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) [20] 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

[⋆] The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-29485-8_9.

[⋆⋆] Partial funding by the European Commission under project H2020-644024 "CLARUS", and support from BAE Systems Advanced Technology Centre is gratefully acknowledged.

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 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 [3, 20] 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 [3], we introduce a new cryptographic primitive called Revocable-Key Dual-policy Attribute-based Encryption (rkDPABE) in Section 4.2, and finally, in Section 4.3, we instantiate HPVC using rkDPABE. Additional details, formal security games and proofs can be found in the full version online [2].

2 Background and Related Work

Verifiable computation [10,12,13,15,16,20,24] may be seen as a protocol between a (weak) client C and a server S, resulting in the provably correct computation of $F(x)$ by the server for the client's choice of F and x. The setup stage may be computationally expensive (amortised over multiple computations) but other operations should be efficient for the client. Some prior work used garbled circuits with fully homomorphic encryption [13,16] or targeted specific functions [10,12,15]. Chung et al. [14] introduced *memory delegation* which is similar to VDC; a client uploads his memory to a server who can update and compute a function F over the entire memory. Backes et al. [8] consider a client that outsources data and requests computations on a data portion. The client can efficiently verify the correctness of the result without holding the input data. Most work requires the client to know the data in order to verify [9, 11, 17, 19]. *Verifiable oblivious storage* [5] 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 [22] using public logs. It is notable that [7] and [11] achieve public verifiability. In independent and concurrent work, Shi et al. [21] use DP-ABE to combine keyword search on encrypted data with the enforcement of an access control policy.

Parno et al. [20] 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. [3] 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 [3, 20] to outsource a Boolean function, F, are based on Key-policy Attribute-based encryption (KP-ABE), which links ciphertexts with attribute sets and decryption keys with a policy; decryption only succeeds if the attributes satisfy the policy. For PVC, two random messages are encrypted and linked to the input data X (represented as attributes) to form the encoded input.

The evaluation key is a pair of decryption keys linked to F and $\overline{F}$ (the complement function of F). Exactly *one* message can be recovered, implying whether F or $\overline{F}$ was satisfied, and hence if $F(X) = 1$ or 0. Ciphertext indistinguishability ensures S cannot return the other message to imply an incorrect result.

3 Hybrid Publicly Verifiable Computation

To accommodate different modes of computation, we define HPVC generically in terms of parameters ω, $\mathbb{O}$, ψ and $\mathbb{S}$. Depending on the mode (and which party provides the input data), $\mathbb{O}$ or $\mathbb{S}$ will encode functions, while ω or ψ encode input data, as detailed in Section 4.1. We retain the single, trusted key distribution centre (KDC) from RPVC [3] who initialises the system for a function family $\mathcal{F}$ resulting in a set of public parameters PP and a master secret key. For each function $F \in \mathcal{F}$, the KDC publishes a delegation key PK_F. It also registers each entity S_i that wants to act as a server by issuing a signing key SK_{S_i}. It may also update PP during any algorithm to reflect changes in the user population.

Depending on the mode, servers either compute functions $\mathbb{O}$ on behalf of clients, or make a dataset ψ available for public querying. The $\mathsf{Certify}$ 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 $\sigma_{F,X}$ and a public verification key $VK_{F,X}$.

A server combines $\sigma_{F,X}$ with its evaluation key to compute $\theta_{F(X)}$ encoding the result $F(X)$. Any entity can verify the correctness of $\theta_{F(X)}$ using $VK_{F,X}$. Verification outputs the result $y = F(X)$ of the computation (if correct) and generates a token $\tau_{F(X)}$ 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.

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

[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.

1. $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^{\ell}, \mathcal{F})$: run by the KDC to establish public parameters PP and a master secret key MK for the system. The inputs are the security parameter ℓ, and the family of functions $\mathcal{F}$ that may be computed;
2. $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate a public delegation key, PK_F, allowing entities to outsource, or request, computations of F;
3. $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to enrol an entity S_i within the system to act as a server. It generates a personalised signing key SK_{S_i};
4. $EK_{(\mathbb{O},\psi),S_i} \xleftarrow{\$} \mathsf{Certify}(\mathtt{mode}, S_i, (\mathbb{O}, \psi), L_i, \mathcal{F}_i, \mathrm{MK}, \mathrm{PP})$: run by the KDC to generate an evaluation key $EK_{(\mathbb{O},\psi),S_i}$ enabling the server S_i to compute on the pair $(\mathbb{O}, \psi)$. The algorithm also takes as input the mode in which it should operate, a set of labels L_i and a set of functions $\mathcal{F}_i$;
5. $(\sigma_{F,X}, VK_{F,X}) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega, \mathbb{S}), L_{F,X}, PK_F, \mathrm{PP})$: run by an entity to request a computation of $F(X)$ from S_i. The inputs are the mode, the pair $(\omega, \mathbb{S})$ representing the computation, a set of labels $L_{F,X} \subseteq L_i$, the delegation key for F and the public parameters. The outputs are an encoded input $\sigma_{F,X}$ and a verification key $VK_{F,X}$;
6. $\theta_{F(X)} \xleftarrow{\$} \mathsf{Compute}(\mathtt{mode}, \sigma_{F,X}, EK_{(\mathbb{O},\psi),S_i}, SK_{S_i}, \mathrm{PP})$: run by an entity S_i to compute $F(X)$. The inputs are the mode, an encoded input, and an evaluation key and signing key for S_i. The output, $\theta_{F(X)}$, encodes the result;
7. $(y, \tau_{F(X)}) \leftarrow \mathsf{Verify}(\theta_{F(X)}, VK_{F,X}, \mathrm{PP})$: run by any entity. The inputs are an encoded output produced by S_i and verification key; the outputs are the computation result $y = F(X)$ if the result was computed correctly, or else $y = \perp$, and a token $\tau_{F(X)}$ which is (accept, S_i) if $\theta_{F(X)}$ is correct, or (reject, S_i) otherwise;
8. $UM \xleftarrow{\$} \mathsf{Revoke}(\tau_{F(X)}, \mathrm{MK}, \mathrm{PP})$: run by the KDC if a misbehaving server is reported. It returns $UM = \perp$ if $\tau_{F(X)} = (\mathsf{accept}, S_i)$. Otherwise, all evaluation keys $EK_{(\cdot,\cdot),S_i}$ for S_i are rendered non-functional and the update material UM is a set of updated evaluation keys $\{EK_{(\mathbb{O},\psi),S'}\}$ for all servers.

3.1 Security Models

We now discuss desirable security properties for HPVC; additional formal models are found in the full paper [2][2]. Public verifiability, revocation and authorised computation are selective notions in line with our rkDPABE scheme introduced in Section 4.2.

Public Verifiability, presented in Game 1, ensures that a server that returns an incorrect result is detected by the verification algorithm so that they can be reported for revocation. The adversary, $\mathcal{A}$, may corrupt other servers, generate

[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 [3].

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

```
 1: $(\omega^\star, \mathbb{O}^\star, \psi^\star, \mathbb{S}^\star, L_{F,X^\star}, \mathtt{mode}) \xleftarrow{\$} \mathcal{A}(1^\ell, \mathcal{F})$
 2: $(\mathrm{PP}, \mathrm{MK}) \xleftarrow{\$} \mathsf{Setup}(1^\ell, \mathcal{F})$
 3: if $(\mathtt{mode} = VDC)$ then $(F \leftarrow \mathbb{S}^\star, X^\star \leftarrow \psi^\star)$
 4: else $(F \leftarrow \mathbb{O}^\star, X^\star \leftarrow \omega^\star)$
 5: $PK_F \xleftarrow{\$} \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$
 6: $(\sigma^\star, VK^\star) \xleftarrow{\$} \mathsf{ProbGen}(\mathtt{mode}, (\omega^\star, \mathbb{S}^\star), L_{F,X^\star}, PK_F, \mathrm{PP})$
 7: $\theta^\star \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma^\star, VK^\star, PK_F, \mathrm{PP})$
 8: $(y, \tau_{\theta^\star}) \leftarrow \mathsf{Verify}(\theta^\star, VK^\star, \mathrm{PP})$
 9: if $(((y, \tau_{\theta^\star}) \neq (\perp, (\mathsf{reject}, \cdot)))$ and $(y \neq F(X^\star)))$ then
10:     return 1
11: else return 0
```

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

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

Definition 2. *The* advantage *of a probabilistic polynomial time adversary $\mathcal{A}$ in the* sPubVerif *game for an HPVC construction, $\mathcal{HPVC}$, for a family of functions $\mathcal{F}$ is defined as:*

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

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

– **Revocation** ensures that a server that has been detected as misbehaving cannot produce a result (even a correct result) that is accepted by a verifier – thus, the server cannot be rewarded for future work. To reflect the revocation mechanism of the rkDPABE primitive, we include a semi-static restriction whereby a list of entities to be revoked at the time of the challenge computation must be declared before the adversary receives oracle access[3].

– **Authorised Computation** extends the model of [1] to the public-key setting to ensure that an unauthorised server cannot produce acceptable results.

[3] This restriction was also used in [6] for revocable KP-ABE, and could be removed if an adaptive, indirectly revocable ABE scheme is found.

Table 1: Parameter definitions for different modes

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

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

4 Instantiating HPVC

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

Following [20], 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.

4.1 Supporting Different Modes

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

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 [20] 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, 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. $P = (\mathtt{Country} = \mathtt{UK}) \vee ((\mathtt{clearance} = \mathtt{Secret}) \wedge (\mathtt{Country} = \mathtt{USA}))$ is satisfied by $s = \{\mathtt{Country} = \mathtt{UK}, \mathtt{Capacity} = \mathtt{3TB}\}$.

Table 2: Example database

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

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

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

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

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.

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

As an example, consider a server S_i that owns the database in Table 2. The dataset D_i represents this as a set of field values for each record in turn: $D_i = \{001, \text{Alice}, 26{,}165, 002, \text{Bob}, 22, 172\}$. S_i publishes data labels $L_i = \{$User ID of record 1, Name of record 1, Age of record 1, Height of record 1, User ID

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

of record 2, Name of record 2, Age of record 2, Height of record 2}. In Table 3, $\mathcal{F}_i$ 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$\}$.

4.2 Revocable Dual-policy Attribute-based Encryption

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

Definition 3. A *Revocable Key Dual-policy Attribute-based Encryption scheme* (rkDPABE) comprises five algorithms:

- $(\mathrm{PP}, \mathrm{MK}) \overset{\$}{\leftarrow} \mathsf{Setup}(1^\ell, \mathcal{U})$: takes the security parameter and attribute universe and generates public parameters PP and a master secret key MK;
- $CT_{(\omega,\mathbb{S}),t} \overset{\$}{\leftarrow} \mathsf{Encrypt}(m, (\omega, \mathbb{S}), t, \mathrm{PP})$: takes as input a message to be encrypted, an objective attribute set ω, a subjective policy $\mathbb{S}$, a time period t and the public parameters. It outputs a ciphertext that is valid for time t;
- $SK_{(\mathbb{O},\psi),\mathrm{ID}} \overset{\$}{\leftarrow} \mathsf{KeyGen}(\mathrm{ID}, (\mathbb{O}, \psi), \mathrm{MK}, \mathrm{PP})$: takes an identity ID, an objective access structure $\mathbb{O}$, a subjective attribute set ψ, the master secret key and the public parameters. It outputs a secret decryption key $SK_{(\mathbb{O},\psi),\mathrm{ID}}$;
- $UK_{R,t} \overset{\$}{\leftarrow} \mathsf{KeyUpdate}(R, t, \mathrm{MK}, \mathrm{PP})$: takes a revocation list R containing all revoked identities, the current time period, the master secret key and public parameters. It outputs updated key material $UK_{R,t}$ which makes the decryption keys $SK_{(\mathbb{O},\psi),\mathrm{ID}}$, for all non-revoked identities ID $\notin$ R, functional to decrypt ciphertexts encrypted for the time t.
- $PT \leftarrow \mathsf{Decrypt}(CT_{(\omega,\mathbb{S}),t}, (\omega, \mathbb{S}), SK_{(\mathbb{O},\psi),\mathrm{ID}}, (\mathbb{O}, \psi), UK_{R,t}, \mathrm{PP})$: takes as input a ciphertext formed for the time period t and the associated pair $(\omega,\mathbb{S})$, a decryption key for entity ID and the associated pair $(\mathbb{O}, \psi)$, an update key for the time t and the public parameters. It outputs a plaintext PT which is the encrypted message m, if and only if the objective attributes ω satisfies the objective access structure $\mathbb{O}$ *and* the subjective attributes ψ satisfies the subjective policy $\mathbb{S}$ *and* the value of t in the update key matches that specified during encryption. If not, PT is set to be a failure symbol $\perp$.

Definition 3 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 full, online version of the paper [2].

4.3 Construction

As mentioned, we base our construction on rkDPABE by encoding inputs as attributes in a universe $\mathcal{U}_x$, and encoding Boolean functions as access structures over $\mathcal{U}_x$. Computations with n-bit outputs can be built from n Boolean functions returning each bit in turn. Negations can be handled by building rkDPABE from non-monotonic ABE [18] or, as here, by adding negated attributes to the universe [23]. For the i^{th} bit of a binary input string $X = x_1 \ldots x_n$, define attributes $A^0_{X,i}$ and $A^1_{X,i} \in \mathcal{U}_x$[5]; X is encoded as $A_X = \{A^j_{X,i} \in \mathcal{U}_x : x_i = j\}$.

Let $\mathcal{U}_l$ be a set of attributes (disjoint from $\mathcal{U}_x$) uniquely labelling each function and data item, and let $\mathcal{U}_{\text{ID}}$ represent server identities. Let g be a one-way function and $\mathcal{DPABE}$ be a revocable key DP-ABE scheme for $\mathcal{F}$ with attribute universe $\mathcal{U} = \mathcal{U}_x \cup \mathcal{U}_l \cup \mathcal{U}_{\text{ID}}$. We initialise two independent DP-ABE systems over $\mathcal{U}$, and define four additional "dummy" attributes to disable modes: T^0_O, T^0_S for the first system, and T^1_O, T^1_S for the second. We denote the complement functions as follows: in RPVC and RPVC-AC, recall $\mathbb{O} = F$ and $\mathbb{S} = \{\{T^0_S\}\}$; we define $\overline{\mathbb{O}} = \overline{F}$ and $\overline{\mathbb{S}} = \{\{T^1_S\}\}$. Similarly, for VDC, $\overline{\mathbb{O}} = \{\{T^1_0\}\}$ and $\overline{\mathbb{S}} = \overline{F}$.

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

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

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

2. FnInit 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).

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

```
$PK_F \leftarrow \text{PP}$
```

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

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

```
$(SK_{\text{Sig}}, VK_{\text{Sig}}) \xleftarrow{\$}$ Sig.KeyGen$(1^\ell)$
$SK_{S_i} \leftarrow SK_{\text{Sig}}$
$L_{\text{Reg}}[S_i][0] \leftarrow L_{\text{Reg}}[S_i][0] \cup VK_{\text{Sig}}$
```

4. Certify first adds an element $(F, \bigcup_{l \in L_i} l)$ to the list $L_{Reg}[S_i][1]$ for each $F \in \mathcal{F}_i$; this publicises the computations that S_i can perform (either functions

[5] Either by defining a large enough $\mathcal{U}_x$ or by hashing strings to elements of the attribute group. Unlike prior schemes [3, 20], we include an identifier of the data X (based on the label $l(x_{i,j})$) in the attribute mapping to specify the data items to be used; alternatively, D_i could be a long bitstring formed by concatenating each data point, and the labels should identify the attributes corresponding to each data point.

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

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

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

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

5. ProbGen chooses messages m_0 and m_1 randomly from the message space. m_0 is encrypted with $(A_\omega, \mathbb{S} \wedge \bigwedge_{l \in L_{F,X}} l)$ in the first DPABE system, whilst m_1 is encrypted with the complement policy and either the first DPABE system for VDC or the second for RPVC (the attributes remain the same as it is the same attribute T^0_O or input data X respectively). The verification key comprises g applied to each message; the one-wayness of g allows the key to be published.

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

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

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

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

```
Parse $EK_{(\mathbb{O},\psi),S_i}$ as $(SK^0_{\mathrm{ABE}}, SK^1_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t}, UK^1_{L_{\mathrm{Rev}},t})$ and $\sigma_{F,X}$ as $(c_0, c_1)$
$d_0 \leftarrow \mathsf{DPABE.Decrypt}(c_0, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
if $\mathtt{mode} = VDC$ then $d_1 \leftarrow \mathsf{DPABE.Decrypt}(c_1, SK^0_{\mathrm{ABE}}, MPK^0_{\mathrm{ABE}}, UK^0_{L_{\mathrm{Rev}},t})$
else $d_1 \leftarrow \mathsf{DPABE.Decrypt}(c_1, SK^1_{\mathrm{ABE}}, MPK^1_{\mathrm{ABE}}, UK^1_{L_{\mathrm{Rev}},t})$
$\gamma \xleftarrow{\$} \mathsf{Sig.Sign}((d_0, d_1, S_i), SK_{S_i})$
$\theta_{(\omega,\mathbb{S}),(\mathbb{O},\psi)} \leftarrow (d_0, d_1, S_i, \gamma)$
```

7. Verify verifies the signature using the verification key for S_i stored in L_{Reg}. If correct, it applies g to each plaintext in $\theta_{F(X)}$ and compares the results to the components of the verification key. If either comparison results in a match (i.e. the server successfully recovered a message), the output token is accept. Otherwise the result is rejected. If m_0 was returned then $F(X) = 1$ as m_0 was encrypted for the non-complemented inputs; if m_1 was returned then $F(X) = 0$.

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

```
Parse $VK_{F,X}$ as $(g(m_0), g(m_1), L_{\mathrm{Reg}})$ and $\theta_{F(X)}$ as $(d_0, d_1, S_i, \gamma)$
if $\mathsf{accept} \leftarrow \mathsf{Sig.Verify}((d_0, d_1, S_i), \gamma, L_{\mathrm{Reg}}[S_i][0])$ then
    if $g(m_0) = g(d_0)$ then return $(y \leftarrow 1, \tau_{F(X)} \leftarrow (\mathsf{accept}, S_i))$
    else if $g(m_1) = g(d_1)$ then return $(y \leftarrow 0, \tau_{F(X)} \leftarrow (\mathsf{accept}, S_i))$
    else return $(y \leftarrow \perp, \tau_{F(X)} \leftarrow (\mathsf{reject}, S_i))$
return $(y \leftarrow \perp, \tau_{F(X)} \leftarrow (\mathsf{reject}, \perp))$
```

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

Algorithm 8 $UM \xleftarrow{\$} \mathsf{HPVC.Revoke}(\tau_{F(X)}, \mathrm{MK}, \mathrm{PP})$

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

Theorem 1. *Given an* IND-sHRSS *secure rkDPABE scheme, a one-way function g, and an* EUF-CMA *signature scheme, this construction is secure in the sense of selective public verifiability, and selective semi-static revocation and selective authorised computation.*

Full proofs of security can be found in the full, online version of the paper [2]. Informally, to prove selective public verifiability, we show that we can replace the message encrypted under the *non-satisfied* function evaluation (i.e. the computation that evaluates to $F(x) \oplus 1$) with a randomly chosen message; due to the IND-CPA-style security of the rkDPABE scheme (implied by the IND-sHRSS property), an adversary cannot learn anything about a message for which the decryption policy is not satisfied. In particular, we can (implicitly) replace the message with the challenge message in an inversion game for the one-way function g and then the verification token for this message is the challenge input in that game. We therefore show that breaking the public verifiability of our construction (i.e. returning the message for the wrong computational result) will allow an adversary to invert the one-way function g.

5 Conclusion

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

Our instantiation, built from a novel use of DP-ABE, captures prior models of PVC [3,20], 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 [4].

ABE was developed to enforce read-only access control policies, and the use of KP-ABE in PVC was a novel and surprising result [20]. 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, 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.

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. Hybrid publicly verifiable computation. Cryptology ePrint Archive, Report 2015/320, 2015. `http://eprint.iacr.org/`.
3. 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.
4. J. Alderman, C. Janson, K. M. Martin, and S. L. Renwick. Extended functionality in verifiable searchable encryption. *IACR Cryptology ePrint Archive*, 2015:975, 2015.
5. 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.
6. 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.
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 &#38; 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. 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.

16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.