Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data

Vipul Goyal* Omkant Pandey† Amit Sahai‡ Brent Waters §

Abstract

As more sensitive data is shared and stored by third-party sites on the Internet, there will be a need to encrypt data stored at these sites. One drawback of encrypting data, is that it can be selectively shared only at a coarse-grained level (i.e., giving another party your private key). We develop a new cryptosystem for fine-grained sharing of encrypted data that we call Key-Policy Attribute-Based Encryption (KP-ABE). In our cryptosystem, ciphertexts are labeled with sets of attributes and private keys are associated with access structures that control which ciphertexts a user is able to decrypt. We demonstrate the applicability of our construction to sharing of audit-log information and broadcast encryption. Our construction supports delegation of private keys which subsumes Hierarchical Identity-Based Encryption (HIBE).

1 Introduction

There is a trend for sensitive user data to be stored by third parties on the Internet. For example, personal email, data, and personal preferences are stored on web portal sites such as Google and Yahoo. The attack correlation center, `dshield.org`, presents aggregated views of attacks on the Internet, but stores intrusion reports individually submitted by users. Given the variety, amount, and importance of information stored at these sites, there is cause for concern that personal data will be compromised. This worry is escalated by the surge in recent attacks and legal pressure faced by such services.

One method for alleviating some of these problems is to store data in encrypted form. Thus, if the storage is compromised the amount of information loss will be limited. One disadvantage of encrypting data is that it severely limits the ability of users to selectively share their encrypted data at a fine-grained level. Suppose a particular user wants to grant decryption access to a party to all of its Internet traffic logs for all entries on a particular range of dates that had a source IP address from a particular subnet. The user either needs

*This research was supported in part by NSF ITR/Cybertrust grants 0456717 and 0627781.

†This research was supported in part by NSF ITR/Cybertrust grants 0456717 and 0627781.

‡This research was supported in part by an Alfred P. Sloan Foundation Research Fellowship, an Intel equipment grant, and NSF ITR/Cybertrust grants 0205594, 0456717 and 0627781.

§This research was supported in part by NSF, the US Army Research Office Grant No. W911NF-06-1-0316, and the Department of Homeland Security (DHS) and the Department of Interior (DOI) under Contract No. NBCHF040146. Any opinions, finding and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

to act as an intermediary and decrypt all relevant entries for the party or must give the party its private decryption key, and thus let it have access to *all* entries. Neither one of these options is particularly appealing. An important setting where these issues give rise to serious problems is *audit logs* (discussed in more detail in Section 7).

Sahai and Waters [34] made some initial steps to solving this problem by introducing the concept of Attributed-Based Encryption (ABE). In an ABE system, a user's keys and ciphertexts are labeled with sets of descriptive attributes and a particular key can decrypt a particular ciphertext only if there is a match between the attributes of the ciphertext and the user's key. The cryptosystem of Sahai and Waters allowed for decryption when at least k attributes overlapped between a ciphertext and a private key. While this primitive was shown to be useful for error-tolerant encryption with biometrics, the lack of expressibility seems to limit its applicability to larger systems.

Our Contribution. We develop a much richer type of attribute-based encryption cryptosystem and demonstrate its applications. In our system each ciphertext is labeled by the encryptor with a set of descriptive attributes. Each private key is associated with an access structure that specifies which type of ciphertexts the key can decrypt. We call such a scheme a Key-Policy Attribute-Based Encryption (KP-ABE), since the access structure is specified in the private key, while the ciphertexts are simply labeled with a set of descriptive attributes.[1]

We note that this setting is reminiscent of secret sharing schemes (see, e.g., [4]). Using known techniques one can build a secret-sharing scheme that specifies that a set of parties must cooperate in order to reconstruct a secret. For example, one can specify a tree access structure where the interior nodes consist of **AND** and **OR** gates and the leaves consist of different parties. Any set of parties that satisfy the tree can reconstruct the secret.

In our construction each user's key is associated with a tree-access structure where the leaves are associated with attributes.[2] A user is able to decrypt a ciphertext if the attributes associated with a ciphertext satisfy the key's access structure. The primary difference between our setting and secret-sharing schemes is that *while secret-sharing schemes allow for cooperation between different parties, in our setting, this is expressly forbidden.* For instance, if Alice has the key associated with the access structure "X **AND** Y", and Bob has the key associated with the access structure "Y **AND** Z", we would not want them to be able to decrypt a ciphertext whose only attribute is Y by colluding. To do this, we adapt and generalize the techniques introduced by [34] to deal with more complex settings. We will show that this cryptosystem gives us a powerful tool for encryption with fine-grained access control for applications such as sharing audit log information.

In addition, we provide a delegation mechanism for our construction. Roughly, this allows any user that has a key for access structure X to derive a key for access structure Y, if and only if Y is more restrictive than X. Somewhat surprisingly, we observe that our construction with the delegation property subsumes Hierarchical Identity-Based Encryption

[1] This contrasts with what we call Ciphertext-Policy Attribute-Based Encryption (CP-ABE), where an access structure (i.e. policy) would be associated to each ciphertext, while a user's private key would be associated with a set of attributes. KP-ABE and CP-ABE systems are useful in different contexts.

[2] In fact, we can extend our scheme to work for any access structure for which a Linear Secret Sharing Scheme exists (see appendix A).

[24, 21] and its derivatives [1].

1.1 Organization

We begin with a discussion of related work in Section 2. Next, we give necessary background information and our definitions of security in Section 3. We then present our first construction and a proof of security in Section 4 and later generalize it to work for any LSSS realizable access structure in Appendix A. We give a construction for the large universe case in Section 5. We then show how to add the delegation property in Section 6. We follow with a discussion of how our system applies to audit logs in Section 7. We discuss the application of our construction to broadcast encryption in Section 8. Finally, we discuss some interesting extensions and open problems in Section 9.

2 Related Work

Fine-grained Access Control. Fine-grained access control systems facilitate granting differential access rights to a set of users and allow flexibility in specifying the access rights of individual users. Several techniques are known for implementing fine grained access control.

Common to the existing techniques (see, e.g., [26, 20, 38, 28, 23, 29] and the references therein) is the fact that they employ a trusted server that stores the data in clear. Access control relies on software checks to ensure that a user can access a piece of data only if he is authorized to do so. This situation is not particularly appealing from a security standpoint. In the event of server compromise, for example, as a result of a software vulnerability exploit, the potential for information theft is immense. Furthermore, there is always a danger of "insider attacks" wherein a person having access to the server steals and leaks the information, for example, for economic gains. Some techniques (see, e.g., [2]) create user hierarchies and require the users to share a common secret key if they are in a common set in the hierarchy. The data is then classified according to the hierarchy and encrypted under the public key of the set it is meant for. Clearly, such methods have several limitations. If a third party must access the data for a set, a user of that set either needs to act as an intermediary and decrypt all relevant entries for the party or must give the party its private decryption key, and thus let it have access to all entries. In many cases, by using the user hierarchies it is not even possible to realize an access control equivalent to monotone access trees.

In this paper, we introduce new techniques to implement fine grained access control. In our techniques, the data is stored on the server in an encrypted form while different users are still allowed to decrypt different pieces of data per the security policy. This effectively eliminates the need to rely on the storage server for preventing unauthorized data access.

Secret-Sharing Schemes. Secret-sharing schemes (SSS) are used to divide a secret among a number of parties. The information given to a party is called the share (of the secret) for that party. Every SSS realizes some access structure that defines the sets of parties who should be able to reconstruct the secret by using their shares.

Shamir [35] and Blakley [7] were the first to propose a construction for secret-sharing schemes where the access structure is a threshold gate. That is, if any t or more parties come together, they can reconstruct the secret by using their shares; however, any lesser number of parties do not get any information about the secret. Benaloh [6] extended Shamir's idea to realize any access structure that can be represented as a tree consisting of threshold gates. Other notable secret-sharing schemes are [25, 15].

In SSS, one can specify a tree-access structure where the interior nodes consist of AND and OR gates and the leaves consist of different parties. Any set of parties that satisfy the tree can come together and reconstruct the secret. Therefore in SSS, collusion among different users (or parties) is not only allowed but required.

In our construction each user's key is associated with a tree-access structure where the leaves are associated with attributes. A user is able to decrypt a ciphertext if the attributes associated with a ciphertext satisfy the key's access structure. In our scheme, contrary to SSS, users should be *unable* to collude in any meaningful way.

Identity-Based Encryption and Extensions. The concept of Attribute-Based Encryption was introduced by Sahai and Waters [34], who also presented a particular scheme that they called Fuzzy Identity-Based Encryption (FIBE). The Fuzzy-IBE scheme builds upon several ideas from Identity-Based Encryption [10, 36, 18]. In FIBE, an identity is viewed as a set of attributes. FIBE allows for a private key for an identity, ω, to decrypt to a ciphertext encrypted with an identity, ω', if and only if the identities ω and ω' are close to each other as measured by the "set overlap" distance metric. In other words, if the message is encrypted with a set of attributes ω', a private key for a set of attributes ω enables decrypting that message, if and only if $|\omega \cap \omega'| \geq d$, where d is fixed during the setup time. Thus, FIBE achieves error tolerance making it suitable for use with biometric identities. However, it has limited applicability to access control of data, our primary motivation for this work. Since the main goal in FIBE is error tolerance, the only access structure supported is a threshold gate whose threshold is fixed at the setup time.

We develop a much richer type of attribute-based encryption. The private keys of different users might be associated with different access structures. Our constructions support a wide variety of access structures (indeed, in its most general form, every LSSS realizable access structure), including a tree of threshold gates.

Yao et. al. [19] show how an IBE system that encrypts to multiple hierarchical identities in a collusion-resistant manner implies a forward secure Hierarchical IBE scheme. They also note how their techniques for resisting collusion attacks are useful in attribute-based encryption. However, the cost of their scheme in terms of computation, private key size, and ciphertext size increases exponentially with the number of attributes. We also note that there has been other work that applied IBE techniques to access control, but did not address our central concern of resisting attacks from colluding users [37, 14].

3 Background

We first give formal definitions for the security of Key-Policy Attribute Based Encryption (KP-ABE). Then we give background information on bilinear maps and our cryptographic assumption. Finally, we give some background on linear secret-sharing schemes

and monotone span programs.

3.1 Definitions

Definition 1 (Access Structure [4]) *Let* $\{P_1, P_2, \ldots, P_n\}$ *be a set of parties. A collection* $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}}$ *is monotone if* $\forall B, C$: if $B \in \mathbb{A}$ *and* $B \subseteq C$ *then* $C \in \mathbb{A}$. *An* access structure *(respectively, monotone access structure) is a collection (respectively, monotone collection)* $\mathbb{A}$ *of non-empty subsets of* $\{P_1, P_2, \ldots, P_n\}$, *i.e.,* $\mathbb{A} \subseteq 2^{\{P_1, P_2, \ldots, P_n\}} \setminus \{\emptyset\}$. *The sets in* $\mathbb{A}$ *are called the* authorized sets, *and the sets not in* $\mathbb{A}$ *are called the* unauthorized sets.

In our context, the role of the parties is taken by the attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes. We restrict our attention to monotone access structures. However, it is also possible to (inefficiently) realize general access structures using our techniques by having the not of an attribute as a separate attribute altogether. Thus, the number of attributes in the system will be doubled. From now on, unless stated otherwise, by an access structure we mean a monotone access structure.

An (Key-Policy) Attribute Based Encryption scheme consists of four algorithms.

Setup This is a randomized algorithm that takes no input other than the implicit security parameter. It outputs the public parameters PK and a master key MK.

Encryption This is a randomized algorithm that takes as input a message m, a set of attributes γ, and the public parameters PK. It outputs the ciphertext E.

Key Generation This is a randomized algorithm that takes as input – an access structure $\mathbb{A}$, the master key MK and the public parameters PK. It outputs a decryption key D.

Decryption This algorithm takes as input – the ciphertext E that was encrypted under the set γ of attributes, the decryption key D for access control structure $\mathbb{A}$ and the public parameters PK. It outputs the message M if $\gamma \in \mathbb{A}$.

We now discuss the security of an ABE scheme. We define a selective-set model for proving the security of the attribute based under chosen plaintext attack. This model can be seen as analogous to the selective-ID model [16, 17, 8] used in identity-based encryption (IBE) schemes [36, 10, 18].

Selective-Set Model for ABE

Init The adversary declares the set of attributes, γ, 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 access structures $\mathbb{A}_j$, where $\gamma \notin \mathbb{A}_j$ 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 γ. The ciphertext is passed 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 2 *An attribute-based encryption scheme is secure in the Selective-Set model of security if all polynomial time adversaries have at most a negligible advantage in the Selective-Set game.*

3.2 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, $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. 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 $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ are both efficiently computable. Notice that the map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

3.3 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 [8, 34] 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 $\mathcal{B}$.

3.4 LSSS and Monotone Span Programs

In a linear secret-sharing scheme [4], realizing an access structure $\mathbb{A}$, a third party called the dealer holds a secret y and distributes the shares of y to parties such that y can be reconstructed by a linear combination of the shares of any authorized set. Further, an unauthorized set has no information about the secret y.

There is a close relation between LSSS and a linear algebraic model of computation called monotone span programs (MSP) [27]. It has been shown that the existence of an efficient LSSS for some access structure is equivalent to the existence of a small monotone span program for the characteristic function of that access structure [27, 4]. The following definition of MSP is a slightly altered version of the one presented in [4].

Definition 3 (Monotone Span Program) *Let $\mathcal{K}$ be a field and $\{x_1, \ldots, x_n\}$ be a set of variables. A* monotone span program *over $\mathcal{K}$ is labeled matrix $\hat{M}(M, \rho)$ where M is a matrix over $\mathcal{K}$, and ρ is a labeling of the rows of M by literals from $\{x_1, \ldots, x_n\}$ (every row is labeled by one literal).*

A monotone span program accepts or rejects an input by the following criterion. For every input set γ of literals, define the submatrix M_γ of M consisting of those rows whose labels are in γ, i.e., rows labeled by some x_i such that $x_i \in \gamma$. The monotone span program $\hat{M}$ accepts γ if and only if $\vec{1} \in \text{span}(M_\gamma)$, i.e., some linear combination of the rows of M_γ gives the all-one vector $\vec{1}$. The MSP $\hat{M}$ computes *a Boolean function f_M if it accepts exactly those inputs γ where $f_M(\gamma) = 1$. The* size *of $\hat{M}$ is the number of rows in M.*

Again, since the role of parties will be assumed by attributes in our context, each row of the matrix M will be labeled by an attribute.

4 Construction 吃 for Access Trees

In the access-tree construction, ciphertexts are labeled with a set of descriptive attributes. Private keys are identified by a tree-access structure in which each interior node of the tree is a threshold gate and the leaves are associated with attributes. (We note that this setting is very expressive. For example, we can represent a tree with "AND" and "OR" gates by using respectively 2 of 2 and 1 of 2 threshold gates.) A user will be able to decrypt a ciphertext with a given key if and only if there is an assignment of attributes from the ciphertexts to nodes of the tree such that the tree is satisfied.

4.1 Access Trees

Access tree $\mathcal{T}$. 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 $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x = num_x$, it is an AND gate. Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$.

To facilitate working with the access trees, we define a few functions. We denote the parent of the node x in the tree by parent(x). The function att(x) is defined only if x is a leaf node and denotes the attribute associated with the leaf node x in the tree. The access tree $\mathcal{T}$ also defines an ordering between the children of every node, that is, the children of a node are numbered from 1 to num. The function index(x) returns such a number associated with the node x. Where the index values are uniquely assigned to nodes in the access structure for a given key in an arbitrary manner.

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 $\mathcal{T}_{x'}(\gamma)$ for all children x' 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 if and only if att$(x) \in \gamma$.

4.2 Our 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 Define the universe of attributes $\mathcal{U} = \{1, 2, \ldots, n\}$. Now, for each attribute $i \in \mathcal{U}$, choose a number t_i uniformly at random from $\mathbb{Z}_p$. Finally, choose y uniformly at random in $\mathbb{Z}_p$. The published public parameters PK are

$$T_1 = g^{t_1}, \ldots, T_{|\mathcal{U}|} = g^{t_{|\mathcal{U}|}}, Y = e(g,g)^y \ .$$

The master key MK is:

$$t_1, \ldots, t_{|\mathcal{U}|}, y \ .$$

Encryption (M, γ, PK) To encrypt a message $M \in \mathbb{G}_2$ under a set of attributes γ, choose a random value $s \in \mathbb{Z}_p$ and publish the ciphertext as:

$$E = (\gamma, E^{'} = MY^s, \{E_i = T_i^s\}_{i \in \gamma}) \ .$$

Key Generation $(\mathcal{T}, \mathrm{MK})$ The algorithm outputs a key that enables the user to decrypt a message encrypted under a set of attributes γ if and only if $\mathcal{T}(\gamma) = 1$. The algorithm proceeds as follows. First choose a polynomial q_x for each node x (including the leaves) 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 node x in the tree, set the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node, that is, $d_x = k_x - 1$. Now, for the root node r, set $q_r(0) = y$ and d_r other points of the polynomial q_r randomly to define it completely. For any other node x, set $q_x(0) = q_{\mathrm{parent}(x)}(\mathrm{index}(x))$ and choose d_x other points randomly to completely define q_x.

Once the polynomials have been decided, for each leaf node x, we give the following secret value to the user:

$$D_x = g^{\frac{q_x(0)}{t_i}} \text{ where } i = \mathrm{att}(x) \ .$$

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

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 and discuss potential performance improvements in the next subsection.

We first define a recursive algorithm $\mathrm{DecryptNode}(E, D, x)$ that takes as input the ciphertext $E = (\gamma, E', \{E_i\}_{i \in \gamma})$, the private key D (we assume the access tree $\mathcal{T}$ is embedded in the private key), and a node x in the tree. It outputs a group element of $\mathbb{G}_2$ or $\perp$.

Let $i = \mathrm{att}(x)$. If the node x is a leaf node then:

$$\mathrm{DecryptNode}(E, D, x) = \begin{cases} e(D_x, E_i) = e(g^{\frac{q_x(0)}{t_i}}, g^{s \cdot t_i}) = e(g,g)^{s \cdot q_x(0)} \text{ if } i \in \gamma \\ \perp \text{ otherwise} \end{cases}$$

We now consider the recursive case when x is a non-leaf node. The algorithm DecryptNode(E, D, x) then 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. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns $\perp$.

Otherwise, we compute:

$$\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, \qquad \text{where } \substack{i=\text{index}(z) \\ S'_x=\{\text{index}(z): z \in S_x\}} \\
&= \prod_{z \in S_x} \left(e(g,g)^{s \cdot q_z(0)}\right)^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} \left(e(g,g)^{s \cdot q_{\text{parent}(z)}(\text{index}(z))}\right)^{\Delta_{i,S'_x}(0)} \qquad \text{(by construction)} \\
&= \prod_{z \in S_x} e(g,g)^{s \cdot q_x(i) \cdot \Delta_{i,S'_x}(0)} \\
&= e(g,g)^{s \cdot q_x(0)} \qquad \text{(using polynomial interpolation)}
\end{aligned}$$

and return the result.

Now that we have defined our function DecryptNode, the decryption algorithm simply calls the function on the root of the tree. We observe that DecryptNode$(E, D, r) = e(g,g)^{ys} = Y^s$ if and only if the ciphertext satisfies the tree. Since, $E' = MY^s$ the decryption algorithm simply divides out Y^s and recovers the message M.

4.3 Efficiency

We now consider the efficiency of the scheme in terms of ciphertext size, private key size, and computation time for decryption and encryption.

The ciphertext overhead will be approximately one group element in $\mathbb{G}_1$ for every element in γ. That is the number of group elements will be equal to the number of descriptive attributes in the ciphertext. Similarly, the encryption algorithm will need to perform one exponentiation for each attribute in γ.

The public parameters in the system will be of size linear in the number of attributes defined in the system. (We will later discuss systems where the set of legal attributes is not dependent upon the public parameters and we give a so called Large-Universe construction in Section 5.) User's private keys will consist of a group element for every leaf in the key's corresponding access tree.

The decryption procedure is by far the hardest to define performance for. In our rudimentary decryption algorithm the number of pairings to decrypt might always be as large as the number of nodes in the tree. However, this method is extremely suboptimal and we now discuss methods to improve upon it.

One important idea is for the decryption algorithm to do some type of exploration of the access tree relative to the ciphertext attributes before it makes cryptographic computations. At the very least the algorithm should first discover which nodes are not satisfied and not bother performing cryptographic operations on them.

The following observation shows how to modify our decryption method to optimize the efficiency. First we find out which leaf nodes we should use in order to minimize the number

of pairing computations as follows. For each node x, define a set $\mathbb{S}_x$. If x is a leaf node, then $\mathbb{S}_x = \{x\}$. Otherwise, let k be the threshold value of x. From among the child nodes of x, choose k nodes $x_1, x_2, \ldots, x_k$ such that $\mathbb{S}_{x_i}$ (for $i = 1, 2, \ldots, k$) are first k sets of the smallest size. Then for non-leaf node x, $\mathbb{S}_x = \{x' : x' \in \mathbb{S}_{x_i}, i = 1, 2, \ldots, k\}$. The set $\mathbb{S}_r$ corresponding to the root node r denotes the set of leaf nodes that should be used in order to minimize the number of pairing computations. Next, we notice that in the given decryption algorithm, Lagrange coefficients (Δ_{i,S'_x}) from various levels get multiplied in the exponent in a certain way in $\mathbb{Z}_p$. Thus, instead of exponentiating at each level, for each leaf node $x \in \mathbb{S}_r$, we can keep track of which Lagrange coefficients get multiplied with each other. Using this we can compute the final exponent f_x for each leaf node $x \in \mathbb{S}_r$ by doing multiplication in $\mathbb{Z}_p$. Now F_r is simply $\prod_{x \in \mathbb{S}_r} e(D_x, E_{\mathrm{att}(x)})^{f_x}$. This reduces the number of pairing computations and exponentiations to $|\mathbb{S}_r|$. Thus, decryption is dominated by $|\mathbb{S}_r|$ pairing computations.

The number of group elements that compose a user's private key grows linearly with the number of leaf nodes in the access tree. The number of group elements in a ciphertext grows linearly with the size of the set we are encrypting under. Finally, the number of group elements in the public parameters grows linearly with the number of attributes in the defined universe. Later, we provide a construction for large universes where all elements in $\mathbb{Z}_p^*$ can be used as attributes, yet the size of public parameters only grows linearly in a parameter n that we set to be the maximum possible size of γ.

4.4 Proof of Security

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

Theorem 1 *If an adversary can break our scheme in the Attribute-based Selective-Set 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-Set 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 generator g. 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 the universe, $\mathcal{U}$ is defined.

Init The simulator $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ chooses the set of attributes γ it wishes to be challenged upon.

Setup The simulator sets the parameter $Y = e(A, B) = e(g, g)^{ab}$. For all $i \in \mathcal{U}$, it sets T_i as follows: if $i \in \gamma$, it chooses a random $r_i \in \mathbb{Z}_p$ and sets $T_i = g^{r_i}$ (thus, $t_i = r_i$); otherwise it chooses a random $\beta_i \in \mathbb{Z}_p$ and sets $T_i = g^{b\beta_i} = B^{\beta_i}$ (thus, $t_i = b\beta_i$). It then gives the public parameters to $\mathcal{A}$.

Phase 1 $\mathcal{A}$ adaptively makes requests for the keys corresponding to any access structures $\mathcal{T}$ such that the challenge set γ does not satisfy $\mathcal{T}$. Suppose $\mathcal{A}$ makes a request for the secret key for an access structure $\mathcal{T}$ where $\mathcal{T}(\gamma) = 0$. To generate the secret key, $\mathcal{B}$ needs to assign a polynomial Q_x of degree d_x for every node in the access tree $\mathcal{T}$.

We first define the following two procedures: PolySat and PolyUnsat.

PolySat($\mathcal{T}_x, \gamma, \lambda_x$) This procedure sets up the polynomials for the nodes of an access subtree with satisfied root node, that is, $\mathcal{T}_x(\gamma) = 1$. The procedure takes an access tree $\mathcal{T}_x$ (with root node x) as input 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 d_x for the root node x. It sets $q_x(0) = \lambda_x$ and then sets rest of the points randomly to completely fix q_x. Now it sets polynomials for each child node x' of x by calling the procedure PolySat($\mathcal{T}_{x'}, \gamma, q_x(\text{index}(x'))$). Notice that in this way, $q_{x'}(0) = q_x(\text{index}(x'))$ for each child node x' of x.

PolyUnsat($\mathcal{T}_x, \gamma, g^{\lambda_x}$) This procedure sets up the polynomials for the nodes of an access tree with unsatisfied root node, that is, $\mathcal{T}_x(\gamma) = 0$. The procedure takes an access tree $\mathcal{T}_x$ (with root node x) as input 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 d_x for the root node x such that $q_x(0) = \lambda_x$. Because $\mathcal{T}_x(\gamma) = 0$, no more than d_x children of x are satisfied. Let $h_x \leq d_x$ be the number of satisfied children of x. For each satisfied child x' of x, the procedure chooses a random point $\lambda_{x'} \in \mathbb{Z}_p$ and sets $q_x(\text{index}(x')) = \lambda_{x'}$. It then fixes the remaining $d_x - h_x$ points of q_x randomly to completely define q_x. Now the algorithm recursively defines polynomials for the rest of the nodes in the tree as follows. For each child node x' of x, the algorithm calls:

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

Notice that in this case also, $q_{x'}(0) = q_x(\text{index}(x'))$ for each child node x' of x.

To give keys for access structure $\mathcal{T}$, simulator first runs PolyUnsat($\mathcal{T}, \gamma, A$) to define a polynomial q_x for each node x of $\mathcal{T}$. Notice that for each leaf node x of $\mathcal{T}$, we know q_x completely if x is satisfied; if x is not satisfied, then at least $g^{q_x(0)}$ is known (in some cases q_x might be known completely). Furthermore, $q_r(0) = a$.

Simulator now defines the final polynomial $Q_x(\cdot) = bq_x(\cdot)$ for each node x of $\mathcal{T}$. Notice that this sets $y = Q_r(0) = ab$. The key corresponding to each leaf node is given using its polynomial as follows. Let $i = \text{att}(x)$.

$$D_x = \begin{cases} g^{\frac{Q_x(0)}{t_i}} = g^{\frac{bq_x(0)}{r_i}} = B^{\frac{q_x(0)}{r_i}} & \text{if } \text{att}(x) \in \gamma \\ g^{\frac{Q_x(0)}{t_i}} = g^{\frac{bq_x(0)}{b\beta_i}} = g^{\frac{q_x(0)}{\beta_i}} & \text{otherwise} \end{cases}$$

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 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:

$$E = (\gamma, E' = m_\nu Z, \{E_i = C^{r_i}\}_{i \in \gamma})$$

If $\mu = 0$ then $Z = e(g,g)^{abc}$. If we let $s = c$, then we have $Y^s = (e(g,g)^{ab})^c = e(g,g)^{abc}$, and $E_i = (g^{r_i})^c = C^{r_i}$. 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 $\frac{1}{2}\Pr[\mu' = \mu | \mu = 0] + \frac{1}{2}\Pr[\mu' = \mu | \mu = 1] - \frac{1}{2} = \frac{1}{2}(\frac{1}{2} + \epsilon) + \frac{1}{2}\frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$.

Chosen-Ciphertext Security. Our security definitions and proofs have been in the chosen-plaintext model. Similar to [34], we notice that our construction can be extended to the chosen-ciphertext model by applying the technique of using simulation-sound NIZK proofs to achieve chosen-ciphertext security [33]. However, in Section 9 we describe how our delegation mechanism can be used with the techniques of Cannetti, Halevi, and Katz [17] to achieve a much more efficient CCA-2 system.

5 Large Universe Construction

In our previous constructions, the size of public parameters grows linearly with the number of possible attributes in the universe. Combining the tricks presented in section 4 with those in the large universe construction of Sahai and Waters [34], we construct another scheme that uses all elements of $\mathbb{Z}_p^*$ as attributes, yet the public parameters grow only linearly in

a parameter n which we fix as the maximum size of the set we can encrypt under.[3]

This construction not only reduces the size of the public parameters but also allows us to apply a collision resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p^*$ and use arbitrary strings, that were not necessarily considered during public key setup, as attributes. For example we can add any verifiable attribute, such as "Lives in Beverly Hills", to a user's private key.

5.1 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. The data will be encrypted under a set γ of n elements[4] of $\mathbb{Z}_p^*$. Our construction follows.

Setup (n) 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, choose $t_1, \ldots, t_{n+1}$ uniformly at random from $\mathbb{G}_1$. Let N be the set $\{1, 2, \ldots, n+1\}$. Define a function T, as:

$$T(X) = g_2^{X^n} \prod_{i=1}^{n+1} t_i^{\Delta_{i,N}(X)}.$$

Function T can be viewed as the function $g_2^{X^n} g^{h(X)}$ for some n degree polynomial h. The public parameters PK are: $g_1, g_2, t_1, \ldots, t_{n+1}$ and the master key MK is: y.

Encryption (m, γ, PK) To encrypt a message $m \in \mathbb{G}_2$ under a set of attributes γ, choose a random value $s \in \mathbb{Z}_p$ and publish the ciphertext as:

$$E = (\gamma, E' = me(g_1, g_2)^s, E'' = g^s, \{E_i = T(i)^s\}_{i \in \gamma}).$$

Key Generation $(\mathcal{T}, \mathrm{MK}, \mathrm{PK})$ The algorithm outputs a key which enables the user to decrypt a message encrypted under a set of attributes γ, if and only if $\mathcal{T}(\gamma) = 1$. The algorithm proceeds as follows. First choose a 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 node x in the tree, set the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node, that is, $d_x = k_x - 1$. Now for the root node r, set $q_r(0) = y$ and d_r other points of the polynomial q_r randomly to define it completely. For any other node x, set $q_x(0) = q_{\mathrm{parent}(x)}(\mathrm{index}(x))$ and choose d_x other points randomly to completely define q_x.

[3] If we are willing to accept random oracles [5], it is possible to overcome the size-limitation on γ by replacing the function $T(X)$ in our construction (see Setup) by a hash function (see [31] for details). This also improves the efficiency of the system.

[4] With some minor modifications, which we omit for simplicity, we can encrypt to all sets of size $\leq n$.

Once the polynomials have been decided, for each leaf node x, we give the following secret values to the user:

$$\begin{aligned} D_x &= g_2^{q_x(0)} \cdot T(i)^{r_x} \quad \text{where } i = \text{att}(x) \\ R_x &= g^{r_x} \end{aligned}$$

where r_x is chosen uniformly at random from $\mathbb{Z}_p$ for each node x. The set of above secret pairs is the decryption key D.

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 $E = (\gamma, E', E'', \{E_i\}_{i\in\gamma})$, the private key D (we assume the access tree $\mathcal{T}$ is embedded in the private key), and a node x in the tree. It outputs a group element of $\mathbb{G}_2$ or $\perp$ as follows.

Let $i = \text{att}(x)$. If the node x is a leaf node then:

$$\text{DecryptNode}(E, D, x) = \begin{cases} \frac{e(D_x, E'')}{e(R_x, E_i)} = \frac{e(g_2^{q_x(0)} \cdot T(i)^{r_x}, g^s)}{e(g^{r_x}, T(i)^s)} = \frac{e(g_2^{q_x(0)}, g^s) \cdot e(T(i)^{r_x}, g^s)}{e(g^{r_x}, T(i)^s)} = e(g, g_2)^{s \cdot q_x(0)} & \text{if } i \in \gamma \\ \perp & \text{otherwise} \end{cases}$$

We now consider the recursive case when x is a non-leaf node. The algorithm DecryptNode(E, D, x) then 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. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists then the node was not satisfied and the function returns $\perp$.

Otherwise, we compute:

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i, S'_x}(0)}, \qquad \text{where } {}^{i=\text{index}(z)}_{S'_x = \{\text{index}(z) : z \in S_x\}} \\ &= \prod_{z \in S_x} \left(e(g, g_2)^{s \cdot q_z(0)}\right)^{\Delta_{i, S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e(g, g_2)^{s \cdot q_{\text{parent}(z)}(\text{index}(z))}\right)^{\Delta_{i, S'_x}(0)} \qquad \text{(by construction)} \\ &= \prod_{z \in S_x} e(g, g_2)^{s \cdot q_x(0) \cdot \Delta_{i, S'_x}(0)} \\ &= e(g, g_2)^{s \cdot q_x(0)} \qquad \text{(using polynomial interpolation)} \end{aligned}$$

and return the result.

Now that we have defined our function DecryptNode, the decryption algorithm simply calls the function on the root of the tree. We observe that DecryptNode$(E, D, r) = e(g, g_2)^{ys} = e(g_1, g_2)^s$ if and only if the ciphertext satisfies the tree. Since, $E' = me(g_1, g_2)^s$ the decryption algorithm simply divides out $e(g_1, g_2)^s$ and recovers the message m.

The security proof of this construction follows largely from the security proof in section 4.4. Details are given in appendix B.

It is possible to extend the above construction to realize any LSSS realizable access structure using techniques similar to those in appendix A. A brief outline of the the construction follows.

Algorithms Setup and Encryption remain exactly the same as for the above construction. Key Generation takes an MSP and the master key as input. For each row i of the matrix, it gives the following pair to the user: $(D_i = g_2^{\vec{M}_i \cdot \vec{u}} \cdot T(i)^{r_i}, R_i = g^{r_i})$; where $\vec{u}$ is chosen such that $\vec{1} \cdot \vec{u} = y$ (master key), and r_i is chosen randomly. Decryption and the security proof follow using the tricks of appendix A.

6 Delegation of Private Keys

In our large universe construction, individual users can generate new private keys using their private keys, which can then be delegated to other users. A user which has a private key corresponding to an access tree $\mathcal{T}$ can compute a new private key corresponding to ANY access tree $\mathcal{T}'$ which is *more restrictive* than $\mathcal{T}$ (i.e., $\mathcal{T}' \subseteq \mathcal{T}$). Thus, the users are capable to acting as a local key authority which can generate and distribute private keys to other users.

Computation of a new private key from an existing private key is done by applying a set of basic operations on the existing key. These operations are aimed at step by step conversion of the given private key for an access tree $\mathcal{T}$ to a private key for the targeted access tree $\mathcal{T}'$ (given that $\mathcal{T}' \subseteq \mathcal{T}$). In the following, a (t, n)-gate denotes a gate with threshold t and number of children n. The operations are as follows.

1) Adding a new trivial gate to $\mathcal{T}$

This operation involves adding a new node y *above* an existing node x. The new node y represents a $(1, 1)$ threshold gate which after adding becomes the parent of x. The former parent of x (if x is not the root node), say z, becomes the parent of y.

Since the threshold of y is 1, we are required to associate a 0 degree polynomial q_y with it such that $q_x(0) = q_y(\text{index}(x))$ and $q_y(0) = q_z(\text{index}(y))$. The second condition essentially fixes q_y and the first one is automatically satisfied since z was the parent of x earlier. Hence, no changes to the private key are required for this operation.

2) Manipulating an existing (t, n)-gate in $\mathcal{T}$

This operation involves manipulating a threshold gate so as to make the access structure more restrictive. The operation could be of the following three types.

2.1) Converting a (t, n)-gate to a $(t + 1, n)$-gate with $(t + 1) \leq n$

Consider a node x representing a (t, n)-gate. Clearly, the polynomial q_x has the degree $(t - 1)$ which has to be increased to t. Define a new polynomial q'_x as follows.

$$q'_x(X) = (X + 1)q_x(X)$$

Now, we change the key such that q'_x becomes the new polynomial of the node x. This is done as follows. For every child y of x, compute the constant $C_x = \text{index}(y) + 1$. For every leaf node z in the subtree[5] $\mathcal{T}_y$, compute the new decryption key as

$$D'_z = (D_z)^{C_x}, R'_z = (R_z)^{C_x}$$

[5]Recall that $\mathcal{T}_y$ denotes the subtree defined by y as its root.

The above results in the multiplication of all the polynomials in the subtree $\mathcal{T}_y$ with the constant C_x. Hence, $q'_y(0) = (\text{index}(y) + 1)q_y(0)$ which is indeed a point on the new polynomial q'_x. Note that since $q'_x(0) = q_x(0)$, no changes outside the subtree $\mathcal{T}_x$ are required.

The above procedure effectively changes x from a (t, n)-gate to a $(t + 1, n)$-gate and yields the corresponding new private key.

2.2) Converting a (t, n)-gate to a $(t + 1, n + 1)$-gate

This procedure involves adding a new subtree (with root say z) as a child of a node x while increasing the degree of x by 1 at the same time. Let z be the v^{th} child of x so that $\text{index}(z) = v$. We shall change the polynomial q_x to the following.

$$q'_x(X) = (aX + 1)q_x(X) \text{ where } a = \frac{-1}{v}$$

As in the previous operation, for every (existing) child y of x, the polynomials in the subtree $\mathcal{T}_y$ are multiplied with the appropriate constant $C_x = a.\text{index}(y) + 1$. This ensures that $q'_y(0)$ is indeed a point on q'_x. Further, set $q_z(0) = 0$ $(= q'_x(v))$. Given $q_z(0)$, keys can be created for the subtree $\mathcal{T}_z$ as in the original key generation algorithm. Hence, the keys of the subtrees of all the children (old as well as new) of the node x have been made consistent with the new polynomial q'_x, thus achieving our goal.

2.3) Converting a (t, n)-gate to a $(t, n - 1)$-gate with $t \leq (n - 1)$

This operation involves deleting a child y of a node x. This can be easily achieved just by deleting decryption keys corresponding to all the leaves of $\mathcal{T}_y$ from the original decryption key.

3) Re-randomizing the obtained key

Once we obtain a key for the desired access structure (by applying a set of operations of type 1 and 2), we apply a final re-randomization step to make it independent of the original key from which it was computed. Re-randomization of a node x with a (known) constant C_x is done as follows. Choose a random polynomial p_x of degree[6] d_x such that $p_x(0) = C_x$. Define the new polynomial q'_x as $q'_x(X) = q_x(X) + p_x(X)$. We have to change the key such that q'_x becomes the new polynomial of node x. This is done by recursively re-randomizing every child y of x with the constant $C_y = p_x(\text{index}(y))$. If y is a leaf node, the new decryption key corresponding to y is computed as follows.

$$D'_y = D_y.{g_2}^{C_y}.T(i)^{r_y},\ R'_y = R_y.g^{r_y}$$

Where $i = \text{att}(y)$ and r_y is chosen randomly.

Now, re-randomization of the private key is done just by re-randomizing the root node r with the constant $C_r = 0$. The key obtained is the final key ready to be distributed to other users.

[6]Recall that d_x is the degree of the polynomial q_x associated with the node x

In the following theorem, we prove that the above set of operations is complete. That is, give a key for an access tree $\mathcal{T}$, this set of operations is sufficient to compute a key for ANY access tree $\mathcal{T}'$ which is more restrictive than $\mathcal{T}$.

Theorem 2 (Completeness Theorem) *The given set of operations is complete.*

PROOF: We can obtain a key for an access tree $\mathcal{T}'$ from a key for $\mathcal{T}$ using the following general technique. Add a new trivial gate x (using operation 1) above the root node of $\mathcal{T}$ so that the new gate becomes the root node. Now we apply operation 2.2 to x to convert it from a $(1, 1)$-gate to a $(2, 2)$-gate. The new subtree added as a child of x corresponds to the access tree $\mathcal{T}'$. Finally, we re-randomize the key obtained (operation 3). This gives us a key for the access structure ($\mathcal{T}$ AND $\mathcal{T}'$). However, since $\mathcal{T}'$ is more restrictive than $\mathcal{T}$, this access structure is equivalent to $\mathcal{T}'$ itself. Hence, we have obtained a key for the access structure $\mathcal{T}'$. We point out that this is a general method for obtaining a more restrictive access structure; in practice users will likely use the delegation tools described above in a more refined manner to achieve shorter private key sizes and faster decryption times.

In the above setting, we may imagine an entity having multiple private keys (procured from different entities). We note that it is possible to use these multiple keys (for different access structures) to compute a key for the targeted access structure. Given n keys for the access trees $\mathcal{T}_1$, $\mathcal{T}_2$, ... ,$\mathcal{T}_n$, using an operation similar to operation 1, we can connect them to obtain a single tree with an OR gate being the root node and $\mathcal{T}_1$, $\mathcal{T}_2$, ... ,$\mathcal{T}_n$ each being a child subtree of that OR gate. Thus, we obtain a single key for the access structure $\mathcal{T} = (\mathcal{T}_1$ OR $\mathcal{T}_2$ OR ... OR $\mathcal{T}_n)$. This key can then be used to generate new private keys.

7 Audit Log Application

An important application of KP-ABE deals with secure forensic analysis: One of the most important needs for electronic forensic analysis is an "audit log" containing a detailed account of all activity on the system or network to be protected. Such audit logs, however, raise significant security concerns: a comprehensive audit log would become a prized target for enemy capture. Merely encrypting the audit log is not sufficient, since then any party who needs to legitimately access the audit log contents (for instance a forensic analyst) would require the secret key – thereby giving this single analyst access to essentially all secret information on the network. Such problematic security issues arise in nearly every secure system, and particularly in large-scale networked systems such as the Global Information Grid, where diverse secret, top secret, and highly classified information will need to appear intermingled in distributed audit logs.

Our KP-ABE system provides an attractive solution to the audit log problem. Audit log entries could be annotated with attributes such as, for instance, the name of the user, the date and time of the user action, and the type of data modified or accessed by the user action. Then, a forensic analyst charged with some investigation would be issued a secret key associated with a particular "access structure" – which would correspond to the key allowing for a particular kind of encrypted search; such a key, for example, would only open audit log records whose attributes satisfied the condition that "the user name is Bob,

OR (the date is between October 4, 2005 and October 7, 2005 AND the data accessed pertained to naval operations off the coast of North Korea)". Our system would provide the guarantee that even if multiple rogue analysts collude to try to extract unauthorized information from the audit log, they will fail.

A more concrete example audit-log application of our ABE system would be to the ArmyCERT program, which uses netflow logs [30]. Basically, an entry is created for every flow (e.g. TCP connection), indexed by seven attributes: source IP address, destination IP address, L3 protocol type, source port, destination port, ToS byte (DSCP), and input logical interface (ifIndex). These aspects of every flow are in the clear, and the payload can be encrypted using our ABE system with these fields as attributes.

Note that in our scheme, we would need to assume that the attributes associated with audit log entries would be available to all analysts.[7] This may present a problem in highly secret environments where even attributes themselves would need to be kept hidden from analysts. We leave the problem of constructing KP-ABE systems where attributes associated with ciphertexts remain secret as an important open problem.

8 Application to Broadcast Encryption: Targeted Broadcast

We describe a new broadcast scenario that we call *targeted broadcast*. Consider the following setting.

- A broadcaster broadcasts a sequence of different items, each one labeled with a set of attributes describing the item. For instance, a television broadcaster might broadcast an episode of the show "24", and label this item with attributes such as the name of the program ("24"), the genre ("drama"), the season, the episode number, the year, month, and date of original broadcast, the current year, month, and date, the name of the director, and the name of the producing company.

- Each user is subscribed to a different "package". The user package describes an access policy, which along with the set of attributes describing any particular item being broadcast, determine whether or not the user should be able to access the item. For example, a television user may want to subscribe to a package that allows him view episodes of "24" from either the current season or Season 3. This could be encoded as policy as ("24" **AND** ("Season:5" **OR** "Season:3")).

The essential idea of Targeted Broadcast is to enjoy the economies-of-scale offered by a broadcast channel, while still being able to deliver programming targeted at the needs or wishes of individual users. The growing acceptability of such a model can be seen by the rising popularity of DVR systems such as TiVo, which allow users to easily record only the programming they want in order to watch it later. In the case of television, taking the approach that we envision here would allow for much more flexibility than just allowing users to select what channels they like.

Our KP-ABE system naturally offers a targeted broadcast system. A new symmetric key would be chosen and used to encrypt each item being broadcast, and then the KP-ABE

[7] We observe that this does not mean that the attributes need be "public." Our KP-ABE system's ciphertexts could be re-encrypted, with a key that corresponds to the general clearance level of all analysts.

system would be used to encrypt the symmetric key with the attributes associated with the item being broadcast. The KP-ABE system would precisely allow the flexibility we envision in issuing private keys for the unique needs of each user.

It is worth mentioning that handling such a situation with the best known broadcast encryption schemes [11, 22] (which allow encrypting to an arbitrary subset of users) is quite inefficient in comparison. The efficiency of such systems is dependent on the size of the authorized user set or the number of users in the system [11, 22], and would also require the broadcaster to refer to its database of user authorizations each time a different item is to be encrypted for broadcast. In our scheme, the encryption of an item would depend only on the properties of that item. The broadcaster could in principle even forget about the levels of access granted to each user after preparing a private key for the user.

9 Discussion and Extensions

We discuss various extensions to our scheme and open problems.

Achieving CCA-Security and HIBE from Delegation We briefly outline how we can achieve efficient CCA-2 security and realize the Hierarchical Identity-Based Encryption by applying delegation techniques to the large universe construction.

To achieve CCA-2 security an encyrptor will chooses a set γ of attributes to encrypt the message under and then generate a public/private key pair for a one time signature scheme. We let VK denote the bitstring represenation of the public key and let γ' be the set $\gamma \cup \text{VK}$. The encryptor encrypts the ciphertext under the attributes γ' and then signs the ciphertext with the private key and attaches the signature and the public key description. Suppose a user has a key for access structure X wishes to decrypt. The user first checks that the ciphertext is signed under VK and rejects the ciphertext otherwise. Then it creates an new key for the access structure of "X AND CCA : VK". By similar arguments to those in Canetti, Halevi, and Katz [17] this gives chosen-ciphertex security. We can also use other methods [12, 13] to achieve greater efficiency.

We can realize a HIBE by simply managing the the assignment of attributes in a careful manner. For example, to encrypt to the hierarchical identity "edu:ucla" one can encrypt to the set of attributes { "1-edu", "2-ucla" }. Someone who has the top-level key for edu will have a policy that requires the attribute "1-edu" to be present. To delegate a key for "edu:ucla" it simply creates a policy for "1-edu"AND "2-ucla" using our delegation techniques. We view the fact that a primitive HIBE follows so simply from our scheme as an attestation to the power of these techniques.

Ciphertext-Policy Attribute-Based Encryption. In this work, we considered the setting where ciphertexts are associated with sets of attributes, whereas user secret keys are associated with policies. As we have discussed, this setting has a number of natural applications. Another possibility is to have the reverse situation: user keys are associated with sets of attributes, whereas ciphertexts are associated with policies. We call such systems Ciphertext-Policy Attribute-Based Encryption (CP-ABE) systems. We note that the construction of Sahai and Waters [34] was most naturally considered in this framework.

CP-ABE systems that allow for complex policies (like those considered here) would have a number of applications. An important example is a kind of sophisticated Broadcast Encryption, where users are described by (and therefore associated with) various attributes. Then, one could create a ciphertext that can be opened only if the attributes of a user match a policy. For instance, in a military setting, one could broadcast a message that is meant to be read only by users who have a rank of Lieutenant or higher, and who were deployed in South Korea in the year 2005. We leave constructing such a system as an important open problem.

Searching on Encrypted Data. Our current constructions do not hide the set of attributes under which the data is encrypted. However, if it were possible to hide the attributes, then viewing attributes as keywords in such a system would lead to the first general keyword-based search on encrypted data [9]. A search query could potentially be any monotone boolean formula of any number of keywords. We leave the problem of hiding the set of attributes as open.

References

[1] Michel Abdalla, Dario Catalano, Alexander W. Dent, John Malone-Lee, Gregory Neven, and Nigel P. Smart. Identity-based encryption gone wild. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 2006.

[2] S.G. Akl and P.D. Taylor. Cryptographic Solution to a Multi Level Security Problem. In *Advances in Cryptology – CRYPTO*, 1982.

[3] H. Anton and C. Rorres. *Elementary Linear Algebra, 9th Edition.* 2005.

[4] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM conference on Computer and Communications Security (ACM CCS)*, pages 62–73, 1993.

[6] J. Benaloh and Leichter J. Generalized Secret Sharing and Monotone Functions. In *Advances in Cryptology – CRYPTO*, volume 403 of *LNCS*, pages 27–36. Springer, 1988.

[7] G. R. Blakley. Safeguarding cryptographic keys. In *National Computer Conference*, pages 313–317. American Federation of Information Processing Societies Proceedings, 1979.

[8] 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.

[9] D. Boneh, G.D. Crescenzo, R. Ostrovsky, and G. Persiano. Public-Key Encryption with Keyword Search. In *Advances in Cryptology – Eurocrypt*, volume 3027 of *LNCS*, pages 506–522. Springer, 2004.

[10] 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.

[11] D. Boneh, C. Gentry, and B. Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Advances in Cryptology – CRYPTO*, volume 3621 of *LNCS*, pages 258–275. Springer, 2005.

[12] Dan Boneh and Jonathan Katz. Improved efficiency for cca-secure cryptosystems built using identity-based encryption. In *CT-RSA*, pages 87–103, 2005.

[13] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In *ACM Conference on Computer and Communications Security*, pages 320–329, 2005.

[14] Robert W. Bradshaw, Jason E. Holt, and Kent E. Seamons. Concealing complex policies with hidden credentials. In *ACM Conference on Computer and Communications Security*, pages 146–157, 2004.

[15] E. F. Brickell. Some ideal secret sharing schemes. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 6:105–113, 1989.

[16] 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.

[17] 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.

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

[19] Y. Dodis, N. Fazio, A. Lysyanskaya, and D.F. Yao. ID-Based Encryption for Complex Hierarchies with Applications to Forward Security and Broadcast Encryption. In *ACM conference on Computer and Communications Security (ACM CCS)*, pages 354–363, 2004.

[20] Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *ESWS*, pages 342–356, 2004.

[21] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[22] D. Halevy and A. Shamir. The LSD Broadcast Encryption Scheme. In *Advances in Cryptology – CRYPTO*, volume 2442 of *LNCS*, pages 47–60. Springer, 2002.

[23] Hugh Harney, Andrea Colgrove, and Patrick Drew McDaniel. Principles of policy in secure groups. In *NDSS*, 2001.

[24] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer, 2002.

[25] M. Ito, A. Saito, and T. Nishizeki. Secret Sharing Scheme Realizing General Access Structure. In *IEEE Globecom*. IEEE, 1987.

[26] Myong H. Kang, Joon S. Park, and Judith N. Froscher. Access control mechanisms for inter-organizational workflow. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 66–74, New York, NY, USA, 2001. ACM Press.

[27] M. Karchmer and A. Wigderson. On Span Programs. In *The Eighth Annual Structure in Complexity Theory*, pages 102–111, 1993.

[28] Jiangtao Li, Ninghui Li, and William H. Winsborough. Automated trust negotiation using cryptographic credentials. In *ACM Conference on Computer and Communications Security*, pages 46–57, 2005.

[29] Patrick Drew McDaniel and Atul Prakash. Methods and limitations of security policy reconciliation. In *IEEE Symposium on Security and Privacy*, pages 73–87, 2002.

[30] Cisco Networks. `http://netflow.cesnet.cz/n_netflow.php`.

[31] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure Atrribute-Based Systems. In *ACM conference on Computer and Communications Security (ACM CCS)*, 2006. To appear.

[32] V.V. Prasolov. *Problems and Theorems in Linear Algebra*. American Mathematical Society, 1994.

[33] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *IEEE Symposium on Foundations of Computer Science*, 1999.

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

[35] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

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

[37] Nigel P. Smart. Access control using pairing based cryptography. In *CT-RSA*, pages 111–121, 2003.

[38] Ting Yu and Marianne Winslett. A unified scheme for resource protection in automated trust negotiation. In *IEEE Symposium on Security and Privacy*, pages 110–122, 2003.

A Construction for Any LSSS-realizable Access Structure

Consider an access structure for which there exists a linear secret-sharing scheme that realizes it. It is known that for every LSSS realizable access structure, there exists a monotone span program (MSP) that computes the corresponding boolean function and vice versa [4]. Thus, such an access structure can be represented by a monotone span program.

In an attempt to accommodate more general and complex access structures, we present attribute based encryption for the class of access structures which can be represented by polynomial size monotone span programs. The access structure will be represented in the form of a monotone span program $\hat{M}(M, \rho)$. Each row of M will be labeled by an attribute from $\mathcal{U}$ and $\rho(i)$ will denote the label of i^{th} row $\vec{M}_i$.

A.1 Our Construction

The data will be encrypted under a set of attributes γ. The user should be able to decrypt the data if and only if he holds the keys for an MSP $\hat{M}(M, \rho)$ such that $f_M(\gamma) = 1$; here f_M denotes the function that $\hat{M}$ computes.

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. Our construction follows:

Setup Define the universe of attributes $\mathcal{U} = \{1, 2, \ldots, n\}$. Now, for each attribute $i \in \mathcal{U}$, choose a number t_i uniformly at random from $\mathbb{Z}_p$. Finally, choose y uniformly at random in $\mathbb{Z}_p$. The published public parameters PK are:

$$T_1 = g^{t_1}, \ldots, T_{|\mathcal{U}|} = g^{t_{|\mathcal{U}|}}, Y = e(g, g)^y$$

The master key MK is:

$$t_1, \ldots, t_{|\mathcal{U}|}, y$$

Encryption (m, γ, PK) To encrypt a message $m \in \mathbb{G}_2$ under γ, choose a random value $s \in \mathbb{Z}_p$ and publish the ciphertext as:

$$E = (\gamma, E^{'} = mY^s, \{E_i = T_i^s\}_{i \in \gamma})$$

Key Generation $(\hat{M}, \text{MK}, \text{PK})$ Input to the key generation algorithm is an access structure in the form of an MSP $\hat{M}(M, \rho)$. Let the dimensions of M be $d \times \ell$. Choose a random vector $\vec{u}$ from $\mathbb{Z}_p^\ell$ such that $\vec{1} \cdot \vec{u} = y$, that is, $u = (u_1, u_2, \ldots, u_\ell)$ such that $\sum_{i=1}^{\ell} u_i = y$. For each row vector $\vec{M}_i$, give the following secret value to the user:

$$D_i = g^{\frac{\vec{M}_i \cdot \vec{u}}{t_{\rho(i)}}}$$

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

Decryption (E, D, PK) To decrypt the ciphertext $E = (\gamma, E', \{E_i\}_{i\in\gamma})$ with the decryption key D, proceed as follows. Since $f_M(\gamma) = 1$, the span program $\hat{M}$ accepts γ. Thus, there exist coefficients $\alpha_i \in \mathbb{Z}_p$ such that,

$$\sum_{\rho(i)\in\gamma} \alpha_i \cdot \vec{M}_i = \vec{1}$$

Now,

$$\begin{aligned}
E'/\prod_{\rho(i)\in\gamma} e(D_i, E_{\rho(i)})^{\alpha_i} &= E'/\prod_{\rho(i)\in\gamma} e(g^{\frac{\vec{M}_i\cdot\vec{u}}{t_{\rho(i)}}}, g^{st_{\rho(i)}})^{\alpha_i} \\
&= mY^s/\prod_{\rho(i)\in\gamma} e(g,g)^{s\alpha_i\cdot\vec{M}_i\cdot\vec{u}} \\
&= mY^s/e(g,g)^{s\cdot(\sum_{\rho(i)\in\gamma}\alpha_i\cdot\vec{M}_i)\cdot\vec{u}} \\
&= mY^s/e(g,g)^{s\cdot(\vec{1}\cdot\vec{u})} \\
&= me(g,g)^{sy}/e(g,g)^{sy} \\
&= m.
\end{aligned}$$

A.2 Proof of Security

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

Theorem 3 *If an adversary can break our scheme in the Attribute-based Selective-Set 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-Set 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 generator g. 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 the universe, $\mathcal{U}$, is defined.

Init The simulator $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ chooses the set of attributes γ it wishes to be challenged upon.

Setup The simulator sets the parameter $Y = e(A, B) = e(g,g)^{ab}$. For all $i \in \mathcal{U}$, it sets T_i as follows: if $i \in \gamma$, it chooses a random r_i and sets $T_i = g^{r_i}$ (thus, $t_i = r_i$); otherwise it chooses a random β_i and sets $T_i = g^{b\beta_i} = B^{\beta_i}$ (thus, $t_i = b\beta_i$).

It then gives the public parameters to $\mathcal{A}$. Notice that from $\mathcal{A}$'s view all parameters are chosen at random as in the construction.

Phase 1 $\mathcal{A}$ adaptively makes requests for several access structures represented in the form of MSP such that γ is accepted by none of them.

Suppose $\mathcal{A}$ makes a request for the secret key for an MSP $\hat{M}(M, \rho)$ such that $f_M(\gamma) = 0$. Let the dimensions of M be $d \times \ell$. Let M_γ be the submatrix of M consisting of only those rows whose label is in γ. To generate the secret key, $\mathcal{B}$ has to fix a random vector $u = (u_1, u_2, \ldots, u_\ell)$ such that $\vec{1} \cdot \vec{u} = ab$. To do that, first define a vector $v = (v_1, v_2, \ldots, v_\ell)$ such that $v_i = b\lambda_i$, for λ_i chosen uniformly at random from $\mathbb{Z}_p$. Now, consider the following proposition [3, 32],

Proposition 1 *A vector $\vec{\pi}$ is independent of a set of vectors represented by a matrix N if and only if there exists a vector $\vec{w}$ such that $N\vec{w} = \vec{0}$ while $\vec{\pi} \cdot \vec{w} \neq 0$.*

Since $\vec{1}$ is independent of M_γ, there exists a vector $\vec{w}$ such that $M_\gamma \vec{w} = \vec{0}$ and $\vec{1} \cdot \vec{w} \neq 0 (= h,$ say). Such a vector can be efficiently computed [3, 32]. Let $w = (w_1, w_2, \ldots, w_\ell)$. Finally define the vector $\vec{u}$ as follows:

$$\vec{u} = \vec{v} + \psi \vec{w} \qquad \text{where } \psi = \frac{ab - b\sum_{k=1}^{\ell} \lambda_k}{h}$$

Notice that,

$$u_i = v_i + \psi w_i = b\lambda_i + \frac{ab - b\sum_{k=1}^{\ell} \lambda_k}{h} \cdot w_i$$

Let $M_j = (x_{j1}, x_{j2}, \ldots, x_{j\ell})$. Give the secret key for the row M_j as follows. If $\rho(j) \in \gamma$, then:

$$D_j = B^{\phi_1} \qquad \text{where } \phi_1 = \frac{\sum_{i=1}^{\ell} x_{ji}\lambda_j}{r_{\rho(j)}}$$

Note that ϕ_1 is completely known. This is a legitimate key because,

$$\frac{\vec{M_j} \cdot \vec{u}}{t_{\rho(j)}} = \frac{\vec{M_j} \cdot (\vec{v} + \psi\vec{w})}{t_{\rho(j)}} = \frac{\vec{M_j} \cdot \vec{v} + \psi(\vec{M_j} \cdot \vec{w})}{t_{\rho(j)}} = \frac{\vec{M_j} \cdot \vec{v} + \psi \cdot 0}{t_{\rho(j)}} = b \cdot \frac{\sum_{i=1}^{\ell} x_{ji}\lambda_i}{r_{\rho(j)}} = b\phi_1$$

If $\rho(j) \notin \gamma$, then:

$$D_j = A^{\phi_2} g^{\phi_3} \qquad \text{where } \phi_2 = \frac{\sum_{i=1}^{\ell} x_{ji}}{h\beta_{\rho(j)}}, \quad \phi_3 = \frac{\sum_{i=1}^{\ell} x_{ji}(h\lambda_i - \sum_{k=1}^{\ell} \lambda_k)}{h\beta_{\rho(j)}}$$

Note that ϕ_2, ϕ_3 are completely known. This is a legitimate key because,

$$\frac{\vec{M_j} \cdot \vec{u}}{t_{\rho(j)}} = \frac{b\sum_{i=1}^{\ell} x_{ji}\left(\lambda_i + \frac{a - \sum_{k=1}^{\ell} \lambda_k}{h}\right)}{b\beta_{\rho(j)}} = a\left(\frac{\sum_{i=1}^{\ell} x_{ji}}{h\beta_{\rho(j)}}\right) + \left(\frac{\sum_{i=1}^{\ell} x_{ji}(h\lambda_i - \sum_{k=1}^{\ell} \lambda_k)}{h\beta_{\rho(j)}}\right) = a\phi_2 + \phi_3$$

Therefore, the simulator is able to construct a private key for the MSP $\hat{M}$. Furthermore, the distribution of the private key for $\hat{M}$ 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:

$$E = (\gamma, E' = m_\nu Z, \{E_i = C^{r_i}\}_{i \in \gamma})$$

If $\mu = 0$ then $Z = e(g,g)^{abc}$. If we let $s = c$, then we have $Y^s = (e(g,g)^{ab})^c = e(g,g)^{abc}$, and $E_i = (g^{r_i})^c = C^{r_i}$. 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 adversary's 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 Large Universe Construction

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

Theorem 4 *If an adversary can break our scheme in the Attribute-based Selective-Set 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-Set 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 generator g. 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.

Init The simulator $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ chooses the challenge set, γ, an n element set of members of $\mathbb{Z}_p$.

Setup The simulator assigns the public parameters $g_1 = A$ and $g_2 = B$. It then chooses a random n degree polynomial $f(X)$ and calculates an n degree polynomial $u(X)$ as follows:

set $u(X) = -X^n$ for all $X \in \gamma$ and $u(X) \neq -X^n$ for some other X. Because, $-X^n$ and $u(X)$ are two n degree polynomials, they will have at most n points in common or they are the same polynomial. This construction ensures that $\forall X, u(X) = -X^n$ if and only if $X \in \gamma$.

Now, the simulator sets $t_i = g_2^{u(i)} g^{f(i)}$ for all $i = 1$ to $n+1$. Because $f(X)$ is a random n degree polynomial, all t_i will be chosen independently at random as in the construction. Implicitly we have: $T(i) = g_2^{i^n + u(i)} g^{f(i)}$.

Phase 1 $\mathcal{A}$ adaptively makes requests for several access structures such that γ passes through none of them. Suppose $\mathcal{A}$ makes a request for the secret key for an access structure $\mathcal{T}$ 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 access tree.

Recall the functions PolySat and PolyUnsat from the proof of small universe case. The simulator simply runs PolyUnsat$(\mathcal{T}, \gamma, A)$ on the main tree $\mathcal{T}$. This defines a polynomial q_x for each node x of $\mathcal{T}$ such that $q_r(0) = a$. Furthermore, for each leaf node x of $\mathcal{T}$, we know the polynomial q_x completely if x is satisfied, and at least $g^{q_x(0)}$ if x is not satisfied.

Simulator now defines the main polynomial $Q_x(\cdot) = q_x(\cdot)$ for each node x of $\mathcal{T}$. Notice that this sets $y = Q_r(0) = a$. The key corresponding to each leaf node x is given using its polynomial as follows. Let $i = \text{att}(x)$.

- If $i \in \gamma$.

  $$D_x = g_2^{Q_x(0)} T(i)^{r_x} = g_2^{q_x(0)} T(i)^{r_x}$$
  $$R_x = g^{r_x}$$

  Where r_x is chosen at random from $\mathbb{Z}_p$.

- If $i \notin \gamma$. Let $g_3 = g^{Q_x(0)} = g^{q_x(0)}$.

  $$\begin{aligned} D_x &= g_3^{\frac{-f(i)}{i^n + u(i)}} (g_2^{i^n + u(i)} g^{f(i)})^{r'_x} \\ R_x &= g_3^{\frac{-1}{i^n + u(i)}} g^{r'_x} \end{aligned}$$

  Where r'_x is chosen at random from $\mathbb{Z}_p$.

The value $i^n + u(i)$ will be non-zero for all $i \notin \gamma$. This follows from our construction of $u(x)$. Above key components are legitimate because if we set $r_x = r'_x - \frac{q_x(0)}{i^n + u(i)}$ then,

$$\begin{aligned} D_x &= g_3^{\frac{-f(i)}{i^n + u(i)}} (g_2^{i^n + u(i)} g^{f(i)})^{r'_x} \\ &= g^{\frac{-q_x(0) \cdot f(i)}{i^n + u(i)}} (g_2^{i^n + u(i)} g^{f(i)})^{r'_x} \\ &= g_2^{q_x(0)} (g_2^{i^n + u(i)} g^{f(i)})^{\frac{-q_x(0)}{i^n + u(i)}} (g_2^{i^n + u(i)} g^{f(i)})^{r'_x} \\ &= g_2^{q_x(0)} (g_2^{i^n + u(i)} g^{f(i)})^{r'_x - \frac{q_x(0)}{i^n + u(i)}} \\ &= g_2^{q_x(0)} (T(i))^{r_x} \\ &= g_2^{Q_x(0)} T(i)^{r_x} \end{aligned}$$

and,

$$R_x = g_3^{\frac{-1}{i^n+u(i)}} g^{r'_x} = g^{r'_x - \frac{q_x(0)}{i^n+u(i)}} = g^{r_x}$$

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:

$$E = (\gamma, E^{'} = m_\nu Z, E'' = C, \{E_i = C^{f(i)}\}_{i\in\gamma})$$

If $\mu = 0$ then $Z = e(g,g)^{abc}$. Then the ciphertext is:

$$E = (\gamma, E^{'} = m_\nu e(g,g)^{abc}, E'' = g^c, \{E_i = (g^c)^{f(i)} = T(i)^c\}_{i\in\gamma})$$

This is a valid ciphertext for the 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 $\nu^{'}$ 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 $\frac{1}{2}\Pr[\mu^{'} = \mu|\mu = 0] + \frac{1}{2}\Pr[\mu^{'} = \mu|\mu = 1] - \frac{1}{2} = \frac{1}{2}(\frac{1}{2} + \epsilon) + \frac{1}{2}\frac{1}{2} - \frac{1}{2} = \frac{1}{2}\epsilon$.