Ciphertext-Policy Attribute-Based Encryption

John Bethencourt, Amit Sahai, Brent Waters

Ciphertext-Policy Attribute-Based Encryption

John Bethencourt
Carnegie Mellon University
bethenco@cs.cmu.edu

Amit Sahai *
UCLA
sahai@cs.ucla.edu

Brent Waters †
SRI International
bwaters@csl.sri.com

Abstract

In several distributed systems a user should only be able to access data if a user posses a certain set of credentials or attributes. Currently, the only method for enforcing such policies is to employ a trusted server to store the data and mediate access control. However, if any server storing the data is compromised, then the confidentiality of the data will be compromised. In this paper we present a system for realizing complex access control on encrypted data that we call Ciphertext-Policy Attribute-Based Encryption. By using our techniques encrypted data can be kept confidential even if the storage server is untrusted; moreover, our methods are secure against collusion attacks. Previous Attribute-Based Encryption systems used attributes to describe the encrypted data and built policies into user's keys; while in our system attributes are used to describe a user's credentials, and a party encrypting data determines a policy for who can decrypt. Thus, our methods are conceptually closer to traditional access control methods such as Role-Based Access Control (RBAC). In addition, we provide an implementation of our system and give performance measurements.

1 Introduction

In many situations, when a user encrypts sensitive data, it is imperative that she establish a specific access control policy on who can decrypt this data. For example, suppose that the FBI public corruption offices in Knoxville and San Francisco are investigating an allegation of bribery involving a San Francisco lobbyist and a Tennessee congressman. The head FBI agent may want to encrypt a sensitive memo so that only personnel that have certain credentials or attributes can access it. For instance, the head agent may specify the following access structure for accessing this information: (("Public Corruption Office" **AND** ("Knoxville" **OR** "San Francisco")) **OR** (management-level $>$ 5) **OR** "Name: Charlie Eppes").

By this, the head agent could mean that the memo should only be seen by agents who work at the public corruption offices at Knoxville or San Francisco, FBI officials very high up in the management chain, and a consultant named Charlie Eppes.

As illustrated by this example, it can be crucial that the person in possession of the secret data be able to choose an access policy based on specific knowledge of the underlying data. Furthermore, this person may not know the exact identities of all other people who should be able to access the data, but rather she may only have a way to describe them in terms of descriptive attributes or credentials.

Traditionally, this type of expressive access control is enforced by employing a trusted server to store data locally. The server is entrusted as a reference monitor that checks that a user presents proper certification before allowing him to access records or files. However, services are increasingly storing data in a distributed fashion across many servers. Replicating data across several locations has advantages in both performance and reliability. The drawback of this trend is that it is increasingly difficult to guarantee the security of data using traditional methods; when data is stored at several locations, the chances that one of them has been compromised increases dramatically. For these reasons we would like to require that sensitive data is stored in an encrypted form so that it will remain private even if a server is compromised.

Most existing public key encryption methods allow a party to encrypt data to a particular user, but are unable to efficiently handle more expressive types of encrypted access control such as the example illustrated above.

*Supported the US Army Research Office under the CyberTA Grant No. W911NF-06-1-0316.

†Supported by NSF CNS-0524252 and the US Army Research Office under the CyberTA Grant No. W911NF-06-1-0316.

Our contribution. In this work, we provide the first construction of a *ciphertext-policy attribute-based encryption (CP-ABE)* to address this problem, and give the first construction of such a scheme. In our system, a user's private key will be associated with an arbitrary number of attributes expressed as strings. On the other hand, when a party encrypts a message in our system, they specify an associated access structure over attributes. A user will only be able to decrypt a ciphertext if that user's attributes pass through the ciphertext's access structure. At a mathematical level, access structures in our system are described by a monotonic "access tree", where nodes of the access structure are composed of threshold gates and the leaves describe attributes. We note that **AND** gates can be constructed as n-of-n threshold gates and **OR** gates as 1-of-n threshold gates. Furthermore, we can handle more complex access controls such as numeric ranges by converting them to small access trees (see discussion in the implementation section for more details).

Our techniques. At a high level, our work is similar to the recent work of Sahai and Waters [24] and Goyal et al. [15] on key-policy attribute based encryption (KP-ABE), however we require substantially new techniques. In key-policy attribute based encryption, ciphertexts are associated with sets of descriptive attributes, and users' keys are associated with policies (the reverse of our situation). *We stress that in key-policy ABE, the encryptor exerts no control over who has access to the data she encrypts, except by her choice of descriptive attributes for the data.* Rather, she must trust that the key-issuer issues the appropriate keys to grant or deny access to the appropriate users. In other words, in [24, 15], the "intelligence" is assumed to be with the key issuer, and not the encryptor. In our setting, the encryptor must be able to intelligently decide who should or should not have access to the data that she encrypts. As such, the techniques of [24, 15] do not apply to our setting, and we must develop new techniques.

At a technical level, the main objective that we must attain is *collusion-resistance*: If multiple users collude, they should only be able to decrypt a ciphertext if at least one of the users could decrypt it on their own. In particular, referring back to the example from the beginning of this Introduction, suppose that an FBI agent that works in the terrorism office in San Francisco colludes with a friend who works in the public corruption office in New York. We do not want these colluders to be able to decrypt the secret memo by combining their attributes. This type of security is the *sine qua non* of access control in our setting.

In the work of [24, 15], collusion resistance is insured by using a secret-sharing scheme and embedding independently chosen secret shares into each private key. Because of the independence of the randomness used in each invocation of the secret sharing scheme, collusion-resistance follows. In our scenario, users' private keys are associated with *sets* of attributes instead of access structures over them, and so secret sharing schemes do not apply.

Instead, we devise a novel private key randomization technique that uses a new two-level random masking methodology. This methodology makes use of groups with efficiently computable bilinear maps, and it is the key to our security proof, which we give in the generic bilinear group model [6, 28].

Finally, we provide an implementation of our system to show that our system performs well in practice. We provide a description of both our API and the structure of our implementation. In addition, we provide several techniques for optimizing decryption performance and measure our performance features experimentally.

Organization. The remainder of our paper is structured as follows. In Section 2 we discuss related work. In Section 3 we our definitions and give background on groups with efficiently computable bilinear maps. We then give our construction in Section 4. We then present our implementation and performance measurements in Section 5. Finally, we conclude in Section 6.

2 Related Work

Sahai and Waters [24] introduced attribute-based encryption (ABE) as a new means for encrypted access control. In an attribute-based encryption system ciphertexts are not necessarily encrypted to one particular user as in traditional public key cryptography. Instead both users' private keys and ciphertexts will be associated with a set of attributes or a policy over attributes. A user is able to decrypt a ciphertext if there is a "match" between his private key and the ciphertext. In their original system Sahai and Waters presented a Threshold ABE system in which ciphertexts were labeled with a set of attributes S and a user's private key was associated with both a threshold parameter k and another set of attributes S'. In order for a user to decrypt a ciphertext at least k attributes must overlap between the ciphertext and his private keys. One of the primary original motivations for this was to design an error-tolerant (or Fuzzy) identity-based encryption [27, 7, 12] scheme that could use biometric identities.

The primary drawback of the Sahai-Waters [24] threshold ABE system is that the threshold semantics are not very expressive and therefore are limiting for designing more general systems. Goyal et al. introduced the idea of a more general *key-policy* attribute-based encryption system. In their construction a ciphertext is associated with a set of attributes and a user's key can be associated with any monotonic tree-access structure. [1] The construction of Goyal et al. can be viewed as an extension of the Sahai-Waters techniques where instead of embedding a Shamir [26] secret sharing scheme in the private key, the authority embeds a more general secret sharing scheme for monotonic access trees. Goyal et. al. also suggested the possibility of a ciphertext-policy ABE scheme, but did not offer any constructions.

Pirretti et al. [23] gave an implementation of the threshold ABE encryption system, demonstrated different applications of attribute-based encryption schemes and addressed several practical notions such as key-revocation. In recent work, Chase [11] gave a construction for a multi-authority attribute-based encryption system, where each authority would administer a different domain of attributes. The primary challenge in creating multi-authority ABE is to prevent collusion attacks between users that obtain key components from different authorities. While the Chase system used the threshold ABE system as its underlying ABE system at each authority, the problem of multi-authority ABE is in general orthogonal to finding more expressive ABE systems.

In addition, there is a long history of access control for data that is mediated by a server. See for example, [18, 14, 30, 20, 16, 22] and the references therein. We focus on encrypted access control, where data is protected even if the server storing the data is compromised.

Collusion Resistance and Attribute-Based Encryption The defining property of Attribute-Based Encryption systems are their resistance to collusion attacks. This property is critical for building cryptographic access control systems; otherwise, it is impossible to guarantee that a system will exhibit the desired security properties as there will exist devastating attacks from an attacker that manages to get a hold of a few private keys. While we might consider ABE systems with different flavors of expressibility, prior work [24, 15] made it clear that collusion resistance is a required property of any ABE system.

Before attribute-based encryption was introduced there were other systems that attempted to address access control of encrypted data [29, 8] by using secret sharing schemes [17, 9, 26, 5, 3] combined with identity-based encryption; however, these schemes did not address resistance to collusion attacks. Recently, Kapadia, Tsang, and Smith [19] gave a cryptographic access control scheme that employed proxy servers. Their work explored new methods for employing proxy servers to hide policies and use non-monontonic access control for small universes of attributes. We note that although they called this scheme a form of CP-ABE, the scheme does not have the property of collusion resistance. As such, we believe that their work should not be considered in the class of attribute-based encryption systems due to its lack of security against collusion attacks.

[1] Goyal et al. show in addition how to construct a key-policy ABE scheme for any linear secret sharing scheme.

3 Background

We first give formal definitions for the security of ciphertext policy attribute based encryption (CP-ABE). Next, we give background information on bilinear maps. Like the work of Goyal et al. [15] we define an access structure and use it in our security definitions. However, in these definitions the attributes will describe the users and the access structures will be used to label different sets of encrypted data.

3.1 Definitions

Definition 1 (Access Structure [1]) *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 ciphertext-policy attribute based encryption scheme consists of four fundamental algorithms: Setup, Encrypt, KeyGen, and Decrypt. In addition, we allow for the option of a fifth algorithm Delegate.

Setup. The setup algorithm takes no input other than the implicit security parameter. It outputs the public parameters PK and a master key MK.

Encrypt(PK, M, $\mathbb{A}$). The encryption algorithm takes as input the public parameters PK, a message M, and an access structure $\mathbb{A}$ over the universe of attributes. The algorithm will encrypt M and produce a ciphertext CT such that only a user that possesses a set of attributes that satisfies the access structure will be able to decrypt the message. We will assume that the ciphertext implicitly contains $\mathbb{A}$.

Key Generation(MK, S). The key generation algorithm takes as input the master key MK and a set of attributes S that describe the key. It outputs a private key SK.

Decrypt(PK, CT, SK). The decryption algorithm takes as input the public parameters PK, a ciphertext CT, which contains an access policy $\mathbb{A}$, and a private key SK, which is a private key for a set S of attributes. If the set S of attributes satisfies the access structure $\mathbb{A}$ then the algorithm will decrypt the ciphertext and return a message M.

Delegate(SK, $\tilde{S}$). The delegate algorithm takes as input a secret key SK for some set of attributes S and a set $\tilde{S} \subseteq S$. It output a secret key $\tilde{\text{SK}}$ for the set of attributes $\tilde{S}$.

We now describe a security model for ciphertext-policy ABE schemes. Like identity-based encryption schemes [27, 7, 12] the security model allows the adversary to query for any private keys that cannot be used to decrypt the challenge ciphertext. In CP-ABE the ciphertexts are identified with access structures and the private keys with attributes. It follows that in our security definition the adversary will choose to be challenged on an encryption to an access structure $\mathbb{A}^*$ and can ask for any private key S such that S does not satisfy $\mathbb{S}^*$. We now give the formal security game.

Security Model for CP-ABE

Setup. The challenger runs the Setup algorithm and gives the public parameters, PK to the adversary.

Phase 1. The adversary makes repeated private keys corresponding to sets of attributes $S_1, \ldots, S_{q_1}$.

Challenge. The adversary submits two equal length messages M_0 and M_1. In addition the adversary gives a challenge access structure $\mathbb{A}^*$ such that none of the sets $S_1, \ldots, S_{q_1}$ from Phase 1 satisfy the access structure. The challenger flips a random coin b, and encrypts M_b under $\mathbb{A}^*$. The ciphertext CT^* is given to the adversary.

Phase 2. Phase 1 is repeated with the restriction that none of sets of attributes $S_{q_1+1}, \ldots, S_q$ satisfy the access structure corresponding to the challenge.

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 ciphertext-policy attribute-based encryption scheme is secure if all polynomial time adversaries have at most a negligible advantage in the above game.*

3.2 Bilinear Maps

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

Let $\mathbb{G}_0$ and $\mathbb{G}_1$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}_0$ and e be a bilinear map, $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. The bilinear map e has the following properties:

1. Bilinearity: for all $u, v \in \mathbb{G}_0$ 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 $\mathbb{G}_0$ is a bilinear group if the group operation in $\mathbb{G}_0$ and the bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ are both efficiently computable. Notice that the map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

4 Our Construction

In this section we provide the construction of our system. We begin by describing the model of access trees and attributes for respectively describing ciphertexts and private keys. Next, we give the description of our scheme. Finally, we follow with a discussion of security, efficiency, and key revocation. We provide our proof of security in Appendix A.

4.1 Our Model

In our construction private keys will be identified with a set S of descriptive attributes. A party that

wishes to encrypt a message will specify through an access tree structure a policy that private keys must satisfy in order to decrypt.

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 private key to nodes of the tree such that the tree is satisfied. We use the same notation as [15] to describe the access trees, even though in our case the attributes are used to identify the keys (as opposed to the data).

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 $\mathsf{parent}(x)$. The function $\mathsf{att}(x)$ 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 $\mathsf{index}(x)$ 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 $\mathsf{att}(x) \in \gamma$.

4.2 Our Construction

Let $\mathbb{G}_0$ be a bilinear group of prime order p, and let g be a generator of $\mathbb{G}_0$. In addition, let $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$ 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 additionally employ a hash function $H : \{0,1\}^* \to \mathbb{G}_0$ that we will model as a random oracle. The function will map any attribute described as a binary string to a random group element. Our construction follows.

Setup. The setup algorithm will choose a bilinear group $\mathbb{G}_0$ of prime order p with generator g. Next it will choose two random exponents $\alpha, \beta \in \mathbb{Z}_p$. The public key is published as:

$$\text{PK} = \mathbb{G}_0, g, h = g^\beta, f = g^{1/\beta}, e(g,g)^\alpha$$

and the master key MK is (β, g^α). (Note that f is used only for delegation.)

Encrypt(PK, M, $\mathcal{T}$). The encryption algorithm encrypts a message M under the tree access structure $\mathcal{T}$. The algorithm first chooses 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, $d_x = k_x - 1$.

Starting with the root node R the algorithm chooses a random $s \in \mathbb{Z}_p$ and sets $q_R(0) = s$. Then, it chooses d_R other points of the polynomial q_R randomly to define it completely. For any other node x, it sets $q_x(0) = q_{\mathsf{parent}(x)}(\mathsf{index}(x))$ and chooses d_x other points randomly to completely define q_x.

Let, Y be the set of leaf nodes in $\mathcal{T}$. The ciphertext is then constructed by giving the tree access structure $\mathcal{T}$ and computing

$$\text{CT} = \big(\mathcal{T},\ \tilde{C} = Me(g,g)^{\alpha s},\ C = h^s,\\ \forall y \in Y: \quad C_y = g^{q_y(0)}, C'_y = H(\mathsf{att}(y))^{q_y(0)}\big).$$

KeyGen(MK, S). The key generation algorithm will take as input a set of attributes S and output a key that identifies with that set. The algorithm first chooses a random $r \in \mathbb{Z}_p$, and then random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. Then it computes the key as

$$\text{SK} = \big(D = g^{(\alpha+r)/\beta},\\ \forall j \in S: \quad D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}\big).$$

Delegate(SK, $\tilde{S}$). The delegation algorithm takes in a secret key SK, which is for a set S of attributes, and another set $\tilde{S}$ such that $\tilde{S} \subseteq S$. The secret key is of the form $\text{SK} = (D,\ \forall j \in S:\ D_j, D'_j)$. The algorithm

chooses random $\tilde{r}$ and $\tilde{r}_k \forall k \in \tilde{S}$. Then it creates a new secret key as

$$\tilde{\text{SK}} = (\tilde{D} = Df^{\tilde{r}}, \quad \forall k \in \tilde{S}: \quad \tilde{D}_k = D_k g^{\tilde{r}} H(k)^{\tilde{r}_k}, \tilde{D}'_k = D'_k g^{\tilde{r}_k}).$$

The resulting secret key $\tilde{\text{SK}}$ is a secret key for the set $\tilde{S}$. Since the algorithm re-randomizes the key, a delegated key is equivalent to one received directly from the authority.

Decrypt(CT, SK). 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 $\text{DecryptNode}(\text{CT}, \text{SK}, x)$ that takes as input a ciphertext $\text{CT} = (\mathcal{T}, \tilde{C}, C, \forall y \in Y: \ C_y, C'_y)$, a private key SK, which is associated with a set S of attributes, and a node x from $\mathcal{T}$.

If the node x is a leaf node then we let $i = \mathsf{att}(x)$ and define as follows: If $i \in S$, then

$$\begin{aligned}\text{DecryptNode}(\text{CT}, \text{SK}, x) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} \\ &= \frac{e\left(g^r \cdot H(i)^{r_i}, h^{q_x(0)}\right)}{e(g^{r_i}, H(i)^{q_x(0)})} \\ &= e(g,g)^{rq_x(0)}.\end{aligned}$$

If $i \notin S$, then we define $\text{DecryptNode}(\text{CT}, \text{SK}, x) = \perp$.

We now consider the recursive case when x is a non-leaf node. The algorithm $\text{DecryptNode}(\text{CT}, \text{SK}, x)$ then proceeds as follows: For all nodes z that are children of x, it calls $\text{DecryptNode}(CT, \text{SK}, z)$ 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

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

and return the result.

Now that we have defined our function DecryptNode, we can define the decryption algorithm. The algorithm begins by simply calling the function on the root node R of the tree $\mathcal{T}$. If the tree is satisfied by S we set $A = \text{DecryptNode}(\text{CT}, \text{SK}, r) = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$. The algorithm now decrypts by computing

$$\tilde{C}/(e(C,D)/A) = \tilde{C}/\left(e\left(h^s, g^{(\alpha+r)/\beta}\right)/e(g,g)^{rs}\right) = M.$$

4.3 Discussion

We now provide a brief discussion about the security intuition for our scheme (a full proof is given in Appendix A), our scheme's efficiency, and how we might handle key revocation.

Security intuition. As in previous attribute-based encryption schemes the main challenge in designing our scheme was to prevent against attacks from colluding users. Like the scheme of Sahai and Waters [24] our solution randomizes users private keys such that they cannot be combined; however, in our solution the secret sharing must be embedded into the ciphertext instead to the private keys. In order to decrypt an attacker clearly must recover $e(g,g)^{\alpha s}$. In order to do this the attacker must pair C from the ciphertext with the D component from some user's private key. This will result in the desired value $e(g,g)^{\alpha s}$, but blinded by some value $e(g,g)^{rs}$. This value can be blinded out if and only if enough the user has the correct key components to satisfy the secret sharing scheme embedded in the ciphertext. Collusion attacks won't help since the blinding value is randomized to the randomness from a particular user's private key.

While we described our scheme to be secure against chosen plaintext attacks, the security of our scheme can efficiently be extended to chosen ciphertext attacks by applying a random oracle technique such as that of the the Fujisaki-Okamoto transformation [13]. Alternatively, we can leverage the delegation mechanism of our scheme and apply the Cannetti, Halevi, and Katz [10] method for achieving CCA-security.

Efficiency. The efficiencies of the key generation and encryption algorithms are both fairly straightforward. The encryption algorithm will require two exponentiations for each leaf in the ciphertext's access tree. The ciphertext size will include two group elements for each tree leaf. The key generation algorithm requires two exponentiations for every attribute given to the user, and the private key consists of two group elements for

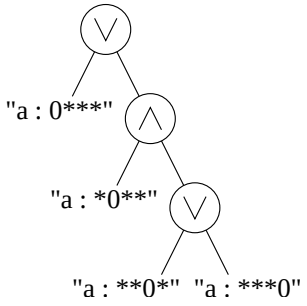


Figure 1. Policy tree implementing the integer comparison "a < 11".

every attribute. In its simplest form, the decryption algorithm could require two pairings for every leaf of the access tree that is matched by a private key attribute and (at most[2]) one exponentiation for each node along a path from such a leaf to the root. However, there might be several ways to satisfy a policy, so a more intelligent algorithm might try to optimize along these lines. In our implementation description in Section 5 we described various performance enhancements.

Key-revocation and numerical attributes. Key-Revocation is typically a difficult issue in identity-based encryption [27, 7] and related schemes. The core challenge is that since the party encrypting the data does not obtain the receiver's certificate on-line, he is not able to check if the the receiving party is revoked. In attribute-based encryption the problem is even more tricky since several different users might match the decryption policy. The usual solution is to append to each of the identities or descriptive attributes a date for when the attribute expires. For instance, Pirretti et al. [23] suggest extending each attribute with an expiration date. For example, instead of using the attribute "Computer Science" we might use the attribute "Computer Science: Oct 17, 2006".

This type of method has a several shortcomings. Since the attributes incorporate an exact date there must be agreement on this between the party encrypting the data and the key issuing authority. If we wish for a party to be able to specify policy about revocation dates on a fine-grained scale, users will be forced to go often to the authority and maintain a large amount of private key storage, a key for every time period.

Ideally, we would like an attribute-based encryption system to allow a key authority to give out a single key with some expiration date X rather than a separate key for every time period before X. When a party encrypts a message on some date Y, a user with a key expiring on date X should be able to decrypt iff $X \geq Y$ and the rest of the policy matches the user's attributes. In this manner, different expiration dates can be given to different users and there does not need to be any close coordination between the parties encrypting data and the authority.

This sort of functionality can be realized by extending our attributes to support numerical values and our policies to support integer comparisons. To represent a numerical attribute "a $= k$" for some n-bit integer k we convert it into a "bag of bits" representation, producing n (non-numerical) attributes which specify the value of each bit in k. As an example, to give out a private key with the 4-bit attribute "a $= 9$", we would instead include "a : 1***", "a : *0***", "a : **0*", and "a : ***1" in the key. We can then use policies of AND and OR gates to implement integer comparisons over such attributes, as shown for "a < 11" in Figure 1. There is a direct correspondence between the bits of the constant 11 and the choice of gates. Policies for $\leq$, $>$, $\geq$, and $=$ can be implemented similarly with at most n gates, or possibly fewer depending on the constant. It is also possible to construct comparisons between two numerical attributes (rather than an attribute and a constant) using roughly $3n$ gates, although it is less clear when this would be useful in practice.

5 Implementation

In this section we discuss practical issues in implementing the construction of Section 4, including several optimizations, a description of the toolkit we have developed, and measurements of its performance.

5.1 Decryption Efficiency Improvements

While little can be done to reduce the group operations necessary for the setup, key generation, and encryption algorithms, the efficiency of the decryption algorithm can be improved substantially with novel techniques. We explain these improvements here and later give measurements showing their effects in Section 5.3.

Optimizing the decryption strategy. The recursive algorithm given in Section 4 results in two pairings for each leaf node that is matched by a private key attribute, and up to one exponentiation for every node occurring along the path from such a node to the root (not including the root). The final step after the recursive portion adds an additional pairing. Of course, at each internal node with threshold k, the results from

[2]Fewer exponentiations may occur if there is an unsatisfied internal node along the path.

all but k of its children are thrown away. By considering ahead of time which leaf nodes are satisfied and picking a subset of them which results in the satisfaction of the entire access tree, we may avoid evaluating DecryptNode where the result will not ultimately be used.

More precisely, let M be a subset of the nodes in an access tree $\mathcal{T}$. We define $\mathsf{restrict}(\mathcal{T}, M)$ to be the access tree formed by removing the following nodes from $\mathcal{T}$ (while leaving the thresholds unmodified). First, we remove all nodes not in M. Next we remove any node not connected to the original root of $\mathcal{T}$ along with any internal node x that now has fewer children than its threshold k_x. This is repeated until no further nodes are removed, and the result is $\mathsf{restrict}(\mathcal{T}, M)$. So given an access tree $\mathcal{T}$ and a set of attributes γ that satisfies it, the natural problem is to pick a set M such that γ satisfies $\mathsf{restrict}(\mathcal{T}, M)$ and the number of leaves in M is minimized (considering pairing to be the most expensive operation). This is easily accomplished with a straightforward recursive algorithm that makes a single traversal of the tree. We may then use DecryptNode on $\mathsf{restrict}(\mathcal{T}, M)$ with the same result.

Direct computation of DecryptNode. Further improvements may be gained by abandoning the DecryptNode function and making more direct computations. Intuitively, we imagine flattening out the tree of recursive calls to DecryptNode, then combining the exponentiations into one per (used) leaf node. Precisely, let $\mathcal{T}$ be an access tree with root r, γ be a set of attributes, and $M \subseteq \mathcal{T}$ be such that γ satisfies $\mathsf{restrict}(\mathcal{T}, M)$. Assume also that M is minimized so that no internal node has more children than its threshold. Let $L \subseteq M$ be the leaf nodes in M. Then for each $\ell \in L$, we denote the path from ℓ to r as

$$\rho(\ell) = (\ell, \mathsf{parent}(\ell), \mathsf{parent}(\mathsf{parent}(\ell)), \ldots r) \ .$$

Also, denote the set of siblings of a node x (including itself) as $\mathsf{sibs}(x) = \{\, y \mid \mathsf{parent}(x) = \mathsf{parent}(y) \,\}$. Given this notation, we may proceed to directly compute the result of
$\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}, r)$. First, for each $\ell \in L$, compute z_ℓ as follows.

$$z_\ell = \prod_{\substack{x \in \rho(\ell) \\ x \neq r}} \Delta_{i,S}(0) \qquad \text{where } \begin{matrix} i = \mathsf{index}(x) \\ S = \{\, \mathsf{index}(y) \mid y \in \mathsf{sibs}(x) \,\} \end{matrix}$$

Then

$$\mathrm{DecryptNode}(\mathrm{CT}, \mathrm{SK}, r) = \prod_{\substack{\ell \in L \\ i = \mathsf{att}(\ell)}} \left(\frac{e(D_i, C_\ell)}{e(D'_i, C'_\ell)} \right)^{z_\ell} \ .$$

Using this method, the number of exponentiations in the entire decryption algorithm is reduced from $|M|-1$ (i.e., one for every node but the root) to $|L|$. The number of pairings is $2|L|$.

Merging pairings. Still further reductions (this time in the number of pairings) are possible by combining leaves using the same attribute. If $\mathsf{att}(\ell_1) = \mathsf{att}(\ell_2) = i$ for some ℓ_1, ℓ_2 in L, then

$$\begin{aligned} &\left(\frac{e(D_i, C_{\ell_1})}{e(D'_i, C'_{\ell_1})} \right)^{z_{\ell_1}} \cdot \left(\frac{e(D_i, C_{\ell_2})}{e(D'_i, C'_{\ell_2})} \right)^{z_{\ell_2}} \\ =& \frac{e(D_i, C_{\ell_1}^{z_{\ell_1}})}{e(D'_i, C_{\ell_1}^{\prime\, z_{\ell_1}})} \cdot \frac{e(D_i, C_{\ell_2}^{z_{\ell_2}})}{e(D'_i, C_{\ell_2}^{\prime\, z_{\ell_2}})} \\ =& \frac{e(D_i, C_{\ell_1}^{z_{\ell_1}} \cdot C_{\ell_2}^{z_{\ell_2}})}{e(D'_i, C_{\ell_1}^{\prime\, z_{\ell_1}} \cdot C_{\ell_2}^{\prime\, z_{\ell_2}})} \ . \end{aligned}$$

Using this fact, we may combine all the pairings for each distinct attribute in L, reducing the total pairings to $2m$, where m is the number of distinct attributes appearing in L. Note, however, that the number of exponentiations increases, and some of the exponentiations must now be performed in $\mathbb{G}_0$ rather than $\mathbb{G}_1$. Specifically, if m' is the number of leaves sharing their attribute with at least one other leaf, we must perform $2m'$ exponentiations in $\mathbb{G}_0$ and $|L| - m'$ in $\mathbb{G}_1$, rather than zero and $|L|$ respectively. If exponentiations in $\mathbb{G}_0$ (an elliptic curve group) are slower than in $\mathbb{G}_1$ (a finite field of the same order), this technique has the potential to increase decryption time. We further investigate this tradeoff in Section 5.3.

5.2 The `cpabe` Toolkit

We have implemented the construction of Section 4 as a convenient set of tools we call the `cpabe` package [4], which has been made available on the web under the GPL. The implementation uses the Pairing Based Cryptography (PBC) library [21].[3] The interface of the toolkit is designed for straightforward invocation by larger systems in addition to manual usage. It provides four command line tools.

`cpabe-setup`
: Generates a public key and a master key.

`cpabe-keygen`
: Given a master key, generates a private key for a set of attributes, compiling numerical attributes as necessary.

[3]PBC is in turn based on the GNU Multiple Precision arithmetic library (GMP), a high performance arbitrary precision arithmetic implementation suitable for cryptography.

```
$ cpabe-keygen -o sara_priv_key pub_key master_key \
    sysadmin it_department 'office = 1431' 'hire_date = '`date +%s`

$ cpabe-keygen -o kevin_priv_key pub_key master_key \
    business_staff strategy_team 'executive_level = 7' \
    'office = 2362' 'hire_date = '`date +%s`

$ cpabe-enc pub_key security_report.pdf
    (sysadmin and (hire_date < 946702800 or security_team)) or
    (business_staff and 2 of (executive_level >= 5, audit_group, strategy_team))
```

Figure 2. Example usage of the `cpabe` toolkit. Two private keys are issued for various sets of attributes (normal and numerical) using `cpabe-keygen`. A document is encrypted under a complex policy using `cpabe-enc`.

`cpabe-enc`
: Given a public key, encrypts a file under an access tree specified in a policy language.

`cpabe-dec`
: Given a private key, decrypts a file.

The `cpabe` toolkit supports the numerical attributes and range queries described in Section 4.3 and provides a familiar language of expressions with which to specify access policies. These features are illustrated in the sample usage session of Figure 2.

In this example, the `cpabe-keygen` tool was first used to produce private keys for two new employees, "Sara" and "Kevin". A mix of regular and numerical attributes were specified; in particular shell backticks were used to store the current timestamp (in seconds since 1970) in the "hire_date" attribute. The `cpabe-enc` tool was then used to encrypt a security sensitive report under a complex policy (in this case specified on the standard input). The policy allows decryption by sysadmins with at least a certain seniority (hired before January 1, 2000) and those on the security team. Members of the business staff may decrypt if they are in the audit group and the strategy team, or if they are in one of those teams and are an executive of "level" five or more. So in this example, Kevin would be able to use the key stored as `kevin_priv_key` to decrypt the resulting document, but Sara would not be able to use hers to decrypt the document.

As demonstrated by this example, the policy language allows the general threshold gates of the underlying scheme, but also provides AND and OR gates for convenience. These are appropriately merged to simplify the tree, that is, specifying the policy "(a and b) and (c and d and e)" would result in a single gate. The tools also handle compiling numerical attributes to their "bag of bits" representation and comparisons into their gate-level implementation.

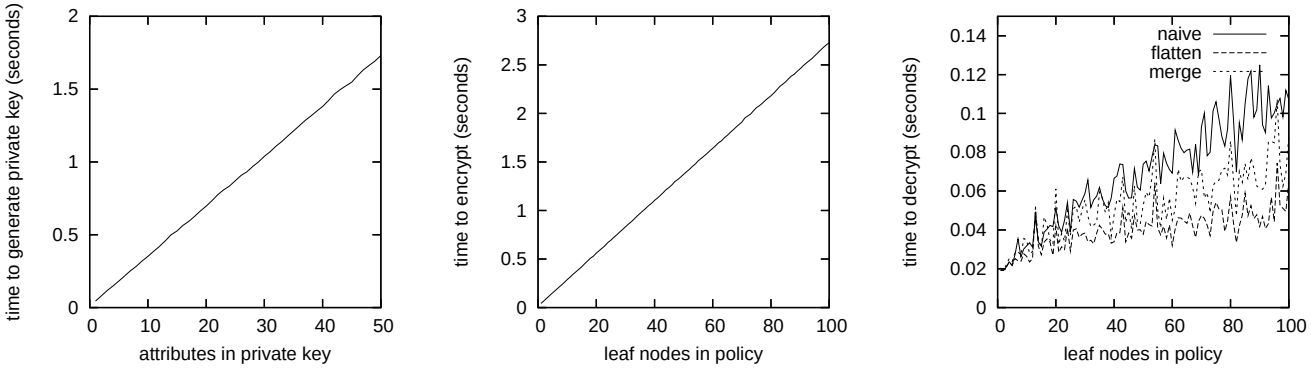
5.3 Performance Measurements

We now provide some information on the performance achieved by the `cpabe` toolkit. Figure 3 displays measurements of private key generation time, encryption time, and decryption time produced by running `cpabe-keygen`, `cpabe-enc`, and `cpabe-dec` on a range of problem sizes. The measurements were taken on a modern workstation.[4] The implementation uses a 160-bit elliptic curve group based on the supersingular curve $y^2 = x^3 + x$ over a 512-bit finite field. On the test machine, the PBC library can compute pairings in approximately 5.5ms, and exponentiations in $\mathbb{G}_0$ and $\mathbb{G}_1$ take about 6.4ms and 0.6ms respectively. Randomly selecting elements (by reading from the Linux kernel's `/dev/urandom`) is also a significant operation, requiring about 16ms for $\mathbb{G}_0$ and 1.6ms for $\mathbb{G}_1$.

As expected, `cpabe-keygen` runs in time precisely linear in the number of attributes associated with the key it is issuing. The running time of `cpabe-enc` is also almost perfectly linear with respect to the number of leaf nodes in the access policy. The polynomial operations at internal nodes amount to a modest number of multiplications and do not significantly contribute to the running time. Both remain quite feasible for even the largest problem instances.

The performance of `cpabe-dec` is somewhat more interesting. It is slightly more difficult to measure in the absence of a precise application, since the decryption time can depend significantly on the particular access trees and set of attributes involved. In an at-

[4] The workstation's processor is a 64-bit, 3.2 Ghz Pentium 4.



(a) Key generation time.

(b) Encryption time.

(c) Decryption time with various levels of optimization.

Figure 3. Performance of the `cpabe` toolkit.

tempt to average over this variation, we ran `cpabe-dec` on a series of ciphertexts that had been encrypted under randomly generated policy trees of various sizes. The trees were generated by starting with only a root node, then repeatedly adding a child to a randomly selected node until the desired number of leaf nodes was reached. At that point random thresholds were selected for each internal node. Since the time to decrypt also depends on the particular attributes available, for each run of `cpabe-dec`, we selected a key uniformly at random from all keys satisfying the policy. This was accomplished by iteratively taking random subsets of the attributes appearing in leaves of the tree and discarding those that did not satisfy it. A series of runs of `cpabe-dec` conducted in this manner produced the running times displayed in Figure 3 (c).

These measurements give some insight into the effects of the optimizations described in Section 5.1 (all of which are implemented in the system). The line marked "naive" denotes the decryption time resulting from running the recursive DecryptNode algorithm and arbitrarily selecting nodes to satisfy each threshold gate. By ensuring that the final number of leaf nodes is minimized when making these decisions and replacing the DecryptNode algorithm with the "flattened" algorithm to reduce exponentiations, we obtain the improved times denoted "flatten". Perhaps most interestingly, employing the technique for merging pairings between leaf nodes sharing the same attribute, denoted "merge", actually increases running time in this case, due to fact that exponentiations are more expensive in $\mathbb{G}_0$ than in $\mathbb{G}_1$.

In summary, `cpabe-keygen` and `cpabe-enc` run in a predictable amount of time based on the number of attributes in a key or leaves in a policy tree. The performance of `cpabe-dec` depends on the specific access tree of the ciphertext and the attributes available in the private key, and can be improved by some of the optimizations considered in Section 5.1. In all cases, the toolkit consumes almost no overhead beyond the cost of the underlying group operations and random selection of elements. Large private keys and policies are possible in practice while maintaining reasonable running times.

6 Conclusions and Open Directions

We created a system for Ciphertext-Policy Attribute Based Encryption. Our system allows for a new type of encrypted access control where user's private keys are specified by a set of attributes and a party encrypting data can specify a policy over these attributes specifying which users are able to decrypt. Our system allows policies to be expressed as any monotonic tree access structure and is resistant to collusion attacks in which an attacker might obtain multiple private keys. Finally, we provided an implementation of our system, which included several optimization techniques.

In the future, it would be interesting to consider attribute-based encryption systems with different types of expressibility. While, Key-Policy ABE and Ciphertext-Policy ABE capture two interesting and complimentary types of systems there certainly exist other types of systems. The primary challenge in this line of work is to find a new systems with elegant forms of expression that produce more than an arbitrary combination of techniques.

One limitation of our system is that it is proved secure under the generic group heuristic. We believe an important endeavor would be to prove a system secure under a more standard and non-interactive assumption. This type of work would be interesting even if it resulted in a moderate loss of efficiency from our existing system.

References

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

[2] 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.

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

[4] J. Bethencourt, A. Sahai, and B. Waters. The `cpabe` toolkit. `http://acsc.csl.sri.com/cpabe/`.

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

[6] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[7] 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.

[8] R. W. Bradshaw, J. E. Holt, and K. E. Seamons. Concealing complex policies with hidden credentials. In *ACM Conference on Computer and Communications Security*, pages 146–157, 2004.

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

[10] 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.

[11] M. Chase. Multi-authority attribute-based encryption. In *(To Appear) The Fourth Theory of Cryptography Conference (TCC 2007)*, 2007.

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

[13] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999.

[14] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. 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.

[15] 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.

[16] H. Harney, A. Colgrove, and P. D. McDaniel. Principles of policy in secure groups. In *NDSS*, 2001.

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

[18] M. H. Kang, J. S. Park, and J. 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.

[19] A. Kapadia, P. Tsang, and S. Smith. Attribute-based publishing with hidden credentials and hidden policies. In *NDSS*, 2007.

[20] J. Li, N. Li, and W. H. Winsborough. Automated trust negotiation using cryptographic credentials. In *ACM Conference on Computer and Communications Security*, pages 46–57, 2005.

[21] B. Lynn. The Pairing-Based Cryptography (PBC) library. `http://crypto.stanford.edu/pbc`.

[22] P. D. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. In *IEEE Symposium on Security and Privacy*, pages 73–87, 2002.

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

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

[25] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

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

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

[28] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

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

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

[31] R. Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

A Security Proof

In this section, we use the generic bilinear group model of [6, 28] and the random oracle model [2] to argue that no efficient adversary that acts generically on the groups underlying our scheme can break the security of our scheme with any reasonable probability. At an intuitive level, this means that if there are any vulnerabilities in our scheme, then these vulnerabilities must exploit specific mathematical properties of elliptic curve groups or cryptographic hash functions used when instantiating our construction.

While from a security standpoint, it would be preferable to have a proof of security that reduces the problem of breaking our scheme to a well-studied complexity-theoretic problem, there is reason to believe that such reductions will only exist for more complex (and less efficient) schemes than the one we give here. We also stress that ours is the first construction which offers the security properties we are proposing here; we strongly encourage further research that can place this kind of security on a firmer theoretical foundation.

The generic bilinear group model. We follow [6] here: We consider two random encodings ψ_0, ψ_1 of the additive group $\mathbb{F}_p$, that is injective maps $\psi_0, \psi_1 : \mathbb{F}_p \to \{0,1\}^m$, where $m > 3\log(p)$. For $i = 0, 1$ we write $\mathbb{G}_i = \{\psi_i(x) : x \in \mathbb{F}_p\}$. We are given oracles to compute the induced group action on $\mathbb{G}_0, \mathbb{G}_1$ and an oracle to compute a non-degenerate bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. We are also given a random oracle to represent the hash function H. We refer to $\mathbb{G}_0$ as a generic bilinear group.

The following theorem gives a lower bound on the advantage of a generic adversary in breaking our CP-ABE scheme.

Theorem 1 *Let $\psi_0, \psi_1, \mathbb{G}_0, \mathbb{G}_1$ be defined as above. For any adversary A, let q be a bound on the total number of group elements it receives from queries it makes to the oracles for the hash function, groups $\mathbb{G}_0$ and $\mathbb{G}_1$, and the bilinear map e, and from its interaction with the CP-ABE security game. Then we have that the advantage of the adversary in the CP-ABE security game is $O(q^2/p)$.*

Proof. We first make the following standard observation, which follows from a straightforward hybrid argument: In the CP-ABE security game, the challenge ciphertext has a component $\tilde{C}$ which is randomly either $M_0 e(g,g)^{\alpha s}$ or $M_1 e(g,g)^{\alpha s}$. We can instead consider a modified game in which $\tilde{C}$ is either $e(g,g)^{\alpha s}$ or $e(g,g)^{\theta}$, where θ is selected uniformly at random from $\mathbb{F}_p$, and

the adversary must decide which is the case. It is clear that any adversary that has advantage ϵ in the CP-ABE game can be transformed into an adversary that has advantage at least $\epsilon/2$ in the modified CP-ABE game. (To see this consider two hybrids: one in which the adversary must distinguish between $M_0 e(g,g)^{\alpha s}$ and $e(g,g)^{\theta}$; another in which it must distinguish between $e(g,g)^{\theta}$ and $M_1 e(g,g)^{\alpha s}$. Clearly both of these are equivalent to the modified game above.) From now on, we will bound the adversary's advantage in the modified game.

We now introduce some notation for the simulation of the modified CP-ABE game. Let $g = \psi_0(1)$ (we will write g^x to denote $\psi_0(x)$, and $e(g,g)^y$ to denote $\psi_1(y)$ in the future).

At setup time, the simulation chooses α, β at random from $\mathbb{F}_p$ (which we associate with the integers from 0 to $p-1$). Note that if $\beta = 0$, an event that happens with probability $1/p$, then setup is aborted, just as it would be in the actual scheme. The public parameters $h = g^{\beta}$, $f = g^{1/\beta}$, and $e(g,g)^{\alpha}$ are sent to the adversary.

When the adversary (or simulation) calls for the evaluation of H on any string i, a new random value t_i is chosen from $\mathbb{F}_p$ (unless it has already been chosen), and the simulation provides g^{t_i} as the response to $H(i)$.

When the adversary makes its j'th key generation query for the set S_j of attributes, a new random value $r^{(j)}$ is chosen from $\mathbb{F}_p$, and for every $i \in S_j$, new random values $r_i^{(j)}$ are chosen from $\mathbb{F}_p$. The simulator then computes: $D = g^{(\alpha + r^{(j)})/\beta}$ and for each $i \in S_j$, we have $D_i = g^{r^{(j)} + t_i r_i^{(j)}}$ and $D_i' = g^{r_i^{(j)}}$. These values are passed onto the adversary.

When the adversary asks for a challenge, giving two messages $M_0, M_1 \in \mathbb{G}_1$, and the access tree $\mathbb{A}$, the simulator does the following. First, it chooses a random s from $\mathbb{F}_p$. Then it uses the linear secret sharing scheme associated with $\mathbb{A}$ (as described in Section 4) to construct shares λ_i of s for all relevant attributes i. We stress again that the λ_i are all chosen uniformly and independently at random from $\mathbb{F}_p$ subject to the linear conditions imposed on them by the secret sharing scheme. In particular, the choice of the λ_i's can be perfectly simulated by choosing ℓ random values $\mu_1, \ldots \mu_\ell$ uniformly and independently from $\mathbb{F}_p$, for some value of ℓ, and then letting the λ_i be fixed public linear combinations of the μ_k's and s. We will often think of the λ_i as written as such linear combinations of these independent random variables later.

Finally, the simulation chooses a random $\theta \in \mathbb{F}_p$, and constructs the encryption as follows: $\tilde{C} = e(g,g)^{\theta}$ and $C = h^s$. For each relevant attribute i, we have $C_i = g^{\lambda_i}$, and $C_i' = g^{t_i \lambda_i}$. These values are sent to the adversary.

(Note, of course, that if the adversary asks for a decryption key for a set of attributes that pass the challenge access structure, then the simulation does not issue the key; similarly if the adversary asks for a challenge access structure such that one of the keys already issued pass the access structure, then the simulation aborts and outputs a random guess on behalf of the adversary, just as it would in the real game.)

We will show that with probability $1 - O(q^2/p)$, taken over the randomness of the the choice of variable values in the simulation, the adversary's view in this simulation is identically distributed to what its view would have been if it had been given $\tilde{C} = e(g,g)^{\alpha s}$. We will therefore conclude that the advantage of the adversary is at most $O(q^2/p)$, as claimed.

When the adversary makes a query to the group oracles, we may condition on the event that (1) the adversary only provides as input values it received from the simulation, or intermediate values it already obtained from the oracles, and (2) there are p distinct values in the ranges of both ϕ_0 and ϕ_1. (This event happens with overwhelming probability $1 - O(1/p)$.) As such, we may keep track of the algebraic expressions being called for from the oracles, as long as no "unexpected collisions" happen. More precisely, we think of an oracle query as being a rational function $\nu = \eta/\xi$ in the variables $\theta, \alpha, \beta, t_i$'s, $r^{(j)}$'s, $r_i^{(j)}$'s, s, and μ_k's. An unexpected collision would be when two queries corresponding to two distinct formal rational functions $\eta/\xi \neq \eta'/\xi'$ but where due to the random choices of these variables' values, we have that the values of η/ξ and η'/ξ' coincide.

We now condition on the event that no such unexpected collisions occur in either group $\mathbb{G}_0$ or $\mathbb{G}_1$. For any pair of queries (within a group) corresponding to distinct rational functions η/ξ and η'/ξ', a collision occurs only if the non-zero polynomial $\eta\xi' - \xi\eta'$ evaluates to zero. Note that the total degree of $\eta\xi' - \xi\eta'$ is in our case at most 5. By the Schwartz-Zippel lemma [25, 31], the probability of this event is $O(1/p)$. By a union bound, the probability that any such collision happens is at most $O(q^2/p)$. Thus, we can condition on no such collision happening and still maintain $1 - O(q^2/p)$ of the probability mass.

Now we consider what the adversary's view would have been if we had set $\theta = \alpha s$. We will show that subject to the conditioning above, the adversary's view would have been identically distributed. Since we are in the generic group model where each group element's representation is uniformly and independently chosen, the only way that the adversary's view can differ in the

$t_i t_{i'}$	$\lambda_i t_{i'}$	$t_i t_{i'} \lambda_{i'}$	$t_i r^{(j)} + t_i t_{i'} r_{i'}^{(j)}$
$t_i r_{i'}^{(j)}$	t_i	$\alpha + r^{(j)}$	$\alpha s + s r^{(j)}$
$\lambda_i \lambda_{i'}$	$t_i \lambda_i \lambda_{i'}$	$\lambda_{i'} r^{(j)} + \lambda_{i'} t_i r_i^{(j)}$	$\lambda_i r_i^{(j)}$
λ_i	$t_i t_{i'} \lambda_i \lambda_{i'}$	$t_i \lambda_i r_i^{(j)} + t_i t_{i'} \lambda_i r_{i'}^{(j)}$	$t_i \lambda_i r_{i'}^{(j)}$
$t_i \lambda_i$	$(r^{(j)} + t_i r_i^{(j)})(r^{(j)} + t_{i'} r_{i'}^{(j')})$	$(r^{(j)} + t_i r_i^{(j)}) r_{i'}^{(j')}$	$r^{(j)} + t_i r_i^{(j)}$
$r_i^{(j)} r_{i'}^{(j')}$	$r_i^{(j)}$	s	

Table 1. Possible query types from the adversary.

case of $\theta = \alpha s$ is if there are two queries ν and ν' into $\mathbb{G}_1$ such that $\nu \neq \nu'$ but $\nu|_{\theta=\alpha s} = \nu'|_{\theta=\alpha s}$. We will show that this never happens. Suppose not.

Recall that since θ only occurs as $e(g,g)^\theta$, which lives in $\mathbb{G}_1$, the only dependence that ν or ν' can have on θ is by having some additive terms of the form $\gamma'\theta$, where γ' is a constant. Therefore, we must have that $\nu - \nu' = \gamma\alpha s - \gamma\theta$, for some constant $\gamma \neq 0$. We can then artificially add the query $\nu - \nu' + \gamma\theta = \gamma\alpha s$ to the adversary's queries. But we will now show that the adversary can *never* construct a query for $e(g,g)^{\gamma\alpha s}$ (subject to the conditioning we have already made), which will reach a contradiction and establish the theorem.

What is left now is to do a case analysis based on the information given to the adversary by the simulation. For sake of completeness and ease of reference for the reader, in Table 1 we enumerate over all rational function queries possible into $\mathbb{G}_1$ by means of the bilinear map and the group elements given the adversary in the simulation, *except those in which every monomial involves the variable* β, since β will not be relevant to constructing a query involving αs. Here the variables i and i' are possible attribute strings, and the variables j and j' are the indices of secret key queries made by the adversary. These are given in terms of λ_i's, not μ_k's. The reader may check the values given in Table 1 against the values given in the simulation above.

In the group $\mathbb{G}_1$, in addition to the polynomials in the table above, the adversary also has access to 1 and α. The adversary can query for arbitrary linear combinations of these, and we must show that none of these polynomials can be equal to a polynomial of the form $\gamma\alpha s$. Recall that $\gamma \neq 0$ is a constant.

As seen above, the only way that the adversary can create a term containing αs is by pairing $s\beta$ with $(\alpha + r^{(j)})/\beta$ to get the term $\alpha s + s r^{(j)}$. In this way, the adversary could create a query polynomial containing $\gamma\alpha s + \sum_{j \in T} \gamma_j s r^{(j)}$, for some set T and constants $\gamma, \gamma_j \neq 0$.

In order for the adversary to obtain a query polynomial of the form $\gamma\alpha s$, the adversary must add other linear combinations in order to cancel the terms of the form $\sum_{j \in T} \gamma_j s r^{(j)}$.

We observe (by referencing the table above) that the only other term that the adversary has access to that could involve monomials of the form $s r^{(j)}$ are obtained by pairing $r^{(j)} + t_i r_i^{(j)}$ with some $\lambda_{i'}$, since the $\lambda_{i'}$ terms are linear combinations of s and the μ_k's.

In this way, for sets T'_j and constants $\gamma_{(i,j,i')} \neq 0$, the adversary can construct a query polynomial of the form:

$$\gamma\alpha s + \sum_{j \in T} \left(\gamma_j s r^{(j)} + \sum_{(i,i') \in T'_j} \gamma_{(i,j,i')} \left(\lambda_{i'} r^{(j)} + \lambda_{i'} t_i r_i^{(j)} \right) \right) + \text{other terms}$$

Now, to conclude this proof, we do the following case analysis:

Case 1 There exists some $j \in T$ such that the set of secret shares $L_j = \{\lambda_{i'} : \exists i : (i,i') \in T'_j\}$ do not allow for the reconstruction of the secret s.

If this is true, then the term $s r^{(j)}$ will not be canceled, and so the adversary's query polynomial cannot be of the form $\gamma\alpha s$.

Case 2 For all $j \in T$ the set of secret shares $L_j = \{\lambda_{i'} : \exists i : (i,i') \in T'_j\}$ do allow for the reconstruction of the secret s.

Fix any $j \in T$. Consider S_j, the set of attributes belonging to the j'th adversary key request. By the assumption that no requested key should pass the challenge access structure, and the properties of the secret sharing scheme, we know that the set $L'_j = \{\lambda_i : i \in S_j\}$ cannot allow for the reconstruction of s.

Thus, there must exist at least one share $\lambda_{i'}$ in L_j such that $\lambda_{i'}$ is linearly independent of L'_j when written in terms of s and the μ_k's. By the case analysis, this means that in the adversary's query there is a term of the form $\lambda_{i'} t_i r_i^{(j)}$ for some $i \in S_j$.

However, (examining the table above), there is no term that the adversary has access to that can

cancel this term. Therefore, any adversary query polynomial of this form cannot be of the form $\gamma\alpha s$.
□