Access Control in Publicly Verifiable Outsourced Computation

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

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

Abstract

Publicly Verifiable Outsourced Computation (PVC) allows devices with restricted resources to delegate expensive computations to more powerful external servers, and to verify the correctness of results. Whilst highlybeneficial in many situations, this increases the visibility and availability of potentially sensitive data, so we may wish to limit the sets of entities that can view input data and results. Additionally, it is highly unlikely that all users have identical and uncontrolled access to all functionality within an organization. Thus there is a need for access control mechanisms in PVC environments.

In this work, we define a new framework for Publicly Verifiable Outsourced Computation with Access Control (PVC-AC). We formally define algorithms to provide different PVC functionality for each entity within a large outsourced computation environment, and discuss the forms of access control policies that are applicable, and necessary, in such environments, as well as formally modelling the resulting security properties. Finally, we give an example instantiation that (in a black-box and generic fashion) combines existing PVC schemes with symmetric Key Assignment Schemes to cryptographically enforce the policies of interest.

1 Introduction

The increasing use of mobile devices as general computing devices, "big data" and cloud computing results in a growing discrepancy between the resources of end-users and those required for computational tasks. As a result, client devices need to be able to efficiently delegate computations to a server and verify the correctness of results. Consider, for example, a company operating a "bring your own device" policy for smartphones and tablets that may be unable to perform complex computations locally. Instead, the company subscribes to an external (untrusted) company providing software-as-a-service (SaaS), but needs to ensure that results remain correct. Alternatively, in the context of battlefield communications, a squadron of soldiers may be deployed with light-weight devices to gather data from their surroundings and send it to regional servers for analysis before receiving tactical commands. Those servers may not be fully trusted, e.g. if part of a coalition network, so soldiers must be assured that the command has been computed correctly. This setting is known as *Verifiable Outsourced Computation* (VC) [16, 20, 10].

Recent work has proposed extensions to VC where there exist large pools of delegators, known as Publicly Verifiable Outsourced Computation (PVC) [20], and servers [2]. As with any multi-user setting, we may wish to control access to resources. In this paper, we show that not only is this setting of multi-user VC well-suited to the cryptographic enforcement of access control policies, but that such policies fulfil a natural and vital role in protecting outsourced computations. Specifically in the setting of multi-user VC, we may wish to (i) restrict the

computations that may be outsourced by delegators; and (ii) restrict the computations a server may perform. The first need stems from separation of duties and the observation that, within an organization, it is extremely unlikely that all users have equal, uncontrolled access to all functionality. We may restrict the set of delegators that may outsource a computation to those that are authorized internally to compute it (if given sufficient resources). The second requirement arises, for example, from servers being selected from the pool for a particular job without the delegator necessarily having prior knowledge. Thus delegators may not authenticate the server beforehand (in contrast with prior schemes where a single server was chosen with which to set up a VC system) and have less control over which servers may operate on their data. The sensitivity of the data or other requirements, such as the physical location or resources of the server, may limit the servers that should be permitted to perform the computation.

Some VC settings distinguish between delegators and verifiers [2]. Delegators (or distinguished verifiers) may learn the result of the computation, whereas standard verifiers may only confirm that the result was computed correctly. Again, in a multi-user VC setting, we may wish to restrict the users that: (i) verify the result; and (ii) learn the result. When operating on sensitive data, this second restriction ensures that read access to the generated results is limited to those that satisfy the access control policy e.g. only entities that may read the input data may read the output.

A third motivation for access control in the VC setting is that computational services may be charged for (e.g. in subscription-based utility computing [17, 21]) and that service providers may offer different levels of service to different clients (e.g. different levels may provide access to different functions or computational resources). We must ensure that only valid subscribers may access each tier of service.

In many multi-user settings for access control to stored data [7, 22], servers enforce access control policies by authenticating users and granting or denying access based on *access control lists* or *capability lists*. This reference monitor is not appropriate in the multi-user VC setting since the servers are assumed to be untrusted and may have a vested interest in violating the policies. We instead use a cryptographic enforcement mechanism for access control policies where cryptographic keys are used to protect objects and restrict access — the access control mechanism reduces to the appropriate distribution of keys to authorized entities. We use the trusted Key Distribution Center (KDC) introduced in the Revocable Publicly Verifiable Outsourced Computation model (RPVC) [2] and extend its duties to instantiate the access control mechanism. In RPVC, the KDC issued keys that enable the delegation and evaluation of particular functions; we additionally require it to issue keys appropriate to the access control policy that enables read and write access to certain components of the PVC system. For example, input data for particular functions may be protected such that only authorized servers may read the data and hence perform the computation.

Note that in an RPVC scheme, the KDC implicitly provides some access control in that servers are certified to perform specific functions through the generation of evaluation keys. However, no access control is applied to delegators – *any* entity can outsource an evaluation of *any* function for which the KDC has published delegation information (essentially due to the use of asymmetric cryptographic primitives). In particular, a delegator may request a computation that the delegator itself is not authorized to perform.

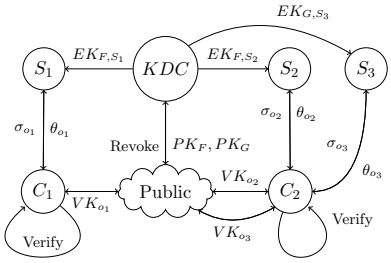
Cryptographic enforcement mechanisms are particularly appropriate when the objects and policies are relatively static (such that additional keys need not be generated and objects need not be re-encrypted). In the context of VC, we may assume that the set of functions that may be evaluated is fixed (a given VC construction can implement a specified family of functions) and that the input data to each function is also static (limited to the set of 'valid' inputs to that function). Thus, the set of objects (function evaluations in VC) is static, and policies will

primarily be specified in terms of these computations. Thus multi-user VC is a very natural setting in which to use cryptographic access control. However, VC leads to a somewhat novel application of these mechanisms as we will see in Section 3.

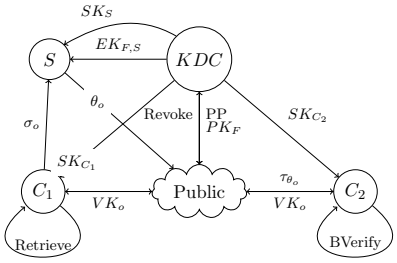
Related work Clear et al. [11] considered policies over delegators only and in a non-verifiable, multi-input outsourced computation setting using homomorphic Ciphertext-Policy ABE and fully homomorphic encryption. In independent and concurrent work, Xu et al.[23] also addressed the necessity for access control in the setting of verifiable computation, but limited their scope to non-public verifiable computation (i.e. not the full multi-user setting) enforcing access control on delegators only. We believe that the PVC setting also necessitates that delegators may specify access control requirements on those *servers* that may be selected for a given computation and, especially, limits must be placed on *verifiers* that may learn the output. Xu et al. discuss their notion purely in terms of using CP-ABE as the enforcement mechanism and did not discuss the form of the policies; in contrast, we discuss in detail the types of policies that may be of interest in these settings and present these in terms of generic graph-based access control policies. These may be enforced by a variety of enforcement mechanisms, including symmetric KASs as used here (which may well be more efficient that the pairing-based CP-ABE approach). We believe that we present a more generic treatment which, importantly, extends to the multi-user PVC setting and incorporates a more natural view of hierarchical access control, as well as additional security notions.

Structure of the paper We begin with some preliminary background material before, in Section 3, discussing example access control policies that are relevant to the setting of multi-user VC. We formulate policies in a generic fashion (irrespective of the cryptographic enforcement mechanism employed) in terms of authorization labels and graph-based policies over entities and computations. Then, in Section 4, we extend prior literature [16, 20, 2] to formally define a framework for PVC-AC. This results in novel security notions that we introduce in Section 5. Finally, in Section 6, we provide an example construction that provably meets the stated security goals. Our construction generically extends any RPVC scheme to use a symmetric Key Assignment Scheme (KAS). When instantiated with a PVC scheme built from Key-policy Attribute-based Encryption [2, 20], this provides a pragmatic blend of symmetric and asymmetric primitives – the symmetric KAS enables the efficient derivation of keys, each associated to a security label, while policies are specified in terms of these labels. A trusted authority on entities, the KDC, issues keys to each entity corresponding to the highest security label for which they are authorized, and they may derive "lower" keys as required. These keys may cryptographically protect input values and verification keys such that only entities satisfying the associated policies may operate on them. Unauthorised servers or eavesdroppers may not even learn the input values, thus exposure of data is limited to explicitly authorised (trusted) entities. Moreover, this restriction enforces access control policies on the part of the delegator device. To encrypt the inputs with a symmetric key associated with the policy, the delegator must be able to derive this key in the first place. Thus, the delegator's security clearance must be at least the classification of the function in order to derive this keyfrom that issued by the KDC.

We use the following notation.We write $y \leftarrow A(\cdot)$ to denote running a probabilistic algorithm A and assigning the result to y. We use PPT to denote probabilistic polynomial-time and say that $\mathrm{negl}(\cdot)$ is a negligible function of its input. $\mathcal{A}^{\mathcal{O}}$ is used to denote the adversary $\mathcal{A}$ being provided with oracle access.



(a) The operation of RPVC



(b) The operation of PVC-AC

Figure 1: The operation of RPVC and PVC-AC

2 Background

2.1 Revocable PVC

Non-interactive verifiable computation (VC), introduced by Gennaro et al. [16],is a protocol between two PPT parties: a *client* C and a *server* S. A successful protocol run results in the provably correct computation of $F(x)$ by the server for an input x given by the client. More specifically [16]:

1. C computes evaluation information EK_F given to S to enable the computation of F (pre-processing)
2. C sends the encoded input σ_o to S (input preparation)
3. S computes $y = F(x)$ using EK_F and σ_o and returns an encoded output θ_o (output computation)
4. C checks whether θ_o encodes $F(x)$ (verification)

KeyGen may be computationally expensive but the remaining operations should be efficient for the client. The cost of setup is amortized over multiple computations of F.

Parno et al. [20] introduced Publicly Verifiable Outsourced Computation (PVC) to allow multiple clients to delegate computations. Alderman et al. [2] extended PVC to consider a wider framework, Revocable PVC (RPVC). Here large pools of clients and servers exist and jobs may be submitted to the pool; an appropriate server is selected by a system-dependent mechanism. This model introduces a trusted entity known as a Key Distribution Center (KDC) that performs costly set-up operations and issues appropriate keys. RPVC comprises the algorithms Setup, FnInit, Register, Certify, ProbGen, Compute, BVerif, Retrieve and Revoke illustrated in Figure 1a and corresponding to the following:

1. The KDC generates public parameters, issues personalised secret keys, and evaluation keys to servers and publishes function delegation information.
2. To outsource the evaluation of $F(x)$, a delegator C, sends an encoded input σ_o to a server S, and publishes a verification token and output retrieval key for the computation.
3. S uses σ_o and an evaluation key for F to produce an encoded output (sent to C, a manager, or published depending on the system architecture).
4. Any entity can use the verification token to verify correctness without learning the value of $F(x)$ unless in possession of a retrieval key (blind verification). If S cheated they may report S to the KDC for revocation.

5. If blind verification was successful, a party possessing the output retrieval key can recover $F(x)$.
6. The KDC may revoke a cheating server to prevent it from performing computations (and hence from receiving any reward for future work).

2.2 Cryptographic Enforcement of Access Control Policies

2.2.1 Graph-based Access Control

A *partially ordered set* (*poset*) is a set L equipped with a reflexive, anti-symmetric and transitive binary relation $\leqslant$. We may write $x < y$ if $x \leqslant y$ and $x \neq y$, and write $y \geqslant x$ if $x \leqslant y$. We say that x *covers* y, written $y \lessdot x$, if $y < x$ and no z exists in L such that $y < z < x$. The *Hasse Diagram* of a poset $(L, \leqslant)$ is the directed acyclic graph $H = (L, \lessdot)$ where vertices are labelled by the elements of L and an edge connects vertex v to w if and only if $w \lessdot v$. Let U be a set of entities, O be a set of resources to be protected, and $(L, \leqslant)$ be a poset of security labels. Let $\lambda : U \cup O \to L$ be a labelling function assigning a security label to each entity and object. The tuple $(L, \leqslant, U, O, \lambda)$ then denotes an *information flow policy* which can be represented by the H. Henceforth we refer to such policies as *graph-based access control policies*. The policy requires that information flow from objects to entities preserves the partial ordering relation — an entity $u \in U$ may read an object $o \in O$ if and only if $\lambda(u) \geqslant \lambda(o)$ i.e. there exists a directed path from $\lambda(u)$ to $\lambda(o)$ in H. Note that this statement is the *simple security property* of the Bell-LaPadula security model [4]. Thus an entity assigned clearance label x is prevented from accessing objects classified with label y if $y > x$.

2.2.2 Key Assignment Schemes

A *Key Assignment Scheme* (KAS) [1] provides a generic, cryptographic enforcement mechanism for such policies where a unique cryptographic key is associated to each node (representing a security label) in H. A KAS eases the problem of key distribution by allowing a trusted center to distribute a single key to each entity, who may combine this key with public information to derive any additional keys for which the user is authorized. More formally, for an information flow policy $(L, \leqslant)$ [13]:

- $\mathsf{MakeKeys}(1^\kappa, (L, \leqslant)) \to \kappa_L$: returns a labelled set of encryption keys $\{\kappa_x : x \in L\}$, denoted κ_L.
- $\mathsf{MakeSecrets}(1^\kappa, (L, \leqslant)) \to \omega_L$: returns a labelled set of secret values $\{\omega_x : x \in L\}$, denoted ω_L.
- $\mathsf{MakePublicData}(1^\kappa, (L, \leqslant)) \to Pub_{(L,\leqslant)}$: returns a set of data $Pub_{(L,\leqslant)}$ published by the trusted authority
- $\mathsf{GetKey}(x, y, \omega_x, Pub_{(L,\leqslant)}) \to \kappa_y$: takes two nodes, $x, y \in L$, the secret information for x, ω_x, and the public information, $Pub_{(L,\leqslant)}$ and returns κ_y if $y \leqslant x$.

A well-known KAS construction (known as an *iterative key encrypting* (IKE) KAS [13]) publishes encrypted keys. In particular, for each $y < x$, such that no z exists with $y < z < x$, $\mathsf{Encrypt}_{\kappa_x}(\kappa_y)$ is published. Then for any $x > y$, the key for each node on a path from x to y can be derived (in an iterative fashion) if κ_x is known. *Key indistinguishability* (KI) [3] is a crucial security property if derived keys are used in other protocols: given a set of keys $\kappa_{x_1}, \ldots, \kappa_{x_n}$, an adversary should not be able to distinguish between the key for a challenge node $x^\star$ (not a descendent of any x_i) and a randomly chosen key. Recently, Freire et al. [15] suggested a new

Game 1 $\mathbf{Exp}_{\mathcal{A}}^{S\text{-}KI}[1^\kappa, (L, \leqslant)]$:

$b \xleftarrow{\$} \{0,1\}$
$\kappa_L \leftarrow \mathsf{MakeKeys}(1^\kappa, (L, \leqslant))$
$\omega_L \leftarrow \mathsf{MakeSecrets}(1^\kappa, (L, \leqslant))$
$Pub_{(L,\leqslant)} \leftarrow \mathsf{MakePublicData}(1^\kappa, (L, \leqslant))$
$Q = \epsilon$
$v^\star \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Corrupt}}(\cdot,\perp)}(Pub_{(L,\leqslant)})$
for all $v_i \in Q$ **do**
    if $v^\star \leqslant v_i$ **then return** 0
if $b = 0$ **then** $k^\star = k_{v^\star}$
else $k^\star \xleftarrow{\$} \mathcal{K}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Corrupt}}(\cdot,v^\star)}(k^\star)$
if $b' = b$ **then return** 1
else return 0

Oracle Query 1 $\mathcal{O}^{\mathsf{Corrupt}}(v_i, v^\star)$:

if $v^\star \not\leqslant v_i$ **then**
    $Q = Q \cup v_i$
    return $(\kappa_{v_i}, \omega_{v_i})$
else if $v_i > v^\star$ **then**
    return κ_{v_i}
else
    return $\perp$

KI security definition for hierarchial KASs called *Strong Key Indistinguishability* (S-KI) which strengthens the definition in [3] to capture a wider range of realistic attacks. S-KI requires that the adversary $\mathcal{A}$ is not able to gain any information about a key which it should not have access to *even if* $\mathcal{A}$ has access to keys associated to all other classes which are predecessors of the target class in the hierachy. In Game 1 we formally capture the notion of S-KI for an adaptive adversary.

Definition 1. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT) is defined as:*

$$Adv^{\text{S-KI}}_{\mathcal{A},\mathcal{KAS}}(1^\kappa, (L, \leqslant)) = \Pr[\text{S-KI}^{\mathcal{A}}_{\mathcal{KAS}}(1^\kappa, (L, \leqslant))] - \tfrac{1}{2}.$$

A KAS is secure in the sense of strong key indistinguishability against adaptive adversaries (S-KI) *if for all PPT adversaries $\mathcal{A}$, $Adv^{\text{S-KI}}_{\mathcal{A},\mathcal{KAS}}(1^\kappa, (L, \leqslant)) \leq \text{negl}(\kappa)$.*

2.3 Authenticated Encryption

In this paper, we use two forms of encryption scheme: a public-key Key-Policy Attribute-based Encryption (KP-ABE) [18, 19] scheme is used as the foundation of the Publicly Verifiable Outsourced Computation functionality, as introduced by Parno et al. [20] – decryption of a ciphertext succeeds only if a function is satisfied by an input; the second form is a symmetric authenticated encryption scheme to preserve confidentiality of messages from unauthorized readers and to achieve data origin authentication. We refer the reader to the cited literature for an introduction to KP-ABE as we use this only in a black-box manner as a component of an existing RPVC scheme. In this section, we give some brief background on authenticated encryption in the symmetric setting and introduce the security notions we shall require.

Game 2 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CPA}}[1^\kappa]$:

1: $b \xleftarrow{\$} \{0,1\}$
2: $SK^\star \leftarrow \mathsf{KeyGen}(1^\kappa)$
3: $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\text{LoR}}(\cdot,\cdot,SK^\star)}(1^\kappa)$
4: **return** $b' = b$

Oracle Query 2 $\mathcal{O}^{\text{LoR}}(m_0, m_1, SK^\star)$:

1: **return** $\mathsf{Encrypt}(m_b, SK^\star)$

Game 3 $\mathbf{Exp}_{\mathcal{A}}^{\text{IND-CCA}}[1^\kappa]$:

1: $b \xleftarrow{\$} \{0,1\}$
2: $L = \epsilon$
3: $SK^\star \leftarrow \mathsf{KeyGen}(1^\kappa)$
4: $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\text{LoR}}(\cdot,\cdot,SK^\star),\mathsf{Decrypt}(\cdot,SK^\star)}(1^\kappa)$
5: **return** $b' = b$

Oracle Query 3 $\mathcal{O}^{\text{LoR}}(m_0, m_1, SK^\star)$:

1: $CT \leftarrow \mathsf{Encrypt}(m_b, SK^\star)$
2: $L = L \cup CT$
3: **return** CT

2.3.1 Symmetric Encryption

A symmetric encryption scheme [5] $\mathcal{SE}$ comprises three algorithms: KeyGen, Encrypt and Decrypt. KeyGen is a randomized algorithm that takes a security parameter κ and returns a secret key SK; Encrypt is a randomized[1] algorithm taking as input a message m and the secret key, and returning a ciphertext CT; finally, the Decrypt algorithm is deterministic and takes a ciphertext and the secret key and returns the corresponding plaintext m or a failure symbol $\perp$. For correctness, we require that, for all $SK \leftarrow \mathsf{KeyGen}(1^\kappa)$ and messages m in the messagespace, $m \leftarrow \mathsf{Decrypt}(\mathsf{Encrypt}(m, SK), SK)$. A symmetric authenticated encryption scheme is syntactically identical but considers integrity as well as privacy of messages, and $\perp$ is returned if the ciphertext is deemed 'inauthentic'.

Two standard notions of privacy for symmetric encryption schemes are *Indistinguishability under Chosen Plaintext Attack* (IND-CPA) and *Indistinguishability under Chosen Ciphertext Attack* (IND-CCA). These are given in Games 2 and 3 respectively. In each, the challenger chooses a random bit b and generates a key $SK^\star$. The adversary is given the security parameter (so can generate other keys itself) and oracle access to LoR which takes two messages of equal length and returns the encryption of message m_b. The adversary must guess the value of b. In the IND-CCA notion, the adversary is also given a decryption oracle for the challenge key but may not query a ciphertext that was output from the LoR oracle to avoid a trivial win.

Definition 2. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT) for $\mathbf{X} \in \{\text{IND-CPA, IND-CCA}\}$ is defined as:*

$$Adv^{\mathbf{X}}_{\mathcal{A},\mathcal{SE}}(1^\kappa) = \Pr[\mathbf{X}^{\mathcal{A}}_{\mathcal{SE}}(1^\kappa)] - \tfrac{1}{2}.$$

A symmetric encryption scheme is secure in the sense of $\mathbf{X}$ *if for all PPT adversaries $\mathcal{A}$, $Adv^{\mathbf{X}}_{\mathcal{A},\mathcal{SE}}(1^\kappa) \leq \text{negl}(\kappa)$.*

Bellare and Namprempre [5] considered two notions of integrity for symmetric encryption schemes: *Integrity of Plaintexts* (INT-PTXT) and *Integrity of Ciphertexts* (INT-CTXT) under chosen message attacks. INT-PTXT requires it to be hard to create a ciphertext which

[1] Encrypt could also be a stateful algorithm that updates and maintains a state between calls.

Oracle Query 4 $\mathcal{O}^{\mathsf{Decrypt}(CT,SK^\star)}$:

1: **if** $CT \in L$ **then return** $\perp$
2: **return** $\mathsf{Decrypt}(CT, SK^\star)$

Game 4 $\mathbf{Exp}_{\mathcal{A}}^{\text{INT-PTXT}}[1^\kappa]$:

1: $L = \epsilon$
2: $SK^\star \leftarrow \mathsf{KeyGen}(1^\kappa)$
3: $CT^\star \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Encrypt}(\cdot,SK^\star)},\text{Ver}(\cdot)}(1^\kappa)$
4: **return** $((\text{Ver}(CT^\star) = 1) \wedge (\mathsf{Decrypt}(CT^\star, SK^\star) \notin L))$

Oracle Query 5 $\mathcal{O}^{\mathsf{Encrypt}}(m, SK^\star)$:

1: $CT \leftarrow \mathsf{Encrypt}(m, SK^\star)$
2: $L = L \cup m$
3: **return** CT

Oracle Query 6 $\mathcal{O}^{\text{Ver}(CT)}$:

1: **if** $\perp \not\leftarrow \mathsf{Decrypt}(CT, SK^\star)$ **then return** 1
2: **else return** 0

decrypts to a message never encrypted by a legitimate sender, while INT-CTXT requires it to be hard to create a ciphertext that was not previously generated by a legitimate sender (regardless of the underlying plaintext). We will only require the first notion here, which is given in Game 4. The adversary is given both an encryption oracle and a verification oracle which returns 1 if the queried ciphertext is not deemed inauthentic. The adversary wins if it can produce a ciphertext that is deemed authentic and that decrypts to a message not previously queried to the encryption oracle.

Definition 3. *The* advantage *of an adversary $\mathcal{A}$ running in probabilistic polynomial time (PPT) is defined as:*

$$Adv_{\mathcal{A},\mathcal{SE}}^{\text{INT-PTXT}}(1^\kappa) = \Pr[\text{INT-PTXT}_{\mathcal{SE}}^{\mathcal{A}}(1^\kappa)].$$

A symmetric encryption scheme is secure in the sense of integrity of plaintexts *if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A},\mathcal{SE}}^{\text{INT-PTXT}}(1^\kappa) \leq \text{negl}(\kappa)$.*

In this work, we will require an authenticated symmetric encryption scheme that is secure in the sense of IND-CPA $\wedge$ INT-PTXT. As noted by Bellare and Namprempre [5], such a scheme can be constructed from an IND-CPA symmetric encryption scheme and a weakly or strongly unforgeable MAC using the generic composition techniques of *MAC-then-Encrypt* or *Encrypt-then-MAC*. As this latter composition is shown to be secure for all security choices of the encryption and MAC schemes and is more standard, we will adopt this notion in this paper. Thus, let $\mathcal{SE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a symmetric authenticated encryption scheme constructed from a symmetric encryption scheme $\mathcal{SE}' = (\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}')$ and a MAC $\mathcal{MAC} = (\mathsf{KeyGen}'', \mathsf{Tag}, \mathsf{Verify})$. KeyGen outputs two keys: SK_E corresponding to KeyGen' and SK_M relating to KeyGen''[2]. The encryption operation is defined as $\mathsf{Encrypt}(m, SK) = C \| \mathsf{Tag}(C, SK_M)$ for $C \leftarrow \mathsf{Encrypt}'(m, SK_E)$.

3 Policies

We now discuss the type of access control policies that are relevant in a multi-user verifiable outsourced computation environment. We consider graph-based policies where "objects" to be

[2]Note that in our setting where keys are derived from a KAS, the KAS could simply output strings that can be split into SK_E and SK_M.

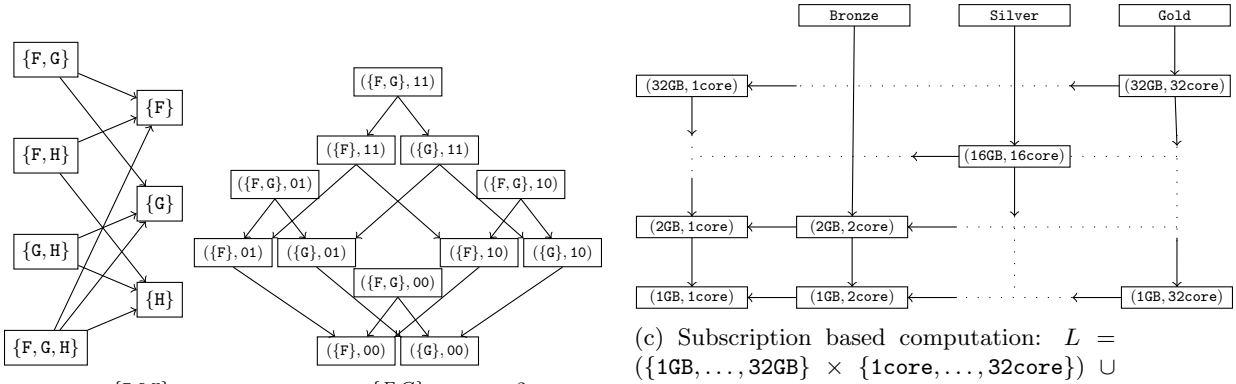
protected are not data files, as in traditional access control policies, but outsourced computations and their results. The "user" population comprises the sets of delegators $\mathcal{C}$, computational servers $\mathcal{S}$, and verifiers $\mathcal{V}$. We are interested in specifying and enforcing policies that restrict: (i) which functions a delegator may delegate; (ii) which functions a server may evaluate; (iii) which outputs a verifier may read. As noted in the introduction, we distinguish between entities that may learn a computational result and those that may only verify correctness (blind verify). In this work, we assume that *any* entity may perform the blind verification step but that only a restricted subset of verifiers should be able to retrieve the actual output value[3].

There has been considerable research in recent years on the cryptographic enforcement of access control policies [13, 14, 1]. Informally, access is regulated in a distributed fashion by issuing appropriate cryptographic keys to authorized subjects, rather than a centralized reference monitor mediating all attempts to access protected resources. Cryptographic access control generally focuses on read access and is regarded as being particularly appropriate when the protected content is read often but written rarely (since this would require re-encryption). In the context of VC, we will use cryptographic access control in somewhat unusual ways. Rather than storing static encrypted data and ensuring that only authorized users hold the relevant decryption key, we will encrypt dynamic messages within a protocol execution. In particular, to enforce policies restricting the computations that may be outsourced, a delegator must use an appropriate key to encrypt input data. Without the appropriate encryption, the input will be discarded by the server. The enforcement of policies for performing computations is achieved by distributing keys to servers that can be used to decrypt encrypted inputs. Without decryption, the server will be unable to read the input data and evaluate the function. The enforcement of (read) policies on outputs uses cryptographic access control in a more conventional fashion; results are published and protected via encryption with an appropriate key.

Cryptographic access control has been particularly widely studied in the context of information flow and graph-based access control policies (see Section 2.2). We restrict our focus to graph-based policies as these have been shown to encompass many notions of access control that are desirable in practice including information flow policies, role-based access control and attribute-based access control [12]. Recall that we define "user" population as the sets of delegators $\mathcal{C}$, computational servers $\mathcal{S}$, and verifiers $\mathcal{V}$. We define a security function $\lambda : \mathcal{C} \cup \mathcal{S} \cup \mathcal{V} \cup \mathcal{O} \rightarrow L$ where $\mathcal{O}$ is the family of computations that may be outsourced and $(L, \leqslant)$ is a poset of security labels. This function assigns a label from L, representing the security classification, to each delegator, server and computation in the PVC-AC system.

We first consider, in Section 3.1, the form of polices that restrict the computations a delegator may outsource and that a server may compute. We begin by examining simple policies over the choice of functions, before considering more fine-grained policies over sets of input values and other security posets. Then, in Section 3.2, we discuss policies that restrict the computation results a verifier may learn. Although we consider a variety of policies capturing a range of conditions on entities and other contextual information, this amounts to altering the policy poset and the labels assigned to each entity and computation; the definition in Section 4 and the instantiation in Section 6 is generic and encompasses any of the following forms of policy. We differentiate between computation policies over delegators and servers, denoted $\lambda^C(\cdot)$, and verification policies over delegators and verifiers, denoted $\lambda^V(\cdot)$. For ease of notation, we use λ to denote λ^C in Section 3.1, and to denote λ^V in Section 3.2.

[3]The same techniques that we present could also be applied to protect the (blind) verification keys to restrict the entities that may validate correctness of a computation.



(a) $\mathtt{L} = 2^{\{\mathtt{F},\mathtt{G},\mathtt{H}\}}$ (b) $L = 2^{\{F,G\}} \times \{0,1\}^2$ (c) Subscription based computation: $L = (\{\mathtt{1GB}, \ldots, \mathtt{32GB}\} \times \{\mathtt{1core}, \ldots, \mathtt{32core}\}) \cup \{\mathtt{Gold}, \mathtt{Silver}, \mathtt{Bronze}\}$

Figure 2: Example Posets of Security Labels

3.1 Delegation and Computation Policies

3.1.1 Policies over Functions

We begin by considering a simple case where policies are formulated purely in terms of the functions being computed. In simple terms, we associate each delegator C and server S with a set of functions $\lambda(C) \subseteq \mathcal{F}$ and $\lambda(S) \subseteq \mathcal{F}$ respectively. We define a *correctness criterion* that states that C should be able to prepare inputs for all functions $F \in \lambda(C)$ (and similarly for S) i.e. entities should be able to perform all operations that they are authorized for. The *security criterion* requires $\lambda(C), \lambda(S) \geqslant \lambda(F)$ in order to delegate or compute for F respectively, and that a set of unauthorized entities cannot collude to perform an operation that any of them couldn't perform alone.

More formally, we define the set of security labels L to be $2^{\mathcal{F}}$ (the power set of all considered functions). Then, $\lambda(C) \subseteq \mathcal{F}$ defines the set of functions that a delegator C may outsource an evaluation of, $\lambda(S) \subseteq \mathcal{F}$ denotes the functions a server S may compute, and $\lambda(o) = \{F\}$ labels the computation of $F \in \mathcal{F}$. Then, for any $x, y \in L$ we define an order relation$<$ such that $x < y$ if and only if $x \in \mathcal{F}, y \subseteq \mathcal{F}$ and $x \in y$. The corresponding Hasse diagram with $\mathcal{F} = \{F, G, H\}$ is shown in Figure 2a.[4] Any entity E authorized for $\lambda(E)$ is, by the correctness criterion, authorized to operate on all computations such that $\lambda(o) < \lambda(E)$. For example, in Figure 2a, an entity authorised for the set $\lambda(E) = \{F, G\}$ is authorized for the functions F and G as expected. Each label $l \in L$ will be associated with a key κ_l. To outsource a computation o of $F(x)$, C prepares the encoding of x using the key $\kappa_o = \{F\}$ which, by the security criterion, C knows if and only if $\lambda(C) \geqslant \lambda(o)$ i.e. if and only if $\{F\} \in \lambda(C)$. To compute $F(x)$, S uses the corresponding key κ_o. As before, S may do this if and only if $\{F\} \in \lambda(S)$.

3.1.2 Policies over Function Inputs

As well as limiting the functions that may be outsourced, we may wish to implement a more fine-grained access control policy determined by input values to functions. For ease of exposition let us assume that all functions have the same domain – for all $F \in \mathcal{F}$, $\mathsf{Dom}(F) = \{0,1\}^n$ for a positive, non-zero integer n.[5] We then redefine the security function such that "objects" are now considered to be pairs (F, x) where $F \in \mathcal{F}$ and $x \in \mathsf{Dom}(F)$ – that is, $\lambda : \mathcal{C} \cup \mathcal{S} \cup (\mathcal{F} \times \{0,1\}^n) \to L$.

We could, for example, let L be $2^{\mathcal{F}} \times \{0,1\}^n$. To define the order relation on L we must first define an ordering on the input data. One choice is to define a co-ordinatewise ordering

[4] Nodes for empty sets are excluded from these figures.

[5] If this is not the case then we can define $n = \max_{F \in \mathcal{F}} |\mathsf{Dom}(F)|$ and then redefine all functions F with $\mathsf{Dom}(F) < n$ to satisfy the required property by, for example, adding fixed points $F(x) = x$ for all $x \in \{0,1\}^n \setminus \mathsf{Dom}(F)$.

on $\{0,1\}^n$: that is $(x_1,\ldots,x_n) \leqslant (y_1,\ldots,y_n)$ if and only if $x_i \leqslant y_i$ for all i. Then for two labels $(\mathcal{G},x)$ and $(\mathcal{G}',x')$ in L, $(\mathcal{G},x) \leqslant (\mathcal{G}',x')$ if and only if $\mathcal{G} = \{F\}$ for some $F \in \mathcal{F}$, $F \in \mathcal{G}'$ and $x \leqslant x'$. This poset could be used to ensure delegators and servers operate over a limited range of data values. Specifically, for an entity E, we define $\lambda(E) = (\mathcal{G},x)$, where $\mathcal{G} \subseteq \mathcal{F}$ and $x \in \{0,1\}^n$, and $\lambda(o) = (\{F\},0^n)$. Then E is authorized to operate on $G(y)$ for all $G \in \mathcal{G}$ and all $y \leqslant x$. The Hasse diagram for this poset with $\mathcal{F} = \{F,G\}$ and $n = 2$ is shown in Figure 2b. Different choice of orderings over inputs lead to different restrictions e.g. one could consider bit strings as integers and use the natural ordering over integers for $x \leq y$.

An alternative would be to let L be the power set $2^{\mathcal{F}\times\{0,1\}^n}$. Then each function may be associated with a different range of permissible inputs. An entity E, is labelled by a set of pairs, each comprising a function label and an associated input label – $\lambda(E) = \{(F_1,x_1),\ldots,(F_m,x_m)\}$. A computation of $F(x)$ is labelled $\lambda(o) = (\{F\},x)$. Then, for any two labels,

$$\{(F_1,x_1),\ldots,(F_m,x_m)\} \leqslant \{(F'_1,x'_1),\ldots,(F'_{m'},x'_{m'})\}$$

if and only if $m = 1$ (the first label is a single pair), $F_1 = F'_i$ for some $i \in \{1,\ldots,m'\}$ and $x_1 \leqslant x'_i$.

We remark that the first choice of poset presented in this section corresponds to the idea of assigning a possible data range to each delegator and then authorizing them to outsource a set of functions on that data. For example, a chinese wall policy may result in employees being provided with separate portions of a database. Then, depending on the employee's role, they are permitted to evaluate a specific set of functions on that data partition. The second choice of poset is more akin to authorizing delegators to outsource a single, specified computation – that is, permitting the entity to evaluate $F(x)$ for only this choice of function and input data. For example, one could imagine an employee being given restricted access to some sensitive data to perform only a particular task, and they should not be able to use the data for other purposes. Finally, we observe that the choice of $L = 2^{\mathcal{F}} \times 2^{\{0,1\}^n}$ extends this second case to a situation where each function can be associated with an arbitrary sets of input values.

3.1.3 Enforcing General Graph-based Policies

We have seen how sets of functions and inputs can be mapped to a graph-based access control policy to restrict the functions a delegator may outsource and a server may compute. However, in practical applications, outsourced computation functionality will be required to integrate with existing workflows and existing access control policies. As an example, a company that operates Role-based Access Control (RBAC) on their local network and wishes to provide access to an external VC system must ensure that the same access control requirements are adhered to within this new environment. Additionally, it may not always be the case that a computation my be considered purely in terms of the function and input data. Indeed, although the evaluation will be uniquely determined by such factors, the level of protection required may depend on other contextual information. As a motivating example, consider a summation function over the integers. The semantic meaning of the integers in question may determine the overall classification of this computation – if the integers are city populations then this may not be classified at all, but troop deployments in different regions may be much more sensitive. We now briefly discuss how to formulate access control policies for multi-user VC settings that incorporate additional security labels relying on the environment or external access control policies.

We can use any poset of security labels L to classify each outsourced computation. For example, consider the total order $M = \{\texttt{Top-Secret},\texttt{Secret},\texttt{Classified},\texttt{Unclassified}\}$ representing the Bell-LaPadula clearance levels [4] and let K to be a set of need-to-know categories.

Then we can enforce the security function $\lambda : \mathcal{C} \cup \mathcal{S} \cup \mathcal{O} \to L = M \times 2^K$ where K specifies the nature, and M specifies the sensitivity, of the computation $o \in \mathcal{O}$. To delegate or compute $F(x)$, we require that the entity E's clearance level is at least the classification of the computation: $\lambda(E) \geqslant \lambda(o)$.

Similarly, it has been shown [12] that the set of roles for a Role-based Access Control Policy can be encoded as a poset. Thus, we could define L to be this role poset to integrate with existing RBAC policies within an organization.

We could also add additional criteria in the mapping to security labels. For example, we could formulate policies dependent on time by setting $\lambda : \mathcal{C} \cup \mathcal{S} \cup (\mathcal{F} \times \mathcal{T}) \to L$, where $\mathcal{T}$ is a set of time periods. We could then set $L = 2^{\mathcal{F} \times \mathcal{T}}$ as in the previous section to allow entities to operate on specific functions only during specified time periods. On the other hand, by defining $L = M$ as above, certain functions can be more highly protected during certain times of the day. In place of a temporal poset $\mathcal{T}$ we could also apply a geo-spatial poset to classify function evaluations differently depending on location data e.g. a function may be more sensitive if being outsourced from a battlefield as opposed to within a secured building. Finally, policies over function inputs can be extended to include characteristics of the input data (as in the averaging example above, contextual information often changes the level of protection required). Let Z be a set of labels describing data types that functions may be computed over, and define $L = 2^{\mathcal{F} \times (\{0,1\}^n \times Z)}$ for example. This enables the same input to the same function to be classified differently and hence to require different authorization from the delegator and the server.

As mentioned in the introduction, one interesting setting for multi-user VC is when users pay for computation-as-a-service [17, 21]. Users may pay a price per computation, in which case they could be issued a relevant key such as those discussed in Section 3.1.2, perhaps with a short time window in which they may perform the computation. Then, when outsourcing the computation, it is protected by this key to prove that the user has paid for this particular computation. An alternative model [17] would be for a server to allow subscriptions to different tiers of service. A user may pay more to subscribe to a higher tier, and then may submit multiple computations relating to this tier or lower. For example, consider a set $T = \{\texttt{Gold}, \texttt{Silver}, \texttt{Bronze}\}$ comprising tiers that users may subscribe to; `Gold` service may allow access to more computational resources, or access during busier periods, than `Silver`, and so on. For simplicity, suppose the resources to be considered are just RAM capacity and the number of processor cores available, and let R and C be sets comprising the available quantities of each. Then, the set of security labels L is set to be $L = (R \times C) \cup T$ — that is, the cartesian product of the available resources, with the addition of labels for each service tier. A user that pays for `Gold` membership can be assigned the label $\lambda(C) = \{\texttt{Gold}\}$. Each computation is labelled by the resources that it requires. The poset ordering over L allows each tier to access different subsets of resources (e.g. if a computation is particularly memory intensive then it may require a higher subscription fee to compute), as illustrated in Figure 2c.

3.2 Verification Policies

Thus far, we have discussed policies restricting the entities that may delegate and evaluate given computations. In the multi-user VC setting, it is equally important to apply access control to the act of verifying the results of a computation and, in particular, learning the output. Clearly, when considering computations over confidential data, it is not always appropriate to publish the results. As a trivial example, outsourcing the identity function could "legitimately" leak confidential data. The scheme of Alderman et al. [2] distinguished between blind verification (to ensure correctness) and output retrieval (to learn results). Here, we assume that all entities

are able to blind verify a result, but the set of entities that may learn the actual output should be restricted[6]. Thus, the set of verifiers we refer to in the following are just those distinguished verifiers that are able to retrieve computation results, and we wish to restrict the results that each may read.

The notion of "no write down" is an important property in many traditional access control policies. In the VC setting, this amounts to ensuring that any verifier should at least have the access rights of the delegating or computing entity, that is $\lambda(C), \lambda(S) \leqslant \lambda(V)$ e.g. a verifier may not read a result arising from input data that he is not also authorized to read. In some cases, however, the KDC may decide that the results of some computations are not as highly classified as the input data, or indeed the act of computing it. For example, statistics of company spending across all departments may be less sensitive than the spending of the research department alone. Alternatively, the results of a computation over classified data may be included in a public document, despite the input data remaining classified The encoded output from the computation may be published with the verification key such that recipients can verify the legitimacy of the published values. It may also be reasonable to expect that only authorised and trusted reviewers of the document should be able to verify, or that some (lower) form of clearance is still required to access the result.

3.2.1 Published Verification Token

In this second case, where verification tokens are published according to some policy, we can use similar posets and security functions to those in Section 3.1. The user population is the set of delegators $\mathcal{C}$ and verifiers $\mathcal{V}$, and objects are encoded outputs. The security function is defined to be $\lambda^V : \mathcal{V} \cup \mathcal{C} \cup \mathcal{O} \to L$, where $\mathcal{O}$ is the set of outsourced computations and $(L, \leqslant)$ is a poset of security labels. As part of the VC functionality, when encoding an input, C also generates an output retrieval key RK_o that enables the result of a computation to be retrieved given the encoded output and the verification key. We will protect this retrieval key and require the verifier to satisfy the verification policy in order to use it – that is $\lambda(V) \geqslant \lambda(o)$ for $o \in \mathcal{O}$. Since the delegator generates the retrieval key, and hence may also use it to retrieve the output, we also assume $\lambda(C) \geqslant \lambda(o)$. This is reasonable given that the delegator must have the right to compute any function that he outsources and hence (resources permitting) could certainly learn the result. As before we can extend the definition of λ and of L to accommodate more fine-grained policies over outputs.

3.2.2 Enforcement of No Write Down Policies

As mentioned, "No Write Down" is an important requirement of many access control policies. It ensures that an entity C may not write (encrypt) an object to a lower classification level $\lambda(o) < \lambda(C)$ as this could constitute a leak of classified data. It is, of course, possible for a delegator to write the result at his (maximal) clearance level and then any verifiers with higher access rights, by the correctness criterion of the poset, will be able to read the result. However, we may want to protect a result at a higher classification level (write up) e.g. when preparing a report that should only be read by management.

To do so, we define two posets: a computation poset $\mathcal{P}_C = (L, \leqslant)$ (as per Section 3.1) and a verification poset $\mathcal{P}_V$. $\mathcal{P}_V$ is constructed by inverting the order relation on $\mathcal{P}_C$ where $\mathcal{P}_V = (L, \geqslant)$ i.e. $x \leqslant y$ in $\mathcal{P}_V$ if and only if $y \leqslant x$ in $\mathcal{P}_C$. Then, delegation and computation of $o \in \mathcal{O}$ are performed using a key associated to $\lambda^C(o) \in \mathcal{P}_C$. Retrieving the result of a

[6]Note, we could use the same techniques to restrict blind verification if desired.

computation requires a key related to $\lambda^V(o) \in \mathcal{P}_V$. A verifier V may then retrieve the output if and only if $\lambda^V(V) \geqslant \lambda^V(o)$.

4 PVC with Access Control

We now formally define the notion of PVC-AC to enforce graph-based access control policies over delegators, servers and verifiers. Generally, as discussed above, we wish to impose restrictions on three activities: first, we wish to specify a (write) policy determining the computations a client is authorized to delegate – this means restricting the inputs a client may correctly prepare; second, we specify a (read) policy that determines the computations a server may compute – that is, encoded inputs he may access; lastly, we specify a (read) policy dictating the outputs a verifier may read. We differentiate between computation policies over delegators and servers, denoted $\lambda^C(\cdot)$, and verification policies over delegators and verifiers, denoted $\lambda^V(\cdot)$. We may also use $\lambda(\cdot)$ to denote both labels where appropriate. Finally, $\mathcal{P}_C$ and $\mathcal{P}_V$ denote posets encoding the computation and verification policies respectively. Recall that the set of outsourced computations is denoted by $\mathcal{O}$. A specific element $o \in \mathcal{O}$ consists of the function F, some input x as well as some auxiliary input (such as a label or additional contextual information).

We extend the functionality of the KDC from [2] to grant access control credentials to delegators, servers and verifiers. We may split the responsibilities between two KDCs: a *computation* KDC (CKDC) that generates function keys for the VC functionality, and an *authorization* KDC (AKDC) that manages access control policies. The AKDC could be a trusted authority on entities to grant relevant permissions, and may be used by multiple systems. For clarity, here we use only one KDC that performs both duties.

Definition 4. *A* Publicly Verifiable Outsourced Computation Scheme with Access Control (PVC-AC) *comprises the following algorithms:*

- $(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{P}_C, \mathcal{P}_V)$: Run by the KDC to generate public parameters for the PVC-AC system as well as secret information used to generate keys.
- $PK_F \leftarrow \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$: Run by the KDC to generate a delegation key, PK_F, for a function F.
- $SK_{\mathrm{ID}} \leftarrow \mathsf{Register}(\mathrm{ID}, \lambda(\mathrm{ID}), \mathrm{MK}, \mathrm{PP})$: Run by the KDC to create a personalised key SK_{ID} for an entity with identifier ID,[7] which grants rights for the label $\lambda(\mathrm{ID})$ according to the computation or verification policy.
- $EK_{F,S} \leftarrow \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$: The KDC creates an evaluation key $EK_{F,S}$ for a function F and server S.
- $(\sigma_o, VK_o, RK_o) \leftarrow \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \mathrm{PP})$: Run by a delegator C to outsource the computation of $F(x)$. C may do so only if it satisfies the computation policy – that is $\lambda^C(C) \geqslant \lambda^C(o)$. It outputs an encoded input σ_o, a public verification key VK_o to verify correctness, and an output retrieval key RK_o which verifiers satisfying $\lambda^V(V) \geqslant \lambda^V(o)$ may use to read $F(x)$.
- $\theta_o \leftarrow \mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), PP)$: Run by a server S holding an evaluation key $EK_{F,S}$, SK_S and an encoded input σ_o to evaluate $F(x)$ and output an encoding, θ_o, of the result. This succeeds if and only if $\lambda^C(S) \geqslant \lambda^C(o)$, that is S satisfies the computation policy for this evaluation.

[7] In future algorithms this will be S, C or V to denote a server, delegator or verifier respectively.

- $(\tilde{y}, \tau_{\theta_o}) \leftarrow \mathsf{Verify}(\theta_o, SK_V, VK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})$:
  Verification comprises two steps:

– $(RT_o, \tau_{\theta_o}) \leftarrow \mathsf{BVerif}(\mathrm{PP}, \theta_o, VK_o)$: Run by *any* verifier to produce a retrieval token RT_o and a token $\tau_{\theta_o} = (\mathsf{accept}, S)$ for a correct result, or $\tau_{\theta_o} = (\mathsf{reject}, S)$ to signify a cheating server S.

– $\tilde{y} \leftarrow \mathsf{Retrieve}(SK_V, RT_o, \tau_{\theta_o}, VK_o, RK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})$: Run by verifiers V in possession of the output retrieval key RK_o and the retrieval token RT_o from BVerif. If $\lambda^V(V) \geqslant \lambda^V(o)$ then V should be able to read the actual result $\tilde{y} = F(x)$ or $\perp$.

- $\{EK_{F,S'}\}$ or $\perp \leftarrow \mathsf{Revoke}(\tau_{\theta_o}, F, \mathrm{MK}, \mathrm{PP})$: If a verifier reports a misbehaving server (i.e. Verify returned $\tau_{\theta_o} = (\mathsf{reject}, S)$), the KDC issues updated evaluation keys $EK_{F,S'}$ to all servers, preventing S from performing further computations. If $\tau_{\theta_o} = (\mathsf{accept}, S)$ then this algorithm should output $\perp$.

We say that a PVC-AC scheme is *correct* if for all functions $F \in \mathcal{F}$, inputs x, honestly generated parameters, and honestly registered entities C, S and V (delegator, certified server and verifier respectively)such that $\lambda^C(C), \lambda^C(S) \geqslant \lambda^C(o)$ and $\lambda^V(C), \lambda^V(V) \geqslant \lambda^V(o)$, if S honestly runs $\mathsf{Compute}$ for F on an encoding of x generated by C, then V running Verify on the output from S will almost certainly output accept and $F(x)$. That is, if all algorithms are run honestly by authorised parties then the verifier should almost certainly accept. More formally we can write

Definition 5 (Correctness)**.** *A Publicly Verifiable Outsourced Computation Scheme with Access Control (PVC-AC) is* correct *for a family of functions $\mathcal{F}$ if for all functions $F \in \mathcal{F}$ and inputs x, where* $\mathrm{negl}(\cdot)$ *is a negligible function of its input:*

$$
\begin{aligned}
&\Pr[(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\lambda), PK_F \leftarrow \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP}),\\
&\quad SK_C \leftarrow \mathsf{Register}(C, \lambda(C), \mathrm{MK}, \mathrm{PP}), SK_S \leftarrow \mathsf{Register}(S, \lambda(S), \mathrm{MK}, \mathrm{PP}),\\
&\quad SK_V \leftarrow \mathsf{Register}(V, \lambda(V), \mathrm{MK}, \mathrm{PP}), EK_{F,S} \leftarrow \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP}),\\
&\quad (\sigma_o, VK_o, RK_o) \leftarrow \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \mathrm{PP}),\\
&\quad (F(x), (\mathsf{accept}, S)) \leftarrow \mathsf{Verify}(\mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), PP), SK_V,\\
&\qquad VK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})]\\
&\quad = 1 - \mathrm{negl}(\lambda),
\end{aligned}
$$

where $\lambda^C(C) \geqslant \lambda^C(o), \lambda^V(C) \geqslant \lambda^V(o), \lambda^C(S) \geqslant \lambda^C(o)$ and $\lambda^V(V) \geqslant \lambda^V(o)$ holds.

5 Security Definitions

We now introduce several security models capturing requirements of PVC-AC. These are important both to ensure that the enforcement mechanism for the access control policies correctly captures the required properties, and that the access control mechanism and the PVC implementation (if built from separate primitives) interact securely. As noted by Ferrara et al. [14], it is not always immediate that (probabilistic) cryptographic enforcement mechanisms can safely implement policies that are generally specified in absolute terms; hence formally defining and proving the required security goals is necessary. Notions of Public Verifiability, Revocation and Vindictive Servers follow naturally from [2] with additional inputs for policy declarations. As these do not rely on the access control properties, the proofs follow naturally. We let o denote an outsourced computation (including descriptors such as the

function, input and environmental factors) such that $\lambda(o)$ encompasses the required clearance to operate on this computation. $\mathcal{A}^{\mathcal{O}}$ denotes $\mathcal{A}$ having oracle access to the following functions: $\mathsf{FnInit}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Register}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{Certify}(\cdot, \cdot, \mathrm{MK}, \mathrm{PP})$, $\mathsf{ProbGen}(\cdot, SK_{(\cdot)}, \cdot, \cdot, \cdot, \mathrm{PP})$, $\mathsf{Compute}(\cdot, \cdot, SK_{(\cdot)}, \cdot, \cdot, PP)$ and $\mathsf{Revoke}(\cdot, \cdot, , \mathrm{MK}, \mathrm{PP})$. Each oracle, aside from Register, simply runs the relevant algorithm, with any restrictions on permissible queries stated below. The Register oracle specified in Oracle Query 7 is used in all games except Authorized Verification which instead uses Oracle Query 8 which uses the verification policy instead of the computation policy. We define the *advantage* and *security* of $\mathcal{A}$ in each of the following gamesas:

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

- *For $\boldsymbol{X} \in \{AO, AC\}$:*

$$Adv^{\mathbf{X}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, F, 1^{\kappa}, q) = \Pr[\mathbf{Exp}^{\mathbf{X}}_{\mathcal{A}}[\mathcal{PVC}_{AC}, F, 1^{\kappa}] = 1]$$

- *For $\boldsymbol{X} \in \{AV, wIP\}$:*

$$Adv^{\mathbf{X}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, F, 1^{\kappa}, q) = |\Pr[\mathbf{Exp}^{\mathbf{X}}_{\mathcal{A}}[\mathcal{PVC}_{AC}, F, 1^{\kappa}] = 1] - \frac{1}{2}|$$

A PVC-AC is secure *against Game $\boldsymbol{X}$ for a function F, if for all PPT adversaries $\mathcal{A}$, $Adv^{\mathbf{X}}_{\mathcal{A}}(\mathcal{PVC}_{AC}, F, 1^{\kappa}, q) \leqslant \mathrm{negl}(\kappa)$.*

5.1 Authorized Outsourcing

In Game 5 we formalize the notion that a delegator may not outsource any computation for which he is not authorized – that is, any computation o where $\lambda^C(o) \not\leqslant \lambda^C(C)$. Clearly, within our framework we cannot make any guarantees about what an entity may do externally. For instance, we cannot enforce that a client does not circumvent the access control policies using an external server not subject to these controls, however we do enforce that to use the provided functionality (i.e. to contract an available server registered in this system) they must prove authorization. This assumption seems reasonable, taking into consideration organizational boundaries – external control should perhaps be enforced by limiting external communication channels (e.g. using a strict firewall). Similarly, we cannot enforce that an entity does not share key content, but we do ensure that such collusion does not enable access that either entity alone could not access. Additionally, due to the revocation functionality, it may be undesirable for a server to share key material as he must trust the additional server not to cheat.

The game proceeds as follows. First the challenger runs the setup procedures for the scheme and registers the adversary for the adversary's choice of security label. The adversary is given the public information, his secret key and oracle access, and outputs a choice of outsourced computation o to attack, including the function and input, F and x. We require that no security label that $\mathcal{A}$ has queried to the Register oracle may allow the derivation of a key for $\lambda^C(o)$ as this would be a trivial win, and we return 0 since the adversary has not found a valid attack target. We also require that any ID that $\mathcal{A}$ queries to its oracle must previously have been queried to the Register oracle (which is in keeping with realistic operation). The ProbGen oracle may not be queried on the challenge computation o. Note that $\mathcal{A}$ may register entities of his choice for any security label that is not an ancestor of the challenge label, and hence does not need oracle access for these labels. BVerif relies only on public information and $\mathcal{A}$ may Register any verifier he likes to learn $\kappa_{\lambda^V(\cdot)}$ to run Retrieve, so these oracles are not required. $\mathcal{C}$ sets up keys for F and simulates registering two entities that are authorized to perform the computation

Game 5 $\mathbf{Exp}^{AO}_{\mathcal{A}}[\mathcal{PVC}_{\mathrm{AC}}, F, 1^\kappa]$:

$L = \epsilon, o = \perp$
$(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{P}_C, \mathcal{P}_V)$;
$\lambda^C(\mathcal{A}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathrm{PP})$;
$SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathcal{A}, \lambda^C(\mathcal{A}), \mathrm{MK}, \mathrm{PP})$;
$L = L \cup \lambda^C(\mathcal{A})$;
$o = (F, x, \mathsf{aux}) \leftarrow \mathcal{A}^{\mathcal{O}}(\lambda^C(\mathcal{A}), SK_{\mathcal{A}}, \mathrm{PP})$;
for all $\lambda^C(v_i) \in L$ **do**
    if $\lambda^C(o) \leqslant \lambda^C(v_i)$ **then**
        return 0
$SK_S \leftarrow \mathsf{Register}(S, \lambda^C(S), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^C(S) \geqslant \lambda^C(o)$;
$SK_V \leftarrow \mathsf{Register}(V, \lambda^V(V), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^V(V) \geqslant \lambda^V(o)$;
$PK_F \leftarrow \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$;
$EK_{F,S} \leftarrow \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$;
$(\sigma_o, VK_o, RK_o) \leftarrow \mathcal{A}^{\mathcal{O}}(o, \lambda(o), \lambda(\mathcal{A}), PK_F, \mathrm{PP})$;
$\theta_o \leftarrow \mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), \mathrm{PP})$;
$(\tilde{y}, \tau_{\theta_o}) \leftarrow \mathsf{Verify}(\theta_o, SK_V, VK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP})$;
if $((\tilde{y}, \tau_{\theta_o}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ **then return** 1
else return 0

Oracle Query 7 $\mathcal{O}^{\mathsf{Register}}(ID, \lambda(ID), \mathrm{MK}, \mathrm{PP})$:

if $(o \neq \perp) \wedge \lambda(ID) \geqslant \lambda^C(o)$ **then**
    return $\perp$
$L = L \cup \lambda(ID)$;
return $SK_{ID} \leftarrow \mathsf{Register}(ID, \lambda(ID), \mathrm{MK}, \mathrm{PP})$

and verification for o respectively. The adversary is then challenged, given all information that a real attacker may learn and oracle access, to output an encoded input that the Compute and Verify algorithms will accept – that is, an unauthorized adversary must produce an input that is accepted and computed on by honest entities.

5.2 Authorized Computation

Authorized Computation, in Game 6, proceeds similarly to Game 5 and captures the notion that a computational result should only be considered valid if generated by an authorized party. The challenger sets up the system and registers the adversary for its choice of security label. $\mathcal{A}$ then chooses a target computation that it is not authorized to compute by *any* of the keys it holds. The challenger simulates two entities: a delegator and a verifier that are authorized for this computation and certifies $\mathcal{A}$ as a server for F. The challenger creates an encoded input by running ProbGen on the adversary's target computation and gives this to $\mathcal{A}$ who must output an encoded output that is accepted by the verifier. Note that although the adversary chooses the input to the computation, and therefore can certainly compute $F(x)$, it still should not be able to convince the verifier to accept. The oracle queries are the same as in the previous section except that ProbGen may be queried for any input (even the challenge computation) and the Compute oracle cannot be run on the challenge input.

5.3 Authorized Verification

In Game 7 we capture the notion that an unauthorized verifier should not be able to learn the output of a computation. This is formulated in an indistinguishability game, where the adversary chooses two computations to give to the challenger. To avoid a trivial win, we require that neither of the associated labels in the verification poset are less than or equal to a label queried to the Register oracle. The challenger chooses *one* of these at random and simulates outsourcing and computing this computation. The adversary is provided with the encoded

Game 6 $\mathbf{Exp}^{AC}_{\mathcal{A}}[\mathcal{PVC}_{\mathrm{AC}}, F, 1^\kappa]$:

$L = \epsilon, o = \perp$
$(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{P}_C, \mathcal{P}_V)$;
$\lambda^C(\mathcal{A}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathrm{PP})$;
$SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathcal{A}, \lambda^C(\mathcal{A}), \mathrm{MK}, \mathrm{PP})$;
$L = L \cup \lambda^C(\mathcal{A})$;
$o = (F, x, \mathsf{aux}) \leftarrow \mathcal{A}^{\mathcal{O}}(\lambda^C(\mathcal{A}), SK_{\mathcal{A}}, \mathrm{PP})$;
for all $\lambda^C(v_i) \in L$ **do**
    if $\lambda^C(o) \leqslant \lambda^C(v_i)$ **then**
        return 0
$SK_C \leftarrow \mathsf{Register}(C, \lambda^C(C), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^C(C) \geqslant \lambda^C(o)$;
$SK_V \leftarrow \mathsf{Register}(V, \lambda^V(V), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^V(V) \geqslant \lambda^V(o)$;
$PK_F \leftarrow \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$;
$EK_{F,\mathcal{A}} \leftarrow \mathsf{Certify}(\mathcal{A}, F, \mathrm{MK}, \mathrm{PP})$;
$(\sigma_o, VK_o, RK_o) \leftarrow \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \mathrm{PP})$;
$\theta_o \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_o, VK_o, o, \lambda^C(o), EK_{F,\mathcal{A}}, PK_F, RK_o, \mathrm{PP})$;
$(\tilde{y}, \tau_{\theta_o}) \leftarrow \mathsf{Verify}(\theta_o, SK_V, VK_o, \lambda^V(V), \lambda^V(o), RK_o, \mathrm{PP})$;
if $((\tilde{y}, \tau_{\theta_o}) \neq (\perp, (\mathsf{reject}, \mathcal{A})))$ **then return** 1
else return 0

Game 7 $\mathbf{Exp}^{AV}_{\mathcal{A}}[\mathcal{PVC}_{\mathrm{AC}}, F, 1^\kappa]$:

$L = \epsilon, o_0 = o_1 = \perp$
$(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^\kappa, \mathcal{P}_C, \mathcal{P}_V)$;
$\lambda^V(\mathcal{A}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathrm{PP})$;
$SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathcal{A}, \lambda^V(\mathcal{A}), \mathrm{MK}, \mathrm{PP})$;
$L = L \cup \lambda^V(\mathcal{A})$;
$(o_0, o_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(SK_{\mathcal{A}}, \mathrm{PP})$ s.t. $\lambda^V(o_0), \lambda^V(o_1) > \lambda^V(v_i)\ \forall \lambda^V(v_i) \in L$;
$b \xleftarrow{\$} \{0, 1\}$;
$SK_C \leftarrow \mathsf{Register}(C, \lambda^C(C), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^C(C) \geqslant \lambda^C(o_b)$;
$SK_S \leftarrow \mathsf{Register}(V, \lambda^C(S), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^C(S) \geqslant \lambda^C(o_b)$;
$PK_F \leftarrow \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$;
$EK_{F,S} \leftarrow \mathsf{Certify}(S, F, \mathrm{MK}, \mathrm{PP})$;
$(\sigma_{o_b}, VK_{o_b}, RK_{o_b}) \leftarrow \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o_b), \mathrm{PP})$;
$\theta_{o_b} \leftarrow \mathsf{Compute}(\sigma_{o_b}, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o_b), \mathrm{PP})$;
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(\theta_{o_b}, VK_{o_b}, RK_{o_b}, \lambda(o_0), \lambda(o_1), SK_{\mathcal{A}}, PK_F, \mathrm{PP})$;
if $b' = b$ **then return** 1
else return 0

Oracle Query 8 $\mathcal{O}^{\mathsf{Register}}(ID, \lambda(ID), \mathrm{MK}, \mathrm{PP})$:

if $(o_b \neq \perp) \wedge \lambda(ID) \geqslant \lambda^V(o_b)$ **then**
    return $\perp$
$L = L \cup \lambda(ID)$;
return $SK_{ID} \leftarrow \mathsf{Register}(ID, \lambda(ID), \mathrm{MK}, \mathrm{PP})$

output from this computation, along with the verification keys and other public information, and must guess which computation was chosen. As the adversary chose the computations, it can work out the results; however, no information about the result should leak from the encoded output unless the verifier is authorized, and hence the adversary should be unable to tell which result the output corresponds to. The oracle queries are the same as in the previous section except that ProbGen and Compute may be queried for any input. The adversary is also given access to a Retrieve oracle (which cannot be queried for the challenge computation) in order to observe outputs for the challenge label.

5.4 Weak Input Privacy

The notion of weak input privacy, captured in Game 8, is not as strong as the input privacy often considered in PVC settings where computational servers learn nothing about the data

they are working on. Instead, we are interested in ensuring that servers (or other entities) that are not authorized to access (compute on) the input data may not learn x. If full input privacy is required then the underlying PVC scheme, such as the KP-ABE based one used as a black box in the construction, could be replaced by one built from a predicate encryption scheme for the same class of functions.

The game begins like the Authorized Verification game with the adversary selecting two computations that he is not authorized for by *any* key that he has queried to the Register oracle. The Compute oracle can be queried for any input except the challenge inputs. The challenger then simulates the outsourcing of one of these computations and gives the adversary the resulting encoded input. The adversary must guess which computation the input corresponds to i.e. which input data is encoded in the input.

Game 8 $\mathbf{Exp}_{\mathcal{A}}^{wIP}[\mathcal{PVC}_{\mathrm{AC}}, F, 1^{\kappa}]$:

$L = \epsilon, o = \perp$
$(\mathrm{PP}, \mathrm{MK}) \leftarrow \mathsf{Setup}(1^{\kappa}, \mathcal{P}_C, \mathcal{P}_V)$;
$\lambda^C(\mathcal{A}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathrm{PP})$;
$SK_{\mathcal{A}} \leftarrow \mathsf{Register}(\mathcal{A}, \lambda^C(\mathcal{A}), \mathrm{MK}, \mathrm{PP})$;
$L = L \cup \lambda^C(\mathcal{A})$;
$(o_0, o_1) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(SK_{\mathcal{A}}, \mathrm{PP})$ s.t. $\lambda^C(o_0), \lambda^C(o_1) > \lambda^C(v_i)\ \forall \lambda^C(v_i) \in L$;
$b \xleftarrow{\$} \{0, 1\}$;
$o = o_b$;
$SK_C \leftarrow \mathsf{Register}(C, \lambda^C(C), \mathrm{MK}, \mathrm{PP})$ *s.t.* $\lambda^C(C) \geqslant \lambda^C(o_b)$;
$PK_F \leftarrow \mathsf{FnInit}(F, \mathrm{MK}, \mathrm{PP})$;
$EK_{F,\mathcal{A}} \leftarrow \mathsf{Certify}(\mathcal{A}, F, \mathrm{MK}, \mathrm{PP})$;
$(\sigma_{o_b}, VK_{o_b}, RK_{o_b}) \leftarrow \mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o_b), \mathrm{PP})$;
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(\sigma_{o_b}, VK_{o_b}, \lambda(o_0), \lambda(o_1), EK_{F,\mathcal{A}}, SK_{\mathcal{A}}, PK_F, \mathrm{PP})$;
if $b' = b$ **then return** 1
else return 0

6 Instantiation

We now provide an example provably secure instantiation of PVC-AC. The approach we take makes generic, black-box use of existing RPVC schemes such as the Key-policy Attribute-based Encryption approaches [2, 20], and introduces the use of a symmetric Key Assignment Scheme (KAS) to restrict the behaviour of entities.

6.1 Informal Overview

The graph-based policies discussed in Section 3 assign labels to each entity and outsourced computation. The ordering relation and the correctness criterion ensures that entities are authorized for all appropriate computations (those that are descendants of their label in the Hasse diagram of the poset). A KAS is designed to enforce such policies, and assigns a key to each label. Each entity is provided with a key corresponding to their label, and they may derive keys for all descendants. As per the security criterion, users may not collude to derive keys for which they are not authorized.

In our setting, we restrict the computations a delegator may outsource, the computations a server may perform, and the results a verifier may learn. We use two independent KASs instantiated over the computation and verification posets, $\mathcal{P}_C$ and $\mathcal{P}_V$ respectively. Delegators and servers are issued a key according to their label in the computation poset, while delegators and verifiers are given a key from the verification KAS. Appropriate keys from the computation KAS are used to encrypt the encoded input for a computation. To encode an input in a manner that will be accepted by a server, delegators must encrypt the encoded input using

the key, $\kappa_{\lambda^C(o)}$, associated to the computation within the computation poset. To do so, the delegator must be able to derive $\kappa_{\lambda^C(o)}$ and hence must hold a key at that level or higher i.e. $\lambda(C) \geqslant \lambda(o)$ – delegators must be authorized by the KDC to outsource the computation. Similarly, only authorized servers can derive the decryption key to access the encoded input and perform the computation. By the IND-CPA security of the symmetric encryption scheme used, no information about the encoded input is learnt by an unauthorized entity. To enforce verification policies, delegators generate a retrieval key during delegation, and encrypt this using an appropriate key from the verification KAS. Then, only verifiers that can derive this key may decrypt the ciphertext to recover the retrieval key and learn the output.

6.2 Formal Details

Let $\mathcal{RPVC}$ = (RPVC.Setup, RPVC.FnInit, RPVC.Register, RPVC.Certify, RPVC.ProbGen, RPVC.Compute, RPVC.BVerif, RPVC.Retrieve, RPVC.Revoke) be an RPVC scheme, as defined by Alderman et al. [2], for a class of functions $\mathcal{F}$. Let $\mathcal{SE}$= (SE.KeyGen, SE.Encrypt, SE.Decrypt) be an authenticated symmetric encryption scheme secure in the sense of IND-CPA$\wedge$INT-PTXT, and let $\mathcal{KAS}$ = (KAS.MakeKeys, KAS.MakeSecrets, KAS.MakePublicData, KAS.GetKey) be a Key Assignment Scheme secure against Strong-Key Indistinguishability[8] whose keys are compatible with $\mathcal{SE}$. Finally, let $\mathcal{P}_C$ denote the poset encoding computation policies (e.g. $(L, \leqslant)$), and similarly let $\mathcal{P}_V$ denote the poset encoding verification policies (e.g. $(L, \geqslant)$)[9]. Then, for the same class of functions $\mathcal{F}$, there is a PVC-AC scheme $\mathcal{PVC}_{\text{AC}}$ = (PVC_{AC}.Setup, PVC_{AC}.FnInit, PVC_{AC}.Register, PVC_{AC}.Certify, PVC_{AC}.ProbGen, PVC_{AC}.Compute, PVC_{AC}.Verify, PVC_{AC}.Revoke) defined in Algorithms 1-9.

Alg. 1 Setup$(1^\kappa, \mathcal{P}_C, \mathcal{P}_V) \rightarrow (\text{PP}, \text{MK})$

1: $(\text{PP}', \text{MK}') \leftarrow$ RPVC.Setup(1^κ)
2: $\kappa_{\mathcal{P}_C} \leftarrow$ KAS.MakeKeys$(1^\kappa, \mathcal{P}_C)$
3: $\omega_{\mathcal{P}_C} \leftarrow$ KAS.MakeSecrets$(1^\kappa, \mathcal{P}_C)$
4: $Pub_{\mathcal{P}_C} \leftarrow$ KAS.MakePublicData$(1^\kappa, \mathcal{P}_C)$
5: $\kappa_{\mathcal{P}_V} \leftarrow$ KAS.MakeKeys$(1^\kappa, \mathcal{P}_V)$
6: $\omega_{\mathcal{P}_V} \leftarrow$ KAS.MakeSecrets$(1^\kappa, \mathcal{P}_V)$
7: $Pub_{\mathcal{P}_V} \leftarrow$ KAS.MakePublicData$(1^\kappa, \mathcal{P}_V)$
8: Set $\text{PP} = (\text{PP}', Pub_{\mathcal{P}_C}, Pub_{\mathcal{P}_V})$
9: Set $\text{MK} = (\text{MK}', \kappa_{\mathcal{P}_C}, \omega_{\mathcal{P}_C}, \kappa_{\mathcal{P}_V}, \omega_{\mathcal{P}_V})$

Alg. 2 FnInit$(F, \text{MK}, \text{PP}) \rightarrow PK_F$

1: $PK_F \leftarrow$ RPVC.FnInit$(F, \text{MK}', \text{PP}')$

Alg. 3 Register$(\text{ID}, \lambda(\text{ID}), \text{MK}, \text{PP}) \rightarrow \text{SK}_{\text{ID}}$

1: **if** ID is a server **then**
2: $\quad SK'_{\text{ID}} \leftarrow$ RPVC.Register$(\text{ID}, \text{MK}', \text{PP}')$
3: $\quad SK_{\text{ID}} = (SK'_{\text{ID}}, \kappa_{\lambda^C(\text{ID})}, \omega_{\lambda^C(\text{ID})})$
4: **else if** ID is a delegator **then**
5: $\quad SK_{\text{ID}} = (\kappa_{\lambda^C(\text{ID})}, \omega_{\lambda^C(\text{ID})}, \kappa_{\lambda^V(\text{ID})}, \omega_{\lambda^V(\text{ID})})$
6: **else**
7: $\quad SK_{\text{ID}} = (\kappa_{\lambda^V(\text{ID})}, \omega_{\lambda^V(\text{ID})})$

Alg. 4 Certify$(S, F, \text{MK}, \text{PP}) \rightarrow EK_{F,S}$

1: $EK_{F,S} \leftarrow$ RPVC.Certify$(S, F, \text{MK}', \text{PP}')$

[8] It has been shown that S-KI is equivalent to KI [9]. We make use of the additional queries in the S-KI game, but due to the equivalence this is not a strengthening of the assumptions. It in interesting to note that the form of the S-KI game is useful as a proof technique besides the original motivation to reflect realistic attacks.

[9] We use $\kappa_{\mathcal{P}_C}$ to denote the set of all keys generated by the KAS for the computation poset $\mathcal{P}_C$ and, for a label $\lambda^C(\text{ID}) \in \mathcal{P}_{\text{C}}$, the associated key is $\kappa_{\lambda^C(\text{ID})} \in \kappa_{\mathcal{P}_C}$.

Alg. 5 $\mathsf{ProbGen}(x, SK_C, PK_F, \lambda(C), \lambda(o), \mathrm{PP}) \rightarrow (\sigma_o, VK_o, RK_o)$

$(\sigma'_o, VK_o, RK'_o) \leftarrow \mathsf{RPVC.ProbGen}(x, PK_F, \mathrm{PP}')$
$\kappa_{\lambda^C(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^C(C), \lambda^C(o), \omega_{\lambda^C(C)}, \mathrm{PP})$
$\kappa_{\lambda^V(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^V(C), \lambda^V(o), \omega_{\lambda^V(C)}, \mathrm{PP})$
if $(\kappa_{\lambda^C(o)} \neq \perp)$ and $(\kappa_{\lambda^V(o)} \neq \perp)$ **then**
    $\sigma_o = (\lambda^C(o), \mathsf{SE.Encrypt}(\sigma'_o, \kappa_{\lambda^C(o)}))$
    $RK_o = (\lambda^V(o), \mathsf{SE.Encrypt}(RK'_o, \kappa_{\lambda^V(o)}))$

Alg. 6 $\mathsf{Compute}(\sigma_o, EK_{F,S}, SK_S, \lambda^C(S), \lambda^C(o), \mathrm{PP}) \rightarrow \theta_o$

Parse σ_o as $(\lambda^C(o), c)$
$\kappa_{\lambda^C(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^C(S), \lambda^C(o), \omega_{\lambda^C(S)}, \mathrm{PP})$
if $\kappa_{\lambda^C(o)} = \perp$ **then**
    return $\perp$
else
    $\sigma'_o \leftarrow \mathsf{SE.Decrypt}(c, \kappa_{\lambda^C(o)})$
    if $\sigma'_o = \perp$ **then return** $\perp$
    else $\theta_o \leftarrow \mathsf{RPVC.Compute}(\sigma'_o, EK_{F,S}, SK_S, PP')$

Alg. 7 $\mathsf{BVerif}(\theta_o, VK_o, \mathrm{PP}) \rightarrow (RT_o, \tau_{\theta_o})$:

$(RT_o, \tau_{\theta_o}) \leftarrow \mathsf{RPVC.BVerif}(\theta_o, VK_o, \mathrm{PP}')$

Alg. 8 $\mathsf{Retrieve}(RT_o, \tau_{\theta_o}, VK_o, RK_o, \lambda^V(V), \lambda^V(o), \mathrm{PP}) \rightarrow \tilde{y}$:

Parse RK_o as $(\lambda^V(o), e)$
$\kappa_{\lambda^V(o)} \leftarrow \mathsf{KAS.GetKey}(\lambda^V(V), \lambda^V(o), \omega_{\lambda^V(V)}, \mathrm{PP})$
if $\kappa_{\lambda^V(o)} = \perp$ **then**
    return $\perp$
else
    $RK'_o \leftarrow \mathsf{SE.Decrypt}(e, \kappa_{\lambda^V(o)})$
    $\tilde{y} \leftarrow \mathsf{RPVC.Retrieve}(RT_o, \tau_{\theta_o}, VK_o, RK'_o, \mathrm{PP}')$

Alg. 9 $\mathsf{Revoke}(\tau_{\theta_o}, F, \mathrm{MK}, \mathrm{PP}) \rightarrow \{EK_{F,S'}\}$ or $\perp$

return $\mathsf{RPVC.Revoke}(\tau_{\theta_o}, F, \mathrm{MK}, \mathrm{PP}')$

We now give a theorem and proof that the construction presented above is secure against the games presented in Section 5.

Theorem 1. *Given any Strong-Key-Indistinguishability secure KAS, any RPVC scheme secure in the sense of Public Verifiability, Revocation, and Blind Verification, and an authenticated symmetric encryption scheme secure in the sense of* IND-CPA $\wedge$ INT-PTXT*, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms* 1-9*. Then $\mathcal{PVC}_{AC}$ is secure in the sense of Public Verifiability, Revocation, Blind Verification, Authorized Outsourcing, Authorized Computation, Weak Input Privacy, and Authorized Verification.*

Informally, the security proofs follow from the security of the underlying RPVC scheme and the S-KI and IND-CPA security properties. The proofs for Public Verifiability, Revocation and Blind Verification are similar to the proofs given by Alderman et al. [2] up to some syntactical changes. We now present a formal proof for Theorem 1.

6.3 Security Proofs

Lemma 1. *Given a secure RPVC scheme, a symmetric authenticated encryption scheme secure in the sense of* IND-CPA $\wedge$ INT-PTXT *and a KAS secure against Strong-Key Indistinguishability, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 1–9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of Authorized Outsourcing (Game 5).*

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage δ in the Authorized Outsourcing game and let $\mathcal{A}_{SE}$ be an adversary playing the IND-CPA game with a challenger $\mathcal{C}$ against $\mathcal{SE}$. We first transition to a slightly modified version of the Authorized Outsourcing game, showing

a negligible difference between the two. We can then use an adversary against this modified game to break the IND-CPA security of the symmetric encryption scheme.

- **Game 0.** This is the Authorized Outsourcing game as defined in Section 5.1.
- **Game 1.** This is identical to **Game 0**, except that we replace the key $\kappa_{\lambda^C(o)}$ for the challenge computation o with a random key $\kappa^\star$ drawn uniformly from the keyspace.

Game 0 to Game 1 This game hop relies on the Strong Key Indistinguishability (S-KI) of the KAS. Suppose an adversary $\mathcal{A}_{VC}$ exists that can distinguish **Game 0** from **Game 1** with non-negligible advantage δ. Then there exists an adversary $\mathcal{A}_{SE}$ that breaks the S-KI of the KAS also with advantage δ. $\mathcal{A}_{VC}$ will play either **Game 0** or **Game 1** with $\mathcal{A}_{SE}$ acting as the challenger, and must guess correctly which game he is playing. $\mathcal{A}_{SE}$ in turn will play the S-KI game with a challenger $\mathcal{C}$. This reduction is important to ensure that no additional information about the encryption key is leaked through the construction.

1. $\mathcal{C}$ begins by selecting a bit $b \xleftarrow{\$} \{0,1\}$ which will determine whether the challenge key is real or random, and hence whether $\mathcal{A}_{VC}$ plays **Game 0** or **Game 1**. $\mathcal{C}$ continues by instantiating the KAS in lines 1 to 5 and giving the public information to $\mathcal{A}_{SE}$.
2. $\mathcal{A}_{SE}$ runs lines 1 and 2 of the Authorized Outsourcing game, and sends $\mathcal{A}_{VC}$ the generated public parameters. During the Setup algorithm in line 2, $\mathcal{A}_{SE}$ will not run lines 2 to 4 of Algorithm 1. Instead, it will set $Pub_{\mathcal{P}_C}$ to be the challenge public information from the S-KI challenger $\mathcal{C}$ and will make use of oracle queries to $\mathcal{C}$ for the remaining parameters.
3. $\mathcal{A}_{VC}$ is then given oracle access. Queries to the FnInit, $\mathsf{Certify}$ and Revoke functions can be answered simply by running the corresponding algorithm. Queries to the remaining algorithms may need to make use of KAS derived keys. $\mathcal{A}_{SE}$ can use its knowledge of the verification poset and the corresponding KAS which it owns where appropriate but it does not know anything other than the public information for the computational poset, because this is the challenge poset for the S-KI game. Thus, $\mathcal{A}_{SE}$ will issue $\mathsf{Corrupt}$ oracle queries to retrieve the necessary keys and secret information for the queried identity label. Since $\mathcal{A}_{VC}$ has not yet chosen a challenge computation o, and $\mathcal{A}_{SE}$ has not yet chosen a challenge node $v^\star$, the $\mathsf{Corrupt}$ oracle will always return a valid key and secret pair, which $\mathcal{A}_{SE}$ can use to form a full, valid response for $\mathcal{A}_{VC}$.
4. $\mathcal{A}_{VC}$ will select a challenge security label $\lambda^C(\mathcal{A})$ for itself, which $\mathcal{A}_{SE}$ will register it for using another $\mathsf{Corrupt}$ oracle query to $\mathcal{C}$ – that is, it will query $\mathcal{O}^{\mathsf{Corrupt}}(\lambda^C(\mathcal{A}), \perp)$ in the S-KI game. $\mathcal{C}$ will add $\lambda^C(\mathcal{A})$ to the list Q and return $(\kappa_{\lambda^C(\mathcal{A})}, \omega_{\lambda^C(\mathcal{A})})$. Additionally, $\mathcal{A}_{SE}$ may run $\mathsf{RPVC.Register}(\mathcal{A}, \mathrm{MK}', \mathrm{PP}')$ to create $SK'_{\mathcal{A}}$. Finally, $\mathcal{A}_{SE}$ sets $\mathrm{SK}_{\mathcal{A}} = (SK'_{\mathcal{A}}, \kappa_{\lambda^C(\mathcal{A})}, \omega_{\lambda^C(\mathcal{A})})$ as well as updating the list to $L = L \cup \lambda^C(\mathcal{A})$.
5. $\mathcal{A}_{VC}$ is again given oracle access which $\mathcal{A}_{SE}$ can respond to as above. Eventually, $\mathcal{A}_{VC}$ will output a challenge computation o. $\mathcal{A}_{SE}$ will send $\lambda^C(o)$ to $\mathcal{C}$ as the challenge node $v^\star$ for the S-KI game. Note that this is a valid challenge in the S-KI game since, in the Authorized Outsourcing game, $\mathcal{A}_{VC}$ may not choose a challenge computation o for which he is authorized – that is, any o such that $\lambda^C(o) \leqslant \lambda^C(v_i)$ for any $\lambda^C(v_i)$ queried to the $\mathsf{Register}$ oracle. This corresponds precisely to the condition on the choice of $v^\star$ in the S-KI game. $\mathcal{C}$ will return a key $\kappa^\star$ which corresponds either to the real key $\kappa_{\lambda^C(o)}$ or a random key from the keyspace chosen according to the bit b chosen at the beginning of the game.

6. $\mathcal{A}_{SE}$ must now simulate a server and a verifier. It does this by first registering two such entities, S and V respectively, that are authorized to compute and verify o respectively. It sets $\lambda^C(S) = \lambda^C(o)$ and hence $\kappa_{\lambda^C(S)} = \kappa^\star$. Note that this is the only part of SK_S that is required, and in particular $\omega_{\lambda^C(S)}$ is not required (as this is only needed for deriving keys, and S will only need to use $\kappa^\star$). To register S, $\mathcal{A}_{SE}$ runs $\mathsf{Register}$ and sets $SK_S = (SK'_S, \kappa^\star, \perp)$. Furthermore, to register V, $\mathcal{A}_{SE}$ sets $SK_V = (\kappa_{\lambda^V(V)}, \omega_{\lambda^V(V)})$.

7. $\mathcal{A}_{SE}$ then runs lines 12 and 13 as written in the Authorized Outsourcing game. It can also run the $\mathsf{Compute}$ algorithm as it knows $\kappa_{\lambda^C(o)}$ from $\kappa_{\lambda^C(C)}$, and Verify as it owns the verification KAS. $\mathcal{A}_{SE}$ can therefore return the result of the game to $\mathcal{A}_{VC}$ as expected.

Now, $\mathcal{A}_{VC}$ has been provided all information that an adversary against the Authorized Outsourcing game would be given and will guess which game he is playing with advantage δ. Notice that the distribution of the game generated by $\mathcal{A}_{SE}$ is precisely that of **Game 0** if $b = 0$ (i.e. the real KAS key is used), and is precisely that of **Game 1** otherwise, if a random key was chosen. Thus, $\mathcal{A}_{SE}$ can simply forward $\mathcal{A}_{VC}$'s guess to $\mathcal{C}$ as its guess in the S-KI game. Thus we conclude that if $\mathcal{A}_{VC}$ guesses correctly with non-negligible advantage δ, then $\mathcal{A}_{SE}$ can break the Strong-Key Indistinguishability of the KAS also with non-negligible advantage δ. Since we assumed the KAS was S-KI secure, such an adversary may not exist and hence **Game 0** is indistinguishable from **Game 1** except with at most a negligible advantage $\epsilon \leqslant 1 - \delta$.

Reduction to INT-PTXT We have shown that, from the adversary's point of view, **Game 1** is almost (with negligible distinguishing advantage) identical to **Game 0**. Thus, we may run the adversary against **Game 1** instead with at most an ϵ loss in the tightness of the reduction. In essence, we have now removed any information leakage from the KAS. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ in **Game 1**. We now show that we can construct an adversary $\mathcal{A}_{SE}$ that breaks the INT-PTXT security of the authenticated symmetric encryption scheme $\mathcal{SE}$ using $\mathcal{A}_{VC}$ as a subroutine. Let $\mathcal{C}$ be the INT-PTXT challenger for $\mathcal{A}_{SE}$ who in turn acts as the challenger in **Game 1** for $\mathcal{A}_{VC}$.

1. $\mathcal{C}$ begins by initializing the list L and running $\mathsf{SE.KeyGen}(1^\kappa)$ to generate a key $\kappa^\star$. It sends the security parameter 1^κ to $\mathcal{A}_{SE}$.

2. $\mathcal{A}_{SE}$ must now initialize Game 1 for $\mathcal{A}_{VC}$. Informally, it will set the KAS key for the label $\lambda^C(o)$ to be the random key $\kappa^\star$ chosen by $\mathcal{C}$. However, the challenge label is unknown until $\mathcal{A}_{VC}$ chooses it, whilst the public parameters and oracle access must be provided before this choice. Thus, we require $\mathcal{A}_{SE}$ to guess the challenge label during Setup so that the correct key can be implicitly set to be the INT-PTXT challenge key (and all encryptions under that key can be formed using oracle access to $\mathcal{C}$). If the number of labels in the poset is N, where N is polynomial in the security parameter (as the scheme must be efficiently instantiable), then $\mathcal{A}_{SE}$ may guess $\lambda^C(o)$ with probability at least $\frac{1}{N}$. Assuming that the guess is correct, we proceed as follows.

3. $\mathcal{A}_{SE}$ runs $\mathsf{PVC_{AC}.Setup}$ as given in Algorithm 1 with the modification that the key for the guessed label in the computation poset, $\kappa_{\lambda^C(o)}$ is implicitly set to be the key generated by $\mathcal{C}$ in the IND-CPA game and the KAS is made to be consistent with this choice. Of course, $\mathcal{A}_{SE}$ does not have this key but will ensure that any operations using it will be performed using its oracles to $\mathcal{C}$. Since the challenge key, and hence any corresponding secret derivation information, is unknown, it is not trivial to construct a KAS incorporating this key. However, notice that the Authorized Outsourcing game (and by extension **Game 1**) does not permit the adversary to query any label that is an ancestor of the challenge label

in the computation poset. Thus, KAS keys for the set of ancestors will not be needed, and a KAS can simply be instantiated over the remaining nodes (and the public information for the ancestor set simulated, as the keys cannot be derived, the public information need not be functionally correct). Remaining keys can simply be generated using the security parameter. From the adversarial point of view, this will be indistinguishable from the real games.

4. $\mathcal{A}_{VC}$ is given the generated public parameters and oracle access which $\mathcal{A}_{SE}$ may respond to as follows:

   - FnInit, Certify, and Revoke queries can be handled by simply calling the relevant algorithm as these have no dependence on the KAS.
   - If a Register query is made for a label $\lambda(ID) \geqslant \lambda^C(o)$ as guessed by $\mathcal{A}_{SE}$ then $\mathcal{A}_{SE}$ aborts the game since $\mathcal{A}_{VC}$ would not then be able to choose $\lambda^C(o)$ as its challenge computation, and hence $\mathcal{A}_{SE}$'s guess was incorrect. Otherwise, $\mathcal{A}_{SE}$ holds the relevant KAS keys and may respond by running as specified in Oracle Query 7.
   - By the oracle restrictions specified for the Authorized Outsourcing game, $\mathcal{A}_{VC}$ may not query the ProbGen oracle on the challenge computation $\lambda^C(o)$. Hence, if such a query is made at this point, then $\lambda^C(o)$ would not be chosen as the challenge computation and $\mathcal{A}_{SE}$'s guess would be incorrect, and so the game is aborted. For any other choice of computation queried to the ProbGen oracle, $\mathcal{A}_{SE}$ holds the KAS key (or generated symmetric key) and may run Algorithm 5 as written.
   - Observe that if a query is made to the Compute oracle for the challenge computation, then the adversary has either submitted a malformed encoded input, and $\perp$ should be returned, or the adversary has submitted a correctly formed encoded input, and hence has already won the game. For all other queries, $\mathcal{A}_{SE}$ can use one of the keys it holds to respond to the challenge by running Algorithm 6.

5. $\mathcal{A}_{VC}$ eventually outputs a choice of label which $\mathcal{A}_{SE}$ can register him for, as in the oracle query above, and $\mathcal{A}_{VC}$ is again given oracle access which is handled as before. It eventually outputs a challenge computation, and if this is not the computation chosen by $\mathcal{A}_{SE}$ at the beginning of the game, then the game is aborted. Otherwise, the choice of computation label is valid since the game has not already been aborted during the oracle queries.

6. $\mathcal{A}_{SE}$ should now register two entities: a computational server S and a verifier V. However, these will not be required in the following, so $\mathcal{A}_{SE}$ can simulate registering them with labels $\lambda^C(o)$ and $\lambda^V(o)$ respectively and update the public parameters accordingly without actually requiring the correct keys for these entities (as the adversary will not see any output from these entities other than their presence in the lists in the public parameters). $\mathcal{A}_{SE}$ can also run the FnInit and Certify algorithms as written.

7. $\mathcal{A}_{VC}$ is then provided with all relevant information and is given oracle access again. Queries can be handled as above, but Register queries now return $\perp$ if the queried label is an ancestor of the computation label in the poset i.e. exactly when $\mathcal{A}_{SE}$ does not hold a KAS key for the queried label.

8. $\mathcal{A}_{VC}$ finally outputs a forged encoded input σ_o.

Now, observe that from the point of view of $\mathcal{A}_{VC}$, the game has been simulated correctly up to this point and that $\mathcal{A}_{SE}$ has not made any queries to its Encrypt oracle in the INT-PTXT game. Thus, if the ciphertext c within σ_o decrypts successfully (which it must for $\mathcal{A}_{VC}$ to

win the Authorized Outsourcing game), then the verification oracle Ver(c) will output 1 *and* the decrypted message has certainly not been queried to the Encrypt oracle, and thus $\mathcal{A}_{SE}$ can forward c as its answer in the INT-PTXT game and win with exactly the advantage of $\mathcal{A}_{VC}$ in the Authorized Outsourcing game. This is assuming that $\mathcal{A}_{SE}$ correctly guessed the challenge computation label so that the game did not abort. Thus, for a poset of polynomial size N, the advantage of $\mathcal{A}_{SE}$ is $\frac{\delta}{N}$ which is non-negligible if $\mathcal{A}_{VC}$ has non-negligible advantage δ in the Authorized Outsourcing game. However, since we assumed the authenticated symmetric encryption scheme to be secure, such an adversary with non-negligible advantage against the Authorized Outsourcing game cannot exist.

Thus, we conclude, the overall advantage against the Authorized Outsourcing game is the sum of the distinguishing advantage between **Game 0** and **Game 1**, and the advantage in the reduction to INT-PTXT, both of which we have shown to be negligible. Therefore, the overall advantage against the Authorized Outsourcing game is negligible.

□

We remark that in the above proof we required $\mathcal{A}_{SE}$ to correctly guess the label that $\mathcal{A}_{VC}$ would select ahead of time. This is very similar to the notions of security commonly found in functional encryption primitives, particularly Identity-based Encryption [8] and Attribute-based Encryption [18, 6]. In these settings, it is common to refer to the method of guessing the correct label as *complexity leveraging* which results in a polynomial loss in the tightness of the reduction. An alternative, which could equally be taken here, is to formulate a weaker *selective* notion of security in which the adversary must select the challenge label at the beginning of the game before seeing the public parameters.

We also note that the restriction on queries to the ProbGen oracle (i.e. that the challenge computation cannot be queried) is stronger than required for our particular construction due to the Encrypt oracle available in the INT-PTXT game. An alternative would be to allow queries to ProbGen for the challenge computation but require that the final adversarial output is distinct from those given in response to these queries.

Lemma 2. *Given a secure RPVC scheme, an authenticated symmetric encryption scheme secure in the sense of* IND-CPA∧INT-PTXT *and a KAS secure against Strong Key-Indistinguishability, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 1–9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of Authorized Computation (Game 6).*

Proof. Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage δ in the Authorized Computation game and let $\mathcal{A}_{SE}$ be an adversary playing the IND-CPA game with a challenger $\mathcal{C}$ against $\mathcal{SE}$. We define the following sequence of games and show that we can perform a sequence of game hops with negligible difference between each successive pairs of games, and thereby construct an adversary $\mathcal{A}_{SE}$ that, using $\mathcal{A}_{VC}$ as a subroutine, can break the IND-CPA property with non-negligible advantage.

- **Game 0.** This is the Authorized Computation game as defined in Section 5.2.
- **Game 1.** This is identical to **Game 0**, except that we replace the key $\kappa_{\lambda^C(o)}$ for the challenge computation o with a random key $\kappa^\star$ drawn uniformly from the keyspace.
- **Game 2.** This is the same as **Game 1** with the modification that, in ProbGen, the encoded input is generated by either encrypting the proper encoded input from the RPVC functionality σ'_o, or a random message of the same length as σ'_o.

Game 0 to Game 1 This game hop relies on the Strong Key-Indistinguishability of the KAS and proceeds very similarly to the corresponding hop in the proof of Lemma 1. As such, we do not state the full proof here but leave it as an easy exercise. We conclude that if $\mathcal{A}$ distinguishes **Game 0** from **Game 1** with non-negligible advantage δ, then an adversary can use $\mathcal{A}$ as a subroutine to break the Strong Key-Indistinguishability of the KAS also with non-negligible advantage δ. Since the KAS is assumed S-KI secure, such an adversary may not exist and hence **Game 0** is indistinguishable from **Game 1** except with at most a negligible advantage $\epsilon \leqslant 1 - \delta$.

Game 1 to Game 2 We have shown that, from the adversary's point of view, **Game 1** is almost (with negligible distinguishing advantage) identical to **Game 0**. Thus, we may run the adversary against **Game 1** instead to remove any information leakage from the KAS. We now show that an adversary cannot distinguish **Game 1** from **Game 2** with more than a negligible probability, which then removes any information leakage from the encrypted, encoded input. Suppose an adversary $\mathcal{A}_{VC}$ exists that can distinguish **Game 1** from **Game 2** with non-negligible advantage δ. Then there exists an adversary $\mathcal{A}_{SE}$ that breaks the IND-CPA security of the symmetric encryption scheme $\mathcal{SE}$ also with advantage δ. $\mathcal{A}_{VC}$ will play either **Game 1** or **Game 2** with $\mathcal{A}_{SE}$ acting as the challenger, and must guess correctly which game he is playing. $\mathcal{A}_{SE}$ in turn will play the IND-CPA game with a challenger $\mathcal{C}$. This reduction removes any information about the encoded input that may leak through the encryption process.

1. $\mathcal{C}$ begins by choosing a random bit b (which ultimately will determine which of **Game 1** and **Game 2** is being played) and running $\mathsf{SE.KeyGen}(1^\kappa)$ to generate a key $\kappa^\star$. It sends the security parameter 1^κ to $\mathcal{A}_{SE}$.

2. $\mathcal{A}_{SE}$ must now initialize **Game b+1** for $\mathcal{A}_{VC}$. Informally, it will set the KAS key for the label $\lambda^C(o)$ to be the random key $\kappa^\star$ chosen by $\mathcal{C}$. However, the challenge label is unknown until $\mathcal{A}_{VC}$ chooses it, whilst the public parameters and oracle access must be provided before this choice. Thus, we require $\mathcal{A}_{SE}$ to guess the challenge label during Setup so that the correct key can be implicitly set to be the IND-CPA challenge key (and all encryptions under that key can be formed using $\mathcal{C}$). If the number of labels in the poset is N, where N is polynomial in the security parameter (as the scheme must be efficiently instantiable), then $\mathcal{A}_{SE}$ may guess $\lambda^C(o)$ with probability at least $\frac{1}{N}$. Assuming that the guess is correct, we proceed as follows.

3. $\mathcal{A}_{SE}$ runs $\mathsf{PVC_{AC}.Setup}$ as given in Algorithm 1 except that the key for the guessed label in the computation poset, $\kappa_{\lambda^C(o)}$ is implicitly set to be $\kappa^\star$ and the KAS is made consistent with this choice. Note that the Authorized Computation game (and by extension **Game b+1**) does not permit the adversary to query any label that is an ancestor of the challenge label in the computation poset. Thus a KAS can be instantiated over the remaining nodes (and the public information for the ancestor set simulated – as the keys cannot be derived, the public information need not be functionally correct). Remaining keys can simply be generated using the security parameter. From the adversarial point of view, this will be indistinguishable from the real games.

4. $\mathcal{A}_{VC}$ is given the generated public parameters and oracle access which $\mathcal{A}_{SE}$ responds to as follows:

    - FnInit, $\mathsf{Certify}$, and Revoke queries can be handled by simply calling the relevant algorithm.

- If a Register query is made for a label $\lambda(ID) \geqslant \lambda^C(o)$ then $\mathcal{A}_{SE}$ aborts the game since $\mathcal{A}_{VC}$ would not then be able to choose $\lambda^C(o)$ as its challenge computation, and hence $\mathcal{A}_{SE}$'s guess was incorrect. Otherwise, $\mathcal{A}_{SE}$ holds the relevant KAS keys and may respond by running Oracle Query 7.
- ProbGen queries for a computation labelled $\lambda(o)$ can be handled by running Algorithm 5 with the exception that lines 2 and 5 are simulated as follows. Line 2 is not run at all as $\mathcal{A}_{SE}$ may not hold the challenge key, and line 5 is run using an oracle query to the IND-CPA LoR oracle for the choice of messages $m_0 = m_1 = \sigma'_o$. For any other queried computation, $\mathcal{A}_{SE}$ holds the KAS key (or generated symmetric key) and may run Algorithm 5 as written.
- Observe that by the INT-PTXT property of the authenticated symmetric encryption scheme, $\mathcal{A}_{VC}$ cannot form a valid ciphertext (one that will decrypt to anything other than $\perp$) without holding the encryption key. The encryption keys that $\mathcal{A}_{VC}$ may hold are precisely those revealed through Register queries, and since queries may not be made for labels $\lambda(ID) \geqslant \lambda^C(o)$, each query to the Compute oracle will be for a label belonging to the KAS instantiated by $\mathcal{A}_{SE}$. Hence, if $\lambda(ID) \geqslant \lambda^C(o)$, $\mathcal{A}_{SE}$ returns $\perp$ and otherwise it uses its knowledge of the KAS to respond genuinely by running Algorithm 6.

5. $\mathcal{A}_{VC}$ eventually outputs a choice of label which $\mathcal{A}_{SE}$ can register him for in the same manner as in the oracle query above, and $\mathcal{A}_{VC}$ is again given oracle access which is handled as before. It eventually outputs a challenge computation, and if this is not the computation chosen by $\mathcal{A}_{SE}$ in Step 2, then the game is aborted. Otherwise, the choice of computation label is valid since the game has not already been aborted during the oracle queries.

6. $\mathcal{A}_{SE}$ should now register two entities: a delegator C and a verifier V. However, these will not be required in the following, so $\mathcal{A}_{SE}$ may simulate registering them with labels $\lambda^C(o)$ and $\lambda^V(o)$ respectively and update the public parameters accordingly without actually requiring the correct keys for these entities (as the adversary will not see any output from these entities other than their presence in the lists in the public parameters). $\mathcal{A}_{SE}$ can also run the FnInit and Certify algorithms as written.

7. $\mathcal{A}_{SE}$ must now run ProbGen for the challenge computation o to generate an encoded input σ_o. To do so, it will make use of the LoR oracle provided by $\mathcal{C}$ in the IND-CPA game. $\mathcal{A}_{SE}$ will run lines 1, 3 and 4 as written to generate a problem encoding σ'_o. It sets $m_0 = \sigma'_o$ and chooses another message m_1 of the same length uniformly at random from the message space. These are queried to the LoR oracle which will return the encryption of message m_b for the challenger's choice of b from Step 1. $\mathcal{A}_{SE}$ can then use this response to form σ_o and form RK_o using the verification KAS key which it holds.

8. All relevant information is passed to $\mathcal{A}_{VC}$ who is also given oracle access. This is again handled in the same manner as before. Eventually, $\mathcal{A}_{VC}$ will output a guess b' reflecting that it believes it is playing **Game b′ + 1**.

Observe that if the challenger's random bit $b = 0$, then this is precisely **Game 1** (since the real encoded input was encrypted by $\mathcal{C}$). If, on the other hand, $b = 1$ then $\mathcal{A}_{VC}$ is provided with the encryption of a random message which does not relate to the computation at all, which is precisely the setting of **Game 2**. Now, we assumed that $\mathcal{A}_{VC}$ could distinguish **Game 1** from **Game 2** with non-negligible probability δ. Since these two games correspond

directly to the challenger's choice of bit b, $\mathcal{A}_{SE}$ can simply forward $\mathcal{A}_{VC}$'s guess b' to $\mathcal{C}$ and win the IND-CPA game with non-negligible advantage δ. However, as we assumed that $\mathcal{SE}$ was IND-CPA $\wedge$ INT-PTXT secure, this is a contradiction and hence an adversary with non-negligible distinguishing advantage cannot exist.

Reduction to Public Verifiability We have shown negligible distinguishing advantages in the transitions from the Authorized Computation game to **Game 2**. Thus we may run an adversary against **Game 2** instead. By moving to this modified game, we have removed any information leakage from the KAS and from the ciphertext encrypting the encoded input. Thus the only information that remains that could aid an adversary in the Authorized Computation game is the verification key and the output retrieval key. Intuitively however, the underlying RPVC scheme was designed such that these components *do not* leak information that aids the adversary in producing a fraudulent result. We now show that, since the adversary is not authorized for the challenge computation, even if it holds a valid evaluation key it cannot produce a valid response. If it could do so, then an adversary could be constructed to break the Public Verifiability of the RPVC scheme.

Let $\mathcal{A}_{VC}$ be an adversary with non-negligible advantage δ against **Game 2**. Then we construct an adversary $\mathcal{A}_{PV}$ that breaks the Public Verifiability of the RPVC scheme with advantage $\frac{\delta}{2}$. Let $\mathcal{C}$ play the Public Verifiability game with $\mathcal{A}_{PV}$ who in turn acts as the challenger for $\mathcal{A}_{VC}$ in **Game 2**.

1. $\mathcal{C}$ begins by choosing a bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, it instantiates $\mathcal{A}_{PV}$ for the function $F^\star = F$. Otherwise, $\mathcal{A}_{PV}$ is instantiated for $F^\star = \overline{F}$.

2. $\mathcal{C}$ also runs $\mathsf{RPVC.Setup}$ on the security parameter and $\mathsf{RPVC.FnInit}$ for the function $F^\star$. It sends the resulting parameters PP' and $PK_{F^\star}$ to $\mathcal{A}_{PV}$.

3. $\mathcal{A}_{PV}$ initialises the list L and the variable o and must then simulate running the $\mathsf{PVC_{AC}.Setup}$ algorithm. It implicitly sets MK' to be that generated by $\mathcal{C}$, and sets up the KASs in the same manner as in the prior transition from **Game 1** to **Game 2** by guessing the correct challenge label with probability at least $\frac{1}{N}$.

4. $\mathcal{A}_{PV}$ sends $\mathcal{A}_{VC}$ the public parameters PP and provides oracle access as follows:

   - Queries to the FnInit, $\mathsf{Certify}$ and Revoke oracles can be forwarded to $\mathcal{C}$ and the response returned to $\mathcal{A}_{VC}$.
   - Queries to the $\mathsf{Register}$ oracle can be run as in Oracle Query 7 with the exception of line 2 of the $\mathsf{Register}$ algorithm for which $\mathcal{A}_{PV}$ makes a $\mathsf{RPVC.Register}$ oracle query for the queried identity.
   - Queries to the $\mathsf{ProbGen}$ oracle can be handled simply by running Algorithm 5. Note that $\mathcal{A}_{PV}$ owns all information related to SK_C for all delegators, and the $\mathsf{RPVC.ProbGen}$ algorithm only requires public information.
   - Queries to the $\mathsf{Compute}$ oracle can be run as written in Algorithm 6.

   Eventually $\mathcal{A}_{VC}$ will output its choice of computation label $\lambda^C(\mathcal{A}_{VC})$.

5. $\mathcal{A}_{PV}$ now registers $\mathcal{A}_{VC}$ for this label. To do so, it can use its knowledge of the KASs it instantiated, except for line 2 where it instead makes a $\mathsf{RPVC.Register}$ oracle query for $ID = \mathcal{A}_{VC}$.

6. $\mathcal{A}_{PV}$ updates the list L with the chosen label and gives $\mathcal{A}_{VC}$ the secret key. $\mathcal{A}_{VC}$ is also given oracle access which is handled by $\mathcal{A}_{PV}$ as above.

7. Eventually, $\mathcal{A}_{VC}$ outputs a choice of challenge computation o, including the input data x. If this is not the same as the guess made by $\mathcal{A}_{PV}$ in Step 3 then the game is aborted. Otherwise, $\mathcal{A}_{PV}$ checks that the label of this computation is valid in accordance with the queries made above. If so, it forwards x to $\mathcal{C}$ as the challenge input $x^\star$ in the Public Verifiability game.

8. $\mathcal{C}$ runs RPVC.ProbGen on $x^\star$ and gives the resulting parameters to $\mathcal{A}_{PV}$ and again provides oracle access.

9. $\mathcal{A}_{PV}$ must now register a delegator and a verifier to act in the following stages. To do so, it simply runs Algorithm 3 since it knows all required information from the KASs it instantiated. It simulates running FnInit either using the $PK_{F^\star}$ provided by $\mathcal{C}$ in Step 2 (if $F^\star = F$) or by querying the RPVC.FnInit oracle, and makes a query to the Certify oracle provided by $\mathcal{C}$ for the identity $\mathcal{A}_{VC}$ and function F. Finally, $\mathcal{A}_{PV}$ runs ProbGen as given in Algorithm 5 with the exception of choosing a random message instead of the encoded input, as per **Game 2**. All generated parameters are passed to $\mathcal{A}_{VC}$, and oracle access is provided as before.

10. Eventually, $\mathcal{A}_{VC}$ finishes querying and outputs θ_o which it believes will appear to be a valid result despite being unauthorized. $\mathcal{A}_{PV}$ forwards θ_o to $\mathcal{C}$ as its guess in the Public Verifiability game.

Now, in order for $\mathcal{A}_{PV}$ to win the Public Verifiability game, it must produce a valid output for the unsatisfied function F or $\overline{F}$ on input $x^\star$. Assuming that $\mathcal{A}_{VC}$ is a successful adversary against **Game 2**, θ_o is a valid result for $F(x)$ with non-negligible probability δ. If $F(x) = b$, where b is the random bit chosen by $\mathcal{C}$, then $\mathcal{A}_{PV}$ is instantiated on the unsatisfied function. Thus, if the game is not aborted (i.e. $\mathcal{A}_{PV}$ guessed the challenge computation label correctly), $\mathcal{A}_{PV}$ wins with probability $\Pr[b = F(x) \wedge \mathbf{Exp}^{\mathbf{Game\ 2}}_{\mathcal{A}_{VC}}[\mathcal{PVC}_{\mathrm{AC}}, F, 1^\kappa] = 1] = \frac{\delta}{2}$. Thus the overall probability of $\mathcal{A}_{PV}$ winning, including guessing the computation correctly is at least $\frac{\delta}{2N}$ which is non-negligible. However, since we assumed the underlying RPVC scheme to be secure in the sense of Public Verifiability, such an adversary $\mathcal{A}_{VC}$ with non-negligible advantage in **Game 2** cannot exist.

Finally, we observe that each transition to **Game 2** had a negligible distinguishing advantage and the final reduction showed a negligible advantage against **Game 2**, and so we conclude that the scheme is secure in the sense of Authorized Computation.

□

We note that the final reduction in this proof could achieve a tighter bound if we consider the actual mechanism used to achieve Public Verifiability in the underlying RPVC scheme (e.g. the application of a one-way function [20, 2]), but this would not be in a black-box manner.

Lemma 3. *Given a secure RPVC scheme, a KAS secure in the sense of Strong-Key Indistinguishability and an authenticated symmetric encryption scheme secure in the sense of* IND-CPA ∧ INT-PTXT*, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 1–9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of Authorized Verification (Game 7).*

Proof. Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ in the Authorized Verification game and $\mathcal{A}_{SE}$ is an adversary playing the IND-CPA game with a challenger $\mathcal{C}$ against $\mathcal{SE}$. We first define the following modified game and show that an adversary has negligible

distinguishing advantage between the two. We can therefore employ an adversary against this modified game to break the IND-CPA security of the symmetric encryption scheme.

- **Game 0.** This is the Authorized Verification game as defined in Section 5.3.
- **Game 1.** This is identical to **Game 0**, except that we replace the key $\kappa_{\lambda^V(o)}$ for the challenge computation o in the verification poset with a random key $\kappa^\star$ drawn uniformly from the keyspace.

Game 0 to Game 1 This transition relies on the Strong Key-Indistinguishability of the KAS. The proof is very similar to that in the proof of Lemma 1, and so we leave this as an easy exercise. If $\mathcal{A}$ can distinguish **Game 0** from **Game 1** with non-negligible advantage δ, then an adversary using $\mathcal{A}$ as a subroutine can break the Strong Key-Indistinguishability of the KAS with the same non-negligible advantage δ. However, as the KAS is assumed S-KI secure, such an adversary may not exist and **Game 0** is indistinguishable from **Game 1** except with at most a negligible advantage $\epsilon \leqslant 1 - \delta$.

Reduction to IND-CPA Suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage against the Authorized Verification game. We now show that we can construct an adversary $\mathcal{A}_{SE}$ that breaks the IND-CPA security of the symmetric encryption scheme $\mathcal{SE}$ using $\mathcal{A}_{VC}$ as a subroutine. From the adversary's point of view, **Game 1** is indistinguishable from **Game 0** with negligible distinguishing advantage. Therefore, $\mathcal{A}_{SE}$ may simulate **Game 1** instead to remove any information leakage from the KAS. Let $\mathcal{C}$ be the IND-CPA challenger for $\mathcal{A}_{SE}$ who in turn acts as the challenger in **Game 1** for $\mathcal{A}_{VC}$ who succeeds with non-negligible probability δ.

1. $\mathcal{C}$ first chooses a bit β uniformly at random and generates a key $\kappa^\star \leftarrow \mathsf{SE.KeyGen}(1^\kappa)$. It sends the security parameter 1^κ to $\mathcal{A}_{SE}$.
2. $\mathcal{A}_{SE}$ must now initialize **Game 1** for $\mathcal{A}_{VC}$. It will implicitly set the KAS key for the label $\lambda^V(o)$ to be $\kappa^\star$. However, as $\mathcal{A}_{VC}$ has not yet chosen his challenge labels, $\mathcal{A}_{SE}$ must guess one of these labels in order to generate the public parameters and provide oracle access. $\mathcal{A}_{SE}$ may guess $\lambda^V(o)$ with probability at least $\frac{2}{N}$ (as $\mathcal{A}_{VC}$ will select two labels, and $\mathcal{A}_{SE}$ must match one of them) for a poset of size N nodes. Assuming that the guess is correct, we proceed as follows.
3. $\mathcal{A}_{SE}$ runs $\mathsf{PVC_{AC}.Setup}$ as given in Algorithm 1 except that the key for the guessed label in the verification poset, $\kappa_{\lambda^V(o)}$ is implicitly set to be $\kappa^\star$ and the KAS is made consistent with this choice. Note that in the Authorized Verification game (and by extension **Game 1**) the adversary may not query any label that is an ancestor of the challenge label $\lambda^V(o_b)$ in the verification poset. Thus $\mathcal{A}_{SE}$ can instantiate a KAS over the remaining nodes and simulate the public information for the ancestor set – as keys for this set cannot be derived, the public information need not be functionally correct. $\mathcal{A}_{SE}$ can also use the security parameter to generate remaining keys in this set. From the adversarial point of view, this is indistinguishable from the real games.
4. $\mathcal{A}_{SE}$ sends the resulting public parameters to $\mathcal{A}_{VC}$ and provides oracle access as follows:
   - For queries to FnInit, $\mathsf{Certify}$, and Revoke $\mathcal{A}_{SE}$ can simply call the relevant algorithm.
   - If a $\mathsf{Register}$ query is made for a label $\lambda(ID) \geqslant \lambda^V(o)$ then $\mathcal{A}_{SE}$ aborts the game as $\mathcal{A}_{VC}$ will not be able to choose $\lambda^V(o)$ as one of its challenge computations, and hence $\mathcal{A}_{SE}$'s guess was incorrect. Otherwise, $\mathcal{A}_{SE}$ holds the relevant KAS keys and may respond by running Oracle Query 8.

- A ProbGen query for a computation labelled $\lambda^V(o)$ can be handled by running Algorithm 5 with the exception of lines 3 and 6 which are simulated as follows. $\mathcal{A}_{SE}$ makes an oracle query to the IND-CPA LoR oracle for the message pair $m_0 = m_1 = RK'_o$. For all other queries, $\mathcal{A}_{SE}$ holds the correct keys and may honestly run Algorithm 5.
- Compute queries can be handled by running Algorithm 6 since it relies only on the computation poset which is owned by $\mathcal{A}_{SE}$.

5. $\mathcal{A}_{VC}$ eventually outputs a choice of label which $\mathcal{A}_{SE}$ can register him for as per the Register oracle query above, and $\mathcal{A}_{VC}$ is again given oracle access which is handled as above. Eventually it chooses two challenge computations o_0 and o_1 and, if neither is the same as that chosen earlier by $\mathcal{A}_{SE}$, the game is aborted. Otherwise, the choices are valid since the game has not already been aborted during the oracle queries. Instead of choosing the bit b at random, it can be set such that o_b corresponds to $\mathcal{A}_{SE}$'s guess of challenge computation.

6. $\mathcal{A}_{SE}$ now simulates registering two entities: a delegator C and a server S. However, as these will not be required in the following, $\mathcal{A}_{SE}$ may simulate registering the delegator with label $\lambda^C(o_b)$ and update the public parameters accordingly without requiring valid keys for the delegator (as the adversary will not see any output from the delegator other than being listed in the public parameters). For the server, $\mathcal{A}_{SE}$ runs Register and sets $SK_S = (SK'_S, \kappa^\star, \perp)$ $\mathcal{A}_{SE}$ can also run the FnInit and Certify algorithms as written.

7. $\mathcal{A}_{SE}$ now runs ProbGen for the challenge computation o_b to encode σ_{o_b}. It runs lines 1, 2,4 and 5 as written to generate a problem encoding σ_o and retrieval key RK'_o. It sets $m_0 = RK'_o$ and randomly chooses another message m_1 of the same length from the message space. These are given as input to the LoR oracle which returns the encryption of message m_β for the challenger's random choice of β. Using this response, $\mathcal{A}_{SE}$ can create RK_o.

8. $\mathcal{A}_{SE}$ must finally run Compute on the resulting encoded input, which it can do honestly as the algorithm does not rely on the verification poset. $\mathcal{A}_{VC}$ then receives all relevant information and oracle access, which is again handled as before. Eventually, $\mathcal{A}_{VC}$ outputs a guess b' of b.

Observe that if $\mathcal{A}_{SE}$ guessed correctly and $\beta = 0$, then this is precisely **Game 1**, and $\mathcal{A}_{VC}$ wins with non-negligible probability δ. Thus $\mathcal{A}_{SE}$ wins with non-negligible probability $\frac{2\delta}{N}$. If $\beta = 1$, on the other hand, the encoded input provided to $\mathcal{A}_{VC}$ is the encryption of a random message completely unrelated to the retrieval key. In this case, by the Blind Verification property of the underlying RPVC construction, $\mathcal{A}_{VC}$ may only have a negligible advantage ϵ (over random guessing) at learning the result of the computation. Thus, if $\mathcal{A}_{VC}$ outputs $b' = b$, then $\mathcal{A}_{SE}$ should output a guess of $\beta' = 0$, and otherwise should guess $\beta' = 1$. If $\beta = 0$ then $\mathcal{A}_{SE}$ wins with probability $\frac{2\delta}{N}$, and if $\beta = 1$ then $\mathcal{A}_{SE}$ wins with probability $1 - \epsilon$. Thus, $\mathcal{A}_{SE}$ wins in both cases with non-negligible probability.

The overall advantage against the Authorized Verification game is the sum of the distinguishing advantage between **Game 0** and **Game 1**, and the advantage in the reduction to IND-CPA, both of which we have shown to be negligible. Therefore, the overall advantage against the Authorized Verification game is negligible.

□

Lemma 4. *Given a secure RPVC scheme, a KAS secure in the sense of Strong-Key Indistinguishability and an* IND-CPA *secure symmetric encryption scheme, let $\mathcal{PVC}_{AC}$ be the PVC-AC scheme defined in Algorithms 1–9. Then $\mathcal{PVC}_{AC}$ is secure in the sense of Weak Input Privacy (Game 8).*

Proof. Assume that $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ in the Weak Input Privacy game. We show that we can use this to construct an adversary, $\mathcal{A}_{SE}$, with non-negligible advantage in the IND-CPA game. Let $\mathcal{C}$ be the challenger for $\mathcal{A}_{SE}$ and let $\mathcal{A}_{SE}$ act as the challenger for $\mathcal{A}_{VC}$. We first transition to a modified version of the Weak Input Privacy game, and show that $\mathcal{A}_{VC}$ has negligible distinguishing advantage between these games. We then construct $\mathcal{A}_{SE}$ which uses $\mathcal{A}_{VC}$ against this modified game to break the IND-CPA security of the symmetric encryption scheme.

- **Game 0:** This is the Weak Input Privacy game as given in Game 8.
- **Game 1:** This is identical to **Game 0** except that the key $\kappa_{\lambda^C(o_b)}$ for the challenge computation o_b is replaced by a random key $\kappa^\star$ drawn uniformly from the keyspace.

Game 0 to Game 1 This game hop relies on the Strong-Key Indistinguishability of the KAS. Suppose an adversary $\mathcal{A}_{VC}$ exists that can distinguish **Game 0** from **Game 1** with non-negligible advantage ϵ. Then there exists an adversary $\mathcal{A}_{KI}$ that breaks the S-KI security of the KAS also with advantage ϵ. A challenger $\mathcal{C}$ will choose either **Game 0** or **Game 1** and interact with $\mathcal{A}_{KI}$ in the S-KI game; $\mathcal{A}_{KI}$ will act as the challenger in either **Game 0** or **Game 1** for $\mathcal{A}_{VC}$ who must guess which game he is playing.

1. $\mathcal{C}$ chooses a bit $b \xleftarrow{\$} \{0,1\}$ which will dictate whether the challenge key is real or random, and hence ultimately that $\mathcal{A}_{VC}$ will be playing **Game b**. $\mathcal{C}$ instantiates the KAS for the S-KI game by running lines 1 to 5 of Game 1 and gives the public information to $\mathcal{A}_{KI}$.
2. $\mathcal{A}_{KI}$ must initialize the wIP game for $\mathcal{A}_{VC}$. To do so, it initializes the variables L and o and then simulates running Setup as written with the exception of lines 2 to 4. Instead, it sets $Pub_{\mathcal{P}_C}$ to be the public information generated by the challenger, and will make use of oracle queries to $\mathcal{C}$ for the remaining KAS secrets and keys.
3. $\mathcal{A}_{VC}$ is given the public parameters and oracle access. To respond to queries to the FnInit, $\mathsf{Certify}$ and Revoke functions, $\mathcal{A}_{VC}$ can simply run the relevant algorithm, as these do not rely on the KAS. Queries to other functions may require knowledge of KAS derived keys. Now, $\mathcal{A}_{VC}$ owns the verification KAS information but only holds public information for the computational poset. Thus, it will issue $\mathsf{Corrupt}$ queries for relevant labels to $\mathcal{C}$ for the computational poset (which is the challenge poset in the S-KI game). Since neither $\mathcal{A}_{VC}$ or $\mathcal{A}_{KI}$ have chosen their respective challenges, the $\mathsf{Corrupt}$ oracle will always return a valid result which $\mathcal{A}_{KI}$ can use to form a valid response.
4. Eventually, $\mathcal{A}_{VC}$ will choose a security label $\lambda^C(\mathcal{A}_{VC})$ which $\mathcal{A}_{KI}$ must register him for. To do so, $\mathcal{A}_{KI}$ makes another $\mathsf{Corrupt}$ query to $\mathcal{C}$ of the form $\mathcal{O}^{\mathsf{Corrupt}}(\lambda^C(\mathcal{A}_{VC}), \perp)$. $\mathcal{C}$ adds $\lambda^C(\mathcal{A}_{VC})$ to the list Q and returns $(\kappa_{\lambda^C(\mathcal{A}_{VC})}, \omega_{\lambda^C(\mathcal{A}_{VC})})$. $\mathcal{A}_{KI}$ may run $\mathsf{RPVC.Register}(\mathcal{A}_{VC}, \mathrm{MK}', \mathrm{PP}')$ himself, as well as update the list $L = L \cup \lambda^C(\mathcal{A}_{VC})$.
5. $\mathcal{A}_{VC}$ is again given oracle access which is handled as above. Eventually, it will output two choices of computations o_0 and o_1. $\mathcal{A}_{KI}$ will choose *one* of these, $o = o_j$ uniformly at random and forward $\lambda^C(o)$ to $\mathcal{C}$ as its choice of challenge label, $v^\star$. Note that this choice is, by definition, valid in the S-KI game since $\mathcal{A}_{VC}$ was not allowed to choose either message if $\lambda^C(o_j) \leqslant \lambda^C(v_i)$ for any v_i queried to the $\mathsf{Register}$ oracle (each of which was then queried to the $\mathsf{Corrupt}$ oracle). If the bit b chosen by $\mathcal{C}$ at the start of the game was 0, then $\mathcal{C}$ returns the real key $\kappa^\star = \kappa_{\lambda^C(o_j)}$; otherwise, $\mathcal{C}$ returns a random key $\kappa^\star$ drawn uniformly from the keyspace.

6. $\mathcal{A}_{KI}$ must now simulate registering a delegator C. It first chooses $\lambda^C(C) = \lambda^C(o_j)$ such that $\kappa_{\lambda^C(C)} = \kappa^\star$. Note that $\mathcal{A}_{KI}$ can provide the relevant key and secret from the verification poset which it owns, and that $\omega_{\lambda^C(C)}$ will not be needed, as no further keys shall need deriving.

7. $\mathcal{A}_{KI}$ can run FnInit and $\mathsf{Certify}$ as written as these do not rely on the computational KAS. To run $\mathsf{ProbGen}$ on the computation o_j, $\mathcal{A}_{KI}$ can run lines 1, 5 and 6 of Algorithm 5 using the keys defined in the previous step. All resulting parameters are given to $\mathcal{A}_{VC}$.

At this point, $\mathcal{A}_{VC}$ has been given all information that an adversary against the Weak Input Privacy game would be given and must attempt to distinguish which of **Game 0** and **Game 1** he has been playing. By assumption, $\mathcal{A}_{VC}$ succeeds at this with non-negligible advantage ϵ. Notice that the distribution of the above game generated by $\mathcal{A}_{KI}$ is precisely that of **Game 0** if the bit b chosen by $\mathcal{C}$ was 0 (i.e. the real KAS key was used), and is precisely **Game 1** if b was 1 (i.e. a random key was used). As such, $\mathcal{A}_{KI}$ can forward the distinguishing guess of $\mathcal{A}_{VC}$ to $\mathcal{C}$ to break the S-KI security of the KAS also with non-negligible advantage ϵ. However, as we assumed that the KAS was S-KI secure, such an adversary may not exist and so no adversary may exist that can distinguish **Game 0** from **Game 1** with non-negligible advantage.

Reduction to IND-CPA We have shown that no adversary can distinguish between **Game 0** and **Game 1** with non-negligible advantage. Thus, we may run an adversary against **Game 0** against **Game 1** instead with at most a negligible loss ϵ in the tightness of the reduction. That is, we can use a truly random key to form the challenge ciphertext which removes any information leakage from the KAS. Let us now suppose $\mathcal{A}_{VC}$ is an adversary with non-negligible advantage δ against **Game 1**. We construct an adversary $\mathcal{A}_{SE}$ that breaks the IND-CPA security of the symmetric encryption scheme using $\mathcal{A}_{VC}$ as a subroutine. Let $\mathcal{C}$ be the IND-CPA challenger for $\mathcal{A}_{SE}$ who in turn acts as the challenger for $\mathcal{A}_{VC}$.

1. $\mathcal{C}$ chooses a bit $\beta \xleftarrow{\$} \{0,1\}$ and generates a key $\kappa^\star$ by running $\mathsf{SE.KeyGen}$. It sends the security parameter to $\mathcal{A}_{SE}$ and provides oracle access to the LoR function.

2. $\mathcal{A}_{SE}$ must now initialize **Game 1** for $\mathcal{A}_{VC}$. Informally, it will set the challenge KAS key $\kappa_{\lambda^C(o_\beta)}$ to be the random $\kappa^\star$ chosen by $\mathcal{C}$. However, as this label is currently unknown and the public parameters and oracle access must be granted before the choice is made, $\mathcal{A}_{SE}$ must make a guess, $\widetilde{\lambda^C(o_\beta)}$, for the correct challenge label to assign the random key to. This choice is correct with probability at least $\frac{1}{N}$ where the computation poset comprises N labels and where N is polynomial in the security parameter (to enable efficient instantiation).

3. $\mathcal{A}_{SE}$ initalizes the parameters L and o, and also initalizes an empty list Q that will be used to store messages queried to the LoR oracle in the IND-CPA game. It runs Setup as written with the following modification. The KAS key for the guessed challenge label $\widetilde{\lambda^C(o_\beta)}$ is implicitly set to be $\kappa^\star$ and the rest of the computation KAS is made consistent with this choice. Since the adversary in **Game 1** is not permitted to query the $\mathsf{Register}$ function for any label that is an ancestor of the challenge label. Thus, keys for ancestors of the challenge label will not need to be derivable and can simply be generated from $\mathsf{SE.KeyGen}$. Therefore, a KAS can simply be instantiated over the remaining (non-ancestor) nodes and the public information for ancestor nodes can be simulated (as keys will not be derived, this need not be functionally correct but must appear to be distributed correctly).

4. $\mathcal{A}_{VC}$ is given the resulting public parameters and access to oracle functionality which $\mathcal{A}_{SE}$ responds to as follows:

   - Queries to FnInit, Certify and Revoke can be handled by simply running the relevant algorithms.
   - If $\mathcal{A}_{VC}$ queries Register for a label $\lambda(ID) \geqslant \widetilde{\lambda^C(o_\beta)}$, as guessed by $\mathcal{A}_{SE}$, then $\mathcal{A}_{SE}$ will abort the game (since $\mathcal{A}_{VC}$ would no longer be able to choose $\widetilde{\lambda^C(o_\beta)}$ as a valid challenge label and hence $\mathcal{A}_{SE}$'s guess was incorrect). For any other Register query, $\mathcal{A}_{SE}$ has generated and holds the relevant KAS keys and may respond honestly.
   - A query to ProbGen for anything other than a computation labelled with $\widetilde{\lambda^C(o_\beta)}$ can be run honestly using the keys he generated during Setup. A computation labelled by $\widetilde{\lambda^C(o_\beta)}$ will require an encryption of σ_o under $\kappa_{\widetilde{\lambda^C(o_\beta)}}$ which is not known to $\mathcal{A}_{SE}$ as it is the challenge key in the IND-CPA game. Therefore, $\mathcal{A}_{SE}$ must make a query to the LoR oracle provided by $\mathcal{C}$ where $m_0 = m_1 = \sigma_o$ to receive a valid ciphertext CT. $\mathcal{A}_{SE}$ also adds the pair (σ_o, CT) to the list Q.
   - By the restriction on Compute oracle queries, $\mathcal{A}_{VC}$ may not query the challenge inputs. Additionally, by the INT-PTXT property of the symmetric encryption scheme, $\mathcal{A}_{VC}$ is unable to form a valid ciphertext for the challenge label without making use of the Encrypt oracle. Thus, if the input to the Compute oracle is for the challenge computation label, then the ciphertext is either malformed (and $\perp$ should be returned), or the encoded input was previously queried to the LoR oracle and $\mathcal{A}_{SE}$ may look up the received ciphertext in the list Q to recover the encrypted input σ_o. For any other computation label, $\mathcal{A}_{SE}$ holds the corresponding KAS keys and can run the Compute algorithm honestly.

5. Eventually, $\mathcal{A}_{VC}$ outputs a choice of label $\lambda^C(\mathcal{A}_{VC})$ which $\mathcal{A}_{SE}$ can register him for as per the Register oracle queries above. The label is added to the list L and $\mathcal{A}_{VC}$ is given its secret key and oracle access which is handled as above.

6. $\mathcal{A}_{VC}$ eventually outputs a choice of two computations o_0 and o_1. If neither computation matches with the label $\widetilde{\lambda^C(o_\beta)}$ chosen by $\mathcal{A}_{SE}$ earlier, then the game is aborted. Otherwise, $\mathcal{A}_{SE}$ checks the labels for validity and aborts if not valid.

7. $\mathcal{A}_{SE}$ must now register a delegator C. However, the role of C will be substituted by oracle queries to $\mathcal{C}$ in the following, so $\mathcal{A}_{SE}$ may simply simulate registering C with label $\widetilde{\lambda^C(o_\beta)}$ by updating any relevant public information. $\mathcal{A}_{SE}$ can also run FnInit and Certify as written.

8. $\mathcal{A}_{SE}$ must now run ProbGen to generate the challenge input. To do so, it runs RPVC.ProbGen on both inputs x_0 and x_1 dictated by o_0 and o_1 respectively, to generate two encoded inputs labelled $m_0 = \sigma_{o_0}$ and $m_1 = \sigma_{o_1}$. It then submits m_0 and m_1 to the IND-CPA LoR oracle provided by $\mathcal{C}$. $\mathcal{C}$ will return the encryption of m_β corresponding to σ_{o_β}, under the key $\kappa_{\lambda^C(o_\beta)}$. $\mathcal{A}_{SE}$ can also encrypt the verification key VK_{o_β} for the o_β matching its guess, using the verification KAS keys that it owns.

9. The (encrypted) encoded input and the verification key are given to $\mathcal{A}_{VC}$ along with oracle access. Queries are handled as above by Register now returns $\perp$ if the queried label is an ancestor of either $\lambda^C(o_0)$ or $\lambda^C(o_1)$ (i.e. precisely when $\mathcal{A}_{SE}$ does not hold the relevant KAS keys). $\mathcal{A}_{VC}$ eventually outputs a guess b' that $o_{b'}$ was chosen.

Now, $\mathcal{A}_{SE}$ can simply forward the guess b' to $\mathcal{C}$ as its guess for β. Observe that if $\mathcal{A}_{SE}$ correctly guessed $\widetilde{\lambda^C(o_\beta)}$, then from $\mathcal{A}_{VC}$s point of view the distribution of the above game is precisely that of **Game 1**. Thus, if $\mathcal{A}_{VC}$ can successfully distinguish which computation was chosen (by $\mathcal{C}$), then it implicitly can decide which plaintext (encoded input) was encrypted during the IND-CPA game. $\mathcal{A}_{SE}$ can correctly guess the challenge label that will be chosen by $\mathcal{C}$ with probability at least $\frac{1}{N^2}$. Thus, since we assumed that $\mathcal{A}_{VC}$ had non-negligible advantage δ against **Game 1**, we have shown how to construct an adversary $\mathcal{A}_{SE}$ with non-negligible advantage δN^2 against the IND-CPA game. However, as the symmetric encryption scheme is assumed to be IND-CPA secure, such an adversary against **Game 1** may exist, and since there is a negligible distinguishing advantage between **Game 0** and **Game 1**, no adversary with non-negligible advantage against the Weak Input Privacy game may exist either.

□

We conclude that combining the results of the Lemmas 1 to 4 proves Theorem 1.

7 Conclusion

We have motivated the need for the cryptographic enforcement of access control policies in the setting of outsourced computation, particularly in the multi-user setting. As developments in VC continue towards such settings, it is vital to enable restrictions to be placed on: the computations that delegators can outsource (both from the perspective of separation of duties, and considering a server providing differing levels of service for different users); the computations a server may perform (such that certain computations, over sensitive data say, may only be performed by a server satisfying a policy); and the verifiers that may learn the output of the result (e.g. ensuring that read access to the newly generated data is handled consistently with the sensitivity of the inputs). We have shown example graph-based access control policies for these scenarios, as well as providing a formal definitional framework, security models, and a provably secure construction built from Key Assignment Schemes.

Future work will consider alternate enforcement mechanisms, such as authentication protocols that enforce graph-based authorisation policies to achieve ticket-based access control to a computational service, or the use of dual-policy ABE to outsource computations (using the KP-ABE policy) and enforcing access control (using the CP-ABE policy). Specific VC scenarios may lead to interesting access control models. One particularly applicable setting for the enforcement of access control policies is *verifiable searchable encryption*. Consider a remote database host that returns verifiably correct results to user queries (computations). In practice it is unlikely that all users should have unrestricted access to the entire database. It is imperative that only authorized users may perform specific queries (those relating solely to their duties and to data for which they have clearance)and that results remain protected to prevent data leakage.

Acknowledgements

We thank Christopher Dearlove, Rachel Player and Gordon Procter for helpful discussions. The first author acknowledges support from BAE Systems Advanced Technology Centre under a CASE Award.

This research was partially sponsored by US Army Research laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official

policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[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 - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, volume 8957 of *Lecture Notes in Computer Science*. Springer, 2014.

[3] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

[4] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, MITRE Corporation, 1973.

[5] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[6] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[7] M. A. Bishop. *Computer Security. Art and Science*. Addison-Wesley Professional, 2002.

[8] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[9] A. Castiglione, A. D. Santis, and B. Masucci. Key indistinguishability vs. strong key indistinguishability for hierarchical key assignment schemes. Cryptology ePrint Archive, Report 2014/752, 2014. `http://eprint.iacr.org/`.

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

[11] M. Clear and C. McGoldrick. Policy-based non-interactive outsourcing of computation using multikey FHE and CP-ABE. In P. Samarati, editor, *SECRYPT*, pages 444–452. SciTePress, 2013.

[12] J. Crampton. Cryptographic enforcement of role-based access control. In P. Degano, S. Etalle, and J. D. Guttman, editors, *Formal Aspects in Security and Trust*, volume 6561 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2010.

[13] J. Crampton, K. M. Martin, and P. R. Wild. On key assignment for hierarchical access control. In *CSFW*, pages 98–111. IEEE Computer Society, 2006.

[14] A. L. Ferrara, G. Fuchsbauer, and B. Warinschi. Cryptographically enforced RBAC. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26-28, 2013*, pages 115–129. IEEE, 2013.

[15] E. S. V. Freire, K. G. Paterson, and B. Poettering. Simple, efficient and strongly KI-secure hierarchical key assignment schemes. In E. Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2013.

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

[17] Google. Google Compute Engine – Cloud Computing & IaaS – Google Cloud Platform. `http://cloud.google.com/compute/`, 2014. [Online; accessed 23-October-2014].

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

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

[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, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012.

[21] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[22] R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.

[23] L. Xu and S. Tang. Verifiable computation with access control in cloud computing. *The Journal of Supercomputing*, 69(2):528–546, 2014.