Table 1. Parameter definitions for different modes of computation

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	$\{l(x_{i,j})\}_{x_{i,j}\in D_i}$	$\{l(x_{i,j})\}_{x_{i,j}\in X}$	$\{(F,\{l(x_{i,j})\}_{x_{i,j}\in \mathsf{Dom}(F))}\}_{F\in\mathcal{F}}$

Definition 4. *The* advantage *of a PPT adversary in the* sAuthComp *game for an hybrid publicly verifiable outsourced computation scheme $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

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

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

4 Instantiating HPVC

In this section, we provide a construction of 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 Key-policy ABE (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.

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.

In this section, we first provide more details about the different modes of computation that are supported by our HPVC scheme. Next we introduce a new cryptographic primitive which acts as our main building block of our construction and finally we provide the concrete instantiation details.

4.1 Supporting Different Modes of Computation

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 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.

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 inputs that 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 $\mathsf{ProbGen}$ 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 $\mathsf{ProbGen}$ 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. $\mathsf{ProbGen}$ 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 $s = \{\texttt{Country} = \texttt{UK}, \texttt{Capacity} = \texttt{3TB}\}$.

VDC. *Verifiable Delegable Computation* 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.

Fig. 4. Example database

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

Fig. 5. 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 $X \subseteq \mathsf{Dom}(F)$[3] 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 4. 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$\}$. In Table 5, $\mathcal{F}_i$ lists the functions and domains that S_i is willing to compute. To find the average age, a delegator queries "Average" on input $X = \{$Age of record 1, Age of record 2$\}$.

Possible applications for which this mode may be useful are the following:

– **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 [25, 11].
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 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

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

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.

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.

In the following we provide a formal definition of a *revocable key dual-policy attribute-based encryption scheme*. Note that we refer to the access structure associated to a decryption key as an *objective* access structure, denoted as $\mathbb{O}$, and the attribute set associated with a ciphertext is referred to as an *objective* attribute set, denoted as ω. Both the objective access structure and attribute set are associated with the KP-ABE functionality. Similarly, we refer to the access structure associated with a ciphertext as a *subjective* access structure, denoted as $\mathbb{S}$, whereas we refer to the attribute set associated to a decryption key as a *subjective* attribute set, denoted as ψ. Thus, both the subjective access structure and attribute set are associated with the CP-ABE functionality.

An indirectly revocable key DP-ABE scheme defines the universe of attributes to be $\mathcal{U} = \mathcal{U}_{\text{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}} \cup \mathcal{U}_{\text{time}} \cup T_{\text{O}} \cup T_{\text{S}}$. In more detail, $\mathcal{U}_{\text{attr}}$ is the "normal" attribute universe for describing ciphertexts and forming access policies, $\mathcal{U}_l$ contains a set of attributes (disjoint from $\mathcal{U}_{\text{attr}}$) that uniquely label each function and each data item, $\mathcal{U}_{\text{time}}$ comprises attributes for time periods, and $\mathcal{U}_{\text{ID}}$ contains attributes encoding entity identities. T_{O} and T_{S} are additional ("dummy") attributes that efficiently enable the DP-ABE scheme to either function as a KP-ABE scheme or CP-ABE scheme. In Section 4.3, we discuss in more detail how those dummy attributes influence the execution of the different modes of computations within our unified HPVC construction.

More formally, an indirectly revocable key DP-ABE scheme is presented in the following definition.

Definition 5. *A* revocable key dual-policy attribute-based encryption (rkDP-ABE) scheme *consists of the following algorithms:*

- $(pp, mk) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathcal{U})$: *this randomised algorithm takes as input the security parameter and the universe of attributes $\mathcal{U}$ and outputs public parameters pp and master secret key mk;*
- $ct_{(\omega,\mathbb{S}),t} \xleftarrow{\$} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, pp)$: *this randomised encryption algorithm inputs a message m, an objective attribute set ω, a subjective policy $\mathbb{S}$, the current time period $t \in \mathcal{U}_{\text{time}}$ and the public parameters pp. It outputs a ciphertext $ct_{(\omega,\mathbb{S}),t}$ that is valid for time t;*
- $sk_{\mathsf{id},(\mathbb{O},\psi)} \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp)$: *this randomised key generation algorithm takes as input an identity $\mathsf{id} \in \mathcal{U}_{\text{ID}}$, an objective access structure $\mathbb{O}$, a subjective attribute set ψ, as well as the master secret key mk and public parameters pp. It outputs a secret decryption key $sk_{\mathsf{id},(\mathbb{O},\psi)}$;*
- $uk_{R,t} \xleftarrow{\$} \mathsf{KeyUpdate}(R, t, mk, pp)$: *this randomised algorithm takes a revocation list $R \subseteq \mathcal{U}_{\text{ID}}$ containing the identities of revoked entities, the current time period t, as well as the master secret key mk and public parameters pp. It outputs updated key material $uk_{R,t}$ which makes the decryption keys $sk_{\mathsf{id},(\mathbb{O},\psi)}$, for all non-revoked identities $\mathsf{id} \notin R$, functional to decrypt ciphertexts encrypted for the time period t;*
- $pt \leftarrow \mathsf{Decrypt}(ct_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), sk_{\mathsf{id},(\mathbb{O},\psi)}, (\mathbb{O}, \psi), uk_{R,t}, pp)$: *this decryption algorithm takes as input a ciphertext $ct_{(\omega,\mathbb{S}),t}$ formed for the time period t and the associated pair $(\omega, \mathbb{S})$, a secret decryption key $sk_{\mathsf{id},(\mathbb{O},\psi)}$ for an entity id and the associated pair $(\mathbb{O}, \psi)$, an update key $uk_{R,t}$ for the current time period t and the public parameters pp. The algorithm outputs a plaintext pt which corresponds to the correct message m, if and only if the objective attributes ω satisfy the objective access structure $\mathbb{O}$* and *the subjective attributes ψ satisfy the subjective access structure $\mathbb{S}$* and *the value of t in the update key matches the one specified during encryption. If not, pt outputs $\perp$.*

Correctness of a revocable key DP-ABE scheme is defined as follows.

Definition 6. *A revocable key DP-ABE scheme is* correct *if for all $m \in \mathcal{M}$, all $\mathsf{id} \in \mathcal{U}_{\mathrm{ID}}$, all $R \subseteq \mathcal{U}_{\mathrm{ID}}$, all access structures $\mathbb{O}, \mathbb{S} \subseteq 2^{\mathcal{U}_{\mathrm{attr}}} \setminus \{\emptyset\}$, all attribute sets $\omega, \psi \subseteq \mathcal{U}_{\mathrm{attr}}$ and all $t \in \mathcal{U}_{\mathrm{time}}$, if $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$ and $\mathsf{id} \notin R$, it holds that*

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

Definition 5 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 the Appendix B.

4.3 Construction

In this section we provide a construction of an HPVC scheme for a family $\mathcal{F}$ of monotone Boolean functions closed under complement using a revocable key dual-policy ABE scheme $\mathcal{RKDPABE}$ in a black box manner comprising the algorithms DPABE.Setup, DPABE.Encrypt, DPABE.KeyGen, DPABE.KeyUpdate and DPABE.Decrypt. We also use a signature scheme with algorithms Sig.KeyGen, Sig.Sign and Sig.Verify, and a one-way function g. Let $\mathcal{U} = \mathcal{U}_{\mathrm{attr}} \cup \mathcal{U}_l \cup \mathcal{U}_{\mathrm{ID}} \cup \mathcal{U}_{\mathrm{time}} \cup T_{\mathrm{O}} \cup T_{\mathrm{S}}$ be the universe of attributes acceptable by the revocable key dual-policy ABE scheme, formed as the union of the following sub-universes, where $\mathcal{U}_{\mathrm{attr}}$ consists of the attributes that form characteristic tuples for input data, $\mathcal{U}_l$ be a set of attributes (disjoint from $\mathcal{U}_{\mathrm{attr}}$) that uniquely label each function and each data item, $\mathcal{U}_{\mathrm{ID}}$ comprises attributes representing entity identifiers, $\mathcal{U}_{\mathrm{time}}$ comprises attributes representing time periods issued by the time source $\mathbb{T}$ and finally T_{O} and T_{S} represent the objective dummy attribute and subjective dummy attribute respectively.

We encode Boolean functions in terms of access structures over $\mathcal{U}_{\mathrm{attr}}$. Computations with n-bit outputs can be built from n Boolean functions returning each bit in turn. We can handle negations by either building rkDPABE from non-monotonic ABE [21] or by adding negated attributes to the universe [26]. We choose to use the latter approach and add negated attributes to $\mathcal{U}_{\mathrm{attr}}$. Thus, for the ith bit of a binary input string $X = x_1 \dots x_n$, we define attributes $A^0_{X,i}$ and $A^1_{X,i} \in \mathcal{U}_{\mathrm{attr}}$ and X is encoded as $A_X = \{A^j_{X,i} \in \mathcal{U}_{\mathrm{attr}} : x_i = j\}$.

In more detail, the dummy attributes T_{O} and T_{S} play generally a crucial role in a DP-ABE scheme as they efficiently enable a DP-ABE scheme to function as either KP-ABE or CP-ABE [6]. For KP-ABE, the subjective policy corresponds to $\mathbb{S} = \{\{T_{\mathrm{S}}\}\}$ and is satisfied by the subjective attribute ψ containing the special attribute T_{S}. Thus, $\mathbb{S}$ is trivially satisfied and decryption (in KP-ABE) only depends on the objective policy and attributes. Similarly, the same holds for CP-ABE where the objective policy corresponds to $\mathbb{O} = \{\{T_{\mathrm{O}}\}\}$ that is trivially satisfied by the objective attribute ω containing the special attribute T_{O}. As noted in [23, 2], we require to establish two distinct ABE schemes to overcome a possible one-sided error in the verification stage. Thus, we initialise two distinct rkDP-ABE systems over $\mathcal{U}$ and hence we define a total of four additional dummy attributes where $T^0_{\mathrm{O}}, T^0_{\mathrm{S}}$ relate to the first rkDP-ABE system, and $T^1_{\mathrm{O}}, T^1_{\mathrm{S}}$ relate to the second rkDP-ABE system. As summarised in Table 1, the function corresponds in the modes RPVC and RPVC-AC to $\mathbb{O} = F$ and $\mathbb{S} = \{\{T^0_{\mathrm{S}}\}\}$. Thus, the complement function for those modes can be defined as $\overline{\mathbb{O}} = \overline{F}$ and $\overline{\mathbb{S}} = \{\{T^1_{\mathrm{S}}\}\}$. Similarly, it follows for the mode VDC that $\mathbb{O} = \{\{T^0_{\mathrm{O}}\}\}$ and $\mathbb{S} = F$, and the the complement can be defined as $\overline{\mathbb{O}} = \{\{T^1_{\mathrm{O}}\}\}$ and $\overline{\mathbb{S}} = \overline{F}$. Each mode operates by encrypting a pair of randomly chosen messages and issuing keys such that the recovery of one message implies whether the encryption of a message was linked to F or $\overline{F}$, and thus whether $F(X) = 1$ or 0. Ciphertext indistinguishability ensures that an adversary cannot cheat by returning the other message. Our HPVC scheme operates in the following way.

1. Setup, presented in Algorithm 1, first forms the attribute universe $\mathcal{U}$ and initialises two rkDPABE schemes over the universe. It further creates an empty two-dimensional array L_{Reg} to list registered

entities, a (empty) list of revoked entities L_{Rev} as well as a time source $\mathbb{T}$ (e.g. a networked clock or counter) to index update keys. The algorithm finally outputs the public parameters pp and master secret key mk comprising of public and secret rkDPABE parameters respectively. Furthermore, the public parameters also contain L_{Reg} and the dummy attributes enabling a client to flexibly switch between the modes of computations by disabling certain parts of the rkDPABE scheme while the master secret key additionally contains the list of revoked entities L_{Rev}. Note that the public parameters may be implicitly updated throughout the execution of all algorithms of an HPVC scheme accommodating any changes in the system population.

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

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

2. FnInit, presented in Algorithm 2, sets the public delegation key pk_F (for all functions F) to be the public parameters for the system (since we use public key primitives). This step is not required in our particular construction, but we retain the algorithm for consistency with prior definitions as well as for generality as other instantiations may require this step.

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

$pk_F \leftarrow pp$

3. Register, presented in Algorithm 3, creates a public-private key pair by calling the KeyGen algorithm of the digital signature scheme. The algorithm provides the server with its own secret signature key and updates $L_{\text{Reg}}[S_i][0]$ to store the verification key for S_i. These prevent servers being impersonated and wrongly revoked.

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

$(sk_{\text{Sig}}, vk_{\text{Sig}}) \leftarrow_{\$} \mathsf{Sig.KeyGen}(1^\lambda)$
$sk_{S_i} \leftarrow sk_{\text{Sig}}$
$L_{\text{Reg}}[S_i][0] \leftarrow L_{\text{Reg}}[S_i][0] \cup vk_{\text{Sig}}$

4. Certify, presented in Algorithm 4, aims to generate an evaluation key $ek_{(\mathbb{O},\psi),S_i}$ for a server S_i. The algorithm first adds an element $(F, \bigcup_{l \in L_i} l)$ to the list $L_{\text{Reg}}[S_i][1]$ for each $F \in \mathcal{F}_i$. This publicises the computations that S_i can perform (either functions in RPVC and RPVC-AC modes, or functions and data labels in VDC). The algorithm removes S_i from the revocation list, gets the current time period from $\mathbb{T}$ and generates a decryption key for $(\mathbb{O}, A_\psi \cup \bigcup_{l \in L_i} l)$ in the first DP-ABE system and A_ψ is the attribute set encoding ψ. The additional attributes for the labels $l \in \mathcal{U}_l$ ensure that a key cannot be used to evaluate computations that do not correspond to these labels. In RPVC and RPVC-AC, this means that a key for a function G cannot evaluate a computation request for $F(X)$. In VDC, it

means that an evaluation key must be issued for a dataset D_i that includes (at least) the specified input data X. It is sufficient to include labels only on the subjective attribute set without also adding them to the objective policy. As these labels are a security measure against a misbehaving server, we amend the servers key but need not take similar measures against the delegator. Delegators are then able to specify the required labels in their created subjective policy. Those labels need to be present in the server's key for a successful evaluation (decryption). The KDC should check that the label corresponds to the input to ensure that a server does not advertise data he does not own. It also generates an update key for the current time period to prove that S_i is not currently revoked. In RPVC and RPVC-AC modes, another pair of keys is generated using the second DP-ABE system for the complement inputs.

Algorithm 4 $ek_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O},\psi), L_i, \mathcal{F}_i, mk, pp)$

for $F \in \mathcal{F}_i$ **do**
  $L_{\mathrm{Reg}}[S_i][1] \leftarrow L_{\mathrm{Reg}}[S_i][1] \cup (F, \bigcup_{l \in L_i} l)$
endfor
$L_{\mathrm{Rev}} \leftarrow L_{\mathrm{Rev}} \setminus S_i$
$t \leftarrow \mathbb{T}$
$sk^0_{\mathrm{ABE}} \leftarrow_{\$} \mathsf{DPABE.KeyGen}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l \in L_i} l), msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
$uk^0_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^0_{\mathrm{ABE}}, mpk^0_{\mathrm{ABE}})$
if ($\mathtt{mode} = \mathrm{RPVC}$) **or** ($\mathtt{mode} = \mathrm{RPVC\text{-}AC}$) **then**
  $sk^1_{\mathrm{ABE}} \leftarrow_{\$} \mathsf{DPABE.KeyGen}(S_i, (\overline{\mathbb{O}}, A_\psi \cup \bigcup_{l \in L_i} l), msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
  $uk^1_{L_{\mathrm{Rev}},t} \leftarrow_{\$} \mathsf{DPABE.KeyUpdate}(L_{\mathrm{Rev}}, t, msk^1_{\mathrm{ABE}}, mpk^1_{\mathrm{ABE}})$
else
  $sk^1_{\mathrm{ABE}} \leftarrow \perp$
  $uk^1_{L_{\mathrm{Rev}},t} \leftarrow \perp$
endif
$ek_{F,S} \leftarrow (sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, uk^1_{L_{\mathrm{Rev}},t})$

5. ProbGen, presented in Algorithm 5, aims to create a problem instance $\sigma_{(\omega,\mathbb{S})}$ that the server can use to evaluate the computation as well as preparing a verification key that enables anyone to verify the server's computational result. The algorithm starts with choosing messages m_0 and m_1 randomly from the message space. The message m_0 is encrypted with $(A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l)$ in the first rkDPABE system, whilst m_1 is encrypted with the complement policy under either the first rkDPABE system for VDC or the second one for RPVC and RPVC-AC depending on the chosen mode of computation. Note that the attributes remain the same as it is the same attribute T^0_{O} or input data X respectively. The algorithm also prepares a public verification key $vk_{(\omega,\mathbb{S})}$. The key is simply generated by applying a one-way function g to each randomly chosen message and also includes a copy of L_{Reg} from the public parameters in case the list is modified between the current time period and the time of verification.

Algorithm 5 $(\sigma_{(\omega,\mathbb{S})}, vk_{(\omega,\mathbb{S})}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, pk_F, pp)$

$(m_0, m_1) \leftarrow_{\$} \mathcal{M} \times \mathcal{M}$
$t \leftarrow \mathbb{T}$
$c_0 \leftarrow_{\$} \mathsf{DPABE.Encrypt}(m_0, (A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l), t, mpk^0_{\mathrm{ABE}})$
if ($\mathtt{mode} = \mathrm{VDC}$) **then**
    $c_1 \leftarrow_{\$} \mathsf{DPABE.Encrypt}(m_1, (A_\omega, \overline{\mathbb{S}} \wedge \bigwedge_{l \in L_{F,X}} l), t, mpk^0_{\mathrm{ABE}})$
else
    $c_1 \leftarrow_{\$} \mathsf{DPABE.Encrypt}(m_1, (A_\omega, \overline{\mathbb{S}} \wedge \bigwedge_{l \in L_{F,X}} l), t, mpk^1_{\mathrm{ABE}})$
endif
$\sigma_{(\omega,\mathbb{S})} \leftarrow (c_0, c_1)$
$vk_{(\omega,\mathbb{S})} \leftarrow (g(m_0), g(m_1), L_{\mathrm{Reg}})$

6. Compute, presented in Algorithm 6, is performed by a server S_i and aims to return the result of the evaluation of a function on some input data. The algorithm attempts to decrypt both ciphertexts of the problem instance $\sigma_{(\omega,\mathbb{S})}$, ensuring that different modes of computation use the correct parameters. Decryption succeeds only if the function evaluates to 1 on the input data X, i.e. the policy is satisfied. Since F and $\overline{F}$ output opposite results on X, exactly one plaintext will correspond to a failure symbol $\perp$. The server signs the results using its personal signing key. Finally, the algorithm outputs the computational result $\theta_{F(X)}$ comprising the two plaintexts, the server id and the server's signature on the output.

Algorithm 6 $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{(\omega,\mathbb{S})}, ek_{(\mathbb{O},\psi),S_i}, sk_{S_i}, pp)$

Parse $\sigma_{(\omega,\mathbb{S})}$ as (c_0, c_1) and $ek_{(\mathbb{O},\psi),S_i}$ as $(sk^0_{\mathrm{ABE}}, sk^1_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, uk^1_{L_{\mathrm{Rev}},t})$
$d_0 \leftarrow \mathsf{DPABE.Decrypt}\left(c_0, sk^0_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, mpk^0_{\mathrm{ABE}}\right)$
if ($\mathtt{mode} = \mathrm{VDC}$) **then**
    $d_1 \leftarrow \mathsf{DPABE.Decrypt}\left(c_1, sk^0_{\mathrm{ABE}}, uk^0_{L_{\mathrm{Rev}},t}, mpk^0_{\mathrm{ABE}}\right)$
else
    $d_1 \leftarrow \mathsf{DPABE.Decrypt}\left(c_1, sk^1_{\mathrm{ABE}}, uk^1_{L_{\mathrm{Rev}},t}, mpk^1_{\mathrm{ABE}}\right)$
endif
$\gamma \leftarrow_{\$} \mathsf{Sig.Sign}(d_0, d_1, S_i, sk_{S_i})$
$\theta_{F(X)} \leftarrow (d_0, d_1, S_i, \gamma)$

7. Verify, presented in Algorithm 7, determines whether the returned computational result is valid or not. The algorithm first checks whether the function F is listed in $L_{\mathrm{Reg}}[S][1]$ to ensure that the server that generated the computational result is authorised to compute F. If this check fails, the result is immediately rejected. Next, the algorithm verifies the signature using the verification key for S_i stored in L_{Reg}. If correct, it applies the one-way function g to each plaintext in $\theta_{F(X)}$ and compares the results to the components of the verification key. If either comparison results in a match (i.e. the server successfully recovered a message), the algorithm creates an acceptance token $\tau_{\theta_{F(X)}} = (\mathtt{accept}, S_i)$ indicating that the server indeed performed the computation correctly. Otherwise the result is rejected, and the algorithm creates a rejection token $\tau_{\theta_{F(X)}} = (\mathtt{reject}, S_i)$ and S_i is reported for revocation. If m_0 was returned then $F(X) = 1$ as m_0 was encrypted for the non-complemented inputs. Otherwise m_1 was returned and thus $F(X) = 0$. Note that this algorithm can be run by any entity since the computational result and verification key are publicly available.

Algorithm 7 $(y, \tau_{\theta_{F(X)}}) \leftarrow \mathsf{Verify}(\theta_{F(X)}, vk_{(\omega,\mathbb{S})}, pp)$

Parse $\theta_{F(x)}$ as (d_0, d_1, S_i, γ) and $vk_{(\omega,\mathbb{S})}$ as $(g(m_0), g(m_1), L_{\text{Reg}})$
if $F \in L_{\text{Reg}}[S_i][1]$ **then**
  if $\texttt{accept} \leftarrow \mathsf{Sig.Verify}((d_0, d_1, S_i), \gamma, L_{\text{Reg}}[S_i][0])$
    if $g(m_0) = g(d_0)$ **return** $(y \leftarrow 1, \tau_{\theta_{F(X)}} \leftarrow (\texttt{accept}, S_i))$
    elseif $g(m_1) = g(d_1)$ **return** $(y \leftarrow 0, \tau_{\theta_{F(X)}} \leftarrow (\texttt{accept}, S_i))$
    else return $(y \leftarrow \perp, \tau_{\theta_{F(x)}} \leftarrow (\texttt{reject}, S_i))$
    endif
  endif
endif
return $(y \leftarrow \perp, \tau_{\theta_{F(x)}} \leftarrow (\texttt{reject}, S_i))$

8. Revoke, presented in Algorithm 8, aims to revoke misbehaving servers by redistributing fresh update keys to all non-revoked servers. The algorithm first checks whether a server S_i should in fact be revoked, i.e. whether it received as input a rejection token $\tau_{\theta_{F(X)}} = (\texttt{reject}, S_i)$. If so, it deletes the list $L_{\text{Reg}}[S_i][1]$ of computations that S_i may perform such that the server is no longer authorised to perform any computations within the system. Additionally, it also adds S_i to the revocation list L_{Rev}, and refreshes the time source $\mathbb{T}$ and samples the new time period. The algorithm then generates new update keys for all non-revoked entities such that non-revoked keys are still functional in the new time period and distributes them accordingly. If the algorithm receives as input an acceptance token indicating that there is no need to revoke any server since computations were performed correctly, it outputs $\perp$.

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

if $\tau_{\theta_{F(X)}} = (\texttt{reject}, S_i)$ **then**
  $L_{\text{Reg}}[S_i][1] \leftarrow \{\epsilon\}$
  $L_{\text{Rev}} \leftarrow L_{\text{Rev}} \cup S_i$
  Refresh $\mathbb{T}$
  $t \leftarrow \mathbb{T}$
  $uk^0_{L_{\text{Rev}},t} \leftarrow_{\$} \mathsf{DPABE.KeyUpdate}(L_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$
  if $(\texttt{mode} = \text{RPVC})$ **or** $(\texttt{mode} = \text{RPVC-AC})$ **then**
    $uk^1_{L_{\text{Rev}},t} \leftarrow_{\$} \mathsf{ABE.KeyUpdate}(L_{\text{Rev}}, t, msk^1_{\text{ABE}}, mpk^1_{\text{ABE}})$
  endif
  for $S' \in \mathcal{U}_{\text{ID}}$ **do**
    Parse $ek_{F,S'}$ as $(sk^0_{\text{ABE}}, sk^1_{\text{ABE}}, uk^0_{L_{\text{Rev}},t-1}, uk^1_{L_{\text{Rev}},t-1})$
    $ek_{F,S'} \leftarrow (sk^0_{\text{ABE}}, sk^1_{\text{ABE}}, uk^0_{L_{\text{Rev}},t}, uk^1_{L_{\text{Rev}},t})$
  endfor
  return $um \leftarrow \{ek_{F,S'}\}_{S' \in \mathcal{U}_{\text{ID}}}$
else
  return $\perp$

Theorem 1. *Given an* IND-sHRSS *secure* rkDPABE *scheme for a class of monotone Boolean functions* $\mathcal{F}$ *closed under complement, an* EUF-CMA *secure signature scheme and a one-way function* g*. Let* $\mathcal{HPVC}$ *be the hybrid publicly verifiable outsourced computation scheme as defined in Algorithms 1–8. Then* $\mathcal{HPVC}$ *is secure in the sense of selective public verifiability (Figure 1), and selective semi-static revocation (Figure 2) and selective authorised computation (Figure 3).*

5 Proofs of Security

In this section we present the full proof of Theorem 1 by providing proofs of security for the notions of selective public verifiability, selective semi-static revocation and selective authorised computations.

5.1 Selective Public Verifiability

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

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

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

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

Game 0 *to* **Game 1**. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**, both with parameters $(\mathcal{HPVC}, 1^\lambda, \mathcal{F})$. Suppose otherwise, that $\mathcal{A}_{\text{HPVC}}$ can distinguish the two games with non-negligible advantage δ. We then show that it is possible to construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{HPVC}}$ as a subroutine to break the IND-sHRSS security of the (indirectly) revocable key DP-ABE scheme formalised in Figure 6. Note that we only focus on the modes RPVC and VDC, and the mode RPVC-AC can be seen as a special case of the mode RPVC as we can assume the adversary being authorised to evaluate a challenge computation. We consider a challenger $\mathcal{C}$ playing the IND-sHRSS game (Figure 6) with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ in turn acts as a challenger for $\mathcal{A}_{\text{HPVC}}$. Given the above parameters the entities interact in the following way.

1. $\mathcal{A}_{\text{HPVC}}$ declares its choice of challenge parameters $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode})$ including a set of labels $L_{F,X^\star}$ and the mode of computation `mode` detailing in which mode the challenge needs to be generated.
2. $\mathcal{A}_{\text{ABE}}$ must send a challenge attribute set and policy $(\widetilde{\omega}, \widetilde{\mathbb{S}})$, and a challenge time period $\widetilde{t}$ to the challenger as $\mathcal{A}_{\text{ABE}}$'s challenge input for the IND-sHRSS game of the rkDP-ABE scheme. Recall from Table 1 that in case `mode` = VDC the challenge subjective policy $\mathbb{S}^\star$ corresponds to the function F and the subjective attribute set ψ corresponds to the challenge input data $X^\star \subseteq D_i$. Also following Table 1, in case `mode` = RPVC the challenge objective policy $\mathbb{O}^\star$ corresponds to the function F and the objective attribute set ω corresponds to the challenge input data $X^\star$. In either mode, the other challenge input parameters correspond to either dummy attributes or dummy policies, and these dummy policies are trivially satisfied by the dummy attributes (cf. Section 4.3). As usual, $\mathcal{A}_{\text{ABE}}$ computes $r = F(X^\star)$.
   - If `mode` = VDC, we need to set the challenge input pair to the IND-sHRSS game of the rkDP-ABE scheme such that the pair is not satisfied by the challenge input $X^\star$ and thus need to set $\widetilde{\mathbb{S}}$ to be unsatisfied.
     - If $r = 1$: we set

$$\widetilde{\omega} = A_{\omega^\star} = \{T_{\text{O}}\},$$

and

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

- If $r = 0$: we set

$$\widetilde{\omega} = A_{\omega^\star} = \{T_{\mathrm{O}}\},$$

and

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

– If $\texttt{mode} = \mathrm{RPVC}$, then we set

$$\widetilde{\omega} = A_{\omega^\star} = A_{X^\star},$$

and

$$\widetilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j = \{\{T_{\mathrm{S}}\}\} \wedge \{l(F)\}.$$

Finally, $\mathcal{A}_{\mathrm{ABE}}$ also sets its challenge $(\widetilde{\omega}, \widetilde{\mathbb{S}})$ for the time period $\widetilde{t} = 1$ for the IND-sHRSS game and sends all challenge parameters to the challenger $\mathcal{C}$.

3. $\mathcal{C}$ runs the $\mathsf{DPABE.Setup}$ algorithm to generate $mpk_{\mathrm{ABE}}, msk_{\mathrm{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\mathrm{ABE}}$.
4. $\mathcal{A}_{\mathrm{ABE}}$ initialises its target revocation list $\overline{R}$ which is initially empty and sends it to $\mathcal{C}$, and simulates running $\mathsf{HPVC.Setup}$ such that the outcome is consistent with the previously generated mpk_{ABE}. If $\texttt{mode} = \mathrm{VDC}$, it sets $mpk^0_{\mathrm{ABE}} \leftarrow mpk_{\mathrm{ABE}}$ as provided by the challenger and implicitly sets $msk^0_{\mathrm{ABE}} \leftarrow msk_{\mathrm{ABE}}$. Note that any use of msk^0_{ABE} will be simulated using oracle calls to the challenger. If $\texttt{mode} = \mathrm{RPVC}$, it sets $mpk^r_{\mathrm{ABE}} \leftarrow mpk_{\mathrm{ABE}}$ as issued by $\mathcal{C}$, and implicitly sets $msk^r_{\mathrm{ABE}} \leftarrow msk_{\mathrm{ABE}}$ to be the key held by the challenger. In either case, $\mathcal{A}_{\mathrm{ABE}}$ executes $\mathsf{DPABE.Setup}$ itself to generate a second DP-ABE system.
5. $\mathcal{A}_{\mathrm{ABE}}$ runs $\mathsf{HPVC.FnInit}$ as detailed in Algorithm 2.
6. $\mathcal{A}_{\mathrm{ABE}}$ must generate a challenge problem instance for $\mathcal{A}_{\mathrm{HPVC}}$ as the output of $\mathsf{HPVC.ProbGen}$. To do so, $\mathcal{A}_{\mathrm{ABE}}$ samples three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. $\mathcal{A}_{\mathrm{ABE}}$ provides m_0 and m_1 as its choice of challenge to $\mathcal{C}$, and receives back the encryption, $ct^\star$, of *one* of these messages ($m_{b^\star}$ for $b^\star \xleftarrow{\$} \{0,1\}$, where $b^\star$ is chosen by the challenger), under the challenge attribute set and policy $(\widetilde{\omega}, \widetilde{\mathbb{S}})$ and challenge time period $\widetilde{t}$. More formally, $ct^\star \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m_{b^\star}, (\widetilde{\omega}, \widetilde{\mathbb{S}}), \widetilde{t}, mpk_{\mathrm{ABE}})$. It needs to assign $ct^\star$ to be one of the ciphertexts c or c' that form the challenge problem instance (encoded input) $\sigma_{F,X^\star}$ using the correct ABE system parameters. $\mathcal{A}_{\mathrm{ABE}}$ chooses a random bit $s \xleftarrow{\$} \{0,1\}$ which intuitively corresponds to its guess for the challenger's choice of $b^\star$.
   – If $\texttt{mode} = \mathrm{VDC}$, we need to distinguish the following cases.
     - If $r = 1$, $\mathcal{A}_{\mathrm{ABE}}$ generates

$$c \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m', \widetilde{\omega}, \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\mathrm{ABE}})$$

     and sets $c' = ct^\star$. It also sets $vk = g(m')$ and $vk' = g(m_s)$.
     - If $r = 0$, $\mathcal{A}_{\mathrm{ABE}}$ sets $c = ct^\star$ and generates

$$c' \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\mathrm{ABE}}).$$

     It also sets $vk = g(m_s)$ and $vk' = g(m')$.
   – If $\texttt{mode} = \mathrm{RPVC}$, $\mathcal{A}_{\mathrm{ABE}}$ sets $c = ct^\star$ and generates

$$c' \xleftarrow{\$} \mathsf{DPABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l \in L_{F,X^\star}} l, \widetilde{t}, mpk^1_{\mathrm{ABE}}).$$

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

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

7. $\mathcal{A}_{\text{HPVC}}$ receives all outputs from the above $\mathsf{HPVC.ProbGen}$ algorithm, and then is provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds in the following way:
   - $\mathsf{HPVC.FnInit}(\cdot, mk, pp)$ and $\mathsf{HPVC.Register}(\cdot, mk, pp)$ are executed as specified in Algorithms 2 and 3.
   - $\mathsf{HPVC.Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$: in order to generate an evaluation key for the queried parameters, $\mathcal{A}_{\text{ABE}}$ needs to request queries to the KeyGen oracle in the rkDP-ABE game formalised in Figure 6. $\mathcal{A}_{\text{ABE}}$ updates first the usual list entries and then sets $\mathbb{O}' = \mathbb{O}$ and $\psi' = A_\psi \cup \bigcup_{l_j \in L_i} l_j$ and requests an oracle query to the challenger for $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}', \psi'), mk, pp)$ as specified in Figure 6. The challenger shall generate a rkDP-ABE decryption key if and only if $\widetilde{\omega} \notin \mathbb{O}'$ or $\psi' \notin \widetilde{\mathbb{S}}$ or $S_i \in \overline{R}$. Note that $S_i \notin \overline{R}$ is fulfilled since we chose $\overline{R}$ to be empty. By construction, the condition $\psi' \in \widetilde{\mathbb{S}}$ is satisfied only if the labels $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$. As specified above, if the labels do not satisfy this relation then $\psi' \notin \widetilde{\mathbb{S}}$ and the challenger may generate the key, which $\mathcal{A}_{\text{ABE}}$ will receive as sk^0_{ABE}.
     If, on the other hand, the labels do satisfy this relation then we have the following cases depending on the chosen mode.
     - If $\mathtt{mode}$ = VDC, then from the above relation $\{l_k\}_{l_k \in L_{F,X^\star}} \subseteq \{l_j\}_{l_j \in L_i}$ it follows that $\{l(x_{i,k})\}_{x_{i,k} \in X^\star} \subseteq \{l(x_{i,j})\}_{x_{i,j} \in D_i}$ and thus it follows that $X^\star \subseteq D_i$. Thus, this means that by the uniqueness of the labels within the system, $\mathcal{A}_{\text{HPVC}}$ has requested an evaluation key for a superset of the challenge input set $X^\star$, i.e. the set D_i that contains the challenge input set $X^\star$ and possibly some more additional data points. If $X^\star \subseteq D_i$, then the data set D_i must satisfy either F or $\overline{F}$ in order to satisfy $\widetilde{\mathbb{S}}$. However, $\widetilde{\mathbb{S}}$ was chosen in such a way that it is not satisfied by $X^\star$ and thus also not by D_i. Hence, the challenger may generate a valid key which $\mathcal{A}_{\text{ABE}}$ stores as sk^0_{ABE}.
     - If $\mathtt{mode}$ = RPVC, then (as specified in Table 1) both sets L_i and $L_{F,X^\star}$ are singleton sets. Thus, from $\{l_j\}_{l_j \in L_i} \supseteq \{l_k\}_{l_k \in L_{F,X^\star}}$ it follows that $L_i = L_{F,X^\star} = \{l(F)\}$. By the uniqueness of the labels within the system, it then follows that $\mathbb{O} = \mathbb{O}^\star$ which means that $\mathcal{A}_{\text{HPVC}}$ has requested an evaluation key for the challenge function F. However, in step 4, the challenger got assigned the ABE system with master secret key msk^r_{ABE} such that $\mathbb{O}^\star$ is not satisfied by the challenge input $\widetilde{\omega}$. Therefore, also $\mathbb{O}'$ is not satisfied either by the challenge input $\widetilde{\omega}$ and hence the challenger may generate a valid key which $\mathcal{A}_{\text{ABE}}$ stores as sk^r_{ABE}.

     $\mathcal{A}_{\text{ABE}}$ needs further to make queries to a $\mathsf{KeyUpdate}$ oracle $\mathcal{O}^{\mathsf{KeyUpdate}}$ to the challenger in order to obtain an update key. The challenger returns a valid key if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Observe that the second condition is satisfied since $\overline{R} = \epsilon$ and hence is a subset of Q_{Rev}. Hence a challenger may generate a valid update key.
     Also if $\mathtt{mode}$ = RPVC, then $\mathcal{A}_{\text{ABE}}$ additionally generates a secret key sk^{1-r}_{ABE} by itself using the parameters of the second DP-ABE system which it owns for the pair $(\overline{\mathbb{O}}, \overline{\psi})$.
   - $\mathsf{HPVC.Revoke}(\tau_{\theta_{F(X)}}, mk, pp)$: whenever a Revoke query is requested, $\mathcal{A}_{\text{ABE}}$ executes Algorithm 8 as specified except it requires to make a $\mathsf{KeyUpdate}$ oracle query to the challenger for the update key that relates to the ABE system owned by $\mathcal{C}$. If $\mathtt{mode}$ = RPVC, this is the key $uk^r_{L_{\text{Rev}},t}$, and if $\mathtt{mode}$ = VDC, this is the key $uk^0_{L_{\text{Rev}},t}$. The challenger may create a valid update key if and only if $t \neq q_t$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Since $\overline{R}$ was defined to be an empty list and hence is a subset of Q_{Rev} the challenger may always return a valid update key.

   Eventually $\mathcal{A}_{\text{HPVC}}$ finishes its query phase and outputs a guess $\theta^\star$ which it believes to be a valid forgery.
8. $\mathcal{A}_{\text{ABE}}$ parses the guess $\theta^\star$ as (d, d', S_i, γ). One of the values d and d' will be $\perp$ (by construction) and we denote the other value (non-$\perp$) by Y. Observe that, since $\mathcal{A}_{\text{HPVC}}$ is assumed to be a successful adversary against selective public verifiability, the non-$\perp$ value, Y, that it will return will be the plaintext m_s since the challenge access structure was always set to be unsatisfied on the challenge input. Thus, if $g(Y) = g(m_s)$, $\mathcal{A}_{\text{ABE}}$ outputs a guess $b' = s$ and otherwise guesses $b' = (1 - s)$.

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

the verification key comprises the one-way function g applied to a legitimate message m' and a random message $m_{1-b^\star}$ that is unrelated to both ciphertexts.

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

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

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

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

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

Hence,

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

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

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

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

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

Now, since $\mathcal{A}_{\text{HPVC}}$ is assumed to be successful, it will output a forgery comprising the plaintext that was encrypted under the unsatisfied function (F or $\overline{F}$) that evaluates to 0. By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{\text{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\text{HPVC}}$ has against the selective public verifiability game.

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

5.2 Selective, Semi-static Revocation

Lemma 2. *The $\mathcal{HPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective, semi-static revocation (Figure 2) under the same assumptions as in Theorem 1.*

Proof. In this proof, we aim to perform a reduction from the the selective, semi-static revocation game (Figure 2) to the IND-sHRSS security of the underlying revocable key DP-ABE scheme (Figure 6). We wish the prove this reduction by achieving a contradiction and therefore we assume that $\mathcal{A}_{\text{HPVC}}$ is an adversary with non-negligible probability against the selective, semi-static revocation game when instantiated by Algorithms 1–8, and making q_t Revoke queries. We show that we can construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{HPVC}}$ as a sub-routine to break the IND-sHRSS security of the indirectly revocable key DP-ABE scheme. Note that as in the previous proof, we only focus on the modes RPVC and VDC, and the mode RPVC-AC can be seen as a special case of the mode RPVC as we can assume the adversary being authorised to evaluate a challenge computation. Let $\mathcal{C}$ be a challenger playing the IND-sHRSS game with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ acts as a challenger for $\mathcal{A}_{\text{HPVC}}$.

1. $\mathcal{A}_{\text{RPVC}}$ declares its choice of challenge input parameters $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \texttt{mode})$ for a challenge computation $F(X^\star)$ including a set of labels $L_{F,X^\star}$ and the mode of computation $\texttt{mode}$ detailing in which mode the challenge needs to be generated.
2. $\mathcal{A}_{\text{ABE}}$ initialises an (empty) list Q_{Rev} of currently revoked entities and sets the current time period $t = 1$. Next, $\mathcal{A}_{\text{ABE}}$ needs to form its own challenge input for the IND-sHRSS game. $\mathcal{A}_{\text{ABE}}$ sets its challenge for the time period $\widetilde{t} = q_t$, and it forms $\widetilde{\omega} = A_{\omega^\star}$ and $\widetilde{\mathbb{S}} = \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j$. Finally, it sends $(\widetilde{t}, (\widetilde{\omega}, \widetilde{\mathbb{S}}))$ to the challenger.
3. $\mathcal{C}$ runs the $\mathsf{DPABE.Setup}$ algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.
4. $\mathcal{A}_{\text{ABE}}$ simulates running $\mathsf{HPVC.Setup}$ such that the outcome is consistent with the previously generated mpk_{ABE} from $\mathcal{C}$. It executes the algorithm as detailed with the exception of line 2, since msk^0_{ABE} and mpk^0_{ABE} were already generated by the challenger.
5. $\mathcal{A}_{\text{ABE}}$ runs $\mathsf{HPVC.FnInit}$ as detailed in Algorithm 2.
6. $\mathcal{A}_{\text{HPVC}}$ chooses a challenge revocation list $\overline{R}$, which $\mathcal{A}_{\text{ABE}}$ forwards to $\mathcal{C}$.
7. $\mathcal{A}_{\text{HPVC}}$ is provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds in the following way:
   – $\mathsf{HPVC.FnInit}(\cdot, mk, pp)$ and $\mathsf{HPVC.Register}(\cdot, mk, pp)$ are executed as specified in Algorithms 2 and 3.
   – Queries of the form $\mathsf{HPVC.Certify}(\texttt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the $\mathsf{Certify}$ oracle as specified in Figure 2. $\mathcal{A}_{\text{ABE}}$ executes Algorithm 4 as detailed except lines 6 and 7, as these rely on the master secret key msk^0_{ABE} held by the challenger. In order to simulate line 6, $\mathcal{A}_{\text{ABE}}$ requires to make a KeyGen oracle query of the form $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (\mathbb{O}, A_\psi \cup \bigcup_{l_k \in L_i} l_k), msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. The challenger responds by running the KeyGen oracle as detailed in Figure 6 which returns a valid key if and only if $\widetilde{\omega} \notin \mathbb{O}$ or $A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \widetilde{\mathbb{S}}$ or $S_i \in \overline{R}$. Now if $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \widetilde{\mathbb{S}})$ is fulfilled then the challenger can return a valid decryption key. By

construction, we observe that the condition $\psi \in \widetilde{\mathbb{S}}$ is satisfied only if $\{l_k\}_{l_k \in L_i} \supseteq \{l_j\}_{l_j \in L_{F,X^\star}}$. By the uniqueness of the label within the system this implies $L_{F,X^\star} \subseteq L_i$. However, in this case, the first condition in the "if" statement in the Certify oracle in Figure 2 is satisfied and thus $\mathcal{A}_{\text{ABE}}$ would have returned $\perp$ without querying KeyGen if $S_i \notin \overline{R}$ to avoid certifying $\mathcal{A}_{\text{HPVC}}$ for the challenge computation. If $(A_\psi \cup \bigcup_{l_k \in L_i} l_k \in \widetilde{\mathbb{S}})$ is satisfied at the point of making a KeyGen query, then $S_i \in \overline{R}$, and thus the challenger can respond to all queries made to it during this phase with a valid key.

In order to simulate line 7, $\mathcal{A}_{\text{ABE}}$ makes a query to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. Here the challenger responds as detailed in Figure 6 which returns a valid update key if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Recall that $\mathcal{A}_{\text{ABE}}$ chose $\widetilde{t} = q_t$, and at the point of calling the KeyUpdate oracle, the list of currently revoked entities corresponds to $Q_{\text{Rev}} \leftarrow Q_{\text{Rev}} \setminus S_i$. Therefore, if the challenger returns $\perp$ in response to this query, then $\mathcal{A}_{\text{ABE}}$ would already have returned $\perp$ as a response to the Certify oracle (Figure 2) as a result of the second condition in the "if" statement. Hence, for all queries made to the challenger, a valid update key is returned.

– Queries of the form $\mathsf{HPVC.Revoke}(\tau_{\theta_{F(X^\star)}}, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the Revoke oracle as specified in Figure 2. In order to simulate running the algorithm, $\mathcal{A}_{\text{ABE}}$ executes Algorithm 8 with the exception of line 6. Here $\mathcal{A}_{\text{ABE}}$ is required to make KeyUpdate oracle calls to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. Note that the Revoke oracle in Figure 2 returns $\perp$ if $t = q_t$ and $\overline{R} \not\subseteq Q_{\text{Rev}} \setminus S_i$. This corresponds directly to the conditions that $\mathcal{C}$ cannot form a valid update key through a KeyUpdate oracle call (Figure 6) since $t = \widetilde{t}$ and $\overline{R} \not\subseteq Q_{\text{Rev}}$. However, since S_i was already removed from the list of currently revoked entities Q_{Rev}, $\mathcal{C}$ can form a valid update key and $\mathcal{A}_{\text{ABE}}$ can simulate the remainder of the algorithm.

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

9. $\mathcal{A}_{\text{ABE}}$ must now generate a challenge for $\mathcal{A}_{\text{HPVC}}$. $\mathcal{A}_{\text{ABE}}$ chooses three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. It then sends m_0 and m_1 to $\mathcal{C}$ as its choice of challenge for the IND-sHRSS game. $\mathcal{C}$ chooses a random bit $b^\star \xleftarrow{\$} \{0,1\}$ and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{b^\star}, \widetilde{\omega}, \mathbb{S}^\star \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\text{ABE}})$. $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and generates depending on the chosen mode the second ciphertext. If $\texttt{mode} = \text{VDC}$, then $c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^0_{\text{ABE}})$. In case $\texttt{mode} = \text{RPVC}$, then $c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \widetilde{\omega}, \overline{\mathbb{S}^\star} \wedge \bigwedge_{l_j \in L_{F,X^\star}} l_j, \widetilde{t}, mpk^1_{\text{ABE}})$. Finally, $\mathcal{A}_{\text{ABE}}$ forms the challenge problem instance $\sigma^\star = (c, c')$. $\mathcal{A}_{\text{ABE}}$ selects a bit $s \xleftarrow{\$} \{0,1\}$ and forms the verification key as $vk^\star = (g(m_s), g(m'), L_{\text{Reg}})$. Note that s intuitively corresponds to $\mathcal{A}_{\text{ABE}}$'s guess for $b^\star$.

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

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

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

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

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

Hence,

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

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

5.3 Selective Authorised Computation

Lemma 3. *The $\mathcal{HPVC}$ scheme defined by Algorithms 1–8 is secure in the sense of selective authorised computation (Figure 3) under the same assumptions as in Theorem 1.*

Proof. In this proof, we aim to perform a reduction from the selective authorised computation game (Figure 3) to the IND-sHRSS security of the underlying revocable key DP-ABE scheme (Figure 6). We wish to prove this reduction by achieving a contradiction and therefore we assume that $\mathcal{A}_{\text{HPVC}}$ is an adversary with non-negligible probability against the selective authorised computation game when instantiated by Algorithms 1–8. We show that we can construct an adversary $\mathcal{A}_{\text{ABE}}$ that uses $\mathcal{A}_{\text{HPVC}}$ as a subroutine to break the IND-sHRSS security of the indirectly revocable key DP-ABE scheme. Note that the notion of selective authorised computation is only meaningful as long as the system is run in the RPVC-AC mode. Let $\mathcal{C}$ be a challenger playing the IND-sHRSS game with $\mathcal{A}_{\text{ABE}}$, and $\mathcal{A}_{\text{ABE}}$ acts as a challenger for $\mathcal{A}_{\text{HPVC}}$.

1. $\mathcal{A}_{\text{HPVC}}$ begins by declaring its choice of challenge input parameters for the RPVC-AC mode consisting of F, $X^\star$, the authorisation policy P and the function label $\{l(F)\}$.

2. $\mathcal{A}_{\text{ABE}}$ needs to form its own challenge input for the IND-sHRSS game. Thus, $\mathcal{A}_{\text{ABE}}$ sets its challenge for the time period $\widetilde{t} = 1$, and it forms $\widetilde{\omega} = A_{X^\star}$ and $\widetilde{\mathbb{S}} = P \wedge \{l(F)\}$. Finally, it sends $(\widetilde{t}, (\widetilde{\omega}, \widetilde{\mathbb{S}}))$ to the challenger.
3. $\mathcal{C}$ runs the DPABE.Setup algorithm to generate $mpk_{\text{ABE}}, msk_{\text{ABE}}$ and sends mpk_{ABE} to $\mathcal{A}_{\text{ABE}}$.
4. $\mathcal{A}_{\text{ABE}}$ simulates running HPVC.Setup such that the outcome is consistent with the previously generated mpk_{ABE} from $\mathcal{C}$. It executes the algorithm as detailed with the exception of line 2, since msk^0_{ABE} and mpk^0_{ABE} were already generated by the challenger. $\mathcal{A}_{\text{ABE}}$ chooses an empty list of currently revoked entities $\overline{R}$ and sends it to the challenger.
5. $\mathcal{A}_{\text{ABE}}$ runs HPVC.FnInit as detailed in Algorithm 2.
6. $\mathcal{A}_{\text{ABE}}$ must now generate a challenge for $\mathcal{A}_{\text{HPVC}}$. $\mathcal{A}_{\text{ABE}}$ chooses three distinct, equal length messages m_0, m_1 and m' uniformly at random from the message space. It then sends m_0 and m_1 to $\mathcal{C}$ as its choice of challenge for the IND-sHRSS game. $\mathcal{C}$ chooses a random bit $b^\star \xleftarrow{\$} \{0,1\}$ and returns $ct^\star \xleftarrow{\$} \mathsf{ABE.Encrypt}(m_{b^\star}, \widetilde{\omega}, P \wedge \{l(F)\}, \widetilde{t}, mpk^0_{\text{ABE}})$. $\mathcal{A}_{\text{ABE}}$ sets $c = ct^\star$ and generates itself the second ciphertext by encrypting m' as $c' \xleftarrow{\$} \mathsf{ABE.Encrypt}(m', \widetilde{\omega}, \overline{P} \wedge \{l(F)\}, \widetilde{t}, mpk^1_{\text{ABE}})$. Finally, $\mathcal{A}_{\text{ABE}}$ forms the challenge problem instance $\sigma^\star = (c, c')$. $\mathcal{A}_{\text{ABE}}$ selects a bit $s \xleftarrow{\$} \{0,1\}$ and forms the verification key as $vk^\star = (g(m_s), g(m'), L_{\text{Reg}})$. Note that s intuitively corresponds to $\mathcal{A}_{\text{ABE}}$'s guess for $b^\star$.
7. $\mathcal{A}_{\text{HPVC}}$ receives the resulting parameters from ProbGen and is provided with oracle access to which $\mathcal{A}_{\text{ABE}}$ responds in the following way:

   - HPVC.FnInit$(\cdot, mk, pp)$ and HPVC.Register$(\cdot, mk, pp)$ are executed as specified in Algorithms 2 and 3.
   - Queries of the form HPVC.Certify$(\text{RPVC-AC}, S_i, (F, \psi), \{l(F)\}, \mathcal{F}_i, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ by running the Certify oracle as specified in Figure 3. In case the queried set of subjective attributes ψ satisfy the challenge authorisation policy then $\mathcal{A}_{\text{ABE}}$ returns $\perp$. Otherwise, $\mathcal{A}_{\text{ABE}}$ executes Algorithm 4 as detailed with the exception in lines 6 and 7, as these rely on the master secret key msk^0_{ABE} held by the challenger. In order to simulate line 6, $\mathcal{A}_{\text{ABE}}$ requires to make a KeyGen oracle query of the form $\mathcal{O}^{\mathsf{KeyGen}}(S_i, (F, A_\psi \cup \bigcup_{l_k \in L_i} l_k), msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. The challenger responds by running the KeyGen oracle as detailed in Figure 6 which returns a valid key if and only if $\widetilde{\omega} \notin \mathbb{O}$ or $A_\psi \cup \bigcup_{l_k \in L_i} l_k \notin \widetilde{\mathbb{S}}$ or $S_i \in \overline{R}$. However, for the query to have been made to KeyGen, $\mathcal{A}_{\text{ABE}}$ must not have returned $\perp$ in the Certify oracle request in Figure 3 and therefore $\psi \notin P$, and hence $\psi \notin \widetilde{\mathbb{S}}$. Therefore, the challenger can always return a valid decryption key sk^0_{ABE}.

     In order to simulate line 7 in the Certify algorithm, $\mathcal{A}_{\text{ABE}}$ makes a query to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. Here the challenger responds as detailed in Figure 6 which returns a valid update key if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Since $\overline{R}$ was initially chosen to be empty and thus $\overline{R} \subseteq Q_{\text{Rev}}$ for any Q_{Rev}. Therefore, the challenger can create a valid update key.
   - Queries of the form HPVC.Revoke$(\tau_{\theta_{F(X^\star)}}, mk, pp)$ are handled by $\mathcal{A}_{\text{ABE}}$ executing Algorithm 8 with the exception of line 6. Here $\mathcal{A}_{\text{ABE}}$ is required to make KeyUpdate oracle calls to the challenger of the form $\mathcal{O}^{\mathsf{KeyUpdate}}(Q_{\text{Rev}}, t, msk^0_{\text{ABE}}, mpk^0_{\text{ABE}})$. The challenger returns a valid update key through a KeyUpdate oracle call (Figure 6) if and only if $t \neq \widetilde{t}$ or $\overline{R} \subseteq Q_{\text{Rev}}$. Since $\overline{R}$ was initially chosen to be empty and thus $\overline{R} \subseteq R$ for any R and in particular L_{Rev}. Therefore, the challenger can always create a valid update key.

8. Eventually $\mathcal{A}_{\text{HPVC}}$ finishes its oracle query phase and outputs its guess $\theta^\star$ which corresponds to the result of $F(X^\star)$ protected by an authorisation policy P. Note that $\mathcal{A}_{\text{HPVC}}$ never received a key for a set of authorisation attributes $s \in P$.
9. As $\theta^\star$ should appear valid, by construction it should contain a non-$\perp$ plaintext which we denote by Y. If $g(Y) = g(m_s)$, $\mathcal{A}_{\text{ABE}}$ guesses $b' = s$. Else, $\mathcal{A}_{\text{ABE}}$ guesses $b' = 1 - s$.

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

Now we consider the advantage of $\mathcal{A}_{\text{ABE}}$ playing the IND-sHRSS game. By assumption, $\mathcal{A}_{\text{HPVC}}$ has a non-negligible advantage δ against the selective authorised computation game. It follows

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

Hence,

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

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

6 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 [23, 2], 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 follow up work, we have investigated VDC further 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 attention in the literature, which we believe to be primarily due to its applications being less obvious than for the single-policy ABE schemes. Whilst KP-

and CP-ABE are generally considered in the context of cryptographic access control, it is unclear that the policies enforced by DP-ABE are natural choices for access control. Thus an interesting side-effect of this 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 gratefully acknowledges partial funding by the European Commission under project H2020-644024 "CLARUS", and support from BAE Systems Advanced Technology Centre.
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 Preliminaries

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

Linear Secret Sharing Schemes Secret sharing is a basic and fundamental cryptographic tool that enables a secret s to be shared amongst a set of entities in such a way that all *authorised* sets of entities can combine their individual share in order to reconstruct the secret value s. For example, any k out of the n entities may form an authorised set and thus are able to reconstruct the secret value. Any set of entities that does not form an authorised set cannot learn more than their individual shares and thus cannot reconstruct the secret s. A secret sharing scheme is *linear* if the reconstruction operation is a linear function of the shares. We provide a definition of an *access structure* which is a collection of satisfying sets of a Boolean formula.

Definition 7. *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of parties (or attributes). A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if for all B, C we have that if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotonic access structure) is a collection (respectively, monotone collection) $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the* authorised sets *and the sets not in $\mathbb{A}$ are called* unauthorised sets.

A linear secret sharing scheme can be defined as follows [26].

Definition 8. *Let $\mathcal{P}$ be a set of parties. Let M be a matrix of size $l \times k$. Let $\pi\colon \{1, \ldots, l\} \to \mathcal{P}$ be a function that maps a row to a party for labelling. A secret sharing scheme Π for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$ is a linear secret-sharing scheme (LSSS) in $\mathbb{Z}_p$ and is represented by (M, π) if it consists of two polynomial-time algorithms:*

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

Note that the set $\{(\mu_i)\}_{i \in I}$ can be found in polynomial-time in the size of M [26].

In Appendix B, we will require the following important fact [26]:

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 any authorised set $S \in \mathbb{A}$, the target vector $(1, 0, \ldots, 0)$ is in the span of $I = \{i : \pi(i) \in S\}$. For all unauthorised sets $S \notin \mathbb{A}$, the target vector is not in the span of I, and there exists a polynomial time algorithm that outputs a vector $\boldsymbol{w} = (w_1, \ldots, w_k) \in \mathbb{Z}_p^k$ such that $w_1 = -1$ and for all $i \in I$ it holds that $\boldsymbol{M_i} \cdot \boldsymbol{w} = 0$.*

In Appendix B, we make use of Lagrange interpolation as the reconstruction algorithm for LSSSs. The reconstruction procedure can be defined following Attrapadung and Imai [4] in the following way.

Definition 9. *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 dth 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 Appendix B, 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 Most ABE schemes are instantiated over groups with efficiently computable bilinear maps. Thus, we review the notions of bilinear maps and the hardness assumption on which we base the security of our revocable DP-ABE scheme in Section B. We follow the formalisation in [4, 5].

Definition 10. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative groups of order p, and let g be a generator of $\mathbb{G}$. A* bilinear map *is a map $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that:*

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

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

Definition 11. *Let $\mathbb{G}$ be a bilinear group of prime order p. The* decisional q-bilinear Diffie-Hellman exponent problem *(q-BDHE) in $\mathbb{G}$ is stated as follows. Given a vector*

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

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

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

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

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

B Revocable Dual-policy Attribute-based Encryption

Dual-policy attribute-based encryption was introduced by Attrapadung and Imai [6] and conjunctively combines KP-ABE and CP-ABE such that *both* the secret decryption key and the ciphertext comprise an access structure and an attribute set. The same authors [4] have also introduced the notion of revocation in ABE schemes which have been used to construct a revocable PVC scheme which can revoke misbehaving servers from the system. Recall that the notion of revocation supports two different modes, namely direct revocation and indirect revocation. The former notion enables a client to specify a revocation list at the point of encryption such that periodic re-keying is not necessary but the encryptors must have the knowledge of the specific (current) revocation list. On the other hand, indirect revocation requires a time period to be specified at the point of encryption and needs an authority that issues update key material at each time period in order to enable entities to update their key to stay functional during the time period. In [2], Alderman et al. have focused on the mode of indirect revocation, mainly as it minimises the client's workload as she is not required to maintain a synchronised revocation list. This mode is implemented in the KP-ABE setting by amending the policy including an entity identifier and by embedding the current time period into the ciphertext. Update keys are issued only to non-revoked entities at each time period. Note that only the combination of a secret key with an update key for a time period forms a functional evaluation key that is able to decrypt a ciphertext formed using the time period.

In this part, we aim to implement a revocation mechanism for a DP-ABE scheme. However, in this context, we are able to embed the revocation mechanism into the KP-ABE functionality *or* the CP-ABE functionality. Recall that decryption in DP-ABE is only successful if and only if *both* attribute sets satisfy their corresponding access structure. Thus, in order to prevent the decryption functionality to be successful, it suffices that at least one attribute set does not satisfy the corresponding access structure. The formal definition of a revocable DP-ABE scheme using indirect revocation in the key-policy was already stated in Section 4.2. Here we discuss the required security model as well as provide a concrete construction and respective security proof.

B.1 Security Model

The security model for a rkDP-ABE scheme is a natural extension of the IND-sHRSS security notion for an indirectly revocable KP-ABE scheme and the security notion is presented in Figure 6.

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

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

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

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

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

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

Fig. 6. The IND-sHRSS experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-sHRSS}}\left[\mathcal{RKDPABE}, 1^{\lambda}, \mathcal{U}\right]$

Definition 12. *The* advantage *of a PPT adversary in the* IND-sHRSS *game for a revocable key DP-ABE construction $\mathcal{RKDPABE}$ is defined as:*

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

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

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

B.2 Construction of a rkDP-ABE scheme

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 a LSSS denoted by (N, π).

Let $\mathcal{U}_\mathrm{s}$ and $\mathcal{U}_\mathrm{o}$ be the universe of subjective and objective attributes respectively. The objective attribute universe comprises disjoint sub-universes $\mathcal{N}, \mathcal{T}, \mathcal{M}$ and $\mathcal{U}_\mathrm{ID}$ referring to standard ABE attributes, time periods, messages and entity 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 rkDP-ABE scheme is defined to be $\mathcal{U} = \mathcal{U}_\mathrm{s} \cup \mathcal{U}_\mathrm{o}$. 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 Section A. We construct each algorithm of the rkDPABE scheme as follows:

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

$$F_s(x) = \prod_{j=0}^{m'} h_j^{x^j}, \quad F_o(x) = \prod_{j=0}^{n'} q_j^{x^j}, \quad P(x) = \prod_{j=0}^{d} u_j^{x^j}. \tag{1}$$

   The public parameters are defined as

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

   For each node label $x \in \mathcal{X}$ in the tree, it randomly chooses $a_x \in \mathbb{Z}_p$ and $r_x \in \mathbb{Z}_p$ to define a first degree polynomial $f_x(z) = a_x z + \alpha r_x + \gamma$. The master key is $mk = (\gamma, \alpha, \{a_x, r_x\}_{x \in \mathcal{X}})$.
2. $\mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, pp)$: The encryption algorithm takes as input a LSSS access structure (M, ρ) for the subjective policy $\mathbb{S}$ and an objective attribute set $\omega \subset \mathcal{U}_\mathrm{o}$. Denote the dimensions of M as $l_s \times k_s$ matrix. The algorithm randomly chooses values $s, y_2, \ldots, y_{k_s} \in \mathbb{Z}_p$ and sets $\boldsymbol{u} = (s, y_2, \ldots, y_{k_s})$. It computes $\lambda_i = \boldsymbol{M_i} \cdot \boldsymbol{u}$ (for $i = 1, \ldots, l_s$), where $\boldsymbol{M_i}$ is the vector corresponding to the ith row of M. The ciphertext is then computed as $ct_{(\omega,\mathbb{S}),t} = (C, C^{(1)}, \{C_k^{(2)}\}_{k \in \omega}, \{C_i^{(3)}\}_{i=1,\ldots,l_s}, C^{(4)})$, where

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

   Intuitively, C masks the message by a group element in the target group of the bilinear map formed from the master secret γ and an encryption secret s (to randomise the encryption procedure). Decryption will have to compute this mask to recover the message.
   $C^{(1)}$ provides the encryption secret s. $C_k^{(2)}$ embeds each attribute in the objective set ω into the ciphertext, incorporating the encryption secret s such that attributes from prior ciphertexts cannot be combined with this encryption. Similarly, $C_i^{(3)}$ embeds the subjective policy $\mathbb{S}$ into the ciphertext using the shares of s divided according to $\mathbb{S}$, i.e. s is shared over the set of attributes such that any set of attributes that satisfies $\mathbb{S}$ can reconstruct the encryption secret s. Finally, $C^{(4)}$ links the encryption secret (and hence this particular ciphertext) to the specified time period t such that an update key for t is required to decrypt the ciphertext; this enables the revocation mechanism.
3. $\mathsf{KeyGen}(\mathsf{id}, (\mathbb{O}, \psi), mk, pp)$: The key generation algorithm takes as input a LSSS access structure (N, π) for the objective policy $\mathbb{O}$ and a subjective attribute set $\psi \subset \mathcal{U}_\mathrm{s}$. Let the dimensions of N be denoted by $l_o \times k_o$. The algorithm also takes an identity $\mathsf{id} \in \mathcal{U}_\mathrm{ID}$ which is a leaf in the binary tree. For all $x \in \mathsf{Path}(\mathsf{id})$, the algorithm shares $f_x(1)$ using the LSSS (N, π). To do so, it randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and sets $\boldsymbol{v_x} = (f_x(1), z_{x,2}, \ldots, z_{x,k_o})$. For $i = 1, \ldots, l_o$, it calculates the share $\sigma_{x,i} = \boldsymbol{N_i} \cdot \boldsymbol{v_x}$, where $\boldsymbol{N_i}$ is the vector corresponding to the ith row of N.

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

$$sk_{\mathsf{id},(N,\pi)} = ((D^{(1)}_{x,i}, D^{(2)}_{x,i})_{x\in\mathsf{Path}(\mathsf{id}),i=1,\ldots,l_o}, (D_x, \{D^{(3)}_k\}_{k\in\psi})_{x\in\mathsf{Path}(\mathsf{id})}),$$

where

$$D_x = g^{r_x}, \qquad D^{(1)}_{x,i} = g^{r_{x,i}},$$
$$D^{(2)}_{x,i} = g^{\sigma_{x,i}} F_o(\pi(i))^{r_{x,i}}, \qquad D^{(3)}_k = F_s(k)^{r_x}.$$

Intuitively, r_x and $r_{x,i}$ for each $x \in \mathsf{Path}(\mathsf{id})$ randomises the key for the user id so that users may not collude. D_x and $D^{(1)}_{x,i}$ allow use of these random key values during decryption. $D^{(2)}_{x,i}$ embeds the shares of $f_x(1) = a_x + \alpha r_x + \gamma$ such that only the authorised sets according to $\mathbb{O}$ may reconstruct $f_x(1)$. Finally, $D^{(3)}_k$ embeds the attributes in ψ with the randomness chosen for this particular key. By linking these parameters to the path in a tree, only users for whom a valid update key has been issued (i.e. the non-revoked users) will be able to make use of these parameters to compute $f_x(1)$ for a node x; $f_x(1)$ is required as it contains the master secret γ which is used to cancel with the ciphertext component C to recover the message.

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

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

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

5. $\mathsf{Decrypt}(ct_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), sk_{\mathsf{id},(\mathbb{O},\psi)}, (\mathbb{O}, \psi), uk_{R,t}, pp)$: The decryption algorithm takes as an input the ciphertext $ct_{(\omega,\mathbb{S}),t}$ which contains a subjective access structure (M, ρ) for $\mathbb{S}$ and a set of objective attributes ω, and a decryption key $sk_{\mathsf{id},(N,\pi)}$ which contains a set of subjective attributes ψ and an objective access structure (N, π) for $\mathbb{O}$. Suppose that ψ satisfies (M, ρ), the set ω satisfies (N, π), and that $\mathsf{id} \notin R$ (so that decryption is possible).

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

$$C \cdot \frac{\prod_{i\in I_s} \left(e\left(C^{(3)}_i, D_x\right) \cdot e\left(C^{(1)}, D^{(3)}_{\rho(i)}\right)\right)^{\mu_i}}{\left(\prod_{j\in I_o} \left(\frac{e\left(D^{(2)}_{x,j}, C^{(1)}\right)}{e\left(C^{(2)}_{\pi(j)}, D^{(1)}_{x,j}\right)}\right)^{\nu_j}\right)^{\frac{t}{t-1}} \left(\frac{e\left(U^{(1)}_x, C^{(1)}\right)}{e\left(C^{(4)}, U^{(2)}_x\right)}\right)^{\frac{1}{1-t}}} = m.$$

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

of this product will be the result of computing C' whilst the first will cancel with C to leave only m.

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

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

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

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

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

Then, it follows

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

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

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

B.3 Security Proof

Theorem 2. *The* rkDPABE *construction is secure with respect to indistinguishability against selective-target with semi-static query attack (*IND-sHRSS*), as specified in Figure 6, assuming that the decisional q-BDHE problem is hard.*

The proof follows from 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 11) in $\mathbb{G}$. Recall that we abbreviate g^{a^j} by g_j. The simulator $\mathcal{B}$ is given a random q-BDHE challenge $(g, h, \boldsymbol{y}_{g,a,q}, Z)$ where $\boldsymbol{y}_{g,a,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ and Z is either $e(g_{q+1}, h)$ or a random element in $\mathbb{G}_1$. $\mathcal{B}$ acts as the challenger for $\mathcal{A}$ in the IND-sHRSS game as follows.

1. $\mathcal{A}$ begins by selecting its challenge parameters $(t^\star, \omega^\star, \mathbb{S}^\star)$ where $\mathbb{S}^\star$ is represented by a LSSS $(M^\star, \rho^\star)$. Let the matrix $M^\star$ be of size $l_s^\star \times k_s^\star$, where $m + k_s^\star \leq q$ and let $l_s^\star = l_{s,\mathbf{max}}$ and $|\omega^\star| = n$.
2. $\mathcal{B}$ now simulates running Setup for the rkDPABE scheme, and embeds the challenge policy into the public parameters. It first chooses $\gamma' \xleftarrow{\$} \mathbb{Z}_p$, sets $g^\alpha = g_1 = g^a$, and implicitly defines $\gamma = \gamma' + a^{q+1}$ by defining

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

It then must define the polynomials F_s, F_o and P (as in [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):

$$p_j(x) = \begin{cases} M^\star_{i,j} & \text{for } j \in [1, k_s^\star] \\ 0 & \text{for } j \in [k_s^\star + 1, k_s^\star + m] \end{cases} \tag{2}$$

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

$$p(x) = \sum_{j=0}^{k_s^\star + m} p_j(x) a^j. \tag{3}$$

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

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

The first equality in the above expression follows from equation (1) whilst the second follows by the above definition of h_i. The third equality is obtained by definition of $g_j = g^{a^j}$ and the last one follows by equation (3).
To define F_o, $\mathcal{B}$ randomly picks a polynomial $f'(x) = \sum_{j=0}^{n-1} f'_j x^j$ in $\mathbb{Z}_p[x]$ of degree $n-1$. It then defines $f(x) = \prod_{k \in \omega^\star}(x - k) = \sum_{j=0}^{n-1} f_j x^j$ (which can be computed entirely from $\omega^\star$); note that

$f(x) = 0$ if and only if $x \in \omega^\star$. It defines $q_j = g_q^{f_j} g^{f'_j}$ for $j = [0, n-1]$. Using the above we can finally compute

$$\begin{aligned} F_o(x) &= \prod_{j=0}^{n-1} q_j^{(x^j)} \\ &= \left(\prod_{j=0}^{n-1} g_q^{f_j} \cdot g^{f'_j} \right)^{x^j} \\ &= g_q^{\sum_{j=0}^{n-1} f_j x^j} \cdot g^{\sum_{j=0}^{n-1} f'_j x^j} \\ &= g_q^{f(x)} g^{f'(x)}. \end{aligned}$$

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

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

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

$$\begin{aligned} P(y) &= \prod_{j=0}^{d} u_j^{y^j} \\ &= \left(\prod_{j=0}^{d} (g^a)^{\hat{p}_j} g^{\rho_j} \right)^{y^j} \\ &= (g^a)^{\sum_{j=0}^{d} \hat{p}_j y^j} \; g^{\sum_{j=0}^{d} \rho_j y^j} \\ &= (g^a)^{\hat{p}(y)} g^{\rho(y)}. \end{aligned} \tag{4}$$

The public key pk for the rkDP-ABE scheme is defined to be

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

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

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

$$a_x = \begin{cases} a'_x - \alpha r_x - \gamma & \text{if } x \in \mathcal{X}_{\overline{R}} \\ a'_x - \frac{\alpha r_x - \gamma}{t^\star} & \text{if } x \notin \mathcal{X}_{\overline{R}} \end{cases} \tag{5}$$

Hence,

$$f_x(1) = a_x + \alpha r_x + \gamma = a'_x - \alpha r_x - \gamma + \alpha r_x + \gamma = a'_x \qquad \text{if } x \in \mathcal{X}_{\overline{R}} \tag{6}$$

$$f_x(t^\star) = a_x t^\star + \alpha r_x + \gamma = (a'_x - \frac{\alpha r_x - \gamma}{t^\star}) t^\star + \alpha r_x + \gamma = a'_x t^\star \qquad \text{if } x \notin \mathcal{X}_{\overline{R}} \tag{7}$$

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

– $(\omega^\star \in \mathbb{O})$ and $(\mathsf{id} \in \overline{R})$:
For each $x \in \mathsf{Path}(\mathsf{id})$, note that since $\mathsf{id} \in \overline{R}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (6), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathsf{id})$. $\mathcal{B}$ can therefore compute the key components precisely as in the construction by sharing the value of $f_x(1)$.

– $(\omega^\star \notin \mathbb{O})$ and $(\mathsf{id} \in \overline{R})$:

For each $x \in \mathsf{Path}(\mathsf{id})$, note that, since $\mathsf{id} \in \overline{R}$, $x \in \mathcal{X}_{\overline{R}}$. Hence, from (6), $\mathcal{B}$ can compute $f_x(1)$ for all $x \in \mathsf{Path}(\mathsf{id})$.

$\mathcal{B}$ randomly chooses $r_x \in \mathbb{Z}_p$. It then lets $D_x = g^{r_x}$, and for all $k \in \psi$ lets $D_k^{(3)} = F_s(k)^{r_x}$ as in the construction. Recall that the dimensions of N are $l_o \times k_o$. Since $\omega^\star$ does not satisfy N for this case of the query, and by Proposition 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 (2)) which will be used for creating the share of $f_x(1) = \gamma + \alpha r_x + a_x$ (note that the first element of $\boldsymbol{v_x}$ is indeed $f_x(1)$ by (6)), as in our construction.

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

$$\begin{aligned} D_{x,i}^{(2)} &= g^{\boldsymbol{N_i} \cdot \boldsymbol{v'_x}} F_o(\pi(i))^{r_{x,i}} \\ &= g^{\boldsymbol{N_i} \cdot (\boldsymbol{v_x} + (a'_x)\boldsymbol{a_x})} F_o(\pi(i))^{r_{x,i}} \\ &= g^{\boldsymbol{N_i} \cdot \boldsymbol{v_x}} F_o(\pi(i))^{r_{x,i}}, \end{aligned}$$

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

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

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

Note that, unlike [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 $(\mathsf{id} \notin \overline{R})$:

For each $x \in \mathsf{Path}(\mathsf{id})$, $\mathcal{B}$ does the following. Since ψ does not satisfy $M^\star$, by Proposition 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 $\boldsymbol{M_i} \cdot \boldsymbol{w_x} = 0$ for all i where $\rho(i) \in \psi^\star$. Now, by our definition of $p_j(x)$ in (2), we have that $(p_1(x), \ldots, p_{k_s^\star}(x)) \cdot (w_1, \ldots, w_{k_s^\star}) = 0$.

$\mathcal{B}$ then computes one possible solution of variables $w_{k_s^\star+1}, \ldots, w_{k_s^\star+m}$ for the system of $|\psi|$ equations: for all $x \in \psi$

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

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

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

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

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

$$\begin{aligned}
f_x(1) &= \gamma + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a_x \\
&= \gamma' + \alpha^{q+1} + \alpha r_x + a'_x - \frac{\alpha r_x - \gamma}{t^\star} \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + (\alpha(\frac{t^\star - 1}{t^\star}))r_x \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + \left(\alpha\left(\frac{t^\star-1}{t^\star}\right)\left(r'_x + w_1\left(\frac{t^\star}{t^\star-1}\right)\cdot\alpha^q\right.\right. \\
&\quad \left.\left. + w_2\left(\frac{t^\star}{t^\star-1}\right)\cdot\alpha^{q-1} + \cdots + w_{k^\star_s+m}\left(\frac{t^\star}{t^\star-1}\right)\cdot\alpha^{q-(k^\star_s+m)+1}\right)\right) \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha^{q+1} + \alpha\left(\frac{t^\star-1}{t^\star}\right)r'_x + w_1\alpha^{q+1} + w_2\alpha^q \\
&\quad + \cdots + w_{k^\star_s+m}\alpha^{q-(k^\star_s+m)+2} \\
&= \gamma' + a'_x + \frac{\gamma}{t^\star} + \alpha\left(\frac{t^\star-1}{t^\star}\right)r'_x w_2\alpha^q + \cdots + w_{k^\star_s+m}\alpha^{q-(k^\star_s+m)+2},
\end{aligned}$$

where the α^{q+1} term in γ has cancelled out using Proposition 1 and the third equality followed from using equation (5). The simulator now randomly chooses $z_{x,2}, \ldots, z_{x,k_o} \in \mathbb{Z}_p$ and implicitly lets the vector $\boldsymbol{v_x} = (\gamma + \alpha r_x + a_x, z_{x,2}, \ldots, z_{x,k_o})$ as in the construction.
$\mathcal{B}$ also randomly chooses $r_{x,1}, \ldots, r_{x,l_o} \in \mathbb{Z}_p$ and computes for $i = 1, \ldots, l_o$ the key $D^{(1)}_{x,1} = g^{r_{x,i}}$. The other keys are computed in the following way. We have

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

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 $(p_1(k), \ldots, p_{k^\star_s+m}(k)) \cdot (w_1, \ldots, w_{k^\star_s+m}) = 0$ for all $k \in \psi$.

$$\begin{aligned}
D^{(3)}_k &= D_x^{p_0(k)} \prod_{j=1}^{k^\star_s+m} \left(g_j^{r'_x} \prod_{k\in[1,k^\star_s+m],k\neq j} (g_{q+1-k+j})^{w_k}\right)^{p_j(k)} \\
&= (g^{r_x})^{p_0(k)} \prod_{j=1}^{k^\star_s+m} (g^{r_x})^{\alpha^j p_j(k)} \\
&= \prod_{j=0}^{k^\star_s+m} (g^{r_x})^{p_j(k)\alpha^j} = (g^{r_x})^{\sum_{j=0}^{k^\star_s+m} p_j(k)\alpha^j} \\
&= (g^{r_x})^{p(k)} = F_s(k)^{r_x},
\end{aligned}$$

where the second equality holds by observing that

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

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

– $(\omega^\star \notin \mathbb{O})$ and $(\psi \in \mathbb{S}^\star)$ and $(\mathsf{id} \notin \overline{R})$:
For each $x \in \mathsf{Path}(\mathsf{id})$, $\mathcal{B}$ randomly chooses $r_x \in \mathbb{Z}_p$. It then lets $D_x = g^{r_x}$, and for all $k \in \psi$ lets $D^{(3)}_k = F_s(k)^{r_x}$ 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 (5)), as in our construction.
Now, for all i such that $\pi(i) \in \omega^\star$, $\mathcal{B}$ randomly chooses $r_{x,i} \in \mathbb{Z}_p$ and computes $D^{(1)}_{x,i} = g^{r_{x,i}}$ 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

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

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

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

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

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

where the second last equality follows from equation (4).
$\mathcal{B}$ also defines

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

Note that $f(\pi(i)) \neq 0$ since $\pi(i) \notin \omega^\star$, and so $D^{(1)}_{x,i}$ and $D^{(2)}_{x,i}$ are well defined.
To simulate KeyUpdate queries for time period t and revocation list R, we consider the following cases:

– $t = t^\star$ and $\overline{R} \subseteq R$:
For each $x \in \mathsf{Cover}(R)$, $\mathcal{B}$ chooses a random $r_x \in \mathbb{Z}_p$ and computes $U_x^{(1)} = (g^{a'_x t^\star})P(t^\star)^{r_x}$ and $U_x^{(2)} = g^{r_x}$. Both keys are valid since $\overline{R} \subseteq R$ and thus for all $x \in \mathsf{Cover}(R)$ we have $x \notin \mathcal{X}_{\overline{R}}$. Hence, by (7), $f_x(t^\star) = a'_x t^\star$.
– $t \neq t^\star$:
For each $x \in \mathsf{Cover}(R)$, $\mathcal{B}$ chooses a random $r'_x \in \mathbb{Z}_p$
- If $x \in \mathsf{Cover}(R) \cap \mathcal{X}_{\overline{R}}$, it defines

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

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

Note that $\hat{p}(t) \neq 0$ for $t \neq t^\star$ so this is well defined. We claim that these keys look valid according to the construction with implicit randomness $r_x = r'_x - \frac{a^q(1-t)}{\hat{p}(t)+1-t}$.
Note that, in this case, $x \in \mathcal{X}_{\overline{\mathcal{R}}}$ and hence by (5)

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

Then,

$$\begin{aligned} U_x^{(1)} &\overset{(1)}{=} g^{f_x(t)} P(t)^{r_x} \\ &\overset{(2)}{=} g^{a'_x t + a r_x (1-t) + \gamma'(1-t) + a^{q+1}(1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a r_x (1-t)} g^{a\hat{p}(t) r_x} g^{\rho(t) r_x} \\ &\overset{(3)}{=} g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{a\hat{p}(t) r'_x} \\ &\qquad g^{-a\hat{p}(t)B} g^{\rho(t) r'_x} g^{-\rho(t)B} \\ &\overset{(4)}{=} g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-a(1-t)B} g^{-a\hat{p}(t)B} g^{-\rho(t)B} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)B} g^{-Ba((1-t)+a\hat{p}(t))} \\ &\overset{(5)}{=} g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} \\ &\qquad g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} (g^a)^{-\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)((1-t)+\hat{p}(t))} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} (g^a)^{-(a^q(1-t))} \\ &= g^{a'_x t} g^{\gamma'(1-t)} g^{a^{q+1}(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} g^{-a^{q+1}(1-t)} \\ &= g^{a'_x t} g^{\gamma'(1-t)} P(t)^{r'_x} g^{a(1-t) r'_x} g^{-\rho(t)\left(\frac{a^q(1-t)}{\hat{p}(t)+1-t}\right)} \\ &= (g^{a'_x})^t (g^{\gamma'})^{(1-t)} (g_1^{r'_x})^{(1-t)} g_q^{-\frac{\rho(t)(1-t)}{\hat{p}(t)+1-t}} P(t)^{r'_x}. \end{aligned}$$

Note that equality (1) follows by construction and (2) uses $f_x(t)$ from above and equation (4). Equality (3) follows by replacing r_x with $r'_x - B$ and equality (4) follows from using (4). Equality (5) is valid by using $B = \frac{a^q(1-t)}{\hat{p}(t)+1-t}$.
Then,

$$\begin{aligned} U_x^{(2)} &= g^{r_x} = g^{r'_x} g^{-\frac{a^q(1-t)}{\hat{p}(t)+1-t}} \\ &= (g^{r'_x})(g_q)^{-\frac{1-t}{\hat{p}(t)+1-t}}. \end{aligned}$$

Hence, these keys look valid according to the construction.
- If $x \in \mathsf{Cover}(R) \setminus \mathcal{X}_{\overline{R}}$, it defines

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

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

In this case, by (5), $a_x = a'_x - \frac{\alpha r_x - \gamma}{t^\star}$. 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 $\hat{p}(t^\star) = 0$, we have $C^{(4)} = (g^s)^{\rho(t^\star)} = ((g^a)^{\hat{p}(t^\star)} g^{\rho(t^\star)})^s = P(t^\star)^s$.

5. The challenge ciphertext is given to $\mathcal{A}$ along with oracle access which is handled as in Step 3.
6. $\mathcal{A}$ eventually outputs $b' \in \{0,1\}$ as its guess of b. If $b = b'$ then $\mathcal{B}$ outputs 1 to guess that $Z = e(g_{q+1}, h)$. Otherwise, $\mathcal{B}$ outputs 0 to guess that Z is random.

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