Symmetric Attribute-Based Encryption from Minimal Hardness Assumptions

Riccardo Longo[1], Enrico Sorbera[1,2]

[1] Fondazione Bruno Kessler, Center for Cybersecurity, Trento, Italy,
{rlongo,esorbera}@fbk.eu,
[2] University Of Trento, Department of Mathematics, Trento, Italy

Abstract. We present a novel construction that applies the Ciphertext-Policy Attribute-Based Encryption paradigm in an original symmetric framework, where also the encryptor needs to have enough attributes to be able to produce a ciphertext for a given policy.
The scheme is built from minimal assumptions on collision-resistant hash functions and pseudorandom functions, exploiting the properties of linear secret sharing and polynomial interpolation. Thus, it is natively Post-Quantum secure.
We formally define a novel extended form for access trees, that trades a polynomial space expansion for a more predictable topological structure. This structure enhance the arithmetic possibilities of the associated secret sharing primitive.
Moreover, we propose a comprehensive notation for access trees, sharing and interpolation, which may help in the study of these powerful primitives.

1 Introduction

Attribute Based Encryption (ABE) is a cryptographic primitive introduced by Sahai and Waters in 2005 [11] with the first complete construction obtained in 2006 [7]. It was designed to enrich the granularity of the decryption criterion of cryptographic schemes. In fact, in previous schemes, the access to an encrypted resource was delegated to the possession of the very key used to encrypt it, or the corresponding decryption key in the case of asymmetric schemes. In many real life use cases we want to further control the access to some sensitive resources. A common way to extend this expressivity is by making the access to a resource dependent on the satisfaction of a boolean formula. Which is why, generally, ABE schemes are made of two main ingredients:

1. a set of shares produced under a target access structure,
2. a cryptographic primitive supporting some form of bilinearity.

The shares are employed to represent the logic structure of the boolean formula we want to check on the decrypting user's attributes. They can be constructed in different ways, the most common are LSSS [3] and Shamir Secret Sharing [12]. Usually, the first approach is preferred to the latter, due to its compact representation, still preserving a high expressivity.

The produced shares are encrypted under some asymmetric and linear cryptographic scheme. This encryption is needed to conceal the shares and to enforce a binding between the, usually implicit, knowledge of the shares and the possession of specific attributes. On top of these requirements, bilinear maps are usually exploited to bind together the private key parts of the user, i.e., make sure that parts coming from different keys do not work well together. This is a core security

requirement to avoid collusions among users, which otherwise could combine their attributes to satisfy access structures they would not have access to individually.

The necessity of bilinear maps dramatically restricts the suitable algebraic structures, that is why the majority of schemes proposed in the literature are defined over Elliptic Curves (EC), see e.g. [1,16]. However, elliptic curves and pairings have two main downsides. On one hand, while the underline secret sharing scheme is cheap both in terms of spatial and time consumption, the pairings over elliptic curves are expensive. There are lines of work [2,15] that reduces the number of pairing evaluations per encryption/decryption to a small constant, but, considering also EC multiplications, they still dominate the costs. Moreover, the discrete logarithm problem, which is a key security assumption for EC cryptography, is going to be deprecated in a few years [10] due to the threat of post quantum cryptanalysis and, in particular, Shor's algorithm [13]. To address this security concern, post quantum secure ABE schemes were proposed [8], based on standard hardness assumptions over lattices, such as Learning with Errors (LWE) or its ring version RLWE. Despite these hardness assumptions are considered post quantum secure, thus solving the security concern, schemes based on lattices still incur in large memory consumption [8] and expensive computations due to their vectorial or polynomial nature [6,14].

For these reasons we propose a scheme that achieves key binding without employing bilinear maps or lattice hardness assumptions. Instead, we address this feature by taking advantage once again from the logic structure of secret sharing.

On a high-level, the idea is to restrict ourself to a set of boolean formulae representable by bounded-depth access trees. Then, we can map each of these trees to an extended form, that has a target depth d. The extended form is designed so that each full-height subtree contains a special tree with a fixed topology, which represents a valid assignment of the variables in the formula. In this way, we can design a minimal access tree that acts as a sort of passkey for any valid policy. By producing shares of some random translation during the key generation phase, we can bind each part of the user's key by forcing these parts to follow this minimal common path when used in interpolation.

In this way, we were able to design a CP-ABE scheme using only keyed hash functions, Shamir linear secret sharing and symmetric encryption. This simplification solves both the problems of previous approaches, since it involves only efficient primitives and relies on post-quantum secure and information theoretically secure (ITS) primitives.

Furthermore, the presented scheme is designed to be symmetric, thus introducing a novel framework for ABE. While the classical approach grants users asymmetric rights in terms of encryption and decryption, giving them the possibility to encrypt under generic policies, while they are entitled to decrypt exclusively ciphertexts whose policies are satisfied by the attributes in their private keys, there are use cases where this extra power in encryption is not only superfluous, but actually harmful.

Finally, the proposed construction achieves $\texttt{IND} - \texttt{CCA2}$ security [4], thus avoiding further steps [17] to achieve stricter security definitions, usually required for standard ABE schemes.

The work is organised as follows. We start with some preliminaries in Section 2, in particular, in Section 2.1 we introduce some useful notation for linear secret sharing, and in Section 2.2 we define our comprehensive notation for threshold access trees and formally introduce the aforementioned extended form for an access tree. Then, in Section 3 we define our novel paradigm for symmetric ABE, and we formalise its security in Section 3.1, introducing a security game tailored to the symmetric framework. Our construction is presented in Section 3.2, discussing its expressiveness

and efficiency in Section 3.3. Finally, we delineate future works and draw some conclusions in Section 4.

Full proofs of correctness and security may be found in Section B and Section C respectively, we also provide in Section A the complete procedures for the conversion of access trees into the extended form, and we extensively analyse the theoretical complexity of our construction in Section D.

2 Preliminaries

2.1 Linear Secret Sharing

In this section, we discuss linear secret sharing through Shamir secret sharing over $\mathbb{F}_q$ and introduce some notation and properties.

Let $s \in \mathbb{F}_q$ be the secret value we want to share, let $t > 0$ be the desired threshold and let $\vec{r} = (r_1, \dots, r_{t-1}) \in \mathbb{F}_q^{t-1}$ be a randomly sampled vector, then we define an interpolating polynomial for s with threshold t as:

$$\left[\!\left[s\right]\!\right]_x^{t,\vec{r}} := s + \sum_{i=1}^{t-1} x^i \cdot r_i. \tag{1}$$

In particular, with a little abuse of notation, we can define a set of shares $\{\lambda_i\}_{i\in I}$ of s, with threshold t and randomness $\vec{r}$ in the set of points $\{x_i\}_{i\in I}$, as:

$$\lambda_i := \left[\!\left[s\right]\!\right]_{x_i}^{t,\vec{r}}. \tag{2}$$

We now introduce some well known properties of secret shares that will be needed later on.

Proposition 1. *Let* $\{\lambda_i\}_{i\in I} = \left\{ \left[\!\left[s\right]\!\right]_{x_i}^{t,\vec{r}} \right\}_{i\in I}$*, and suppose* $|I| \geq t$*, then there exist (and are efficiently computable knowing only* t *and* $\{x_i\}_{i\in I}$*) coefficients* $\{w_i\}_{i\in I}$ *such that:*

$$s = \sum_{i\in I} w_i \lambda_i, \tag{3}$$

and we denote this interpolation with the notation:

$$s = \left(\!\left|\{(\lambda_i, x_i)\}_{i\in I}\right|\!\right). \tag{4}$$

This property captures the fact that we can interpolate at least t consistent shares of a secret for threshold t, obtaining the correct result.

For the shares corresponding to a threshold 2 node, another interesting property holds.

Proposition 2. *Let* $\lambda_x = \left[\!\left[s\right]\!\right]_x^{2,r}$ *be a share of a secret* s *with threshold 2, corresponding to an interpolating point* x*. Let* y *a target point s.t.* $y \neq x$*, then we can build a new valid share* λ_y *for* s *corresponding to the point* y *as*

$$\lambda_y = \lambda_x + (y - x)r \tag{5}$$

This property can be easily checked. Despite its triviality, it has a crucial role in this work.

2.2 Threshold Access Trees

An access tree $\mathcal{T} = (N, E)$, where N is a set of nodes and E a set of edges, is a rooted threshold tree whose leaves are attributes a in some set of attributes U and whose satisfaction defines a monotone access structure. For a node $v \in N$ we define the following set of descriptors:

- $\mathtt{t}(v)$ which outputs the value of the threshold for its children, classically it is the label of internal nodes of threshold access trees, or $\perp$ if v is a leaf node;
- $\mathbb{T}(v)$ which outputs the threshold for its parent node, or 1 if it is the root;
- $\mathbb{l}(v)$ which outputs the level of the node, i.e. its height inside the tree, where the root is at level 0;
- $\mathtt{x}(v)$ which outputs the point used for the computation of the share corresponding to that node (i.e. at which point the sharing polynomial of degree $\mathbb{T}(v) - 1$ has been evaluated);
- $\mathbb{p}(v)$ which outputs the parent node of v, or $\perp$ if v is the root;
- $\mathtt{c}(v)$ which outputs the set of children of v.
- $\mathtt{a}(v)$ which outputs the attribute related to the node if the node is a leaf, $\perp$ otherwise.

Moreover, we define $\mathtt{r}(\mathcal{T})$ as the root of $\mathcal{T}$, and we partition the set of nodes of $\mathcal{T}$ in the sets $N_{\mathtt{l}}$ and $N_{\mathtt{int}}$, consisting respectively of the leaves and internal nodes of the tree. We also define $n_{\mathtt{l}} = |N_{\mathtt{l}}|$ and $n_{\mathtt{int}} = |N_{\mathtt{int}}|$. For a node $v \in \mathcal{T}$, let $\mathcal{T}_v$ be the subtree of $\mathcal{T}$ rooted in v. Then, it is easy to verify that $\mathcal{T}_v$ is still an access tree. Let $|\mathcal{T}_v|$ denote the depth of $\mathcal{T}_v$ in the following.

Definition 1 (d-Extended Form). *Consider the set T_d of access trees of depth less than d, with set of attributes $\{a_i\}_{i\in I}$, where $|I| = n$, we can map a tree $\mathcal{T} \in T_d$ to a new tree $\mathcal{T}'$, that we call* d-extended form*, with the following properties:*

- *$\mathcal{T}'$ has depth d;*
- *each node with depth $\ell < d$ is either labelled by a threshold $\{2, \dots, n\}$, or is labelled by $\emptyset_\ell$, where $\{\emptyset_\ell\}_{\ell\in\{1,\dots,d-1\}}$ are special attributes;*
- *each node with depth d is either labelled by an attribute b_i with $i \in I$ and has exactly one sibling node labelled by the attribute c_i, or the node is labelled by c_i and has exactly one sibling labelled b_i, where $\{b_i\}_{i\in I}$ and $\{c_i\}_{i\in I}$ are special artificial attributes;*
- *$\mathcal{T}'$ is minimal, in the sense that $\forall v \in N'$, we have that the tree $\mathcal{T}''$ obtained by removing from $\mathcal{T}'$ its subtree rooted in v is such that $\Gamma'' \subsetneq \Gamma'$,*
- *for any set $A = \{a_{i_1}, \dots, a_{i_s}\}$ we have that $A \in \Gamma$, where Γ is the access structure represented by $\mathcal{T}$, iff $A' = \{\emptyset_i\}_{i\in\{1,\dots,d-1\}} \cup \left\{b_j\right\}_{j\in\{i_1,\dots,i_s\}} \cup \left\{c_j\right\}_{j\in\{i_1,\dots,i_s\}}$ belongs to the access structure Γ' represented by $\mathcal{T}'$.*

We call $\mathcal{O}$ the set $\left\{\emptyset_i\right\}_{i\in\{1,\dots,d-1\}}$. The artificial attributes b_i, c_i are associated to the "original" attribute a_i, therefore with a slight abuse of notation we will associate an attribute also to the internal nodes at level $d-1$, in fact those nodes will have exactly two children, one labelled b_i and the other labelled c_i, therefore it is natural to label the parent with a_i.

Although the use of special attributes $\emptyset_i$ is necessary to push all leaves to the same depth d, the use of the twin attributes b_j and c_j can be somewhat obscure to the reader at this stage. They are introduced for security reasons that are clarified later in this work; on a high level, they are designed to prevent a user from retrieving the content related to leaves for which they do not have the right attributes.

In Section A we present a procedure to obtain the d-extended tree $\mathcal{T}'$, starting from a tree $\mathcal{T} \in T_d$.

Definition 2. *We define the string representation of an access tree $\mathcal{T}$ as the string defined recursively by the following two steps:*

- *if a node is labelled by an attribute a, encode it as 'a'.*
- *if a node is labelled by a threshold t, encode it as $(t;\ s_1,\ \ldots,\ s_l)$, where $\{s_i\}_{i=1}^{l}$ are its children.*

For threshold access trees, we generally want to associate each tree with a set of shares for some target secret s.

The details of the shares generation can be found in Algorithm 1.

Algorithm 1: Shares generation for an access tree

Initialize an empty stack S;
Initialize an empty list P;
$v \leftarrow \mathbb{r}(\mathcal{T})$;
Push(v) ; `// enqueue the root`
$\lambda_v \leftarrow s$;
while $S \neq \emptyset$ **do**
  $v \leftarrow$Pop(S);
  if $\mathbb{t}(v) \neq \perp$ *and* $\mathbb{l}(v) < d-1$ **then**
    $\vec{r} = (r_1, \ldots, r_{\mathbb{t}(v)-1}) \xleftarrow{\$} \mathbb{F}_q^{\mathbb{t}(v)-1}$;
    for $w \in \mathbb{c}(v)$ **do**
      $\mathbb{x}(w) \leftarrow$ PointSampler(w); `// select the interpolating point`
      $\lambda_w \leftarrow \left[\!\left[\lambda_v\right]\!\right]_{\mathbb{x}(w)}^{\mathbb{t}(v)}_{\vec{r}}$;
      Append(P, x); `// save the point, only required when it is sampled`
      Push(S, w); `// push the children in the stack`

The function PointSampler can be designed in different ways depending on whether we want to fix the interpolating points, generate them pseudorandomly, or sampling them with uniform distribution.

With the introduced procedure we are able to associate a secret s with a set of shares related to the nodes of any access tree $\mathcal{T}$. By giving a user the shares corresponding to the leaves, together with the interpolating points, the user is able to reconstruct the secret. To do so, the user chooses a "reconstruction tree", as per the following definition.

Definition 3 (Reconstruction Tree). *A reconstruction tree $\hat{\mathcal{T}}$ is a subtree of $\mathcal{T}$ with the following properties:*

- $\mathbb{r}\left(\hat{\mathcal{T}}\right) = \mathbb{r}(\mathcal{T})$;
- *the leaves of $\hat{\mathcal{T}}$ are a subset of the leaves of $\mathcal{T}$;*
- *$\hat{\mathcal{T}}$ is minimal, in the sense that $\forall v \in \mathcal{T}$, if $\mathbb{t}(v) \neq \perp$ then $|\mathbb{c}(v)| = \mathbb{t}(v)$.*

Note that a reconstruction tree of a tree in d-extended form is still an access tree in d-extended form.

In the following section we present and discuss the main property of trees in d-extended form.

OR trees and shares additivity In this section we explore an additivity property of shares of trees in d-extended form. In particular, given a generic tree $\mathscr{T}$ in d-extend form, and special OR trees $\widetilde{\mathscr{T}}$ that we are about to define, it is possible to sum the shares corresponding to leaves referring to matching attributes, while still being able to reconstruct the secret corresponding to the root of $\mathscr{T}$, up to sharing some further pieces of information.

Let us begin by introducing an OR tree $\widetilde{\mathscr{T}}$. Given a set of attributes $A = \{a_i\}_{i \in I}$, we can define the OR tree related to A as the d-extended form of the tree $(1;\ (a_i)_{i \in I})$ (see Figure 1).

Consider now a generic tree $\mathscr{T}$ corresponding to a policy $\mathbb{A}$ such that $A \in \mathbb{A}$. Let $\hat{\mathscr{T}}$ be a reconstruction tree of $\mathscr{T}$ whose internal nodes at level $d-1$ are all referred to attributes in A. That is, for each node $v \in \hat{\mathscr{T}}$ such that $\mathbb{l}(v) = d-1$ we have that $\mathtt{a}(v) \neq \emptyset_{d-1} \ \Rightarrow\ \mathtt{a}(v) = a_i$ for some $i \in I$. Given a value $s \in \mathbb{F}_q$, let us fix some points $\{\omega_i, \upsilon_i, \varpi_i\}_{i \in I}, \{x_\ell\}_{\ell \in \{1,\dots,d-1\}}, \{y_\ell\}_{\ell \in \{1,\dots,d-2\}}$, all sampled uniformly at random in $\mathbb{F}_q$, and consider the set of shares $\{\lambda_v\}_{v \in \hat{\mathscr{T}}}$ generated as in Algorithm 1 with the following definition of PointSampler$(\cdot)$:

$$\mathtt{x}(v) = \text{PointSampler}(v) = \begin{cases} \upsilon_i & \text{if } \mathbb{l}(v) = d \text{ and } \mathtt{a}(v) = b_i; \\ \varpi_i & \text{if } \mathbb{l}(v) = d \text{ and } \mathtt{a}(v) = c_i; \\ x_\ell & \text{if } \mathbb{l}(v) = \ell \leq d-1 \text{ and } \mathtt{a}(v) = \emptyset_\ell; \\ z_v \xleftarrow{\$} \mathbb{F}_q & \text{if } \mathtt{a}(v) = \perp \text{ or } \mathtt{a}(v) = a_i. \end{cases} \tag{6}$$

Note that the definition is complete.

Given another value $\tau \in \mathbb{F}_q$, consider the collection of randomnesses $\left\{r_v^{(j)}\right\}_{v \in \widetilde{\mathscr{T}}}$ for $j \in \{1, \dots, m\}$. Each set $\left\{r_v^{(j)}\right\}_{v \in \widetilde{\mathscr{T}}}$ allows us to generate a different set of shares $\{\theta_v^{(j)}\}_{v \in \widetilde{\mathscr{T}}}$ of τ according to the OR tree $\widetilde{\mathscr{T}}$ by applying the usual Algorithm 1 with $\left\{r_v^{(j)}\right\}_{v \in \widetilde{\mathscr{T}}}$ as randomness for the sharing and using the following sampler function PointSampler$'(\cdot)$:

$$\text{PointSampler}'(v) = \begin{cases} \text{PointSampler}(v) & \text{if } \mathtt{a}(v) \neq \perp \text{ or } \mathtt{a}(v) = a_i \\ y_\ell & \text{if } \mathbb{l}(v) = \ell < d-1 \text{ and } \mathtt{a}(v) = \perp, \\ \omega_i & \text{if } \mathbb{l}(v) = d-1 \text{ and } \mathtt{a}(v) = a_i; \end{cases} \tag{7}$$

Note that the definition is complete.

By the definition of an OR tree, $\forall a \in \{\mathtt{a}(v) \ :\ v \in \widetilde{\mathscr{T}}\} \setminus \{\perp\}, \exists!\ v \in \widetilde{\mathscr{T}}$ such that $\mathtt{a}(v) = a$, so we can define without ambiguity $\theta_a^{(j)}$ as $\theta_v^{(j)}$.

Proposition 3. *Let the set of attributes A, the trees $\hat{\mathscr{T}}, \widetilde{\mathscr{T}}$, the randomnesses $\left\{\left\{r_v^{(j)}\right\}_{v \in \widetilde{\mathscr{T}}}\right\}_{j \in \{1,\dots,m\}}$, the points $\{\mathtt{x}(v)\}_{v \in \hat{\mathscr{T}}}, \{\mathtt{x}(v)\}_{v \in \widetilde{\mathscr{T}}}$, and the shares $\{\lambda_v\}_{v \in \hat{\mathscr{T}}}$, $\left\{\left\{\theta_v^{(j)}\right\}_{v \in \widetilde{\mathscr{T}}}\right\}_{j \in \{1,\dots,m\}}$ as defined above. Let $\{w_j\}_{j \in \{1,\dots,m\}}$ be such that $\sum_{j=0}^{m} w_j = 0$, then, given:*

- *the sums of shares $\{\Delta_v\}_{v \in \{v\ :\ \mathtt{c}(v) = \emptyset\}}$, with $\Delta_v = \lambda_v + \sum_{j=1}^{m} w_j \theta_{\mathtt{a}(v)}^{(j)}$;*
- *the linear combination of the randomnesses $\{R_\ell\}_{\ell \in \{1,\dots,d-2\}}$, with $R_\ell = \sum_{j=1}^{m} w_j r_{v_\ell}^{(j)}$, where $v_\ell \in \widetilde{\mathscr{T}}$ is the unique node such that $\mathbb{l}(v_\ell) = \ell \bigwedge \mathtt{a}(v_\ell) = \perp$;*
- *the linear combination of the randomnesses $\{R_i\}_{i \in I}$, with $R_i = \sum_{j=1}^{m} w_j r_{v_i}^{(j)}$, where $v_i \in \widetilde{\mathscr{T}}$ is the unique node such that $\mathbb{l}(v_i) = d-1 \bigwedge \mathtt{a}(v_i) = a_i$;*

– *and the points* $\{\mathbb{x}(v)\}_{v\in\widehat{\mathscr{T}}} \cup \{y_\ell\}_{\ell\in\{1,\dots,d-2\}} \cup \{w_i\}_{i\in I}$*;*

it is possible to reconstruct the value s*.*

The proof of this proposition can be found in Section B. In our construction this additivity property is going to play a crucial role in binding the users' key parts, while allowing them to decrypt correctly. In fact, the shares of τ act as a random translation on the shares of a message at each level of the reconstruction tree, except for the root, where τ cancels out, revealing the value stored in the root of $\widehat{\mathscr{T}}$.

2.3 Cryptographically Secure Keyed Hash Functions

A *keyed hash function* is a deterministic function that takes as input a secret key and a message, and outputs a fixed-length digest that behaves like a pseudorandom function. Formally, a keyed hash family is defined as a pair of polynomial-time algorithms

$$\mathscr{H} = (\mathsf{KeyGen}_{\mathsf{H}}, H),$$

where:

– $\mathsf{KeyGen}_{\mathsf{H}}(1^\lambda) \to k$ is a probabilistic polynomial-time (PPT) algorithm that, on input the security parameter λ, outputs a secret key $k \in \{0,1\}^\lambda$.
– $H(k,m) \to h$ is a deterministic polynomial-time algorithm that takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a digest $h \in \mathbb{F}_q$, where n is polynomial in λ.

In the following, we denote $H(k,m)$ as $H_k(m)$, for the sake of simplicity.

A keyed hash function $\mathscr{H}$ is said to be *cryptographically secure* if it satisfies the following properties:

1. **Collision resistance:** It is computationally infeasible for any PPT adversary to find two distinct messages m, m' such that $H_k(m) = H_k(m')$.
2. **Pseudorandomness (PRF security):** The keyed hash behaves as a pseudorandom function. More precisely, for all PPT distinguishers $\mathscr{A}$, the advantage

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathscr{H}}(\mathscr{A}) = \left| \Pr_{k\leftarrow\mathsf{KeyGen}_{\mathsf{H}}(1^\lambda)}[\mathscr{A}^{H_k(\cdot)}(1^\lambda) = 1] - \Pr_{f\leftarrow\mathscr{F}_{\mathrm{rand}}}[\mathscr{A}^{f(\cdot)}(1^\lambda) = 1] \right|$$

   is negligible in λ, where $\mathscr{F}_{\mathrm{rand}}$ denotes the set of all truly random functions with the same domain and range.

In practice, we can exploit a keyed extensible output function (XOF) or a hash with a large enough output to construct H. In fact, suppose that q is a prime such that $2\lambda - 1 < \log_2(q) < 2\lambda$ (so that the probability of collisions is $\sim 2^{-\lambda}$), and we can compute uniform digests of length 3λ bits, then it is sufficient to interpret the digest as an integer and reduce it modulo q to obtain a value in $\mathbb{F}_q$ with negligible bias $2^{-\lambda}$ with respect to a uniform distribution.

2.4 Authenticated Encryption with Additional Data

An *Authenticated Encryption with Additional Data* (AEAD) scheme is a fundamental cryptographic primitive that enables two parties sharing a common secret key to exchange messages over an insecure channel where a part of the message remains confidential and another part stays in cleartext, but both parts are authenticated. Formally, an AEAD scheme with key space $\mathscr{K}$, message space $\mathscr{M}$, and ciphertext space $\mathscr{C}$ is defined as a pair of probabilistic polynomial-time (PPT) algorithms

$$\mathscr{E} = (\mathsf{AuthEnc}, \mathsf{AuthDec}),$$

where:

- $\mathsf{AuthEnc}(k, m, \mathtt{AD})$ is a probabilistic algorithm that, on input a key $k \in \mathscr{K}$, a plaintext $m \in \mathscr{M}$, and additional data $\mathtt{AD} \in \{0,1\}^*$, outputs a ciphertext $\mathtt{C} \in \mathscr{C}$.
- $\mathsf{AuthDec}(k, \mathtt{C}, \mathtt{AD})$ is a deterministic algorithm that, on input a key $k \in \mathscr{K}$, a ciphertext $\mathtt{C} \in \mathscr{C}$, and additional data $\mathtt{AD} \in \{0,1\}^*$, outputs either a decrypted plaintext $m \in \mathscr{M}$ or a special error symbol $\perp$.

From here on we use for brevity the shorthand notation $\mathsf{AuthEnc}_k(m, \mathtt{AD}) = \mathsf{AuthEnc}(k, m, \mathtt{AD})$ and $\mathsf{AuthDec}_k(\mathtt{C}, \mathtt{AD}) = \mathsf{AuthDec}(k, \mathtt{C}, \mathtt{AD})$.

The scheme must satisfy the correctness property $\forall k \in \mathscr{K}$ sampled uniformly at random and for any additional data $\mathtt{AD} \in \{0,1\}^*$:

$$\forall m \in \mathscr{M} : \quad \mathsf{AuthDec}_k(\mathsf{AuthEnc}_k(m, \mathtt{AD}), \mathtt{AD}) = m.$$

In this work we assume the AEAD scheme to be $\mathtt{IND-CCA2}$.

3 Symmetric CP-ABE

We now formally define our Symmetric CP-ABE scheme and adapt the security definition to our new framework.

Definition 4 (Symmetric CP-ABE). *A Symmetric CP-ABE scheme is defined as a quadruple of probabilistic polynomial time algorithms:*

$$\mathscr{S}_{\mathsf{ABE}} = (\mathsf{Setup}_{\mathsf{ABE}}, \mathsf{KeyGen}_{\mathsf{ABE}}, \mathsf{Encrypt}_{\mathsf{ABE}}, \mathsf{Decrypt}_{\mathsf{ABE}}),$$

where:

- $\mathsf{Setup}_{\mathsf{ABE}}(1^\lambda) \to (msk, pp)$ *is a probabilistic algorithm that, on input the security parameter* λ*, outputs a master secret key* msk *and public parameters* pp*;*
- $\mathsf{KeyGen}_{\mathsf{ABE}}(pp, msk, A) \to k_A$ *is a probabilistic algorithm that, on input the public parameters* pp*, the master secret key* msk *and a set of attributes* A*, outputs a user key* k_A*;*
- $\mathsf{Encrypt}_{\mathsf{ABE}}(pp, m, k_A, \mathbb{A}) \to \mathtt{C}$ *is a probabilistic algorithm that, on input the public parameters* pp*, a message* m*, a user key* k_A *and an access policy* $\mathbb{A}$*, outputs* $\mathtt{C}$*, which is either* $\perp$ *or the ciphertext;*
- $\mathsf{Decrypt}_{\mathsf{ABE}}(pp, \mathtt{C}, k'_{A'}) \to m'$ *is a deterministic algorithm that, on input the public parameters* pp*, a ciphertext* $\mathtt{C}$ *and a user key* $k'_{A'}$*, outputs* m'*, which is either* $\perp$ *or the decrypted message.*

The scheme is correct if:

- *given* $(msk, pp) = \mathsf{Setup}_{\mathsf{ABE}}(1^\lambda)$, $k_A = \mathsf{KeyGen}_{\mathsf{ABE}}(pp, msk, A)$, $k'_{A'} = \mathsf{KeyGen}_{\mathsf{ABE}}(pp, msk, A')$;
- *for any policy* $\mathbb{A}$ *that mentions all and only the attributes in a set* $A_{\mathbb{A}}$ *we have that, given* $C = \mathsf{Encrypt}_{\mathsf{ABE}}(pp, k_A, \mathbb{A})$, *then* $C \neq \perp \iff A_{\mathbb{A}} \subseteq A$;
- *for any ciphertext* $C = \mathsf{Encrypt}_{\mathsf{ABE}}(pp, k_A, \mathbb{A})$, $C \neq \perp$, *given* $m' = \mathsf{Decrypt}_{\mathsf{ABE}}(pp, k'_{A'}, C)$, *then* $m = m' \neq \perp \iff A' \in \mathbb{A}$.

Note that we do not explicitly define a universe of attributes: in our construction we have an *open universe* where any string $a \in \{0,1\}^*$ can be an attribute.

3.1 Security

In this section we present a security game, capturing the $\mathtt{IND-CCA2}$ security level. We adapted standards $\mathtt{IND-CCA2}$ games for ABE to our symmetric framework.

Definition 5 ($\mathtt{IND-CCA2}$ Game for Symmetric CP-ABE). *Let* $\mathcal{A}$ *be the adversary and* $\mathcal{C}$ *the challenger, then, the* $\mathtt{IND-CCA2}$ *security game is defined as follows:*

1. $\mathcal{C}$ *runs* $\mathsf{Setup}_{\mathsf{ABE}}$ *obtaining* msk, pp *and sends* pp *to* $\mathcal{A}$.
2. $\mathcal{A}$ *may perform a polynomial number* q_1 *of queries of the following types:*

   Key Query *given a set of attributes* A_i, $\mathcal{C}$ *responds giving the corresponding user key* k_{A_i};

   Encryption Query *given a message* m_i *and a policy* $\mathbb{A}_i$, $\mathcal{C}$ *responds giving the corresponding ciphertext* C_i;

   Decryption Query *given a ciphertext* C_i, $\mathcal{C}$ *responds giving the corresponding decrypted message* m'_i *or* $\perp$ *if* C_i *is not a valid ciphertext.*
3. $\mathcal{A}$ *chooses the challenge: a pair of messages of equal length* $\bar{m}_0, \bar{m}_1$ *and a policy* $\bar{\mathbb{A}}$ *such that* $\forall i \in \{1, \dots, q_1\}$ *if the* i*-th query is a Key Query, then* $A_i \notin \bar{\mathbb{A}}$. $\mathcal{C}$ *flips a random coin* b *and responds with* $\bar{C}$, *the encryption of* $\bar{m}_b$ *according to* $\bar{\mathbb{A}}$.
4. $\mathcal{A}$ *may perform a polynomial number* q_2 *of queries of the same type as in Step 2, with the following trivial restrictions:*

   Key Query *the set of attributes* A_i *must satisfy* $A_i \notin \bar{\mathbb{A}}$;

   Decryption Query $C_i \neq \bar{C}$.
5. $\mathcal{A}$ *outputs the guess* b'.

$\mathcal{A}$ *wins the game if* $b' = b$, *and its advantage in playing the game is defined as:*

$$\mathsf{Adv}(\mathcal{A}) = \mathbb{P}[b' = b] - \mathbb{P}[b' \neq b]. \tag{8}$$

Definition 6 ($\mathtt{IND-CCA2}$ security of Symmetric CP-ABE). *A Symmetric CP-ABE scheme is* $\mathtt{IND-CCA2}$ *Secure if for any PPT adversary* $\mathcal{A}$, *its advantage in playing the game defined in Definition 5 is negligible.*

3.2 Our Construction

We now present our protocol, based on secret sharing, symmetric encryption and keyed hash functions.

Let us fix an integer d, that is a strict upper bound on the depth of any access tree linked to an access policy we want to express. Since we want our scheme to accept an open universe of attributes, we sample an hash key, namely $k_{\mathtt{points}}$ and five different domain separators $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \emptyset, \mathtt{f}\}$. Now, given any attribute a_i, we can associate three interpolating points in $\mathbb{F}_q$ to it:

$$\omega_i = H_{k_{\mathtt{points}}}(\mathtt{a} \| a_i), \tag{9}$$

$$\upsilon_i = H_{k_{\mathtt{points}}}(\mathtt{b} \| a_i), \tag{10}$$

$$\varpi_i = H_{k_{\mathtt{points}}}(\mathtt{c} \| a_i). \tag{11}$$

Similarly, for any level ℓ we define two points in $\mathbb{F}_q$ as

$$x_\ell = H_{k_{\mathtt{points}}}(\emptyset \| \ell), \quad \ell \in \{1, \ldots, d-1\}, \tag{12}$$

$$y_\ell = H_{k_{\mathtt{points}}}(\mathtt{f} \| \ell), \quad \ell \in \{1, \ldots, d-2\}. \tag{13}$$

- $\mathsf{Setup}(1^\lambda, d)$: given a bound d on the depth of representable access tree and a security parameter 1^λ, this algorithm generates a prime so that uniformly random samples from $\mathbb{F}_q$ grant λ bits of security even in case of birthday attacks, that is, it selects a prime q of 2λ bits. Then, it runs $\mathsf{KeyGen}_{\mathsf{H}}(q)$ to obtain a key $k_{\mathtt{attr}}$. Moreover, it samples a random translation $\tau \in \mathbb{F}_q$ and outputs a master secret key $msk = \{k_{\mathtt{attr}}, \tau\}$ and public parameters $pp = \{q, d, k_{\mathtt{points}}\}$.
- $\mathsf{KeyGen}(pp, msk, A)$: given a set of attributes $A = \{a_i\}_{i \in I}$, let $I = \{i_1, \ldots, i_k\}$. Then, this algorithm generates shares of τ under the OR access tree

$$\Gamma = (2;\ \emptyset_1, (2;\ \emptyset_2,\ (\ldots;\ \emptyset_{d-1},\ (2;\ \emptyset_d,\ (2;\ b_{i_1},\ c_{i_1}),\ \ldots,\ (2;\ b_{i_k},\ c_{i_k}))))),$$

  depicted in Figure 1. Recall that b_i and c_i for $i \in I$ are special artificial attributes in one-to-one correspondence with the set of attributes a_i. The shares are generated in the following way:

Algorithm 2: Modified shares for Γ

$s \leftarrow \tau$;
for $\ell \in \{0, \dots, d-3\}$ **do**
  $r_\ell \xleftarrow{\$} \mathbb{F}_q$;
  $\omega_{\ell+1} \leftarrow \left[\!\left[s \right]\!\right]^{2}_{\substack{r_\ell \\ x_{\ell+1}}}$, $\nu_{\ell+1} \leftarrow \left[\!\left[s \right]\!\right]^{2}_{\substack{r_\ell \\ y_{\ell+1}}}$;
  $s \leftarrow \nu_{\ell+1}$;
$r_{d-2} \xleftarrow{\$} \mathbb{F}_q$;
$\omega_{d-1} \leftarrow \left[\!\left[s \right]\!\right]^{2}_{\substack{r_{d-2} \\ x_{d-1}}}$;
for $i \in I$ **do**
  $\rho_i \xleftarrow{\$} \mathbb{F}_q$;
  $\alpha_i \leftarrow \left[\!\left[s \right]\!\right]^{2}_{\substack{r_{d-2} \\ \omega_i}}$;
  $\beta_i \leftarrow \left[\!\left[\alpha_i \right]\!\right]^{2}_{\substack{\rho_i \\ \upsilon_i}}$;
  $\gamma_i \leftarrow \left[\!\left[\alpha_i \right]\!\right]^{2}_{\substack{\rho_i \\ \omega_i}}$;

Note that this procedure coincides with Algorithm 1, but it stresses the notation in use for the shares at different levels in the tree. Finally, it outputs the user key

$$k_A = \left((a_i, k_{a_i}, k_{b_i}, k_{c_i})_{i \in I}, (k_{\emptyset_\ell}, k_{f_\ell})_{\ell \in \{1,\dots,d-2\}}, k_{\emptyset_{d-1}} \right), \tag{14}$$

where

$$k_{b_i} = H_{k_{\mathtt{attr}}}(\mathtt{b} \| a_i) + \beta_i, \qquad k_{c_i} = H_{k_{\mathtt{attr}}}(\mathtt{c} \| a_i) + \gamma_i, \qquad k_{\emptyset_\ell} = H_{k_{\mathtt{attr}}}(\emptyset \| \ell) + \omega_\ell, \tag{15}$$

$$k_{a_i} = H_{k_{\mathtt{attr}}}(\mathtt{a} \| a_i) + \rho_i, \qquad k_{f_\ell} = H_{k_{\mathtt{attr}}}(\mathtt{f} \| \ell) + r_\ell. \tag{16}$$

– $\mathsf{Enc}(pp, m, k_A, \mathbb{A})$: the encryption proceeds as follows:
  1. If the set of attributes occurring in $\mathbb{A}$ is not a subset of A, then the algorithm returns $\perp$.
  2. Otherwise, the algorithm constructs the d-extended form $\mathcal{T}'$ for the tree $\mathcal{T}$ corresponding to $\mathbb{A}$, let I be such that $\{a_i\}_{i \in I}$ be the set of attributes occurring in $\mathbb{A}$.
  3. It samples $r_m \xleftarrow{\$} \mathbb{F}_q$ and computes the secret shares of r_m under $\mathcal{T}'$, using the procedure Algorithm 1 with PointSampler defined as in Equation (6), where the points $\{\upsilon_i, \omega_i\}_{i \in I}$, $\{x_\ell\}_{\ell \in \{1,\dots,d-1\}}$, $\{y_\ell\}_{\ell \in \{1,\dots,d-2\}}$ are those defined in Equations (10) to (13).
  4. Let λ_v be the share computed for the node v, we are only interested in the shares corresponding to the leaf nodes, i.e. $\{\lambda_v\}_{\{v \in N_1\}}$. The algorithm computes:

$$\mathsf{C}_1 = \left\{ \mathsf{c}_{1,v} \right\}_{v \in N_1} = \left\{ \lambda_v + k_{\mathtt{a}(v)} \right\}_{v \in N_1}, \tag{17}$$

  where $k_{\mathtt{a}(v)}$ is a part of the user's key k_A as defined in Equation (15).

5. The procedure computes:

$$\mathtt{C}_2 = \left\{\mathtt{c}_{2,\ell}\right\}_{\ell\in\{1,\ldots,d-2\}} = \left\{k_{f_\ell}\right\}_{\ell\in\{1,\ldots,d-2\}}, \tag{18}$$

$$\mathtt{C}_3 = \left\{\mathtt{c}_{3,i}\right\}_{i\in I} = \left\{k_{a_i}\right\}_{i\in I}, \tag{19}$$

$$\mathtt{C}_4 = (\mathbb{A}, P). \tag{20}$$

   Where P is the set of randomly sampled points used for computing the shares of the internal nodes:

$$P = \{\mathbb{x}(v)\}_{v\in N_{\mathtt{int}}}. \tag{21}$$

6. Let $\mathscr{E}$ be an AEAD scheme with keyspace $\mathscr{K}$, and let $H' : \mathbb{F}_q \to \mathscr{K}$ be a pseudorandom function with uniform distribution of outputs, the algorithm computes:

$$k_m = H'(r_m), \qquad \mathtt{AD} = \mathtt{C}_1\|\mathtt{C}_2\|\mathtt{C}_3\|\mathtt{C}_4. \tag{22}$$

7. The procedure computes the encrypted payload as:

$$\mathtt{C}_0 = \mathsf{AuthEnc}_{k_m}(m, \mathtt{AD}), \tag{23}$$

   And outputs the ciphertext $\mathtt{C} = (\mathtt{C}_0, \mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3, \mathtt{C}_4)$.

– $\mathsf{Dec}(pp, \mathtt{C}, k'_{A'})$ The algorithm extracts the policy $\mathbb{A}$ and the set of internal interpolation points P from $\mathtt{C}_4$. If $A' \notin \mathbb{A}$, i.e. the set of attributes associated to the user key does not satisfy the access policy, then the procedure returns $\perp$. Otherwise, let $\hat{\mathscr{T}}$ be the smallest reconstruction tree for A' and $\mathbb{A}$ (see Definition 3). The intuition is that the decryption procedure combines the user key and the ciphertext elements to cancel out the masks added to the shares, and interpolates them iteratively until it reconstructs the randomness used to derive the key for the authenticated cipher.
  The decryption proceeds in the following way:
  1. The user unmasks the shares corresponding to the leaves, subtracting the corresponding key part, that is, for each $v \in \hat{\mathscr{T}}$ such that $\mathbb{t}(v) = \perp$, the user computes:

$$\Delta_v = \mathtt{c}_{1,v} - k'_{\mathbb{a}(v)} = \lambda_v + k_{\mathbb{a}(v)} - k'_{\mathbb{a}(v)} = \lambda_v + \theta_v - \theta'_v, \tag{24}$$

     where $\theta_v = \beta_i$ and $\theta'_v = \beta'_i$ if $\mathbb{a}(v) = b_i$, $\theta_v = \gamma_i$ and $\theta'_v = \gamma'_i$ if $\mathbb{a}(v) = c_i$.
  2. The user combines and unmasks the randomnesses of the keys by computing:

$$R_\ell = \mathtt{c}_{2,\ell} - k'_{f_\ell} = k_{f_\ell} - k'_{f_\ell} = r_\ell - r'_\ell, \tag{25}$$

$$R_i = \mathtt{c}_{3,i} - k'_{a_i} = k_{a_i} - k'_{a_i} = r_i - r'_i. \tag{26}$$

  3. Noting that $\mathtt{C}_4$ contains the set of points for $\hat{\mathscr{T}}$, while the points for the keys are publicly computable, then the user has all the values required by Proposition 3 and therefore can reconstruct the value r_m.
  4. Finally, the user decrypts the message:

$$m' = \mathsf{AuthDec}_{H'(r_m)}(\mathtt{C}_0, \mathtt{C}_1\|\mathtt{C}_2\|\mathtt{C}_3\|\mathtt{C}_4). \tag{27}$$

The procedure for actually computing the value r_m can be found in the proof of Proposition 3 in Section B. Note that we can generate pseudorandomly the points in P to save space in the ciphertext. It is sufficient to run $\mathsf{KeyGen}_{\mathsf{H}}$, to obtain a sampling key $k_{\mathtt{samp}}$ and modify the definition of the PointSampler as

$$\text{PointSampler'}(v) = \begin{cases} H_{k_{\mathtt{samp}}}(v) & \text{if } \mathtt{a}(v) = \perp \text{ or } \mathtt{a}(v) = a_i, \\ \text{PointSampler}(v) & \text{otherwise,} \end{cases} \tag{28}$$

where we assume that v has a public and unambiguous representation to be used as input of H (for example we can enumerate the nodes of the tree with a breadth-first visit). In this way it is no longer necessary to include the whole set of points in $\mathtt{C}_4$, but it is sufficient to set $P = k_{\mathtt{samp}}$.

Theorem 1 (Security of Our Construction). *If $\mathscr{H}$ is secure (see Section 2.3), $H'(\cdot)$ is a secure pseudo-random function, and $\mathscr{E}$ is* $\mathtt{IND-CCA2}$*, then the Symmetric CP-ABE scheme defined in Section 3.2 is* $\mathtt{IND-CCA2}$ *as per Definition 6.*

The proof can be found in Section C.

3.3 Expressiveness and Efficiency of the Scheme

In asymmetric schemes, a user can typically encrypt under any policy composed of attributes defined in the public key. On the other hand, each user is issued a private key for decryption, which is bound to their specific set of attributes. In our case, instead, each user has their own private key, bound to a set of attributes, to be used in both encryption and decryption.

Let us investigate the capabilities of this key. For the encryption it is easy to characterize the set of boolean formulae the user is allowed to encrypt under. Let A be their set of attributes, then they can express any monotone boolean formula (i.e., any boolean formula that does not use negations) with literals in the set A. For what concerns decryption instead, we can read messages encrypted under any policy for which A is an authorized set.

Beside the precise characterization of the capabilities in encryption and decryption, the scheme is designed for scenarios where there is not a strong separation between read and write rights. That is, users are given keys granting roughly the same capabilities in both encryption and decryption. Thus, the scheme induces a covering over the set of users, that is subgroups of parties that can share information among themselves. Note that this is not a partition since there may be non-empty intersections of subgroups.

Concerning the efficiency of the construction, in Section D we formally analysed the theoretical complexity of the scheme. In the worst-case, the number of operations is quadratic in the number n of attributes involved, while the size is polynomial in n and the selected depth d. In particular, in Section D, we prove that the number of nodes in any d-extended tree is lesser than $dn + d^2$. For the depth d we have a valid lower bound, that is $d \geq \log(n)$, but it is theoretically unbounded from above. Note that, to avoid excessive space overhead, we can divide the target policies into multiple distinct sets, each characterized by a maximum depth d_i. Then, we can run multiple Setup, each with input depth equal to d_i, so that for each subset it holds $d_i = \Theta(n)$ for each possible policy and the space expansion is quasi-linear.

For what concerns the quadratic number of operations reached in the worst-case, note that the computational demand of the involved primitives (i.e., arithmetic in $\mathbb{F}_q$, evaluation of hash functions and symmetric encryption) is very low and, moreover, the number of attributes actually found in real-life policies is very restrained [5] (very often less than 20). Therefore, the scheme should perform really well in practice.

4 Conclusions and Future Directions

In this work we presented a novel symmetric framework for ABE. This framework is designed for scenarios where we want to provide users with roughly equal capabilities in both encryption and decryption.

We provided a new open universe CP-ABE scheme that matches the framework requirements and proved it secure for a symmetric version of the classic $\texttt{IND} - \texttt{CCA2}$ security game for ABE, assuming only the collision resistance of a hash function, the security of a pseudorandom function and a secure AEAD. Since the proposed scheme only employs hardness assumptions considered post quantum secure and ITS primitives, it naturally achieves security against Post-Quantum cryptanalysis.

Note that the proposed scheme natively supports the extension to a multi-authority setting similar to that in [9], where the key generation is not relegated to a single entity, but multiple partially-trusted authorities are required to create a user key. This mitigates against single point of failures and key escrow problematics, since it is not required to trust a single entity anymore. The intuition to achieve this enhanced construction is simply to require each key authority to run the setup on a common set of public parameters, and then issue to users partial keys computed exactly as the user keys as presented in Section 3.2, finally we define user keys as the sum of all partial keys. Note that the encryption and decryption algorithms stay the same.

Our contribution also comprises a novel extended form for access trees, which enables the core key binding property of ABE schemes in a very natural and efficient way. Moreover we propose a granular and comprehensive notation for secret sharing, interpolation and threshold access trees, enhancing the clarity of the discussion on these specific topics.

Future Works To the best of our knowledge, this is the first example of ABE scheme involving only Hash functions, Shamir secret sharing and symmetric encryption, thus opening up a rich landscapes for future works.

A natural continuation of this work is the implementation of the proposed scheme to analyse its practical performances, and study its suitability in various access-control paradigms and real-world scenarios.

A different promising direction, thanks to the flexibility of the scheme, is to investigate the possibility of adding other features usually found in ABE schemes. For example, it would be interesting to study the possibility of adding asymmetric capabilities, to cover also use-cases that require more granularity in distinguishing read and write rights.

Another important feature to be explored is to be able to distribute the trust among multiple attribute-granting authorities delegating the generation of different sets of attributes to different groups of authority, as it is natural in many settings.

Finally, note that the novel way of exploiting secret sharing and d-extended forms to achieve cryptographic binding could be seen as a novel paradigm in symmetric cryptography, and hopefully it will sparkle generalisations which could lead to novel families of post-quantum, pairing-free primitives.

References

1. Shashank Agrawal and Melissa Chase. FAME: Fast Attribute-based Message Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 665–682, New York, NY, USA, October 2017. Association for Computing Machinery.

2. Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography – PKC 2011*, pages 90–108, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
3. Amos Beimel. *Secure schemes for secret sharing and key distribution.* PhD thesis, Technion—Israel Institute of Technology, Haifa, 1996.
4. Dan Boneh and Victor Shoup. A graduate course in applied cryptography. Online Draft, 2024.
5. Thang Bui, Anthony Matricia, Emily Contreras, Ryan Mauvais, Luis Medina, and Israel Serrano. Abac lab: An interactive platform for attribute-based access control policy analysis, tools, and datasets [dataset/tool paper]. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies*, SACMAT '25, page 111116, New York, NY, USA, 2025. Association for Computing Machinery.
6. Lorenzo Gasparini, Elia Onofri, Martina Palmucci, and Marco Pedicini. Cross-primitive comparison in CP-ABE: Bilinear pairing versus lattices. *Journal of Mathematical Cryptology*, 20(1):20250051, March 2026.
7. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.
8. Saima Hasib, Akhtar Rasool, Manasi Gyanchandani, Rahul Haripriya, and Lalit Kumar. A structured review of lattice based attribute based encryption methods for post quantum security. *Discover Computing*, 29(1):175, March 2026.
9. Riccardo Longo, Chiara Marcolla, and Massimiliano Sala. Key-policy multi-authority attribute-based encryption. In *International Conference on Algebraic Informatics*, pages 152–164. Springer, 2015.
10. Dustin Moody, Ray Perlner, Andrew Regenscheid, Angela Robinson, and David Cooper. Transition to post-quantum cryptography standards. Technical report, National Institute of Standards and Technology, 2024.
11. Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
12. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612613, November 1979.
13. Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.
14. Gudipati Sravya, Pasupuleti Syam Kumar, and R. Padmavathy. Survey of Post-Quantum Lattice-Based Ciphertext-Policy Attribute-Based Encryption Schemes for Cloud Storage: Taxonomy, Open Issues, and Future Directions. *IEEE Transactions on Services Computing*, 17(6):4540–4557, November 2024.
15. Willy Susilo, Guomin Yang, Fuchun Guo, and Qiong Huang. Constant-size ciphertexts in threshold attribute-based encryption without dummy attributes. *Information Sciences*, 429:349–360, 2018.
16. Marloes Venema, Greg Alpár, and Jaap-Henk Hoepman. Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. *Designs, Codes and Cryptography*, 91(1):165–220, January 2023.
17. Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. Generic constructions for chosen-ciphertext secure attribute based encryption. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography – PKC 2011*, pages 71–89, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

A Extended d-form derivation

We introduce here a complete procedure to obtain a tree $\mathcal{T}'$ in the d-extended form, starting from a tree $\mathcal{T} \in T_d$.

First, let us describe two helper functions: the first formalizes the insertion of a new node inside a tree as the parent of a specific node; the second specifies the insertion of a set of nodes into a tree, as children of a certain node.

Algorithm 3: Insert

Input : Access tree $\mathcal{T} = (N, E)$, a node $v \in N$, a node $v' \notin N$
Output: An access tree $\mathcal{T}'$ with v' in place of v and $\mathcal{T}_v$ as v' children

$\mathbb{l}(v') \leftarrow \mathbb{l}(v)$; // new node inserted at the same level
$\mathbb{T}(v') \leftarrow \mathbb{T}(v)$; // copy parent data
$\mathbb{p}(v') \leftarrow \mathbb{p}(v)$;
if $\mathbb{p}(v) \neq \perp$ **then**
  $\mathbb{c}(\mathbb{p}(v)) \leftarrow (\mathbb{c}(\mathbb{p}(v)) \setminus \{v\}) \cup \{v'\}$; // substitute node in parent's children
else // set new node as root
  $\mathbb{r}(\mathcal{T}) \leftarrow v'$
$\mathbb{c}(v') \leftarrow \{v\}$; // set old node as child of new one
$\mathbb{p}(v) \leftarrow v'$;
for $w \in \mathcal{T}_v$ **do** // adjust level of descendants
  $\mathbb{l}(w) \leftarrow \mathbb{l}(w)+1$;

Algorithm 4: AddChildren

Input : Access tree $\mathcal{T} = (N, E)$, a node $v \in N$, a set of nodes V such that $V \cap N = \emptyset$
Output: An access tree $\mathcal{T}'$ with V added to the children of v

for $w \in V$ **do**
  $\mathbb{p}(w) \leftarrow v$; // set parent
  $\mathbb{T}(w) \leftarrow \mathbb{t}(v)$; // set threshold
  $\mathbb{l}(w) \leftarrow \mathbb{l}(v) + 1$; // set level
$\mathbb{c}(v) \leftarrow \mathbb{c}(v) \cup V$; // add children

Now we can describe how a generic threshold access tree can be translated into our special form.

Algorithm 5: d-Extended form of an Access Tree

Input : Access tree $\mathcal{T}$ with root r, and an integer d s.t. the depth n of $\mathcal{T}$ is less than d
Output: Extended access tree $\mathcal{T}'$
Initialize an empty queue Q;
Enqueue(r) ; // start from root
while $Q \neq \emptyset$ **do**
  $v \leftarrow$ Dequeue(Q);
  $d_1 \leftarrow \mathbb{l}(v)$;
  $d_2 \leftarrow |\mathcal{T}_v|$;
  if $d_1 + d_2 < d - 1$ **then** // sub-tree can be pushed down
    create new nodes v', v'';
    $\mathbb{t}(v') \leftarrow 2$, $\mathbb{t}(v'') \leftarrow \perp$; // set threshold
    $\mathbb{a}(v') \leftarrow \perp$, $\mathbb{a}(v'') \leftarrow \emptyset_{d_1+1}$; // set attributes
    Insert($\mathcal{T}, v, v'$) ; // push down sub-tree
    AddChildren($\mathcal{T}, v', \{v''\}$) ; // add special 'void' attribute at the level
    Enqueue(v) ; // the node has to be processed again
  else
    if $d_1 < d - 1$ **then**
      for $v_i \in \mathbb{c}(v)$ **do** // children will have to be processed
        Enqueue(v_i);
      if $\mathbb{t}(v) = 1$ **then** // increase threshold and add 'void' attribute
        $\mathbb{t}(v) \leftarrow 2$;
        create a new node v'';
        $\mathbb{t}(v'') \leftarrow \perp$;
        $\mathbb{a}(v'') \leftarrow \emptyset_{d_1+1}$;
        AddChildren($\mathcal{T}, v, \{v''\}$);
    else // $d_1 = d - 1$
      // add twin special attribute
      let j such that $\mathbb{a}(v) = a_j$;
      $\mathbb{t}(v) = 2$; // replace with AND gate
      create new nodes v', v'';
      $\mathbb{a}(v') \leftarrow b_j$, $\mathbb{a}(v'') \leftarrow c_j$; // set attributes
      AddChildren($\mathcal{T}, v, \{v', v''\}$) ; // add special twin attributes

Note that the output of the Algorithm 5 represents a valid access structure for the set of attributes $\{b_1, \ldots, b_n\} \cup \{c_1, \ldots, c_n\} \cup \{\emptyset_1, \ldots, \emptyset_{d-1}\}$. We give an example of the output of Algorithm 5, with input an OR gate, in Figure 1.

B Correctness Proof

Here we prove Proposition 3.

Note that the smallest threshold access tree has the form $(1; a)$, therefore for any d-extended form we have $d \geq 2$ (see Definition 1), so let v be any internal node of $\widehat{\mathcal{T}}$ at level $d - 2$, i.e.

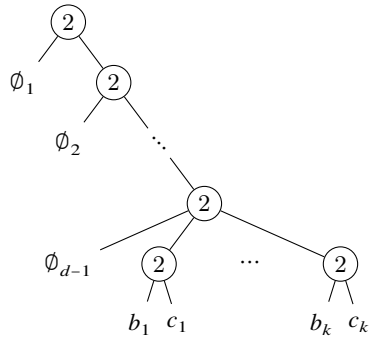

Fig. 1. Example of d-extended form for the tree $(1;\ (a_1,\ \ldots,\ a_k))$

$\mathbb{l}(v) = d - 2 \bigwedge \mathbb{t}(v) \neq \perp$. Since $\hat{\mathscr{T}}$ is a reconstruction tree, we have that $\mathbb{t}(v) = |\mathbb{c}(v)|$, and $\forall z \in \mathbb{c}(v)$, if $\mathbb{a}(z) \neq \emptyset_{d-1}$ then $2 = \mathbb{t}(z) = |\mathbb{c}(z)| = |\{z_1, z_2\}|$, note that there is at least one of such z since $\mathbb{t}(v) \geq 2$. Let i be such that $\mathbb{a}(z) = a_i$, $\mathbb{a}(z_1) = b_i$ and $\mathbb{a}(z_2) = c_i$. We have that:

$$\Delta_{z_1} = \lambda_{z_1} + \sum_{j=1}^{m} w_j \theta_{b_i}^{(j)} = \left[\!\left[\lambda_z \right]\!\right]^{2}_{\upsilon_i} + \sum_{j=1}^{m} w_j \left[\!\left[\theta_{a_i}^{(j)} \right]\!\right]^{2}_{r^{(j)}_{v_{d-1}},\, \upsilon_i}, \tag{29}$$

$$\Delta_{z_2} = \lambda_{z_2} + \sum_{j=1}^{m} w_j \theta_{c_i}^{(j)} = \left[\!\left[\lambda_z \right]\!\right]^{2}_{\omega_i} + \sum_{j=1}^{m} w_j \left[\!\left[\theta_{a_i}^{(j)} \right]\!\right]^{2}_{r^{(j)}_{v_{d-1}},\, \omega_i}. \tag{30}$$

Note that we are summing shares evaluated on the same points, and that $\lambda_{z_1}, \lambda_{z_2}$ are computed with the same (even if unknown) randomness, thus, we can correctly interpolate them. Moreover, thanks to the linearity of the interpolation procedure, we can define Δ_z and $\tilde{\Delta}_{z'}$:

$$\tilde{\Delta}_z = \left(\!\left|\left\{\left(\Delta_{z_1}, \upsilon_i\right), \left(\Delta_{z_2}, \omega_i\right)\right\}\right|\!\right) \tag{31}$$

$$= \left(\!\left|\left\{\left(\left[\!\left[\lambda_z \right]\!\right]^{2}_{\upsilon_i}, \upsilon_i\right), \left(\left[\!\left[\lambda_z \right]\!\right]^{2}_{\omega_i}, \omega_i\right)\right\}\right|\!\right) + \sum_{j=1}^{m} w_j \cdot \left(\!\left|\left\{\left(\left[\!\left[\theta_{a_i}^{(j)} \right]\!\right]^{2}_{r^{(j)}_{v_{d-1}},\, \upsilon_i}, \upsilon_i\right), \left(\left[\!\left[\theta_{a_i}^{(j)} \right]\!\right]^{2}_{r^{(j)}_{v_{d-1}},\, \omega_i}, \omega_i\right)\right\}\right|\!\right) \tag{32}$$

$$= \lambda_z + \sum_{j=1}^{m} w_j \theta_{a_i}^{(j)} = \left[\!\left[\lambda_v \right]\!\right]^{\mathbb{t}(v)}_{\mathbb{x}(z)} + \sum_{j=1}^{m} w_j \left[\!\left[\theta_{v_{d-2}}^{(j)} \right]\!\right]^{2}_{r^{(j)}_{v_{d-2}},\, \omega_i}, \tag{33}$$

from here on, we define v_ℓ for $\ell \in \{0, \ldots, d-2\}$ as the unique internal node in $\widetilde{\mathscr{T}}$ such that $\mathbb{l}(v_\ell) = \ell$, so v_{d-2} in Equation (33) is the unique internal node in $\widetilde{\mathscr{T}}$ at level $d-2$. As we have shown, we can directly interpolate the given shares, resulting in an output with similar shape, in the sense that we still have a linear combination of shares. However, we cannot directly interpolate this sum of shares, since the interpolating point of λ_z differs from the one of $\theta^{(j)}_{v_{d-2}}$. Here, the shared randomnesses come to play, since we can take advantage of Proposition 2 to move the interpolating point of the shares

$\theta^{(j)}_{v_{d-2}}$, to make it match $\mathbb{x}(z)$. To do so, we can compute

$$\Delta_z = \tilde{\Delta}_z + (\mathbb{x}(z) - \omega_i)R_i \tag{34}$$

$$= \left[\!\left[\lambda_v\right]\!\right]^{\mathbb{t}(v)}_{\mathbb{x}(z)} + \sum_{j=1}^{m} w_j \left[\!\left[\theta^{(j)}_{v_{d-2}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \omega_i}} + (\mathbb{x}(z) - \omega_i) \sum_{j=1}^{m} w_j r^{(j)}_{v_i} \tag{35}$$

$$= \left[\!\left[\lambda_v\right]\!\right]^{\mathbb{t}(v)}_{\mathbb{x}(z)} + \sum_{j=1}^{m} w_j (\left[\!\left[\theta^{(j)}_{v_{d-2}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \omega_i}} + (\mathbb{x}(z) - \omega_i) r^{(j)}_{v_i}) \tag{36}$$

$$= \left[\!\left[\lambda_v\right]\!\right]^{\mathbb{t}(v)}_{\mathbb{x}(z)} + \sum_{j=1}^{m} w_j \left[\!\left[\theta^{(j)}_{v_{d-2}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \mathbb{x}(z)}}. \tag{37}$$

Once again we reached the desired form, that allows us to interpolate and reach level $d-2$. When interpolated, this shares give again a value in the form

$$\tilde{\Delta}_v = \lambda_v + \sum_{j=1}^{m} w_j \theta_{v_{d-2}}. \tag{38}$$

Note that if $\exists z' \in \mathbb{c}(v)$ such that $\mathbb{a}(z') = \emptyset_{d-1}$, then we can compute:

$$\Delta_{z'} = \tilde{\Delta}_{z'} = \lambda_{z'} + \sum_{j=1}^{m} w_j \theta^{(j)}_{\emptyset_{d-1}} = \left[\!\left[\lambda_v\right]\!\right]^{\mathbb{t}(v)}_{x_{d-1}} + \sum_{j=1}^{m} w_j \left[\!\left[\theta^{(j)}_{v_{d-2}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ x_{d-1}}}, \tag{39}$$

which has again the exact same form, and can directly be used in interpolation.

If $d = 2$, we have that $\mathbb{r}\left(\widehat{\mathscr{T}}\right) = v$ and $\mathbb{r}\left(\widetilde{\mathscr{T}}\right) = v_{d-2} = v_0$, so by the hypotheses of Proposition 3 we have that $\lambda_v = s$, and $\theta_{v_0} = \tau$, so $\tilde{\Delta}_v = s + \sum_{j=1}^{m} w_j \tau = s$ since $\sum_{j=1}^{m} w_j = 0$, so the proposition is proven in this case.

If $d > 2$, we have that:

$$\tilde{\Delta}_v = \left[\!\left[\lambda_{\mathbb{p}(v)}\right]\!\right]^{\mathbb{T}(v)}_{\mathbb{x}(v)} + \sum_{j=1}^{m} w_j \left[\!\left[\theta^{(j)}_{v_{d-3}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \mathscr{y}_{d-2}}}, \tag{40}$$

despite being the sum of valid shares, we have the usual synchronization problem on the evaluation points: the first one is evaluated in a randomly sampled point $\mathbb{x}(v)$, while the remaining ones are evaluated in a common point $\mathscr{y}_{d-2}$. Thus, we once again take advantage on Proposition 2, using this time the randomness R_{d-2}. That is, we compute the value

$$\Delta_v = \tilde{\Delta}_v + (\mathbb{x}(v) - \mathscr{y}_{d-2})R_{d-2} \tag{41}$$

$$= \left[\!\left[\lambda_{\mathbb{p}(v)}\right]\!\right]^{\mathbb{T}(v)}_{\mathbb{x}(v)} + \sum_{j=1}^{m} w_j \left[\!\left[\theta^{(j)}_{v_{d-3}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \mathscr{y}_{d-2}}} + \left(\mathbb{x}(v) - \mathscr{y}_{d-2}\right) \left(\sum_{j=1}^{m} w_j r^{(j)}_{v_{d-2}} \right) \tag{42}$$

$$= \left[\!\left[\lambda_{\mathbb{p}(v)}\right]\!\right]^{\mathbb{T}(v)}_{\mathbb{x}(v)} + \sum_{j=1}^{m} w_j \left(\left[\!\left[\theta^{(j)}_{v_{d-3}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \mathscr{y}_{d-2}}} + (\mathbb{x}(v) - \mathscr{y}_{d-2}) r^{(j)}_{v_{d-2}} \right) \tag{43}$$

$$= \left[\!\left[\lambda_{\mathbb{p}(v)}\right]\!\right]^{\mathbb{T}(v)}_{\mathbb{x}(v)} + \sum_{j=1}^{m} w_j \left[\!\left[\theta^{(j)}_{v_{d-3}}\right]\!\right]^{2}_{\substack{r^{(j)}_{v_{d-2}} \\ \mathbb{x}(v)}}. \tag{44}$$

Again, if $\exists v' \in \mathbb{c}(\mathbb{p}(v))$ such that $\mathbb{a}(v') = \emptyset_{d-2}$, then we can compute $\Delta_{v'}$ as in Equation (39), so we have all the shares in the correct form to interpolate and compute $\tilde{\Delta}_{\mathbb{p}(v)}$. With the same

procedure we can compute $\Delta_{\bar{v}}$ and $\tilde{\Delta}_{\bar{v}}$ for any node $\bar{v} \in \hat{\mathscr{T}}$ going up until we reach $\mathbb{r}\left(\hat{\mathscr{T}}\right)$ where, as seen for the case $d = 2$, we have that $\tilde{\Delta}_{\mathbb{r}\left(\hat{\mathscr{T}}\right)} = s$, concluding the proof.

C Security Proof

To prove Theorem 1 we need first to prove the security of the sharing scheme at the core of our construction, so let us formalise it.

Definition 7 (Security game for Secret Sharing). *Let $\mathscr{A}$ be the adversary and $\mathscr{C}$ the challenger, then the security game for the Secret Sharing used in the construction of the Symmetric CP-ABE of Section 3.2 is defined as follows:*

1. *$\mathscr{C}$ runs $\mathsf{Setup}_{\mathsf{ABE}}$ obtaining msk, pp and sends pp to $\mathscr{A}$.*
2. *$\mathscr{A}$ may perform a polynomial number q_1 of queries of the following types:*
   Key Query *given a set of attributes A_i, $\mathscr{C}$ responds giving the corresponding user key k_{A_i};*
   Encryption Query *given a policy $\mathbb{A}_i$, $\mathscr{C}$ responds giving the values $(C_1^{(i)}, C_2^{(i)}, C_3^{(i)}, C_4^{(i)})$ corresponding to the ciphertext $C^{(i)}$ computed with an empty message;*
3. *$\mathscr{A}$ chooses the challenge: a pair of values $r_0, r_1 \in \mathbb{F}_q$ and a policy $\bar{\mathbb{A}}$ such that $\forall i \in \{1, \dots, q_1\}$ if the i-th query is a Key Query, then $A_i \notin \bar{\mathbb{A}}$. $\mathscr{C}$ flips a random coin b and responds with $(\bar{C}_1, \bar{C}_2, \bar{C}_3, \bar{C}_4)$, the parts of a ciphertext computed from an empty message according to $\bar{\mathbb{A}}$ and using the randomness $r_m = r_b$ instead of sampling it at random (see Item 3 of the Encryption procedure defined in Section 3.2).*
4. *$\mathscr{A}$ may perform a polynomial number q_2 of queries of the same type as in Step 2, with the following trivial restriction:*
   Key Query *the set of attributes A_i must satisfy $A_i \notin \bar{\mathbb{A}}$, for each $i \in \{1, \dots, q_2\}$;*
5. *$\mathscr{A}$ outputs the guess b'.*

$\mathscr{A}$ wins the game if $b' = b$, and its advantage in playing the game is defined as:

$$\mathsf{Adv}(\mathscr{A}) = \mathbb{P}[b' = b] - \mathbb{P}[b' \neq b]. \tag{45}$$

Definition 8 (Security of Secret Sharing). *The Secret Sharing scheme used in the construction of the Symmetric CP-ABE of Section 3.2 is secure if for any PPT adversary $\mathscr{A}$, its advantage in playing the game defined in Definition 7 is negligible.*

Theorem 2. *If $\mathscr{H}$ is secure (as defined in Section 2.3), then the Secret Sharing used in the construction of the Symmetric CP-ABE of Section 3.2 is secure as per definition 8.*

Proof. Our goal is to show that $\mathscr{A}$ gets no information about r_b from the data at its disposal. So, let us recap which data $\mathscr{A}$ gets:

- From the first step, $\mathscr{A}$ gets the public parameters, in particular, from this data it can compute the points $\{\omega_i, \upsilon_i, w_i\}_{i \in I}$, $\{x_\ell\}_{\ell \in \{1,\dots,d-1\}}$, $\{y_\ell\}_{\ell \in \{1,\dots,d-2\}}$.
- From the challenge it receives:
  - $\{\lambda_v + k_{\mathfrak{a}(v)}\}_{v \in N_1}$;
  - $\{k_{f_\ell}\}_{\ell \in \{1,\dots,d-2\}}$;
  - $\{k_{a_i}\}_{i \in I}$;

- $\{\mathbb{x}(v)\}_{v \in N_{\mathtt{int}}}$

where $\mathcal{T}$ is the d-extended tree associated to the challenge policy $\bar{\mathbb{A}}$, $N_{\mathtt{l}}$ is its set of leaf nodes and $N_{\mathtt{int}}$ is its set of internal nodes. Let also be $\bar{A}$ the set of attributes that occur in $\bar{\mathbb{A}}$.

– From Key Queries (in both phases) it receives:

$$k_{A_j} = \left(\left(a_i, k_{a_i}^{(j)}, k_{b_i}^{(j)}, k_{c_i}^{(j)} \right)_{i \in I_j}, \left(k_{\emptyset_\ell}^{(j)}, k_{f_\ell}^{(j)} \right)_{\ell \in \{1,\dots,d-2\}}, k_{\emptyset_{d-1}}^{(j)} \right)$$

where $A_j \notin \bar{\mathbb{A}}$ $\forall j \in Q_K$, where $Q_K \subseteq \{1, \dots, q_1 + q_2\}$ is the set of indexes of the queries which are key queries.

– From Encryption Queries (in both phases) it receives:

- $\left\{ \lambda_v^{(j)} + k_{\mathtt{a}(v)}^{(j)} \right\}_{v \in N_{\mathtt{l}}^{(j)}}$;
- $\left\{ k_{f_\ell}^{(j)} \right\}_{\ell \in \{1,\dots,d-2\}}$;
- $\left\{ k_{a_i}^{(j)} \right\}_{i \in I_j}$;
- $\{\mathbb{x}(v)\}_{v \in N_{\mathtt{int}}^{(j)}}$

where $\forall j \in Q_E$, where $Q_E \subseteq \{1, \dots, q_1 + q_2\}$ is the set of indexes of the queries which are encryption queries.

Note that only the data obtained in the challenge is related to r_b in any way, so the only hope to obtain information about r_b without guessing is to use the values $\left\{ \lambda_v + k_{\mathtt{a}(v)} \right\}_{v \in N_{\mathtt{l}}}$ (the other parts are also unrelated to r_b). These values are related to r_b via the secret sharing according to $\mathcal{T}$, and they give information only if we manage to interpolate the shares and climb the tree all the way up to the root.

By the restrictions on the challenges and the key queries made after the challenge, we have that $\forall j \in Q_K$, $A_j \notin \bar{\mathbb{A}}$, i.e. there is at least a branch of $\mathcal{T}$ that cannot be satisfied by the attributes in A_j. So, in order to gain information about r_b, $\mathcal{A}$ has to somehow combine the data at its disposal to interpolate shares corresponding to at least one of these unsatisfied branches, and in particular we can focus on leaves for which none of their ancestors is satisfied.

However, each of these shares is combined with another share of an OR tree and a mask, which is the digest of a keyed hash function. While we have hopes to obtain useful results by operating on sums of shares (even if of different trees), the masks will surely not play nice, so it is necessary to remove them from each share that we want to use in the interpolation.

Let us consider one of these values, specifically that associated to the leaf node $\bar{v}$ (note that they are all associated to a leaf), corresponding to the attribute $\bar{a}$, i.e. $\lambda_{\bar{v}} + k_{\bar{a}} = \lambda_{\bar{v}} + H_{k_{\mathtt{attr}}}(\bar{a}) + \theta_{\bar{a}}$ (we omit the domain separation value to simplify the notation), we can remove the mask only by subtracting another piece of information that contains the same mask $H_{k_{\mathtt{attr}}}(\bar{a})$, thanks to the security assumption on the hash function. We now list the different parts of the attacker information containing that specific mask:

– Other pieces of the challenge ciphertext that mention the same attribute, that is: $\lambda_{\bar{v}'} + k_{\bar{a}} = \lambda_{\bar{v}'} + H_{k_{\mathtt{attr}}}(\bar{a}) + \theta_{\bar{a}}$;
– Pieces of queried ciphertexts that mention the same attribute, i.e.: $\lambda_v^{(j)} + k_{\bar{a}} = \lambda_v^{(j)} + H_{k_{\mathtt{attr}}}(\bar{a}) + \theta_{\bar{a}}^{(j)}$, where $\mathtt{a}(v) = \bar{a}$;
– Pieces of keys that mention the same attribute, i.e.: $k_{\bar{a}}^{(j)} = H_{k_{\mathtt{attr}}}(\bar{a}) + \theta_{\bar{a}}^{(j)}$;

Let us define the set of pair of nodes in $\bar{\mathcal{T}}$ that share the same attributes, where $\bar{\mathcal{T}}$ is the tree in d-extended form associated to $\bar{\mathbb{A}}$:

$$\mathcal{P}_{\mathtt{SC},\mathtt{l}} = \{(v, v') \ : \ v, v' \in \bar{\mathcal{T}}, \ \mathtt{a}(v) = \mathtt{a}(v'), \mathtt{a}(v) \in \{b_i, c_i\}_{i \in I} \cup \{\emptyset_\ell\}_{\ell \in \{1,\dots,d-1\}}\}; \tag{46}$$

$$\mathcal{P}_{\mathtt{SC},\mathtt{int}} = \{(v, v') \ : \ v, v' \in \bar{\mathcal{T}}, \ \mathtt{a}(v) = \mathtt{a}(v') = a_i, \ i \in I\}; \tag{47}$$

$$\mathcal{P}_{\mathtt{SC}} = \mathcal{P}_{\mathtt{SC},\mathtt{l}} \cup \mathcal{P}_{\mathtt{SC},\mathtt{int}} \tag{48}$$

If we subtract $\lambda_{\bar{v}}$ and $\lambda_{\bar{v}'}$ for $(\bar{v}, \bar{v}') \in \mathcal{P}_{\mathtt{SC},\mathtt{l}}$, we obtain

$$\Omega_{\bar{v},\bar{v}'} = \lambda_{\bar{v}} - \lambda_{\bar{v}'}. \tag{49}$$

Let us analyse the different cases for $\bar{a} = \mathtt{a}(\bar{v})$. If $\bar{a} \neq \emptyset_\ell$, then let us assume without loss of generality that $\bar{a} = b_i$ for some i, then both $\bar{v}$ and $\bar{v}'$ have a sibling, namely $\bar{z}$ and $\bar{z}'$, which corresponds to the attribute c_i. Thus, by subtracting their related shares as well, we can produce $\{\Omega_{\bar{v},\bar{v}'}, \Omega_{\bar{z},\bar{z}'}\}$, which we can interpolate to obtain the difference of the shares

$$\tilde{\Omega}_{\mathtt{p}(\bar{v}),\mathtt{p}(\bar{v}')} = \lambda_{\mathtt{p}(\bar{v})} - \lambda_{\mathtt{p}(\bar{v}')}. \tag{50}$$

Importantly, since the point we used to generate the shares for nodes at level $d-1$ during encryption are randomized, up to negligible collisions, we have that $\mathtt{x}(\mathtt{p}(\bar{v})) \neq \mathtt{x}(\mathtt{p}(\bar{v}'))$. Thus, no matter how many shares of sibling nodes in this form we generate, they cannot be interpolated as they are. We can also assume without loss of generality that $\mathtt{p}(\mathtt{p}(\bar{v})) \neq \mathtt{p}(\mathtt{p}(\bar{v}'))$, since it would imply that two leaves referring to the same attributes are siblings. We can exclude this cases without restricting the expressiveness of the access trees. Since the grandparents of $\bar{v}$ and $\bar{v}'$ are different, the shares $\lambda_{\mathtt{p}(\bar{v})}$ and $\lambda_{\mathtt{p}(\bar{v})'}$ are also computed using different randomnesses. Thus, the values $\tilde{\Omega}_{\mathtt{p}(\bar{v}),\mathtt{p}(\bar{v}')}$ as they are give the attacker no further information on the unexplored branches of the tree. If, instead, $\bar{a} = \emptyset_\ell$, the difference of the shares can be further interpolated, since the shares are computed over the same point x_ℓ. However, since $\ell \in \{1, \dots, d-1\}$, the attacker cannot produce well formed shares of the siblings of the nodes related to $\emptyset_\ell$. This case has the same behaviour through the proof, so it will not be discussed again for the following cases. We can summarize the information generated by subtracting shares coming form the challenge ciphertext as the set

$$\mathcal{I}_{\mathtt{SC}} = \{\Omega_{v,v'} \ : \ (v, v') \in \mathcal{P}_{\mathtt{SC},\mathtt{l}}\} \cup \{\tilde{\Omega}_{v,v'} \ : \ (v, v') \in \mathcal{P}_{\mathtt{SC},\mathtt{int}}\}. \tag{51}$$

Similarly, to summarize the information that the attacker can retrieve from the common parts of the challenge ciphertext and the queried ciphertext, we define the sets

$$\mathcal{P}_{\mathtt{QC}} = \bigcup_{j \in Q_E} \mathcal{P}^{(j)}_{\mathtt{QC},\mathtt{l}} \cup \mathcal{P}^{(j)}_{\mathtt{QC},\mathtt{int}} \tag{52}$$

$$= \bigcup_{j \in Q_E} \left(\left\{ (v, v') \ : \ v \in \bar{\mathcal{T}}, v' \in \mathcal{T}^{(j)}, \ \mathtt{a}(v) = \mathtt{a}(v'), \mathtt{a}(v) \in \{b_i, c_i\}_{i \in I} \cup \{\emptyset_\ell\}_{\ell \in \{1,\dots,d-1\}} \right\} \right. \tag{53}$$

$$\left. \bigcup \left\{ (v, v') \ : \ v \in \bar{\mathcal{T}}, v' \in \mathcal{T}^{(j)}, \ \mathtt{a}(v) = \mathtt{a}(v') = a_i, \ i \in I \right\} \right), \tag{54}$$

where $\mathcal{T}^{(j)}$ is the d-extended form of the tree associated with A_j. Then, the total information generated using the queried ciphertexts can be summarized as

$$\mathcal{I}_{\mathtt{QC}} = \bigcup_{j \in Q_E} \left\{ \Omega^{(j)}_{v,v'} \ : \ (v, v') \in \mathcal{P}^{(j)}_{\mathtt{QC},\mathtt{l}} \right\} \cup \left\{ \tilde{\Omega}^{(j)}_{v,v'} \ : \ (v, v') \in \mathcal{P}^{(j)}_{\mathtt{QC},\mathtt{int}} \right\}, \tag{55}$$

where

$$\Omega^{(j)}_{v,v'} = \tilde{\Omega}^{(j)}_{v,v'} = \lambda_v - \lambda^{(j)}_{v'} + \theta_v - \theta^{(j)}_{v'}. \tag{56}$$

Note that, once again, in the definition of $\tilde{\Omega}^{(j)}_{v,v'}$, we implicitly assume $\mathtt{x}(v) \neq \mathtt{x}(v')$, while for $\Omega^{(j)}_{v,v'}$, the interpolating points coincide. When considering different ciphertexts, it is important to mention that the attacker could subtract parts coming from $\mathtt{C}^{(j)}_2$ and $\mathtt{C}^{(j)}_3$. However, this information serves the only purpose of allowing key shares synchronization (see Section B). The key shares prevent the attacker to use pieces of different keys in decryption even when the interpolating point can be moved. In fact, whenever the attacker tries to interpolate shares obtained with different keys, the corresponding key shares (that are summed to the ciphertext shares through all the interpolation process) would be generated from different independent polynomials, thus, they cannot be interpolated.

For what concerns the queried keys, we can associate to each of them the set of all the nodes that can be reached by applying the decryption procedure using that key.

More precisely, let k_{A_j} one of such keys, that is $j \in Q_K$. Then, the attacker can unmask the shares of the leaves nodes for the attributes in $A_j \cap \bar{A}$. This set of leaves allows the attacker to partially reconstruct the tree $\bar{\mathcal{T}}$. With a slight abuse of notation we denote this set as $\hat{\mathcal{T}}_j$. In fact, this set is actually a collection of possibly disjointed trees. From the requirement $A_j \notin \bar{\mathbb{A}}$, we know that $\mathtt{r}\left(\bar{\mathcal{T}}\right) \notin \hat{\mathcal{T}}_j$, for all $j \in Q_K$.

Note that while exploring $\bar{\mathcal{T}}$, the attacker could try to reconstruct nodes by evaluating the polynomial associated to their ancestors, rather then interpolating their children as intended. However, this procedure does not allow it to reach the shares of leaves for which it did not possess the attributes in the first place. This is due to the fact that in the d-extended form we substitute each leaf (a_i) with the tree $(2;\ (b_i,\ c_i))$ (see Algorithm 5). Even in the case the attacker is able to visit the node v with threshold 2 that we inserted in place of the old leaf, it cannot further descend to the children of v. In fact, to compute the shares, it would need to know the randomness used during encryption. The only values depending on this randomness are the shares associated to b_i and c_i. Since we assumed that it cannot reach v from below, this implies that it did not possess the right attributes and thus none of these shares, so the leaves cannot be reconstructed.

Coming back to the information generated by using the queried keys on the challenge, we have that for each node $v \in \hat{\mathcal{T}}_j$ for some $j \in Q_K$, the attacker gets the quantity

$$\Delta^{(j)}_v = \lambda_v + \theta_v - \theta^{(j)}_v. \tag{57}$$

Note that the values $\theta^{(j)}_v$ prevent the attacker to interpolate shares coming from sibling nodes chosen in different reconstruction trees, since they are generated from independent polynomials at key generation time. The total information can be then summarized as

$$\mathcal{T}_{\mathtt{K}} = \bigcup_{j \in Q_K} \left\{ \Delta^{(j)}_v \ :\ v \in \hat{\mathcal{T}}_j \right\}. \tag{58}$$

Clearly, the same information can be retrieved also in the case the attacker considers nodes coming from one of the trees related to one of the queried ciphertext. But in this case, all the values involved are unrelated to the secret s. The only secret quantities occurring in the queried ciphertexts are the secret digests, that mask the shares of the leaves and the translation needed for the synchronization of the evaluation points of shares coming from the keys, as in the decryption procedure (see

Proposition 3). However, we have shown how it is impossible to reach the values of shares related to leaves the attacker did not known already by reconstructing parts of the trees. Such shares are the only piece of information related to the masks that the attacker does not know. The exact same argument applies whenever we consider differences of shares coming from trees related to different queried ciphertexts.

Finally, we have to consider what an attacker can retrieve by subtracting parts of different queried keys. Each pair of keys have in common the digests related to the translations, that is $\left\{k_{f_\ell}^{(j)}\right\}_{\ell\in\{1,\dots,d-2\}}$. Moreover, consider the keys $k_{A_{j_1}}^{(j_1)}$, $k_{A_{j_2}}^{(j_2)}$, for $j_1, j_2 \in Q_K$ different indexes, corresponding to the sets of attributes A_{j_1} and A_{j_2}. Then, the queried keys have also in common the secret digests related to attributes in $A_{j_1} \cap A_{j_2}$ and $\left\{k_{a_i}^{(j)}\right\}_{i\in I_{j_1}\cap I_{j_2}}$. Whenever the attacker takes the differences of shares related to the same attributes $a \in A_{j_1} \cap A_{j_2}$, it gets

$$\theta_a^{(j_1)} - \theta_a^{(j_2)}, \tag{59}$$

which is a valid share of zero with respect to the OR tree structure. Once again, this values are unrelated to the secret s and, by reconstructing the OR tree, the attacker is unable to reach leaves that it did not know already, which are the only parts of the key related to the secret digests.

To conclude the proof, we have to consider what an attacker can obtain by crossing the information it generated in the cases we analysed above. In particular, we can see that every piece of information in the sets $\mathcal{I}_{\mathtt{SC}}, \mathcal{I}_{\mathtt{QC}}, \mathcal{I}_{\mathtt{K}}$, plus the crude data obtained interacting with $\mathscr{C}$, is a linear combination of the following variables:

1. the target secret r_b,
2. the randomnesses of the challenge ciphertext,
3. the randomnesses of the queried ciphertexts,
4. the secret translation τ,
5. the randomnesses of the queried keys,
6. the randomnesses of the keys implicitly used in the challenge and query ciphertexts,
7. the secret digests masking the leaves,

In particular, the values in $\mathcal{I}_{\mathtt{SC}}$ depend on Items 1 and 2, the values in $\mathcal{I}_{\mathtt{QC}}$ depend on Items 1 to 4 and 6, the values in $\mathcal{I}_{\mathtt{K}}$ depend on Items 1, 2 and 4 to 6, while Item 7 occurs in the information directly obtained from $\mathscr{C}$. So each set of information corresponds to a block of linear relationships across the listed variables.

We have already proven that, by using any single block, it is impossible to retrieve any information about r_b, we will now see that this still holds when considering different blocks together. The intuition is that each of the blocks introduces a new variable, which does not depend on r_b. Thus, no matter how they are combined, the global linear system stays underdetermined.

More precisely, we have proven that, to reconstruct the secret, the adversary has to traverse branches of $\mathcal{T}$ masked by digests of attributes that it does not possess. If the attacker tries to retrieve this information from $\mathcal{I}_{\mathtt{K}}$, the values $\Delta_\nu^{(j)}$ introduce the quantities $\theta_\nu^{(j)}$. Each of these quantities introduces a new value $\theta_\nu^{(j)}$, which depends on a query-dependent randomness sampled independently of r_b.

If, instead, the attacker tries to use the information from $\mathcal{I}_{\mathtt{QC}}$, the quantities $\Omega_{\nu,\nu'}^{(j)}$ introduce the value $\lambda_\nu^{(j)}$, which once again is dependent on query-specific randomnesses, sampled independently of r_b. At first glance, it could seem that the attacker is able to isolate the quantities $\lambda_\nu^{(j)}$ from

Equation (56) and Equation (57), thus eliminating the new independent variable and generating further information. This is not the case, since we have proven that the attacker cannot isolate $\lambda_v^{(j)}$ for the leaves v for which $\mathcal{A}$ did not already know the corresponding key part. Those are exactly the conditions that characterise the nodes which would give more information about the challenge ciphertext.

This discussion proves that, supposing the evaluating points and the digests used as masks do not collide, then the attacker gets no information whatsoever on the challenge. We omit the formal standard reduction to the collision resistance assumption on $\mathcal{H}$, but it is straightforward to see that the advantage of the adversary is negligible, inherited by the collision probability of $\mathcal{H}$.

□

Now we are ready to prove the security of our Symmetric CP-ABE presented in Section 3.2.

Proof (of Theorem 1). To prove the security of our Symmetric CP-ABE scheme we reduce ourselve to the $\mathtt{IND-CCA2}$ security of $\mathcal{E}$.

Suppose there exists a polynomial time adversary $\mathcal{A}$, that can attack the scheme with non negligible advantage, then we show how to build a simulator $\mathcal{S}$ that can break the $\mathtt{IND-CCA2}$ security of $\mathcal{E}$.

$\mathcal{S}$ runs $\mathsf{Setup}_{\mathsf{ABE}}$ honestly and gives the public parameters pp to $\mathcal{A}$, then responds normally to the queries of the first phase. When $\mathcal{A}$ sends the challenge $(\bar{m}_0, \bar{m}_1, \bar{\mathbb{A}})$, $\mathcal{S}$ samples $\bar{r} \in \mathbb{F}_q$ uniformly at random and computes the ciphertext parts $\mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3, \mathtt{C}_4$ and $\mathtt{AD}$ exactly as in a normal encryption under $\bar{\mathbb{A}}$) using $\bar{r}$ as the secret randomness to be shared. However, instead of computing $\mathtt{C}_0$ normally using $k_m = H'(\bar{r})$, $\mathcal{S}$ interacts with the challenger $\mathcal{C}$ of the $\mathtt{IND-CCA2}$ game for $\mathcal{E}$, and sends as its challenge $(\bar{m}_0, \bar{m}_1, \mathtt{AD})$. $\mathcal{C}$ responds with $\mathtt{C}_0 = \mathsf{AuthEnc}_k(\bar{m}_b, \mathtt{AD})$, where $k \in \mathcal{K}, b \in \{0,1\}$ are unknown to $\mathcal{S}$. $\mathcal{S}$ then sends to $\mathcal{A}$ the ciphertext $\mathtt{C} = (\mathtt{C}_0, \mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3, \mathtt{C}_4)$ as its answer to the challenge.

During the second phase of queries, $\mathcal{S}$ responds normally for key and encryption queries, while for decryption queries it first reconstructs the message randomness, if it is different from $\bar{r}$ it proceeds normally, otherwise it invokes the decryption oracle of $\mathcal{C}$ to get the answer to forward to $\mathcal{A}$.

Finally, $\mathcal{S}$ forwards to $\mathcal{C}$ the guess b' given by $\mathcal{A}$.

Note that the probability that the key k or the randomness $\bar{r}_m$ have been used by $\mathcal{S}$ to encrypt or decrypt before the challenge is negligible, and that by Theorem 2 the simulation run by $\mathcal{S}$ is indistinguishable from an honest challenger, but the only information related to the challenge ciphertext comes from $\mathcal{C}$, so $\mathcal{S}$ has a non-negligible advantage in winning against $\mathcal{C}$ if and only if $\mathcal{A}$ has a non-negligible advantage in breaking the Symmetric CP-ABE scheme. □

D Complexity Analysis

In this section we analyse and present some bounds on the space and time complexity of the proposed CP-ABE scheme. In particular, we give bounds for how many nodes are added at most by expanding a tree into its d-extended form via Algorithm 5, which give worst-case bounds on the number of elements in keys and ciphertexts, and bounds for the cost of the algorithms of the scheme. Note that these bounds can be strongly improved for specific use cases. They are intended to stress that the worst-case expansion is polynomial in the original number of nodes and target depth d, they generally do not represent the expected behaviour of the algorithms.

We recall the notation introduced in Section 2. Given a tree $\mathcal{T}$, let N be its set of nodes, then, we define $N_{\mathtt{l}}$ and $N_{\mathtt{int}}$ to be the sets of its leaves and internal nodes respectively. Moreover, let $n_{\mathtt{l}}$ and $n_{\mathtt{int}}$ be their cardinality, and $n = n_{\mathtt{l}} + n_{\mathtt{int}}$.

Proposition 4. *Let $\mathcal{T}$ be an access tree with depth m. Let d be a positive integer s.t. $d > m$, then the d-extended form $\mathcal{T}' = (N', E')$ of $\mathcal{T}$ is such that:*

$$n'_{\mathtt{int}} \leq \begin{cases} n_{\mathtt{int}} + (d-1)n_{\mathtt{l}} - \frac{1}{2}(d-2)(d+1) & \text{if } n_{\mathtt{l}} \geq d \\ n_{\mathtt{int}} + n_{\mathtt{l}} + \frac{1}{2}(d-1)(d-2) & \text{otherwise} \end{cases} \tag{60}$$

$$n'_{\mathtt{l}} \leq \begin{cases} dn_{\mathtt{l}} + n_{\mathtt{int}} - \frac{1}{2}(d-2)(d+1) & \text{if } n_{\mathtt{l}} \geq d \\ 2n_{\mathtt{l}} + n_{\mathtt{int}} + \frac{1}{2}(d-1)(d-2) & \text{otherwise} \end{cases} \tag{61}$$

Proof. In Algorithm 5, we add a node in three different points, corresponding to the calls to Insert and AddChildren, that is, to the following cases:

1. we add 1 leaf node when the node we are considering has threshold 1,
2. we add 2 nodes, one leaf and one internal, when we substitute the leaves referring to an attribute a_i at level $d-1$ with a node whose children refer to b_i and c_i,
3. we add 2 nodes, one leaf and one internal, whenever we are considering a node v such that its level plus the depth of the tree rooted in v is lesser than $d-1$.

To conclude the proof it is sufficient to bound the number of occurrences of these cases.

For the first case we have a very simple upper bound given from the number of internal nodes $n_{\mathtt{int}}$ of the input tree. Despite being really broad in practice, this worst case could be met by a tree consisting in a single 1 out of $n-1$ threshold node (i.e. a OR gate). Note that a tree composed only by OR gates can be flatten into a single OR gate, optimising the tree.

The occurrence of second case does not depend on the thresholds used in the boolean formulae or the topology of the tree, but only on the number of attributes in use, that coincides with the number of leaves $n_{\mathtt{l}}$ of the input tree.

The third case, instead, strongly depends on the topology of the access tree, and we have to count how many times we have to *push down* a node. Note that if $m < d-1$ then the root is pushed down $m-d-1$ times, and this automatically pushes down all other nodes without having to insert other nodes. If, instead, we push down a leaf, this does not effect other nodes, so the greatest expansion occurs when we have $m = d-1$ and the maximum number of leaves at level 1, and thus a minimal "crutch" that goes down to level $d-1$. Note that it makes no sense to consider as a *crutch* an access tree with just internal nodes with threshold equal to 1 and a single leaf, because (despite its minimality) it would be far more efficient to flatten it into a single threshold. Therefore, we have to add to the crutch a leaf at every level. Furthermore, to avoid obvious flattening opportunities we have to alternate OR and AND gates: in fact, by associativity, two consecutive ORs or ANDs can be flattened into a single one. In summary, the greatest number of insertion occurs when $n_{\mathtt{int}} = m = d-1$ and $n_{\mathtt{l}} \geq d$, with 2 leaves at level $d-1$, one leaf at level ℓ for $2 \leq \ell < d-1$ and $n_{\mathtt{l}} - d + 1$ leaves at level 1, which causes a total of:

$$\sum_{\ell=2}^{d-2} (d - \ell - 1) + (n_{\mathtt{l}} - d + 1)(d-2)$$

insertions, each adding a leaf and an internal node. Noting that:

$$\sum_{\ell=2}^{d-2}(d-\ell-1)=\sum_{i=1}^{d-3} i=\frac{(d-3)(d-2)}{2},$$

the number of insertions can be rewritten as:

$$n_{\mathtt{l}}(d-2)+(d-2)(1-d)+\frac{(d-3)(d-2)}{2}=n_{\mathtt{l}}(d-2)-\frac{1}{2}(d-2)(d+1).$$

If $n_{\mathtt{l}}<d$ we can use as upper bound $n_{\mathtt{l}}=d$ and obtain $\frac{1}{2}(d-1)(d-2)$.

Putting together the three cases, if $n_{\mathtt{l}}\geq d$ we have:

$$n'_{\mathtt{int}}\leq n_{\mathtt{int}}+n_{\mathtt{l}}+n_{\mathtt{l}}(d-2)-\frac{1}{2}(d-2)(d+1)=n_{\mathtt{int}}+(d-1)n_{\mathtt{l}}-\frac{1}{2}(d-2)(d+1), \tag{62}$$

$$n'_{\mathtt{l}}\leq n_{\mathtt{l}}+n_{\mathtt{int}}+n_{\mathtt{l}}+n_{\mathtt{l}}(d-2)-\frac{1}{2}(d-2)(d+1)=dn_{\mathtt{l}}+n_{\mathtt{int}}-\frac{1}{2}(d-2)(d+1), \tag{63}$$

otherwise:

$$n'_{\mathtt{int}}\leq n_{\mathtt{int}}+n_{\mathtt{l}}+\frac{1}{2}(d-1)(d-2), \tag{64}$$

$$n'_{\mathtt{l}}\leq n_{\mathtt{l}}+n_{\mathtt{int}}+n_{\mathtt{l}}+\frac{1}{2}(d-1)(d-2)=2n_{\mathtt{l}}+n_{\mathtt{int}}+\frac{1}{2}(d-1)(d-2), \tag{65}$$

thus concluding the proof.

The following corollary follows, which gives bounds on the size of ciphertexts and keys:

Corollary 1. *Let d be a positive integer, consider a set of attributes A and a policy $\mathbb{A}$ representable as an access tree $\mathcal{T}$, with $|\mathcal{T}|<d$. Let n_A be the number of attributes occurring in $\mathbb{A}$, counted with multiplicity. Note that, with the notation introduced above, we have $n_{\mathtt{l}}=n_A$, $n_{\mathtt{int}}=n-n_{\mathtt{l}}=n-n_A$. Let k_A be a key generated by calling KeyGen over the set of attributes A and, given a message m, let $C=(C_0,C_1,C_2,C_3,C_3)$ be its encryption under policy $\mathbb{A}$, then*

$$|k_A|=3n_A+2d-3 \tag{66}$$

$$|C_1|\leq\begin{cases}n+(d-1)n_A-\frac{1}{2}(d-2)(d+1) & \text{if } n_A\geq d\\ n+n_A+\frac{1}{2}(d-1)(d-2) & \text{otherwise}\end{cases} \tag{67}$$

$$|C_2|=d-2 \tag{68}$$

$$|C_3|=n_A \tag{69}$$

where $|k_A|$ (resp. $|C_i|$) stands for the number of elements in k_A (resp. in C_i).

We now study the time complexity of the algorithms of our scheme. The results concerning the number of operations in $\mathbb{F}_q$ are summarized in Table 1.

The costs of encryption and the key generation are dominated by the cost of generating shares of a secret under an access structure. In Shamir secret sharing t out of k, shares generation reduces to sampling $t-1$ random values $\in\mathbb{F}_q$ and evaluating a polynomial of degree $t-1$ in k points. Since in realistic use cases k and t are usually too small for sophisticated techniques such as multipoint polynomial evaluation, we assume a computational cost of $t\cdot k$ multiplications and sums, achieved

Algorithm	Multiplications	Sums
Key Generation	$2(d-2)+3n_A+1$	$4(d-2)+6n_A+2$
Encryption	$\sum_{v\in N} \mathbb{t}(v)\,\kappa_v$	$n'_1+1+\sum_{v\in N} \mathbb{t}(v)\,\kappa_v$
Decryption	$n'_{\mathtt{int}}-1+\sum_{v\in N'}(\kappa_v^2+\kappa_v)$	$n'_1+n_A+d+2n'_{\mathtt{int}}-4+\sum_{v\in N'}(\kappa_v^2+\kappa_v)$

Table 1. Exact computational costs in terms of number of $\mathbb{F}_q$ operations. In decryption, $N' = N'_{int} \cup N'_{leaf}$ are the nodes of the reconstruction tree actually used, partitioned in internal nodes and leaves.

by using Horner's rule on each point. Let us consider the sums of these costs among the nodes of a tree $\mathcal{T}$:

$$\sum_{v\in N} \mathbb{t}(v)\,\kappa_v, \tag{70}$$

where $\kappa_v = |\mathbb{c}(v)|$.

For the key generation, note that the access structure is the d-extended form of the policy $(1;\ (a_i)_{i\in I})$, where $\{a_i\}_{i\in I}$ are the input attributes. This step requires $2(d-2)+3n_A+1$ evaluations of a polynomial of degree 1, as can be easily checked in Algorithm 2. Moreover, we have to sum a mask for each key part, for a total of $2(d-2)+3n_A+1$ mask additions.

The number of polynomial evaluations required for encryption depends on the input policy and the subsequent topology of its d-extended form. For an access tree to represent a boolean function of the attributes with non-empty authorized set, we have to require $\mathbb{t}(v) \le \kappa_v$ for each node $v \in N$. Thus, we can give the following upper bound for the cost of encryption:

$$\sum_{v\in N} \mathbb{t}(v)\,\kappa_v \le \sum_{v\in N} \kappa_v^2 \le \left(\sum_{v\in N} \kappa_v\right)^2 \le n^2. \tag{71}$$

This is a very rough bound to capture the asymptotic behaviour. The tightest exact bound we can give for a generic access tree, is the exact value of the worst-case topology, which is easy to see that corresponds to a pure AND policy.

The cost for this worst-case tree can be computed by observing that we need to compute $d-2$ pairs of shares with threshold 2 corresponding to the first levels, then n_1 shares coming from a threshold equal to n_1, and finally n_1 pairs of shares with threshold 2, which give:

$$2(d-2)+n_1^2+2n_1$$

sums and multiplications in $\mathbb{F}_q$, to which we have to add $2n_1+d-2$ sums with key parts. So the final bound is:

$$\text{multiplications} \le 2(d-2)+n_1^2+2n_1, \tag{72}$$

$$\text{sums} \le 3(d-2)+n_1^2+4n_1. \tag{73}$$

Note that we did not consider the sampling of points, which requires one hash per non-root internal node of the d-extended tree, and the cost of the PRF and AEAD encryption.

For the decryption algorithm, given a cyphertext $\mathtt{C}$, let us suppose without loss of generality that the reconstruction tree that we will use is the whole access tree of $\mathtt{C}$, in fact if we prune any

ciphertext to only the elements associated with the reconstruction tree related to the decryption key to be used, we still get a valid ciphertext (of course the original ciphertext has to be used for the integrity checks of the AEAD). Then, we need $|\mathtt{C}_1| + |\mathtt{C}_2| + |\mathtt{C}_3| = n_1' + d - 2 + n_A$ sums to remove the masks from the shares and the translations needed for reconstructing the secret. Then, the decryption algorithm proceeds alternating an interpolation step over a subset of the nodes at a given level, and a step of synchronization, which is a translation by a correcting vector of the same size of the set of internal nodes at the next level. We interpolate once for each level, except for the root, thus d interpolations, and we translate at each level, except for levels d and 0, for a total of $n'_{\mathtt{int}} - 1$ sums. Note that, to be precise, we have an interpolation for each set of siblings at a given level, but we can consider it as a single linear transformation.

Thus, the entire process can be interpreted as the composition of d affine transformations. The first is defined over the set of leaves considered in the first step, and outputs a number of shares that depends on the topology of the access structure of the cyphertext. This set can be characterized as the set of parents of the interpolated nodes. Note that, given a set of k shares, each interpolation over a set of points x_i of a set of siblings nodes v_i requires computing a set of Lagrangian coefficients:

$$w_i = \prod_{\substack{j=0 \\ j \neq i}}^{k} \frac{x_j}{x_j - x_i}, \tag{74}$$

and computing the scalar products between these coefficients and the shares. Hence, given a set of k shares the cost of interpolating them is $k^2 + k$ multiplications over $\mathbb{F}_q$. Unfortunately, we can not reuse the Lagrangian multipliers across decryption of different cyphertext, since the interpolating points are randomized. Thus, the total cost of the interpolation procedure, ignoring the translations, is given by:

$$C(\mathscr{T}) := \sum_{i=1}^{n} (\kappa_i^2 + \kappa_i), \tag{75}$$

where κ_i is the number of children of the i-th node, so that $\sum_i \kappa_i = n - 1$, in fact all the nodes in the tree are children of some node, beside the root. Thus, we can bound the cost as:

$$C(\mathscr{T}) = \sum_{i=1}^{n} \kappa_i^2 + n - 1 \leq (\sum_{i=1}^{n} \kappa_i)^2 + n \leq n^2 + n. \tag{76}$$

Note, however, that Lagrangian multipliers only depend on the interpolating points, thus can be computed before the decryption step, by any party knowing the interpolating point. This opens up the possibility of delegating such computations to a server and to compute them in parallel.

Once again, this bound is loose but catches the quadratic asymptotic behaviour, we can give a tighter bound by considering the worst case topology for a tree in d-extended form. Since the cost function is still a sum of squares, we can consider the same worst-case topology we considered in encryption. The total cost would be dominated by n_A^2. Additionally, for each level in the set $\{1, \dots, d-2\}$, we have to compute the synchronisation, which costs 2 sums and 1 multiplication per internal node.

Note that once the reconstruction tree is selected (and this depends only on the policy and set of attributes), all the interpolating points are known and fixed, since they are shared all together with the tree in the ciphertext. Thus, it is possible to parallelize the computation of the Lagrangian coefficients. On the contrary, the mutual dependency of the shares induced by the access tree structure prevent us to parallelize the interpolation across different levels of the tree. It is still possible to parallelize among group of distinct siblings, once the Lagrangian multipliers are precomputed.

E Tables of Notation

Symbol	Meaning
$\mathcal{A}$	a polynomial time adversary,
$\mathscr{C}$	the challenger,
$\mathcal{P}_{\mathtt{SC}}$	pair of nodes sharing the same attributes in the challenge,
$\Omega_{v,v'}$	difference of shares of nodes of the same tree, with same interpolating points,
$\tilde{\Omega}_{v,v'}$	difference of shares of nodes of the same tree, with different interpolating points,
$\mathcal{T}_{\mathtt{SC}}$	global information deduced by the challenge ciphertext,
$\mathcal{P}_{\mathtt{QC}}$	pair of nodes coming from the challenge and from a queried ciphertext with same attribute,
$\Omega^{(j)}_{v,v'}$, $\tilde{\Omega}^{(j)}_{v,v'}$	difference of shares, but the second node comes from a queried ciphertext,
$\mathcal{T}_{\mathtt{QC}}$	global information deduced by the queried ciphertexts,
$\widehat{\mathcal{T}}_j$	partial reconstruction on the challenge try by using queried key,
$\mathcal{T}_{\mathtt{K}}$	global information deduced by the queried keys.

Table 2. Table summarizing the notation employed in the security proof.

attribute	tree node type	domain separator	interpolating point	share
a_i	internal at level $d-1$	$\mathtt{a}$	ω_i	α_i
b_i	leaf at level d	$\mathtt{b}$	υ_i	β_i
c_i	leaf at level d	$\mathtt{c}$	ω_i	γ_i
$\emptyset_\ell$	leaf at level ℓ	$\emptyset$	x_ℓ	ω_ℓ
f_ℓ	internal at level ℓ	$\mathtt{f}$	y_ℓ	ν_ℓ

Table 3. Additional notation for special attributes

Symbol	Meaning
$\mathcal{T}$	access tree,
N	set of tree nodes,
E	set of tree edges,
$N_{\mathtt{int}}$	set of internal nodes in a tree,
$N_{\mathtt{l}}$	set of leaves in a tree,
$n_{\mathtt{int}}$	number of internal nodes of a tree,
$n_{\mathtt{l}}$	number of leaves of a tree,
v	node in a tree,
$\mathcal{T}_v$	access tree rooted in v,
$\lvert\mathcal{T}_v\rvert$	depth of $\mathcal{T}_v$,
$\mathtt{t}(v)$	threshold of the node v,
$\mathbb{T}(v)$	threshold of the parent of v,
$\mathbb{l}(v)$	level of v in the tree,
$\mathtt{x}(v)$	evaluation point for the secret share related to v,
$\mathtt{p}(v)$, $\mathtt{c}(v)$	parent of v and set of children respectively,
κ_v	cardinality of the set of children of v,
$\mathtt{a}(v)$	attribute related to v, or $\perp$ for the internal nodes,
$\left(\!\left\{(\lambda_i, x_i)\right\}_{i=1}^{t}\!\right)$	interpolation of the t shares λ_i in the points x_i,
$\left[\!\left[s\right]\!\right]_{x}^{t\,r}$	interpolating polynomial for secret s, with randomness r and threshold t,
λ_v	share corresponding to the node v,
θ_v	share corresponding to the node v in an OR tree $\widetilde{\mathcal{T}}$,
Δ_v	difference of λ_v and linear combination of θ_v, computed in the same points,
$\tilde{\Delta}_v$	difference of λ_v and linear combination of θ_v, computed in different points,
$k_{\mathtt{attr}}$	hash key for generating the masks for the shares related to attributes,
$k_{\mathtt{points}}$	hash key for generating the interpolating points,
k_A	a key for the CP-ABE scheme, related to the set of attributes A,
$\mathtt{C}_i$	part of a ciphertext of the CP-ABE scheme.

Table 4. Table summarizing the notation employed.