Bounded Ciphertext Policy Attribute Based Encryption

Vipul Goyal Abhishek Jain Omkant Pandey Amit Sahai

Department of Computer Science, UCLA
{vipul,abhishek,omkant,sahai}@cs.ucla.edu

Abstract

In a ciphertext policy attribute based encryption system, a user's private key is associated with a set of attributes (describing the user) and an encrypted ciphertext will specify an access policy over attributes. A user will be able to decrypt if and only if his attributes satisfy the ciphertext's policy.

In this work, we present the first construction of a ciphertext-policy attribute based encryption scheme having a security proof based on a number theoretic assumption and supporting advanced access structures. Previous CP-ABE systems could either support only very limited access structures or had a proof of security only in the generic group model. Our construction can support access structures which can be represented by a bounded size access tree with threshold gates as its nodes. The bound on the size of the access trees is chosen at the time of the system setup. Our security proof is based on the standard Decisional Bilinear Diffie-Hellman assumption.

1 Introduction

In many access control systems, every piece of data may legally be accessed by several different users. Such a system is typically implemented by employing a trusted server which stores all the data in clear. A user would log into the server and then the server would decide what data the user is permitted to access. However such a solution comes with a cost: what if the server is compromised? An attacker who is successful in breaking into the server can see all the sensitive data in clear.

One natural solution to the above problem is to keep the data on the server encrypted with the private keys of the users who are permitted to access it. However handling a complex access control policy using traditional public key encryption systems can be difficult. This is because the access policy might be described in terms of the *properties or attributes* that a valid user should have rather than in terms of the actual identities of the users. Thus, a priori, one may not even know the exact list of users authorized to access a particular piece of data.

The concept of attribute based encryption (ABE) was introduced by Sahai and Waters [SW05] as a step towards developing encryption systems with high expressiveness. Goyal et al [GPSW06] further developed this idea and introduced two variants of ABE namely ciphertext-policy attribute based encryption (CP-ABE) and key-policy attribute based encryption (KP-ABE). In a CP-ABE system, a user's private key is associated with a set of attributes (describing the *properties* that the user has) and an encrypted ciphertext will specify an access policy over attributes. A user will be able to decrypt if and only if his attributes satisfy the ciphertext's policy. While a construction of KP-ABE was offered by [GPSW06], constructing CP-ABE was left as an important open problem.

Subsequently to Goyal et al [GPSW06], Bethencourt et al [BSW07] gave the first construction of a CP-ABE system. Their construction however only had a security argument in the generic group model. Cheung and Newport [CN07] recently gave a CP-ABE construction supporting limited type of access structures which could be represented by **AND** of different attributes. Cheung and Newport also discussed the possibility of supporting more general access structures with threshold gates. However as they discuss, a security proof of this generalization would involve overcoming several subtleties. In sum, obtaining a CP-ABE scheme for more advanced access structures based on any (even relatively non-standard) number theoretic assumption has proven to be surprisingly elusive.

Our Results. We present the first construction of a ciphertext-policy attribute based encryption scheme having a security proof based on a standard number theoretic assumption and supporting advanced access structures. Our construction can support access structures which can be represented by a bounded size access tree with threshold gates as its nodes. The bound on the size of the access trees is chosen at the time of the system setup and is represented by a tuple (d, num) where d represents the maximum depth of the access tree and num represents the maximum number of children each non-leaf node of the tree might have. We stress that any access tree satisfying these upper bounds on the size can be dynamically chosen by the encryptor. Our construction has a security proof based on the standard Decisional Bilinear Diffie-Hellman (BDH) assumption. We note that previous CP-ABE systems could either support only very limited access structures [CN07] or had a proof of security only in the generic group model [BSW07] (rather than based on a number theoretic assumption).

Techniques. Our construction can be seen as a way to reinterpret the KP-ABE scheme of [GPSW06] with a fixed "universal" tree access structure as a CP-ABE scheme. Such a reinterpretation does not follow directly because in a KP-ABE scheme, the key material for each attribute is "embedded" into the access structure in a unique way depending on where it occurs in the access policy. To overcome this difficulty, we introduce many "copies" of each attribute for every position in the access structure tree where it can occur. This causes a significant increase in private key size, but does not significantly affect ciphertext size. However, since the actual access structure to be used for a particular ciphertext must be embedded into the fixed "universal" tree access structure in the KP-ABE scheme, this causes a blowup in ciphertext size. This effect can be moderated by having multiple parallel CP-ABE schemes with different sized "universal" tree access structures underlying the scheme, which allows for a trade-off between ciphertext size and the size of the public parameters and private keys.

As a result of the issues discussed above, our scheme has significantly worse efficiency than the scheme of [BSW07]. We leave constructing CP-ABE schemes based on number-theoretic assumptions with better efficiency and unbounded access structures as important open problems.

2 Background

We first give formal definitions for the security of Bounded Ciphertext Policy Attribute Based Encryption (BCP-ABE). Then we give background information on bilinear maps and our cryptographic assumption. Like the work of Goyal et al. [GPSW06], we describe our constructions for access trees. Roughly speaking, given a set of attributes $\{P_1, P_2, \ldots, P_n\}$, an access tree is an access structure $\mathcal{T} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$, where each node in the tree represents a threshold gate (see Section 3 for a detailed description). We note that contrary to the work of Goyal et al., in our definitions, users will be identified with a set of attributes while access trees will be used to specify policies for encrypting data.

A Bounded Ciphertext Policy Attribute Based Encryption scheme consists of four algorithms.

Setup (d, num) This is a randomized algorithm that takes as input the implicit security parameter and a pair of system parameters (d, num). These parameters will be used to restrict the access trees under which messages can be encrypted in our system. It outputs the public parameters PK and a master key MK.

Key Generation (γ, MK) This is a randomized algorithm that takes as input – the master key MK and a set of attributes γ. It outputs a decryption key D corresponding to the attributes in γ.

Encryption $(M, \mathrm{PK}, \mathcal{T}')$ This is a randomized algorithm that takes as input – the public parameters PK, a message M, and an access tree $\mathcal{T}'$ over the universe of attributes, with depth $d' \leq d$, and where each non-leaf node x has at most num child nodes. The algorithm will encrypt M and output the ciphertext E. We will assume that the ciphertext implicitly contains $\mathcal{T}'$.

Decryption (E, D) This algorithm takes as input – the ciphertext E that was encrypted under the access tree $\mathcal{T}'$, and the decryption key D for an attribute set γ. If the set γ of attributes satisfies the access tree $\mathcal{T}'$ (i.e. $\gamma \in \mathcal{T}'$), then the algorithm will decrypt the ciphertext and return a message M.

We now discuss the security of a bounded ciphertext-policy ABE scheme. We define a selective-tree model for proving the security of the scheme under the chosen plaintext attack. This model can be seen as analogous to the selective-ID model [CHK03, CHK04, BB04] used in identity-based encryption (IBE) schemes [Sha84, BF01, Coc01].

Selective-Tree Model for BCP-ABE

Let $\mathcal{U}$ be the universe of attributes that is fixed by the security parameter κ. The system parameters d, num are also defined.

Init The adversary declares the access tree $\mathcal{T}'$, that he wishes to be challenged upon.
Setup The challenger runs the Setup algorithm of ABE and gives the public parameters to the adversary.
Phase 1 The adversary is allowed to issue queries for private keys for many attribute sets γ_j, where γ_j does not satisfy the access tree $\mathcal{T}'$ for all j.
Challenge The adversary submits two equal length messages M_0 and M_1. The challenger flips a random coin b, and encrypts M_b with $\mathcal{T}'$. The ciphertext is passed on to the adversary.
Phase 2 Phase 1 is repeated.
Guess The adversary outputs a guess b' of b.

The advantage of an adversary $\mathcal{A}$ in this game is defined as $\Pr[b' = b] - \frac{1}{2}$.

We note that the model can easily be extended to handle chosen-ciphertext attacks by allowing for decryption queries in Phase 1 and Phase 2.

Definition 1 *A bounded ciphertext-policy attribute-based encryption scheme (BCP-ABE) is secure in the Selective-Tree model of security if all polynomial time adversaries have at most a negligible* *advantage in the Selective-Tree game.*

2.1 Bilinear Maps

We present a few facts related to groups with efficiently computable bilinear maps.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}_1$ and e be a bilinear map, $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. The bilinear map e has the following properties:

1.Bilinearity: for all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, g) \neq 1$.

We say that $\mathbb{G}_1$ is a bilinear group if the group operation in $\mathbb{G}_1$ and the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ are both efficiently computable. Notice that the map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

2.2 The Decisional Bilinear Diffie-Hellman (BDH) Assumption

Let $a, b, c, z \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}_1$. The decisional BDH assumption [BB04, SW05] is that no probabilistic polynomial-time algorithm $\mathcal{B}$ can distinguish the tuple $(A = g^a, B = g^b, C = g^c, e(g, g)^{abc})$ from the tuple $(A = g^a, B = g^b, C = g^c, e(g, g)^z)$ with more than a negligible advantage. The advantage of $\mathcal{B}$ is

$$\left| \Pr[\mathcal{B}(A, B, C, e(g, g)^{abc}) = 0] - \Pr[\mathcal{B}(A, B, C, e(g, g)^z)] = 0 \right|$$

where the probability is taken over the random choice of the generator g, the random choice of a, b, c, z in $\mathbb{Z}_p$, and the random bits consumed by $\mathcal{B}$.

3 Access Trees

In our constructions, user decryption keys will be identified with a set γ of attributes. A party who wishes to encrypt a message will specify through an access tree structure a policy that private keys must satisfy in order to decrypt. We now proceed to explain the access trees used in our constructions.

Access Tree. Let $\mathcal{T}$ be a tree representing an access structure. Each non-leaf node of the tree represents a threshold gate, described by its children and a threshold value. If num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x \leq num_x$. For ease of presentation, we use the term *cardinality* to refer to the number of children of a node. Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$.

Let $\Phi_{\mathcal{T}}$ denote the set of all the non-leaf nodes in the tree $\mathcal{T}$. Further, let $\Psi_{\mathcal{T}}$ be the set of all the non-leaf nodes at depth $d - 1$, where d is the depth of $\mathcal{T}$. To facilitate working with the access trees, we define a few functions. We denote the parent of the node x in the tree by $\text{parent}(x)$. The access tree $\mathcal{T}$ also defines an ordering between the children of every node, that is, the children of a node x are numbered from 1 to num_x. The function $\text{index}(x)$ returns such a number associated with a node x, where the index values are uniquely assigned to nodes in an arbitrary manner for a given access structure. For simplicity, we provision that $\text{index}(x) = \text{att}(x)$, when x is a leaf node and $\text{att}(x)$ is the attribute associated with it.

Satisfying an Access Tree. Let $\mathcal{T}$ be an access tree with root r. Denote by $\mathcal{T}_x$ the subtree of $\mathcal{T}$ rooted at the node x. Hence $\mathcal{T}$ is the same as $\mathcal{T}_r$. If a set of attributes γ satisfies the access tree $\mathcal{T}_x$, we denote it as $\mathcal{T}_x(\gamma) = 1$. We compute $\mathcal{T}_x(\gamma)$ recursively as follows. If x is a non-leaf node, evaluate $\mathcal{T}_z(\gamma)$ for all children z of node x. $\mathcal{T}_x(\gamma)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $\mathcal{T}_x(\gamma)$ returns 1 iff $\text{att}(x) \in \gamma$.

This completes the basic description of access trees. The rest of this section further discusses some terminology and functions related to access trees to be used throughout the paper. However, the reader may choose to skip it for now, and come back to it later when needed.

Universal Access Tree. Given a pair of integer values (d, num), define a complete num-ary tree $\mathcal{T}$ of depth d, where each non-leaf node has a threshold value of num. The leaf nodes in $\mathcal{T}$ are empty, i.e., no attributes are assigned to the leaf nodes. Next, $num - 1$ new leaf nodes are attached to each non-leaf node x, thus increasing the cardinality of x to $2 \cdot num - 1$ while the threshold value num is left intact. Choose an arbitrary assignment of *dummy* attributes (explained later in Section 4) to these newly added leaf nodes[1] for each x. The resultant tree $\mathcal{T}$ is called a (d, num)-universal access tree (or simply the universal access tree when d, num are fixed by the system).

Bounded Access Tree. We say that $\mathcal{T}'$ is a (d, num)-bounded access tree if it has a depth $d' \leq d$, and each non-leaf node in $\mathcal{T}'$ exhibits a cardinality at most num.

Normal Form. Consider a (d, num)-bounded access tree $\mathcal{T}'$. We say that $\mathcal{T}'$ exhibits the (d, num)-normal form if (a) it has a depth $d' = d$, and (b) all the leaf nodes in $\mathcal{T}'$ are at depth d. Any (d, num)-bounded access tree $\mathcal{T}'$ can be converted to the (d, num)-normal form (or simply the normal form when d, num are fixed by the system) in the following way in a top down manner, starting from the root node r'. Consider a node x at level l_x in $\mathcal{T}'$. If the depth d_x of the subtree $\mathcal{T}'_x$ is less than $(d - l_x)$, then insert a vertical chain of $(d - l_x - d_x)$ nodes (where each node has cardinality 1 and threshold 1) between x and $\text{parent}(x)$. Repeat the procedure recursively for each child of x. Note that conversion to the normal form does not affect the satisfying logic of an access tree.

Map between Access Trees. Consider a (d, num)-universal access tree $\mathcal{T}$ and another tree $\mathcal{T}'$ that exhibits the (d, num)-normal form. A map between the nodes of $\mathcal{T}'$ and $\mathcal{T}$ is defined in the following way in a top-down manner. First, the root of $\mathcal{T}'$ is mapped to the root of $\mathcal{T}$. Now suppose that a node x' in $\mathcal{T}'$ is mapped to a node x in $\mathcal{T}$. Let $z'_1, \ldots, z'_{num_{x'}}$ be the child nodes of x', ordered according to their index values. Then, for each child node z'_i ($i \in [1, num_{x'}]$) of x' in $\mathcal{T}'$, set the corresponding child node z_i (i.e. with index value $\text{index}(z'_i)$) of x in $\mathcal{T}$ as the map of z'. This procedure is performed recursively, until each node in $\mathcal{T}'$ is mapped to a corresponding node in $\mathcal{T}$. To capture the above node mapping procedure between $\mathcal{T}'$ to $\mathcal{T}$, we define a public function $\text{map}(\cdot)$ that takes a node (or a set of nodes) in $\mathcal{T}'$ as input and returns the corresponding node (or a set of nodes) in $\mathcal{T}$.

We now proceed to describe our constructions. We first consider the case when the attributes in the system are drawn from a fixed-size universe. Later, we will consider the more general setting where the attributes in the system are not restricted to a fixed set.

[1]From now onwards, by *dummy nodes*, we shall refer to the leaf nodes with dummy attributes associated with them.

4 Small Universe Construction

Before we explain the details of our construction, we first present a brief overview highlighting the main intuitions behind our approach.

4.1 Overview of Our Construction

Fixed Tree Structure. For simplicity, let us first consider a very simple and basic access tree $\mathcal{T}$. The tree $\mathcal{T}$ has depth, say, d; and all leaf nodes in the tree are at depth d. Each non-leaf node x is a "k_x-out-of-num_x" threshold gate where k_x and num_x are fixed beforehand. Thus the "structure" of the access tree is *fixed*. However, the leaf nodes of $\mathcal{T}$ are "empty", i.e., no attributes are associated with them. At the time of encryption, an encryptor will assign attributes to each leaf-node, in order to define the access structure completely. That is, once the encryptor assigns an attribute to each leaf node in $\mathcal{T}$, it fixes the set of "authorized sets of attributes". A user having keys corresponding to an authorized set will be able to decrypt a message encrypted under the above access structure.

Recall that $\Psi_{\mathcal{T}}$ denotes the set of non-leaf nodes at depth $d-1$. Each child node z of an $x \in \Psi_{\mathcal{T}}$ will be assigned an attribute j.[2] However, the same j may be assigned to z_1 as well as z_2 where z_1 is a child node of $x_1 \in \Psi_{\mathcal{T}}$, and z_2 is a child node of $x_2 \in \Psi_{\mathcal{T}}$. Thus, any given attribute j may have at most $|\Psi_{\mathcal{T}}|$ distinct parent nodes. Intuitively, these are all the distinct positions under which j can appear as an attribute of a leaf in the tree $\mathcal{T}$. Now, during system setup, we will publish a unique public parameter corresponding to each such appearance of j, for each attribute j. Next, consider a user A with an attribute set γ. Imagine an access tree $\mathcal{T}'$ that has the same "structure" as $\mathcal{T}$, but where additionally each attribute $j \in \gamma$ is attached to a distinct leaf child of each node $x \in \Psi_{\mathcal{T}'}$. A will be assigned a private key that is computed for such an access tree $\mathcal{T}'$ as in the KP-ABE construction of Goyal et al [GPSW06]. Now, suppose that an encryptor $\mathcal{E}$ has chosen an assignment of attributes to the leaf nodes in $\mathcal{T}$ to define it completely. Let $f(j,x)$ be a function that outputs 1 if an attribute j is associated with a leaf child of x and 0 otherwise. Then, $\mathcal{E}$ will compute (using the public parameters published during system setup) and release a ciphertext component $E_{j,x}$ corresponding to an attribute j attached to a leaf child of $x \in \Psi_{\mathcal{T}}$ (i.e., iff $f(j,x) = 1$). A receiver who possesses an authorized set of attributes for the above tree can choose from his private key - the components $D_{j,x}$, such that $f(j,x) = 1$; and use them with corresponding $E_{j,x}$ during the decryption process.

Varying the Thresholds. The above system, although dynamic, may be very limited in its expressibility for some applications. In order to make it more expressible, we can further extend the above system as follows. At the time of encryption, an encryptor is now given the flexibility of choosing the threshold value between 1 and some maximum fixed value, (say) *num* for each node x in the access tree.

As a first step, we will construct $\mathcal{T}$ as a complete *num*-ary tree of depth d, where each non-leaf node is a "*num*-out-of-*num*" threshold gate. As earlier, the leaf nodes in the tree are empty. Next, we introduce a $(num-1)$-sized set of special attributes called *dummy* attributes that are different from the usual attributes (which we will henceforth refer to as *real* attributes[3]). Now, attach

[2]Note that it is useless to allow an encryptor to assign the same attribute j to two child nodes z_1, z_2 of a given x. So we assume, w.l.o.g., that for a given x, an attribute j can be assigned to only one of its child nodes.

[3]As the name suggests, we will identify users with a set of "real" attributes, while the dummy attributes will be used for technical purposes, i.e., varying the threshold of the nodes when needed.

$(num-1)$ leaf nodes to each $x \in \Phi_{\mathcal{T}}$, and assign a dummy attribute to each such newly-added leaf node (henceforth referred to as dummy nodes).

Note that a dummy attribute j may have atmost $|\Phi_{\mathcal{T}}|$ parent nodes. Intuitively, these are all the distinct positions where j can appear as an attribute of a dummy leaf in $\mathcal{T}$. Therefore, during the system setup, for each dummy attribute j, we will publish a unique public parameter corresponding to *each appearance* of j (in addition to the public parameters corresponding to the real attributes as in the previous description). Next, consider a user A with an attribute set γ. Imagine an access tree $\mathcal{T}'$ that is similar to $\mathcal{T}$, but where additionally each attribute $j \in \gamma$ is attached to a distinct leaf child of each node $x \in \Psi_{\mathcal{T}'}$. A will be assigned a private key that is computed for such an access tree $\mathcal{T}'$ as in the KP-ABE construction of Goyal et al [GPSW06] (the difference from the previous description for fixed trees is that here A will additionally receive key-components corresponding to dummy attributes). Now, at the time of encryption, an encryptor E will first choose a threshold value $k_x \leq num$ for each $x \in \Phi_{\mathcal{T}}$. Next, E will choose an assignment of real attributes to the leaf nodes in $\mathcal{T}$, and an arbitrary $(num - k_x)$-sized subset ω_x of dummy child nodes of each $x \in \Phi_{\mathcal{T}}$. Finally, E will release the ciphertext components as in the previous description. E will additionally release a ciphertext component corresponding to each dummy node in ω_x, for each $x \in \Phi_{\mathcal{T}}$. Now, consider a receiver with an attribute set γ. For any $x \in \Phi_{\mathcal{T}}$, if k_x children of x can be satisfied with γ, then the receiver can use the key-components (from his private key) corresponding to the dummy attributes in order to satisfy each dummy leaf $z \in \omega_x$; thus satisfying the num-out-of-num threshold gate x.

Varying Tree Depth and Node Cardinality. Finally, we note that the above system can be further extended to allow an encryptor to choose the depth of the access tree and also the cardinality of each node; thus further increasing the expressibility of the system. To do this, we will assume an upper bound on the maximum tree depth d and the maximum node cardinality num, fixed beforehand. We will then make use of the techniques presented in the latter part of Section 3, to achieve the desired features. Details are given in the construction.

4.2 The Construction

Let $\mathbb{G}_1$ be a bilinear group of prime order p, and let g be a generator of $\mathbb{G}_1$. In addition, let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ denote the bilinear map. A security parameter, κ, will determine the size of the groups. We also define the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set, S, of elements in $\mathbb{Z}_p$: $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$. We will associate each attribute with a unique element in $\mathbb{Z}_p^*$. Our construction follows.

Setup (d, num) This algorithm takes as input two system parameters, namely, (a) the maximum tree depth d, and (b) the maximum node cardinality num. The algorithm proceeds as follows. Define the universe of real attributes $\mathcal{U} = \{1, \ldots, n\}$, and a $(num-1)$-sized universe of dummy attributes[4] $\mathcal{U}^* = \{n+1, \ldots, n+num-1\}$. Next, define a (d, num)-universal access tree $\mathcal{T}$ as explained in the section 3. In the sequel, $d, num, \mathcal{U}, \mathcal{U}^*, \mathcal{T}$ will all be assumed as implicit inputs to all the procedures.

Now, for each real attribute $j \in \mathcal{U}$, choose a set of $|\Psi_{\mathcal{T}}|$ numbers $\{t_{j,x}\}_{x \in \Psi_{\mathcal{T}}}$ uniformly at random from $\mathbb{Z}_p$. Further, for each dummy attribute $j \in \mathcal{U}^*$, choose a set of $|\Phi_{\mathcal{T}}|$ numbers $\{t^*_{j,x}\}_{x \in \Phi_{\mathcal{T}}}$ uniformly at random from $\mathbb{Z}_p$. Finally, choose y uniformly at random in $\mathbb{Z}_p$. The public parameters

[4]Recall the distinction between real attributes and dummy attributes that was introduced in the Overview section.

PK are:

$$Y = e(g,g)^y,\ \{T_{j,x} = g^{t_{j,x}}\}_{j\in\mathcal{U},x\in\Psi_{\mathcal{T}}},\ \{T^*_{j,x} = g^{t^*_{j,x}}\}_{j\in\mathcal{U}^*,x\in\Phi_{\mathcal{T}}}$$

The master key MK is:

$$y,\ \{t_{j,x}\}_{j\in\mathcal{U},x\in\Psi_{\mathcal{T}}},\ \{t^*_{j,x}\}_{j\in\mathcal{U}^*,x\in\Phi_{\mathcal{T}}}$$

Key Generation (γ, MK) Consider a user A with an attribute set γ. The key generation algorithm outputs a private key D that enables A to decrypt a message encrypted under a (d, num)-bounded access tree $\mathcal{T}'$ iff $\mathcal{T}'(\gamma) = 1$.

The algorithm proceeds as follows. For each user, choose a random polynomial q_x for each non-leaf node x in the universal access tree $\mathcal{T}$. These polynomials are chosen in the following way in a top-down manner, starting from the root node r. For each x, set the degree c_x of the polynomial q_x to be one less than the threshold value, i.e., $c_x = num - 1$. Now, for the root node r, set $q_r(0) = y$ and choose c_r other points of the polynomial q_r randomly to define it completely. For any other non-leaf node x, set $q_x(0) = q_{\mathrm{parent}(x)}(\mathrm{index}(x))$ and choose c_x other points randomly to completely define q_x. Once the polynomials have been decided, give the following secret values to the user:

$$\{D_{j,x} = g^{\frac{q_x(j)}{t_{j,x}}}\}_{j\in\gamma,x\in\Psi_{\mathcal{T}}},\ \{D^*_{j,x} = g^{\frac{q_x(j)}{t^*_{j,x}}}\}_{j\in\mathcal{U}^*,x\in\Phi_{\mathcal{T}}}$$

The set of above secret values is the decryption key D.

Encryption $(M, \mathrm{PK}, \mathcal{T}')$ To encrypt a message $M \in \mathbb{G}_2$, the encrypter $\mathcal{E}$ first chooses a (d, num)-bounded access tree $\mathcal{T}'$. $\mathcal{E}$ then chooses an assignment of real attributes to the leaf nodes in $\mathcal{T}'$.

Now, to be able to encrypt the message M with the access tree $\mathcal{T}'$, the encrypter first converts it to the normal form (if required). Next, $\mathcal{E}$ defines a map between the nodes in $\mathcal{T}'$ and the universal access tree $\mathcal{T}$ as explained in section 3. Finally, for each non-leaf node x in $\mathcal{T}'$, $\mathcal{E}$ chooses an arbitrary $(num - k_x)$-sized set ω_x of dummy child nodes of $\mathrm{map}(x)$ in $\mathcal{T}$.

Let $f(j,x)$ be a boolean function such that $f(j,x) = 1$ if a real attribute $j \in \mathcal{U}$ is associated with a leaf child of node $x \in \Psi_{\mathcal{T}'}$ and 0 otherwise. Now, choose a random value $s \in \mathbb{Z}_p$ and publish the ciphertext as:

$$E = (\mathcal{T}',\ E' = M\cdot Y^s,\ \{E_{j,x} = T^s_{j,\mathrm{map}(x)}\}_{j\in\mathcal{U},x\in\Psi_{\mathcal{T}'}:f(j,x)=1},\ \{E^*_{j,x} = T^{*s}_{j,\mathrm{map}(x)}\}_{j=\mathrm{att}(z):z\in\omega_x,x\in\Phi_{\mathcal{T}'}})$$

Decryption (E, D) We specify our decryption procedure as a recursive algorithm. For ease of exposition, we present the simplest form of the decryption algorithm here. The performance of the decryption procedure can potentially be improved by using the techniques explained in [GPSW06].

We define a recursive algorithm $\mathrm{DecryptNode}(E, D, x)$ that takes as input the ciphertext E, the private key D, and a node x in $\mathcal{T}'$. It outputs a group element of $\mathbb{G}_2$ or $\perp$. First, we consider the case when x is a leaf node. Let $j = \mathrm{att}(x)$ and w be the parent of x.

$$\mathrm{DecryptNode}(E,D,x) = \begin{cases} e(D_{j,\mathrm{map}(w)}, E_{j,w}) = e(g^{\frac{q_{\mathrm{map}(w)}(j)}{t_{j,\mathrm{map}(w)}}}, g^{s\cdot t_{j,\mathrm{map}(w)}}) = e(g,g)^{s\cdot q_{\mathrm{map}(w)}(j)} & \text{if } j\in\gamma \\ \perp \text{ otherwise} \end{cases}$$

We now consider the recursive case when x is a non-leaf node in $\mathcal{T}'$. The algorithm proceeds as follows: For all nodes z that are children of x, it calls $\mathrm{DecryptNode}(E, D, z)$ and stores the output as F_z. Additionally, for each dummy node $z \in \omega_x$ (where ω_x is a select set of dummy nodes of $\mathrm{map}(x)$ in $\mathcal{T}$ chosen by the encrypter), it invokes a function $\mathrm{DecryptDummy}(E, D, z)$ that is defined below, and stores the output as F_z. Let j be the dummy attribute associated with z. Then, we have:

$$\mathrm{DecryptDummy}(E,D,z) = e(D^*_{j,\mathrm{map}(x)}, E^*_{j,x}) = e(g^{\frac{q_{\mathrm{map}(x)}(j)}{t^*_{j,\mathrm{map}(x)}}}, g^{s\cdot t^*_{j,\mathrm{map}(x)}}) = e(g,g)^{s\cdot q_{\mathrm{map}(x)}(j)}\ .$$

Let Ω_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. Further, let S_x be the union of the sets Ω_x and ω_x. Thus we have that $|S_x| = num$. If no k_x-sized set Ω_x exists, then the node x was not satisfied and the function returns $\perp$. Otherwise, we compute:

$$\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \qquad \text{where } \substack{i=\text{att}(z) \text{ if } z \text{ is a leaf node} \\ i=\text{index}(\text{map}(z)) \text{ otherwise} \\ S'_x=\{i:z\in S_x\}} \\
&= \prod_{z \in \Omega_x} F_z^{\Delta_{i,S'_x}(0)} \prod_{z \in \omega_x} F_z^{\Delta_{i,S'_x}(0)} \\
&= \begin{cases} \prod_{z\in\Omega_x} (e(g,g)^{s\cdot q_{\text{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} \prod_{z\in\omega_x} (e(g,g)^{s\cdot q_{\text{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} & \text{if } x \in \Psi_{\mathcal{T}'} \\ \prod_{z\in\Omega_x} (e(g,g)^{s\cdot q_{\text{map}(z)}(0)})^{\Delta_{i,S'_x}(0)} \prod_{z\in\omega_x} (e(g,g)^{s\cdot q_{\text{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} & \text{otherwise} \end{cases} \\
&= \begin{cases} \prod_{z\in S_x} e(g,g)^{s\cdot q_{\text{map}(x)}(i)\cdot\Delta_{i,S'_x}(0)} & \text{if } x \in \Psi_{\mathcal{T}'} \\ \prod_{z\in\Omega_x} (e(g,g)^{s\cdot q_{\text{map}(\text{parent}(z))}(\text{index}(\text{map}(z)))})^{\Delta_{i,S'_x}(0)} \prod_{z\in\omega_x} (e(g,g)^{s\cdot q_{\text{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} & \text{otherwise} \end{cases} \\
&= \prod_{z \in S_x} e(g,g)^{s\cdot q_{\text{map}(x)}(i)\cdot\Delta_{i,S'_x}(0)} \\
&= e(g,g)^{s\cdot q_{\text{map}(x)}(0)} \qquad \text{(using polynomial interpolation)}
\end{aligned}$$

and return the result.

Now that we have defined the function DecryptNode, the decryption algorithm simply calls the function on the root r' of the access structure $\mathcal{T}'$. We observe that DecryptNode$(E, D, r') = e(g,g)^{sy}$ iff $\mathcal{T}'(\gamma) = 1$ (note that $F_{r'} = e(g,g)^{s\cdot q_{\text{map}(r')}(0)} = e(g,g)^{s\cdot q_r(0)} = e(g,g)^{sy}$, where r is the root of the universal access tree $\mathcal{T}$). Since $E' = M \cdot e(g,g)^{sy}$, the decryption algorithm simply divides out $e(g,g)^{sy}$ and recovers the message M.

Remark 1 *We note that in a given system, in order to allow encryption with large and highly expressive access trees, the system parameters* (d, num) *must be set to large values. However, it may often be the case that an access tree chosen by an encryptor in the system is too small in comparison to the universal access tree fixed by the system parameters* (d, num)*. This means that the ciphertexts computed with such access trees will have unnecessarily high ciphertext overhead (due to the ciphertext components corresponding to the dummy attributes). We note that this problem can be mitigated by using techniques similar to that described by Ostrovsky et al.* [OSW07]*. Roughly speaking, we can create a system that uses (say) n parallel encryption systems, where each encryption system is setup for a specific value pair* (d_i, num_i)*. An encryptor can now use the encryption system closest to the access tree chosen for encryption, thus limiting the ciphertext overhead. For details, see* [OSW07].

4.3 Proof of Security

We prove that the security of our scheme in the Selective-Tree model reduces to the hardness of the Decisional BDH assumption.

Theorem 1 *If an adversary can break our scheme in the Selective-Tree model, then a simulator can be constructed to play the Decisional BDH game with a non-negligible advantage.*

PROOF: See Appendix A.

5 Non-Monotonic Access Trees

One fundamental limitation of our original construction is that it does not support *negative* constraints in a ciphertext's access formula. Ostrovsky et al [OSW07] gave the first construction of a KP-ABE scheme that supports non-monotonic access structures. We adapt their techniques to our setting in order to allow an encrypter to use non-monotonic ciphertext policies.

Small Universe. With some minor modifications to our small universe construction, we can support non-monotonic access trees. We now proceed to discuss the modified construction. Since most of the details follow from the original construction, we only highlight the modifications.

Let (d, num) define the bound on access trees in the system. During the system setup, we will publish a $|\mathcal{U}|$-sized universe $\bar{\mathcal{U}}$ of *negative* attributes, where each $\bar{j} \in \bar{\mathcal{U}}$ corresponds to the **NOT** of a distinct real attribute $j \in \mathcal{U}$. Note that this will double the total number of real attributes in the system. Now, consider a user A in the original construction with an attribute set γ'. In the modified construction, in addition to each $j \in \gamma'$, we will associate the negative attribute $\bar{j} \in \bar{\mathcal{U}}$ corresponding to the **NOT** of each $j \in \{\mathcal{U} \setminus \gamma'\}$ with the user A. In this manner, $|\mathcal{U}|$ number of attributes will now be associated with each user in the system. Let γ be the $|\mathcal{U}|$-sized attribute set associated with A. A will be assigned a private key corresponding to γ as in the original construction (each negative attribute in γ is treated as an ordinary attribute).

As pointed out by the authors in [OSW07], by applying DeMorgan's law, we can transform a non-monotonic access tree $\mathcal{T}''$ into $\mathcal{T}'$ so that $\mathcal{T}'$ represents the same access scheme as $\mathcal{T}''$, but has **NOTs** only at the leaves, where the attributes are. Further, we can replace an attribute j with its corresponding negative attribute $\bar{j}$ if the above transformation results in a **NOT** gate at the leaf to which j is associated. We refer the reader to [OSW07] for more details. Now consider a (d, num)-bounded non-monotonic access tree $\mathcal{T}''$ chosen by an encrypter. Using the above mechanism, the encrypter first transforms it to $\mathcal{T}'$ such that the interior gates of $\mathcal{T}'$ consist only of positive threshold gates, while both positive and negative attributes may be associated with the leaf nodes. Now that we have reduced the non-monotonic access tree to a monotonic access tree, the encryption and decryption procedure follow as in the original construction. This completes the description of the modified construction.

Supporting any Access Formula of Bounded Polynomial Size. It is known that any access formula can be represented by a non-monotonic NC^1 circuit. It is intuitive to see that any circuit of logarithmic depth can be converted to a tree with logarithmic depth. To this end, we note that our modified construction for non-monotonic access trees can support any access formula of bounded polynomial size.

6 Large Universe Construction

In our previous construction, the size of public parameters corresponding to the real attributes grows linearly with the size of the universe of real attributes. Combining the tricks presented in section 4 with those in the large universe construction of Goyal et al. [GPSW06], we construct another scheme that uses all but $(num-1)$ elements of $\mathbb{Z}_p^*$ as real attributes (the $(num-1)$ remaining elements of $\mathbb{Z}_p^*$ will be used as dummy attributes), yet the public parameters corresponding to the real attributes grow only linearly in a parameter n which we fix as the maximum number of leaf child nodes of a node in an access tree we can encrypt under. However, similar to the previous construction, we define a system parameter num for the maximum node cardinality; this implicitly fixes $n = num$.

This construction not only reduces the size of the public parameters but also allows us to apply a collision resistant hash function $H : \{0,1\}^* \to \{\mathbb{Z}_p^* \setminus \mathcal{U}^*\}$ (where $\mathcal{U}^*$ is a $(num-1)$-sized subset of $\mathbb{Z}_p^*$ corresponding to the dummy attributes) and use arbitrary strings, that were not necessarily considered during public key setup, as real attributes. For example we can add any verifiable attribute, such as "Works for NSA", to a user's private key.

6.1 Description

Let $\mathbb{G}_1$ be a bilinear group of prime order p, and let g be a generator of $\mathbb{G}_1$. Additionally, let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ denote the bilinear map. A security parameter, κ, will determine the size of the groups. Also define the Lagrange coefficient $\Delta_{i,S}$ for $i \in \mathbb{Z}_p$ and a set, S, of elements in $\mathbb{Z}_p$, exactly as before. Our construction follows.

Setup (d, num) Similar to the previous construction, this algorithm takes as input two system parameters, namely, (a) the maximum tree depth d, and (b) the maximum node cardinality num. The algorithm proceeds as follows. Define a $(num-1)$-sized universe of dummy attributes $\mathcal{U}^* = \{p - num + 1, \ldots, p-1\}$. The remaining space of $\mathbb{Z}_p^*$ will be used for real attributes. Next, define a (d, num)-universal access tree $\mathcal{T}$.

Choose a random value $y \in \mathbb{Z}_p$ and let $g_1 = g^y$. Now choose a random element g_2 of $\mathbb{G}_1$. Next, for each node $x \in \Psi_\mathcal{T}$, choose $t_{1,x}, \ldots, t_{n+1,x}$ uniformly at random from $\mathbb{G}_1$. Let N be the set $\{1, 2, \ldots, n+1\}$. Define $|\Psi_\mathcal{T}|$ functions T_x, for $x \in \Psi_\mathcal{T}$, as:

$$T_x(X) = g_2^{X^n} \prod_{i=1}^{n+1} t_{i,x}^{\Delta_{i,N}(X)}.$$

Function T_x can be viewed as the function $g_2^{X^n} g^{h_x(X)}$ for some n degree polynomial h_x. Now, for each dummy attribute $j \in \mathcal{U}^*$, choose a $|\Phi_\mathcal{T}|$-sized set $\{t^*_{j,x}\}_{x \in \Phi_\mathcal{T}}$ of random values in $\mathbb{Z}_p$. The public parameters PK are:

$$g_1, g_2, \{t_{1,x}, \ldots, t_{n+1,x}\}_{x \in \Psi_\mathcal{T}}, \{T^*_{j,x} = g^{t^*_{j,x}}\}_{j \in \mathcal{U}^*, x \in \Phi_\mathcal{T}} \ ,$$

and the master key MK is:

$$y, \{t^*_{j,x}\}_{j \in \mathcal{U}^*, x \in \Phi_\mathcal{T}} \ .$$

Key Generation (γ, MK) Consider a user A with an attribute set γ. The key generation algorithm outputs a private key D that enables A to decrypt a message encrypted under a (d, num)-bounded access tree $\mathcal{T}'$ iff $\mathcal{T}'(\gamma) = 1$.

The algorithm proceeds as follows. For each user, choose a random polynomial q_x for each non-leaf node x in the tree $\mathcal{T}$. These polynomials are chosen in the following way, in a top down manner, starting from the root node r. For each x, set degree c_x of the polynomial q_x to be one less than the threshold value, i.e., $c_x = num - 1$. Now, for the root node r, set $q_r(0) = y$ and set d_r other points randomly to completely define q_r. For any other non-leaf node x, set $q_x(0) = q_{\text{parent}(x)}(\text{index}(x))$ and choose c_x other points randomly to completely define q_x. Once the polynomials have been decided, for each attribute $j \in \gamma$, we give the following secret values to the user:

$$D_{j,x} = g_2^{q_x(j)} \cdot T_x(j)^{r_{j,x}}, \quad R_{j,x} = g^{r_{j,x}}, \ \forall\ x \in \Psi_\mathcal{T}$$

where $r_{j,x}$ is chosen uniformly at random from $\mathbb{Z}_p$. Next, for each dummy attribute $j \in \mathcal{U}^*$, we give the following secret values to the user:

$$D^*_{j,x} = g^{\frac{q_x(j)}{t^*_{j,x}}} \ , \ \forall x \in \Phi_\mathcal{T} \ .$$

Encryption $(M, \mathrm{PK}, \mathcal{T}')$ To encrypt a message $M \in \mathbb{G}_2$, the encrypter $\mathcal{E}$ first chooses a (d, num)-bounded access tree $\mathcal{T}'$. $\mathcal{E}$ then chooses an assignment of real attributes to the leaf nodes in $\mathcal{T}'$. If required, the encrypter converts $\mathcal{T}'$ to the normal form. Now, a map between the nodes in $\mathcal{T}'$ and $\mathcal{T}$ is defined, as described earlier. Finally, for each internal node x in $\mathcal{T}'$, $\mathcal{E}$ chooses an arbitrary $(num - k_x)$-sized set ω_x of dummy child nodes of map(x) in $\mathcal{T}$.

Let $f(j,x)$ be a boolean function such that $f(j,x) = 1$ if a real attribute j is associated with a leaf child of a node $x \in \Psi_{\mathcal{T}'}$ in and 0 otherwise. Choose a random value $s \in \mathbb{Z}_p$ and publish the ciphertext as:

$$E = (\mathcal{T}', \mathrm{map}(\cdot), E' = M.e(g_1, g_2)^s, E'' = g^s, \{E_{j,x} = T_x(j)^s\}_{j,x \in \Psi_{\mathcal{T}'}: f(j,x)=1}, \{E^*_{j,x} = T^{*s}_{j,x}\}_{j=\mathrm{att}(z): z \in \omega_x, x \in \Phi_{\mathcal{T}'}})$$

Decryption (E, D) As for the case of small universe, we first define a recursive algorithm DecryptNode(E, D, x) that takes as input the ciphertext E, the private key D, and a node x in the tree. First, we consider the case when x is a leaf node in $\mathcal{T}'$ with a real attribute attached to it. Let $j = \mathrm{att}(x)$ and w be the parent of x.

$$\mathrm{DecryptNode}(E, D, x) = \begin{cases} \frac{e(D_{j,\mathrm{map}(w)}, E'')}{e(R_{j,\mathrm{map}(w)}, E_{j,w})} = \frac{e(g_2^{q_{\mathrm{map}(w)}(j)} \cdot T_{\mathrm{map}(w)}(j)^{r_{j,\mathrm{map}(w)}}, g^s)}{e(g^{r_{j,\mathrm{map}(w)}}, T_{\mathrm{map}(w)}(j)^s)} = e(g, g_2)^{s \cdot q_{\mathrm{map}(w)}(j)} & \text{if } j \in \gamma \\ \perp \text{ otherwise} \end{cases}$$

We now consider the recursive case when x is an internal node in $\mathcal{T}'$. The algorithm proceeds as follows: For all nodes z that are children of x, it calls DecryptNode(E, D, z) and stores the output as F_z. Additionally, for each $z \in \omega_x$ (where ω_x is a select set of dummy nodes of map(x) in $\mathcal{T}$ chosen by the encrypter), it invokes a function DecryptDummy(E, D, z) that is defined below, and stores the output as F_z. Let j be the dummy attribute associated with z. Then, we have:

$$\mathrm{DecryptDummy}(E, D, z) = e(D^*_{j,\mathrm{map}(x)}, E^*_{j,x}) = e(g_2^{\frac{q_{\mathrm{map}(x)}(j)}{t^*_{j,\mathrm{map}(x)}}}, g^{s \cdot t^*_{j,\mathrm{map}(x)}}) = e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(j)} \ .$$

Now, let Ω_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. Further, let S_x be the union of the sets Ω_x and ω_x. Thus we have that $|S_x| = num$. If no k_x-sized set Ω_x exists, then the node x was not satisfied and the function returns $\perp$. Otherwise, we compute:

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \qquad \text{where } \substack{i=\mathrm{att}(z) \text{ if } z \text{ is a leaf node} \\ i=\mathrm{index}(\mathrm{map}(z)) \text{ otherwise} \\ S'_x=\{i:z\in S_x\}} \\ &= \prod_{z \in \Omega_x} F_z^{\Delta_{i,S'_x}(0)} \prod_{z \in \omega_x} F_z^{\Delta_{i,S'_x}(0)} \\ &= \begin{cases} \prod_{z \in \Omega_x} (e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} \prod_{z \in \omega_x} (e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} & \text{if } x \in \Psi_{\mathcal{T}'} \\ \prod_{z \in \Omega_x} (e(g, g_2)^{s \cdot q_{\mathrm{map}(z)}(0)})^{\Delta_{i,S'_x}(0)} \prod_{z \in \omega_x} (e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} & \text{otherwise} \end{cases} \\ &= \begin{cases} \prod_{z \in S_x} e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(i) \cdot \Delta_{i,S'_x}(0)} & \text{if } x \in \Psi_{\mathcal{T}'} \\ \prod_{z \in \Omega_x} (e(g, g_2)^{s \cdot q_{\mathrm{map}(\mathrm{parent}(z))}(\mathrm{index}(\mathrm{map}(z)))})^{\Delta_{i,S'_x}(0)} \prod_{z \in \omega_x} (e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(i)})^{\Delta_{i,S'_x}(0)} & \text{otherwise} \end{cases} \\ &= \prod_{z \in S_x} e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(i) \cdot \Delta_{i,S'_x}(0)} \\ &= e(g, g_2)^{s \cdot q_{\mathrm{map}(x)}(0)} \qquad \text{(using polynomial interpolation)} \end{aligned}$$

and return the result.

Now that we have defined our function DecryptNode, the decryption algorithm simply calls the function on the root r' of the access structure $\mathcal{T}'$. We observe that DecryptNode$(E, D, r') =$

$e(g, g_2)^{sy}$ iff $\mathcal{T}'(\gamma) = 1$ (note that $F_{r'} = e(g, g_2)^{s \cdot q_{\mathrm{map}(r')}(0)} = e(g, g_2)^{s \cdot q_r(0)} = e(g, g_2)^{sy}$, where $r = \mathrm{map}(r')$). Since $E' = M \cdot e(g, g_2)^{sy}$, the decryption algorithm simply divides out $e(g, g_2)^{sy}$ and recovers the message M.

The security proof of this construction follows largely from the security proof of the small universe construction. Details are given in appendix B.

7 Delegation of Private Keys

Similar to the system of Goyal et al. [GPSW06], our constructions come with the added capability of delegation of private keys. Due to lack of space, we only give a brief discussion on how to achieve delegation of private keys in our system.

Delegation of Private Keys in the Small Universe Construction. With a slight modification to our small universe construction, we can enable delegation of private keys by their owners (i.e. the users in the system). However, for proving the security of the modified construction, we will require the following augmented form of the decisional BDH assumption. Let $a, b, c, z \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}_1$. The augmented decisional BDH assumption is that no probabilistic polynomial-time algorithm $\mathcal{B}$ can distinguish the tuple $(A = g^a, B = g^b, C = g^c, C' = g^{1/c}, e(g, g)^{abc})$ from the tuple $(A = g^a, B = g^b, C = g^c, C' = g^{1/c}, e(g, g)^z)$ with more than a negligible advantage.

Consider a private key D corresponding to an attribute set γ. Roughly speaking, the delegation procedure will involve (a) choosing the key parts in D corresponding to the attributes in (say) γ', where $\gamma' \subseteq \gamma$, and (b) re-randomizing them to obtain the private key D'.

During the system setup, we will additionally publish the public parameters $g^{1/t_{j,x}}, \forall j \in \mathcal{U}, x \in \Psi_{\mathcal{T}}$, and $g^{1/t^*_{j,x}}, \forall j \in \mathcal{U}^*, x \in \Phi_{\mathcal{T}}$. Now, recall that during the key generation process, a random polynomial $q_x(X)$ of degree $c_x = num$ was defined (in a specific manner, as explained in the construction) for each non-leaf node x in the universal access tree $\mathcal{T}$. In order to make the key components in D' independent of D, each polynomial q_x must be re-randomized. Re-randomization for a node x with a (known) constant L_x is done as follows. Choose a random polynomial $p_x(X)$ of degree $c_x = num$ such that $p_x(0) = L_x$. Define the new polynomial q'_x as $q'_x(X) = q_x(X) + p_x(X)$. Now, recursively re-randomize every non-leaf child z of x with the constant $L_z = p_x(\mathrm{index}(z))$.

Now, re-randomization of all the nodes is done by simply re-randomizing the root node r of the universal access tree $\mathcal{T}$ with the constant $L_r = 0$. The new private key components corresponding to a real attribute $j \in \gamma'$ are computed as:

$$D'_{j,x} = D_{j,x} \cdot g^{\frac{p_x(j)}{t_{j,x}}}, \forall x \in \Psi_{\mathcal{T}}$$

Further, the new private key components corresponding to a dummy attribute $j \in \mathcal{U}^*$ are computed as:

$$D^{**}_{j,x} = D^*_{j,x} \cdot g^{\frac{p_x(j)}{t^*_{j,x}}}, \forall x \in \Phi_{\mathcal{T}}$$

Note that since we publish additional public parameters during the system setup, the original security proof (based on the decisional BDH assumption) does not go through. However, given the modified decisional BDH assumption, we are able to construct a security proof of our construction. Details will be provided in the full version.

Delegation of Private Keys in the Large Universe Construction. With a slight modification to our large universe construction, we can enable delegation of private keys by their owners (i.e. the users in the system) under the standard decisional BDH assumption.

Recall that all the parameters corresponding to the dummy attributes in the large universe case are exactly similar to that in the small universe construction. Therefore, in order to re-randomize the key components corresponding to the dummy attributes, we would require additional public parameters $g^{1/t^*_{j,x}}, \forall j \in \mathcal{U}^*, x \in \Phi_{\mathcal{T}}$ (and therefore, the augmented decisional BDH assumption to prove security), as in the small universe case described above. In order to get rid of the additional parameters, we can instead create the parameters corresponding to the dummy attributes in a manner similar to that of the real attributes (in the large universe construction). Specifically, we can define a new function $T^*_x(X) = g_2^{X^n} g^{h^*_x(X)}$ by creating a *num*-degree polynomial $h^*_x(X)$, for each $x \in \Phi_{\mathcal{T}}$. Now, the private key components corresponding to the dummy attributes for any user will be computed as:

$$D^*_{j,x} = g_2^{q_x(j)} \cdot T^*_x(j)^{r^*_{j,x}}, \quad R^*_{j,x} = g^{r^*_{j,x}}, \quad \forall x \in \Phi_{\mathcal{T}}$$

where $r^*_{j,x}$ is chosen uniformly at random from $\mathbb{Z}_p$. The ciphertext components corresponding to the dummy attributes will be modified in a similar manner.

Now, consider a private key D corresponding to an attribute set γ. A new private key D' for an attribute set $\gamma' \subseteq \gamma$ can be computed as follows. First re-randomize all the polynomials in the universal access tree as explained above in the small universe case. Now, the new private key components corresponding to a real attribute $j \in \gamma'$ are computed as:

$$D'_{j,x} = D_{j,x} \cdot g_2^{p_x(j)} \cdot T_x(j)^{r'_{j,x}}, \quad R'_{j,x} = R_{j,x} \cdot g^{r'_{j,x}}, \ \forall x \in \Psi_{\mathcal{T}}$$

where $r'_{j,x}$ is chosen randomly. Further, the new private key components corresponding to a dummy attribute $j \in \mathcal{U}^*$ are computed as:

$$D^{**}_{j,x} = D^*_{j,x} \cdot g_2^{p_x(j)} \cdot T_x(j)^{r^{**}_{j,x}}, \quad R^{**}_{j,x} = R^*_{j,x} \cdot g^{r^{**}_{j,x}}, \ \forall x \in \Psi_{\mathcal{T}}$$

where $r^{**}_{j,x}$ is chosen randomly. We note that the security of the above system can be proven under the standard decisional BDH assumption. Details will be given in the full version.

References

[BB04] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In *Advances in Cryptology – Eurocrypt*, volume 3027 of *LNCS*, pages 223–238. Springer, 2004.

[BF01] D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. In *Advances in Cryptology – CRYPTO*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[CHK03] R. Canetti, S. Halevi, and J. Katz. A Forward-Secure Public-Key Encryption Scheme. In *Advances in Cryptology – Eurocrypt*, volume 2656 of *LNCS*. Springer, 2003.

[CHK04] R. Canetti, S. Halevi, and J. Katz. Chosen Ciphertext Security from Identity Based Encryption. In *Advances in Cryptology – Eurocrypt*, volume 3027 of *LNCS*, pages 207–222. Springer, 2004.

[CN07] Ling Cheung and Calvin Newport. Provably Secure Ciphertext Policy ABE. In *ACM conference on Computer and Communications Security (ACM CCS)*, 2007.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[GPSW06] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute Based Encryption for Fine-Grained Access Conrol of Encrypted Data. In *ACM conference on Computer and Communications Security (ACM CCS)*, 2006.

[OSW07] R. Ostrovsky, A. Sahai, and B. Waters. Attribute Based Encryption with Non-Monotonic Access Structures. In *ACM conference on Computer and Communications Security (ACM CCS)*, 2007.

[Sha84] A. Shamir. Identity Based Cryptosystems and Signature Schemes. In *Advances in Cryptology – CRYPTO*, volume 196 of *LNCS*, pages 37–53. Springer, 1984.

[SW05] A. Sahai and B. Waters. Fuzzy Identity Based Encryption. In *Advances in Cryptology – Eurocrypt*, volume 3494 of *LNCS*, pages 457–473. Springer, 2005.

A Proof of Security: Small Universe Construction

Suppose there exists a polynomial-time adversary $\mathcal{A}$, that can attack our scheme in the Selective-Tree model with advantage ϵ. We build a simulator $\mathcal{B}$ that can play the Decisional BDH game with advantage $\epsilon/2$. The simulation proceeds as follows:

We first let the challenger set the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with an efficient bilinear map e, and a generator g for the group $\mathbb{G}_1$. The challenger flips a fair binary coin μ, outside of $\mathcal{B}$'s view. If $\mu = 0$, the challenger sets $(A, B, C, Z) = (g^a, g^b, g^c, e(g,g)^{abc})$; otherwise it sets $(A, B, C, Z) = (g^a, g^b, g^c, e(g,g)^z)$ for random a, b, c, z.

We assume that the maximum tree depth d, the maximum node cardinality num, the universe of real attributes $\mathcal{U}$, and the universe of dummy attributes $\mathcal{U}^*$ are defined for the system. This implicitly defines the universal access tree $\mathcal{T}$ (see Section 3).

Init The simulator $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ chooses a (d, num)-bounded access tree $\mathcal{T}''$ it wishes to be challenged upon. Next, $\mathcal{A}$ assigns real attributes to the leaf nodes to completely define $\mathcal{T}''$.

Recall that the encryption procedure in our construction involves a pre-processing phase, where, (a) an access tree $\mathcal{T}''$ (to be used for encryption) is converted to its normal form (say) $\mathcal{T}'$, (b) a map is defined between the nodes of $\mathcal{T}'$ and the universal access tree $\mathcal{T}$, and (c) for each non-leaf node x in $\mathcal{T}'$, a $(num - k_x)$-sized set ω_x of dummy child nodes of map(x) is selected.

Before the system setup, the simulator executes the pre-processing phase on the access tree $\mathcal{T}''$ received from $\mathcal{A}$. In the sequel, $\mathcal{T}'$ would refer to the normal form of the access tree $\mathcal{T}''$.

Setup The system setup is done based on the access tree $\mathcal{T}'$ and the universal access tree $\mathcal{T}$. First, the simulator sets the parameters corresponding to the real attributes in the following manner. Let $x' \in \Psi_{\mathcal{T}'}$, and $x = \text{map}(x')$ in $\mathcal{T}$. If $f(j, x') = 1$ for $j \in \mathcal{U}$, set $T_{j,x} = g^{r_{j,x}}$ (thus, $t_{j,x} = r_{j,x}$); otherwise choose a random $\beta_{j,x} \in \mathbb{Z}_p$ and set $T_{j,x} = g^{b\beta_{j,x}} = B^{\beta_{j,x}}$ (thus, $t_{j,x} = b\beta_{j,x}$). Now, consider the sets $\Psi_{\mathcal{T}'}$ and $\Psi_{\mathcal{T}}$. Note that $\Psi_{\mathcal{T}} \supseteq \text{map}(\Psi_{\mathcal{T}'})$. For $x \in (\Psi_{\mathcal{T}} \setminus \text{map}(\Psi_{\mathcal{T}'}))$, choose a random $\beta_{j,x} \in \mathbb{Z}_p$ and set $T_{j,x} = g^{b\beta_{j,x}} = B^{\beta_{j,x}}$.

Next, the simulator sets the parameters corresponding to the dummy attributes in the following manner. Let $x' \in \Phi_{\mathcal{T}'}$, and $x = \text{map}(x')$ in $\mathcal{T}$. Recall that $\omega_{x'}$ is the select set of dummy child nodes of x chosen earlier by the simulator. If $j = \text{att}(z)$ for $j \in \mathcal{U}^*$, $z \in \omega_{x'}$, choose a random $r^*_{j,x} \in \mathbb{Z}_p$ and set $T^*_{j,x} = g^{r^*_{j,x}}$; otherwise choose a random $\beta^*_{j,x} \in \mathbb{Z}_p$ and set $T^*_{j,x} = g^{b\beta^*_{j,x}} = B^{\beta^*_{j,x}}$. Now, consider the sets $\Phi_{\mathcal{T}'}$ and $\Phi_{\mathcal{T}}$. Note that $\Phi_{\mathcal{T}} \supseteq \text{map}(\Phi_{\mathcal{T}'})$. For $x \in (\Phi_{\mathcal{T}} \setminus \text{map}(\Phi_{\mathcal{T}'}))$, choose a random $\beta^*_{j,x} \in \mathbb{Z}_p$ and set $T^*_{j,x} = g^{b\beta^*_{j,x}} = B^{\beta^*_{j,x}}$.

Finally, the simulator sets the parameter $Y = e(A, B) = e(g,g)^{ab}$ and gives all the public parameters to $\mathcal{A}$.

Phase 1 $\mathcal{A}$ adaptively makes requests for the keys corresponding to any attribute set γ such that it does not satisfy the challenge tree $\mathcal{T}'$. Now, suppose that $\mathcal{A}$ makes a request for the private key for an attribute set γ where $\mathcal{T}'(\gamma) = 0$. To generate the private key, $\mathcal{B}$ needs to assign a polynomial Q_x of degree $c_x = num - 1$ for every non-leaf node in the access tree $\mathcal{T}$.

For ease of exposition, we define a variant $\hat{\mathcal{T}}$ of the universal access tree $\mathcal{T}$ that carries the same tree "structure" as $\mathcal{T}$, but is different in terms of assignment of attributes to the leaf nodes, explained as follows. First, consider a node $x = \text{map}(x')$ in $\hat{\mathcal{T}}$[5], where $x' \in \Phi_{\mathcal{T}'}$. For each such x in $\hat{\mathcal{T}}$, the dummy attributes are assigned only to the child nodes $z \in \omega_{x'}$ (instead of all the dummy child nodes of x). Now, consider a leaf node $x = \text{map}(x')$ in $\hat{\mathcal{T}}$, where $x' \in \Phi_{\mathcal{T}'}$. Each such leaf node x in $\hat{\mathcal{T}}$ is assigned the real attribute $\text{att}(x')$, where $\text{att}(x')$ was fixed by the adversary $\mathcal{A}$. This completes the description of $\hat{\mathcal{T}}$.

Now, consider a node x' in $\Phi_{\mathcal{T}'}$, and $x = \text{map}(x')$ in $\hat{\mathcal{T}}$. Suppose that $\mathcal{T}'_{x'}(\gamma) = 1$. This implies that $l_{x'} \geq k_{x'}$ child nodes z' of x' must be satisfied. From the construction of $\hat{\mathcal{T}}$, we have that $l_{x'}$ child nodes $z = \text{map}(z')$ of x are satisfied in $\hat{\mathcal{T}}$. Note that $|\omega_{x'}| \geq num - l_{x'}$. Since $|\omega_{x'}|$ dummy child nodes of x can be satisfied using the dummy attributes, we have that $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 1$. On the other hand, $(\mathcal{T}'_{x'}(\gamma) = 0) \Leftrightarrow (\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 0)$. Further, consider a node $x \in (\Phi_{\hat{\mathcal{T}}} \setminus \text{map}(\Phi_{\mathcal{T}'}))$. Since no attributes are assigned to the leaf nodes in the subtree $\hat{\mathcal{T}}_x$, we have that $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 0$.

We now explain how the simulator assigns a polynomial for every non-leaf node in $\hat{\mathcal{T}}$. Note that this is exactly the same as assigning a polynomial for every non-leaf node in $\mathcal{T}$, since $\hat{\mathcal{T}}$ and $\mathcal{T}$ carry the same structure. First we define the following two procedures: PolySat and PolyUnsat.

PolySat$(\hat{\mathcal{T}}_x, \gamma, \lambda_x)$ This procedure sets up the polynomials for the non-leaf nodes of the access sub-tree $\hat{\mathcal{T}}_x$ with a satisfied root node, that is, $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 1$. The procedure takes as input an access tree $\hat{\mathcal{T}}_x$ (with root node x), along with a set of attributes γ and an integer $\lambda_x \in \mathbb{Z}_p$.

It first sets up a polynomial q_x of degree $c_x = num - 1$ for the root node x. It sets $q_x(0) = \lambda_x$ and then sets the rest of the points randomly to completely fix q_x. Now it sets polynomials for each non-leaf child z of x by calling the procedure PolySat$(\hat{\mathcal{T}}_z, \gamma, q_x(\text{index}(z)))$. Notice that in this way, $q_z(0) = q_x(\text{index}(z))$ for each non-leaf child z of x.

PolyUnsat$(\hat{\mathcal{T}}_x, \gamma, g^{\lambda_x})$ This procedure sets up the polynomials for the non-leaf nodes of the access sub-tree $\hat{\mathcal{T}}_x$ with an unsatisfied root node, that is, $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 0$. The procedure takes as input an access tree $\hat{\mathcal{T}}_x$ (with root node x), along with a set of attributes γ and an element $g^{\lambda_x} \in \mathbb{G}_1$ (where $\lambda_x \in \mathbb{Z}_p$).

It first defines a polynomial q_x of degree $c_x = num - 1$ for the root node x such that $q_x(0) = \lambda_x$. We consider the following two cases:

Case 1. $x \in \text{map}(\Phi_{\mathcal{T}'})$

Because $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 0$, which in turn implies that $\mathcal{T}'_{x'}(\gamma) = 0$ (where $x = \text{map}(x')$), note that no more than $k_{x'} - 1$ children of x' in $\mathcal{T}'_{x'}$ are satisfied. Let $l_{x'} \leq k_{x'} - 1$ be the number of satisfied children z' of x'. This implies that exactly $l_{x'}$ number of child nodes $z = \text{map}(z')$ of x are satisfied in $\hat{\mathcal{T}}$. For each satisfied child z of x, the procedure chooses a random point $\lambda_z \in \mathbb{Z}_p$ and sets $q_x(\text{index}(z)) = \lambda_z$. Further, for each dummy node $z \in \omega_{x'}$, the procedure chooses a random point $\lambda_z \in \mathbb{Z}_p$ and sets $q_x(\text{att}(z)) = \lambda_z$. It then fixes the remaining $c_x - (l_{x'} + |\omega_{x'}|)$ points of q_x randomly to completely fix q_x. Hence, note that for each dummy attribute $j \in \mathcal{U}^*$, $q_x(j)$ is known if $j = \text{att}(z) : z \in \omega_{x'}$; otherwise only $g^{q_x(j)}$ can be obtained by interpolation as only $g^{q_x(0)}$ is known in this case. Now the algorithm recursively defines

[5]Since $\hat{\mathcal{T}}$ and $\mathcal{T}$ exhibit the same tree structure, given a map between the nodes of $\mathcal{T}'$ and $\mathcal{T}$, we can construct an identical map between the nodes of $\mathcal{T}'$ and $\hat{\mathcal{T}}$.

polynomials for the rest of the non-leaf nodes in the sub-tree as follows. For each non-leaf child node z of x, the algorithm calls:

- PolySat$(\hat{\mathcal{T}}_z, \gamma, q_x(\text{index}(z)))$, if z is a satisfied node. Notice that $q_x(\text{index}(z))$ is known in this case.
- PolyUnsat$(\hat{\mathcal{T}}_z, \gamma, g^{q_x(\text{index}(z))})$, if z is not a satisfied node. Notice that only $g^{q_x(\text{index}(z))}$ can be obtained by interpolation as only $g^{q_x(0)}$ is known in this case.

Case 2. $x \in (\Phi_{\hat{\mathcal{T}}} \setminus \text{map}(\Phi_{\mathcal{T}'}))$

In this case, the procedure simply sets the rest of the points randomly to completely fix q_x. Now, it recursively defines the polynomials for each non-leaf child z by calling the procedure PolyUnsat$(\hat{\mathcal{T}}_z, \gamma, g^{q_x(\text{index}(z))})$. Notice that only $g^{q_x(\text{index}(z))}$ can be obtained by interpolation as only $g^{q_x(0)}$ is known in this case. Similarly, for each dummy attribute $j \in \mathcal{U}^*$, only $g^{q_x(j)}$ can be obtained by interpolation as only $g^{q_x(0)}$ is known in this case.

Notice that in this case also, $q_z(0) = q_x(\text{index}(z))$ for each non-leaf child node z of x.

Now, the simulator first runs PolyUnsat$(\hat{\mathcal{T}}, \gamma, A)$ to define a polynomial q_x for each node $x \in (\Phi_{\hat{\mathcal{T}}} \setminus \Psi_{\hat{\mathcal{T}}})$. Therefore, we have that $q_r(0) = a$, where r is the root of $\hat{\mathcal{T}}$. Simulator now defines the final polynomial $Q_x(\cdot) = bq_x(\cdot)$ for each x. Notice that this sets $y = Q_r(0) = ab$. Finally, the keys for each real attribute $j \in \gamma$, and each dummy attribute $j \in \mathcal{U}^*$ are given, as explained below.

The simulator first defines the polynomials for the remaining non-leaf nodes in $\hat{\mathcal{T}}$, i.e. $x \in \Psi_{\hat{\mathcal{T}}}$. We consider the following two cases:
Case 1. $x \in \text{map}(\Psi_{\mathcal{T}'})$

If x is satisfied, i.e. $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 1$, then $q_x(0)$ is known. In this case, the simulator simply sets the rest of the points randomly to completely fix q_x. Otherwise, if x is not satisfied, i.e. $\hat{\mathcal{T}}_x(\gamma \cup \mathcal{U}^*) = 0$, which in turn implies that $\mathcal{T}'_{x'}(\gamma) = 0$ (where $x = \text{map}(x')$), no more than $k_{x'} - 1$ leaf child nodes of x' are satisfied. Let $l_{x'} \leq k_{x'} - 1$ be the number of satisfied children z of x' in $\mathcal{T}'$. This implies that exactly $l_{x'}$ number of (real) leaf child nodes $z = \text{map}(z')$ of x are satisfied in $\hat{\mathcal{T}}$. For each satisfied child z of x, the simulator chooses a random point $\lambda_z \in \mathbb{Z}_p$ and sets $q_x(\text{att}(z)) = \lambda_z$. Simularly, for each dummy node $z \in \omega_{x'}$, the simulator chooses a random point $\lambda_z \in \mathbb{Z}_p$ and sets $q_x(\text{att}(z)) = \lambda_z$. It then fixes the remaining $c_x - (l_{x'} + |\omega_{x'}|)$ points of q_x randomly to completely fix q_x. Then, for each real attribute $j \in \gamma$, we have:

$$D_{j,x} = \begin{cases} g^{\frac{Q_x(j)}{t_{j,x}}} = g^{\frac{bq_x(j)}{r_{j,x}}} = B^{\frac{q_x(j)}{r_{j,x}}} & \text{if } f(j,x) = 1 \\ g^{\frac{Q_x(j)}{t_{j,x}}} = g^{\frac{bq_x(j)}{b\beta_{j,x}}} = g^{\frac{q_x(j)}{\beta_{j,x}}} & \text{otherwise} \end{cases}$$

Case 2. $x \in (\Psi_{\hat{\mathcal{T}}} - \text{map}(\Psi_{\mathcal{T}'}))$

In this case, x is not satisfied, hence only $g^{q_x(0)}$ is known. Now, the simulator simply sets the rest of the points randomly to completely fix q_x. Then, for each real attribute $j \in \gamma$, we have:

$$D_{j,x} = g^{\frac{Q_x(j)}{t_{j,x}}} = g^{\frac{bq_x(j)}{b\beta_{j,x}}} = g^{\frac{q_x(j)}{\beta_{j,x}}}$$

Finally, the key parts for each dummy attribute $j \in \mathcal{U}^*$ are computed in the following manner. We consider the following two cases:
Case 1. $x \in \text{map}(\Phi_{\mathcal{T}'})$

$$D^*_{j,x} = \begin{cases} g^{\frac{Q_x(j)}{t^*_{j,x}}} = g^{\frac{bq_x(j)}{r^*_{j,x}}} = B^{\frac{q_x(j)}{r^*_{j,x}}} & \text{if } j = \text{att}(z) : z \in \omega_x \\ g^{\frac{Q_x(j)}{t^*_{j,x}}} = g^{\frac{bq_x(j)}{b\beta^*_{j,x}}} = g^{\frac{q_x(j)}{\beta^*_{j,x}}} & \text{otherwise} \end{cases}$$

Case 2. $x \in (\Phi_{\hat{\mathcal{T}}} - \text{map}(\Phi_{\mathcal{T}'}))$

$$D^*_{j,x} = g^{\frac{Q_x(j)}{t^*_{j,x}}} = g^{\frac{bq_x(j)}{b\beta^*_{j,x}}} = g^{\frac{q_x(j)}{\beta^*_{j,x}}}$$

Therefore, the simulator is able to construct a private key for the attribute set γ. Furthermore, the distribution of the private key for γ is identical to that in the original scheme.

Challenge The adversary $\mathcal{A}$, will submit two challenge messages m_0 and m_1 to the simulator. The simulator flips a fair binary coin ν, and returns an encryption of m_ν. The ciphertext is output as:

$$E = (\mathcal{T}',\ \text{map}(\cdot),\ E' = m_\nu Z,\ \{E_{j,x} = C^{r_{j,\text{map}(x)}}\}_{j \in \mathcal{U}, x \in \Psi_{\mathcal{T}'} : f(j,x)=1},\ \{E^*_{j,x} = C^{r^*_{j,\text{map}(x)}}\}_{x \in \Phi_{\mathcal{T}'}, j=\text{att}(z): z \in \omega_x})$$

If $\mu = 0$ then $Z = e(g,g)^{abc}$. If we let $s = c$, then we have $Y^s = (e(g,g)^{ab})^c = e(g,g)^{abc}$, $E_{j,x} = (g^c)^{r_{j,\text{map}(x)}} = T^c_{j,\text{map}(x)}$, and $E^*_{j,x} = (g^c)^{r^*_{j,\text{map}(x)}} = T^{*c}_{j,\text{map}(x)}$. Therefore, the ciphertext is a valid random encryption of message m_ν.

Otherwise, if $\mu = 1$, then $Z = e(g,g)^z$. We then have $E' = m_\nu e(g,g)^z$. Since z is random, E' will be a random element of $\mathbb{G}_2$ from the adversaries view and the message contains no information about m_ν.

Phase 2 The simulator acts exactly as it did in Phase 1.

Guess $\mathcal{A}$ will submit a guess ν' of ν. If $\nu' = \nu$ the simulator will output $\mu' = 0$ to indicate that it was given a valid BDH-tuple otherwise it will output $\mu' = 1$ to indicate it was given a random 4-tuple.

As shown in the construction the simulator's generation of public parameters and private keys is identical to that of the actual scheme.

In the case where $\mu = 1$ the adversary gains no information about ν. Therefore, we have $\Pr[\nu \neq \nu' | \mu = 1] = \frac{1}{2}$. Since the simulator guesses $\mu' = 1$ when $\nu \neq \nu'$, we have $\Pr[\mu' = \mu | \mu = 1] = \frac{1}{2}$.

If $\mu = 0$ then the adversary sees an encryption of m_ν. The adversary's advantage in this situation is ϵ by definition. Therefore, we have $\Pr[\nu = \nu' | \mu = 0] = \frac{1}{2} + \epsilon$. Since the simulator guesses $\mu' = 0$ when $\nu = \nu'$, we have $\Pr[\mu' = \mu | \mu = 0] = \frac{1}{2} + \epsilon$.

The overall advantage of the simulator in the Decisional BDH game is $\frac{1}{2}\Pr[\mu' = \mu | \mu = 0] + \frac{1}{2}\Pr[\mu' = \mu | \mu = 1] - \frac{1}{2} = \frac{1}{2}(\frac{1}{2} + \epsilon) + \frac{1}{2}\frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$.

B Proof of Security: Large Universe Construction

We prove that the security of large universe construction in the attribute-based Selective-Tree model reduces to the hardness of the Decisional BDH assumption.

Theorem 2 *If an adversary can break our scheme in the Attribute-based Selective-Tree model, then a simulator can be constructed to play the Decisional BDH game with a non-negligible advantage.*

PROOF: Suppose there exists a polynomial-time adversary $\mathcal{A}$, that can attack our scheme in the Selective-Tree model with advantage ϵ. We build a simulator $\mathcal{B}$ that can play the Decisional BDH game with advantage $\epsilon/2$. The simulation proceeds as follows:

We first let the challenger set the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ with an efficient bilinear map e, and a generator g of $\mathbb{G}_1$. The challenger flips a fair binary coin μ, outside of $\mathcal{B}$'s view. If $\mu = 0$, the challenger sets $(A, B, C, Z) = (g^a, g^b, g^c, e(g,g)^{abc})$; otherwise it sets $(A, B, C, Z) = (g^a, g^b, g^c, e(g,g)^z)$ for random a, b, c, z.

We assume that the maximum tree depth d, the maximum node cardinality num, and the universe of dummy attributes $\mathcal{U}^*$, are defined for the system. This implicitly defines the universal access tree $\mathcal{T}$.

Init The simulator $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ chooses a (d, num)-bounded access tree $\mathcal{T}''$ it wishes to be challenged upon. Next, $\mathcal{A}$ assigns real attributes to the leaf nodes to completely define $\mathcal{T}''$.

Recall that the encryption procedure in our construction involves a pre-processing phase, where, (a) an access tree $\mathcal{T}''$ (to be used for encryption) is converted to its normal form (say) $\mathcal{T}'$, (b) a map is defined between the nodes of $\mathcal{T}'$ and the universal access tree $\mathcal{T}$, and (c) for each non-leaf node x in $\mathcal{T}'$, a $(num - k_x)$-sized set ω_x of dummy child nodes of $\text{map}(x)$ is selected.

Before the system setup, the simulator executes the pre-processing phase on the access tree $\mathcal{T}''$ received from $\mathcal{A}$. In the sequel, $\mathcal{T}'$ would refer to the normal form of the access tree $\mathcal{T}''$.

Setup The system setup is done based on the access tree $\mathcal{T}'$ and the universal access tree $\mathcal{T}$. First the simulator assigns the public parameters $g_1 = A$ and $g_2 = B$. Now, for each node $x \in \Psi_{\mathcal{T}}$, it chooses a random n degree polynomial $f_x(X)$ and calculates an n degree polynomial $u_x(X)$ as follows: set $u_x(X) = -X^n$ for all X s.t. $f(x, X) = 1$ and $u_x(X) \neq -X^n$ otherwise. Because, $-X^n$ and $u_x(X)$ are two n degree polynomials, they will have at most n points in common or they are the same polynomial. This construction ensures that $\forall X, u_x(X) = -X^n$ if and only if $f(x, X) = 1$. Further, for each node $x \in \Psi_{\mathcal{T}}$, the simulator sets $t_{i,x} = g_2^{u_x(i)} g^{f_x(i)}$ for all $i = 1$ to $n+1$. Because $f_x(X)$ is a random n degree polynomial, all $t_{i,x}$ will be chosen independently at random as in the construction. Implicitly we have: $T_x(i) = g_2^{i^n + u_x(i)} g^{f_x(i)}$.

Next, the simulator sets the parameters corresponding to the dummy attributes in the following manner. Let $x' \in \Phi_{\mathcal{T}'}$, and $x = \text{map}(x')$ in $\mathcal{T}$. Recall that $\omega_{x'}$ is the select set of dummy child nodes of x chosen earlier by the simulator. If $j = \text{att}(z)$ for $j \in \mathcal{U}^*$, $z \in \omega_{x'}$, choose a random $r^*_{j,x} \in \mathbb{Z}_p$ and set $T^*_{j,x} = g^{r^*_{j,x}}$; otherwise choose a random $\beta^*_{j,x} \in \mathbb{Z}_p$ and set $T^*_{j,x} = g^{b\beta^*_{j,x}} = B^{\beta^*_{j,x}}$. Now, consider the sets $\Phi_{\mathcal{T}'}$ and $\Phi_{\mathcal{T}}$. Note that $\Phi_{\mathcal{T}} \supseteq \text{map}(\Phi_{\mathcal{T}'})$. For $x \in (\Phi_{\mathcal{T}} \setminus \text{map}(\Phi_{\mathcal{T}'}))$, choose a random $\beta^*_{j,x} \in \mathbb{Z}_p$ and set $T^*_{j,x} = g^{b\beta^*_{j,x}} = B^{\beta^*_{j,x}}$. Finally, the simulator gives the public parameters to $\mathcal{A}$.

Phase 1 $\mathcal{A}$ adaptively makes requests for private keys for several sets of attributes such that none of them satisfy the challenge tree $\mathcal{T}'$. Suppose $\mathcal{A}$ makes a request for the private key for an attribute set γ where $\mathcal{T}'(\gamma) = 0$. To generate the secret key, $\mathcal{B}$ has to fix a polynomial Q_x of degree d_x for every non-leaf node in the universal access tree $\mathcal{T}$. Recall the definition of $\hat{\mathcal{T}}$ from the proof of the small universe case. We will explain how the simulator assigns a polynomial for every non-leaf node in $\hat{\mathcal{T}}$. Note that this is exactly the same as assigning a polynomial for every non-leaf node in $\mathcal{T}$, since $\hat{\mathcal{T}}$ and $\mathcal{T}$ carry the same structure.

Recall the functions PolySat and PolyUnsat from the proof of small universe case. The simulator first runs $\text{PolyUnsat}(\hat{\mathcal{T}}, \gamma, A)$ on the main tree $\hat{\mathcal{T}}$ to define a polynomial q_x for each non-leaf node x of $\hat{\mathcal{T}}$, except for $x \in \Psi_{\hat{\mathcal{T}}}$. Therefore, we have that $q_r(0) = a$. Simulator now defines the main polynomial $Q_x(\cdot) = q_x(\cdot)$ for each non-leaf node x of $\hat{\mathcal{T}}$. Notice that this sets $y = Q_r(0) = a$. Finally, the keys for each real attribute $j \in \gamma$, and each dummy attribute $j \in \mathcal{U}^*$ are given, as explained below.

The simulator first defines the polynomials for the remaining non-leaf nodes in $\hat{\mathcal{T}}$, i.e. $x \in \Psi_{\hat{\mathcal{T}}}$. We consider the following two cases:
Case 1. $x \in \text{map}(\Psi_{\mathcal{T}'})$

The simulator fixes the polynomial for x depending upon whether it is satisfied or not, as explained in the proof of the small universe construction. Then, for each real attribute $j \in \gamma$, we have:

- If $f(j,x)=1$.
$$D_{j,x} = g_2^{Q_x(j)} T_x(j)^{r_{j,x}} = g_2^{q_x(j)} T_x(j)^{r_{j,x}}$$
$$R_{j,x} = g^{r_{j,x}}$$
Where $r_{j,x}$ is chosen at random from $\mathbb{Z}_p$.
- If $f(j,x)=0$. Let $g_3 = g^{Q_x(j)} = g^{q_x(j)}$.
$$\begin{array}{rcl} D_{j,x} & = & g_3^{\frac{-f_x(j)}{j^n+u_x(j)}} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{r'_{j,x}} \\ R_{j,x} & = & g_3^{\frac{-1}{j^n+u_x(j)}} g^{r'_{j,x}} \end{array}$$
Where $r'_{j,x}$ is chosen at random from $\mathbb{Z}_p$.

The value $j^n + u_x(j)$ will be non-zero for all $j \in \gamma$, s.t. $f(j,x)=1$. This follows from our construction of $u_x(X)$. The above key components are legitimate because if we set $r_{j,x} = r'_{j,x} - \frac{q_x(j)}{j^n+u_x(j)}$ then,

$$\begin{array}{rcl} D_{j,x} & = & g_3^{\frac{-f_x(j)}{j^n+u_x(j)}} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{r'_{j,x}} \\ & = & g^{\frac{-q_x(j).f_x(j)}{j^n+u_x(j)}} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{r'_{j,x}} \\ & = & g_2^{q_x(j)} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{\frac{-q_x(j)}{j^n+u_x(j)}} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{r'_{j,x}} \\ & = & g_2^{q_x(j)} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{r'_{j,x} - \frac{q_x(j)}{j^n+u_x(j)}} \\ & = & g_2^{q_x(j)} (T_x(j))^{r_{j,x}} \\ & = & g_2^{Q_x(j)} T_x(j)^{r_{j,x}} \end{array}$$

and,
$$R_{j,x} = g_3^{\frac{-1}{j^n+u_x(j)}} g^{r'_{j,x}} = g^{r'_{j,x} - \frac{q_x(j)}{j^n+u_x(j)}} = g^{r_{j,x}}$$

Case 2. $x \in (\Psi_{\hat{\mathcal{T}}} - \text{map}(\Psi_{\mathcal{T}'}))$

The simulator fixes the polynomial for x as explained in the proof of the small universe construction. Then, for each real attribute $j \in \gamma$, we have:

$$\begin{array}{rcl} D_{j,x} & = & g_3^{\frac{-f_x(j)}{j^n+u_x(j)}} (g_2^{j^n+u_x(j)} g^{f_x(j)})^{r'_{j,x}} \\ R_{j,x} & = & g_3^{\frac{-1}{j^n+u_x(j)}} g^{r'_{j,x}} \end{array}$$

Where $r'_{j,x}$ is chosen at random from $\mathbb{Z}_p$. The above key components are legitimate as explained in the previous case.

Finally, the key parts for each dummy attribute $j \in \mathcal{U}^*$ are computed in the following manner. We consider the following two cases:

Case 1. $x \in \text{map}(\Phi_{\mathcal{T}'})$

$$D^*_{j,x} = \begin{cases} g_2^{\frac{Q_x(j)}{t^*_{j,x}}} = g^{\frac{bq_x(j)}{r^*_{j,x}}} = B^{\frac{q_x(j)}{r^*_{j,x}}} & \text{if } j = \text{att}(z) : z \in \omega_x \\ g_2^{\frac{Q_x(j)}{t^*_{j,x}}} = g^{\frac{bq_x(j)}{b\beta^*_{j,x}}} = g^{\frac{q_x(j)}{\beta^*_{j,x}}} & \text{otherwise} \end{cases}$$

Case 2. $x \in (\Phi_{\hat{\mathcal{T}}} - \mathrm{map}(\Phi_{\mathcal{T}'}))$

$$D^*_{j,x} = g_2^{\frac{Q_x(j)}{t^*_{j,x}}} = g^{\frac{bq_x(j)}{b\beta^*_{j,x}}} = g^{\frac{q_x(j)}{\beta^*_{j,x}}}$$

Therefore, the simulator is able to construct a private key for the access structure $\mathcal{T}$. Furthermore, the distribution of the private key for $\mathcal{T}$ is identical to that of the original scheme.

Challenge The adversary $\mathcal{A}$, will submit two challenge messages m_0 and m_1 to the simulator. The simulator flips a fair binary coin ν, and returns an encryption of m_ν. The ciphertext is output as:

$$E = (\mathcal{T}', \mathrm{map}(\cdot), E' = m_\nu Z, E'' = C, \{E_{j,x} = C^{f_{\mathrm{map}(x)}(j)}\}_{j \in \mathcal{U}, x \in \Psi_{\mathcal{T}'} : f(j,x)=1}, \{E^*_{j,x} = C^{r^*_{j,\mathrm{map}(x)}}\}_{x \in \Phi_{\mathcal{T}'}, j = \mathrm{att}(z) : z \in \omega_x}$$

If $\mu = 0$ then $Z = e(g,g)^{abc}$. If we let $s = c$, then we have $e(g_1, g_2)^s = e(g,g)^{abc}$, $E'' = g^c$, $E_{j,x} = (g^c)^{f_{\mathrm{map}(x)}(j)} = T_{\mathrm{map}(x)}(j)^c$, and $E^*_{j,x} = (g^c)^{r^*_{j,\mathrm{map}(x)}} = T^{*c}_{j,\mathrm{map}(x)}$. Therefore, the ciphertext is a valid encryption of message m_ν under the set γ.

Otherwise, if $\mu = 1$, then $Z = e(g,g)^z$. We then have $E' = m_\nu e(g,g)^z$. Since z is random, E' will be a random element of $\mathbb{G}_2$ from the adversaries view and the message contains no information about m_ν.

Phase 2 The simulator acts exactly as it did in Phase 1.

Guess $\mathcal{A}$ will submit a guess ν' of ν. If $\nu' = \nu$ the simulator will output $\mu' = 0$ to indicate that it was given a valid BDH-tuple otherwise it will output $\mu' = 1$ to indicate it was given a random 4-tuple.

As shown in the construction the simulator's generation of public parameters and private keys is identical to that of the actual scheme.

In the case where $\mu = 1$ the adversary gains no information about ν. Therefore, we have $\Pr[\nu \neq \nu' | \mu = 1] = \frac{1}{2}$. Since the simulator guesses $\mu' = 1$ when $\nu \neq \nu'$, we have $\Pr[\mu' = \mu | \mu = 1] = \frac{1}{2}$.

If $\mu = 0$ then the adversary sees an encryption of m_ν. The adversary's advantage in this situation is ϵ by definition. Therefore, we have $\Pr[\nu = \nu' | \mu = 0] = \frac{1}{2} + \epsilon$. Since the simulator guesses $\mu' = 0$ when $\nu = \nu'$, we have $\Pr[\mu' = \mu | \mu = 0] = \frac{1}{2} + \epsilon$.

The overall advantage of the simulator in the Decisional BDH game is $\frac{1}{2}\Pr[\mu' = \mu | \mu = 0] + \frac{1}{2}\Pr[\mu' = \mu | \mu = 1] - \frac{1}{2} = \frac{1}{2}(\frac{1}{2} + \epsilon) + \frac{1}{2}\frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$.