Collusion-Resistant Group Key Management Using Attribute-Based Encryption* **

Ling Cheung[1], Joseph A. Cooley[2], Roger Khazan[2], and Calvin Newport[1]

[1] MIT Computer Science and Artificial Intelligence Laboratory
{lcheung,cnewport}@theory.csail.mit.edu
[2] MIT Lincoln Laboratory

Abstract. This paper illustrates the use of ciphertext-policy attribute-based encryption (CP-ABE), a recently proposed primitive, in the setting of group key management. Specifically, we use the CP-ABE scheme of Bethencourt, Sahai and Waters to implement flat table group key management. Unlike past implementations of flat table, our proposal is resistant to collusion attacks. We also provide efficient mechanisms to refresh user secret keys (for perfect forward secrecy) and to delegate managerial duties to subgroup controllers (for scalability). Finally, we discuss performance issues and directions for future research.

1 Introduction

IP Multicast is an efficient way of disseminating information to large groups of users. However, it does not provide any access control mechanism: any host can join a multicast group by sending a request-to-join message to the nearby router. To enforce access control, a typical approach is to encrypt group data traffic with a symmetric cryptographic key, which is distributed selectively, to legitimate group members (GMs) only. In other words, the access control problem for group data is transformed into access control on the data encryption key (DEK).

A security concern in this setting is *perfect forward secrecy*: the compromise of a GM at a particular time does not lead to the compromise of past multicast data, provided the data are not stored locally by the compromised GM. This suggests DEKs should be refreshed periodically, with old versions erased, so that access to the current DEK does not imply access to past data.

In a multicast group with dynamic membership (i.e., GMs may join and leave throughout the lifetime of the group), two more security issues arise:

- *Group backward secrecy*: a new GM should not have access to data that were transmitted before he joined.

* This research was sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are not necessarily endorsed by the US Government.

** Cheung was also supported by NSF Award #CCR-0326277. Newport was supported by NSF Award #CCR-0121277 and USAF, AFRL Award #FA9550-04-1-0121.

– *Group forward secrecy*: once a GM leaves the group, he should not have access to future data.

Both perfect forward secrecy and group backward secrecy can be addressed efficiently, for instance, by applying a one-way transformation to the DEK at regular intervals. Since one-way functions cannot be inverted efficiently, a new GM (or an intruder with access to the current DEK) cannot recover old DEKs.

Group forward secrecy, however, is harder to achieve, because the leaving GM knows potentially all DEKS that have been used. Any new key efficiently derivable from old ones is therefore insecure. A typical solution is to have new DEKs generated independently and delivered securely to remaining GMs. Thus, the main challenge becomes the efficiency of selective key distribution. This problem is known in the literature as *group key management*, and has been studied extensively for over a decade. We refer to [1] for a comprehensive survey.

Most existing group key management schemes make use of auxiliary keys, often called key encryption keys (KEKs). Each GM is given a unique subset of KEKs, and the group controller (GC) encrypts the new DEK with a combination of KEKs that ensures only current GMs can decrypt. For example, a naive solution is to assign a unique KEK to each GM and to encrypt the new DEK individually to every remaining GM. More efficient solutions have been proposed (e.g., [2–4]), where the number of encryptions is logarithmic in the size of the group. This is achieved by allowing GMs to hold subsets of KEKs that intersect.

In this paper, we show that a similar concept can be implemented using the ciphertext-policy attribute-based encryption (CP-ABE) primitive proposed by Bethencourt, Sahai and Waters (BSW) [5]. The main idea is to associate each GM with a set of abstract attributes, instead of actual KEKs. An attribute can be any statement regarding the GM; for example, "The 12-th bit in my ID is 0," or "I belong to the subgroup identified by *gid*." Like KEKs, these attributes allow the GC to distinguish current GMs from former/leaving GMs. The GC computes an access structure that is satisfied by the attribute set of every current GM, but not by that of any former/leaving GM. This structure is then used to encrypt the new DEK in the CP-ABE system, which ensures the desired access control.

There are several advantages to using CP-ABE (in particular, the BSW implementation) for group key management. One of them is the decoupling of abstract attributes from actual keys. When a secret key SK is issued for a set S of attributes, the components of SK are computed based on S, but are also masked by a combination of randomization factors unique to SK. As a result, even if two GMs share certain attributes, their secret keys are independent, and hence secret keys need not be refreshed during a join/leave event. This differs from traditional approaches, where affected KEKs must be replaced. Moreover, GMs cannot share their secret keys to increase the number of attributes to which they are associated, as the randomization factors will not match[1]. This allows us to give a collusion-resistant implementation of the flat table scheme, which is vulnerable to collusion attacks when implemented with traditional KEKs [3, 6].

[1] Indeed, collusion attacks are built into the CP-ABE security game of [5], where the adversary may query for secret keys both before and after submitting the challenges.

We also use the Delegate algorithm of [5] to implement subgroup controllers (SGCs). This algorithm generates new secret keys by re-randomizing an existing secret key, without access to the master secret. Thus, the master secret resides only at the central GC, which further improves security. In addition, we describe an efficient technique to refresh an entire BSW encryption system, including the public parameters, master secret key and user secret keys. We use this to achieve perfect forward secrecy. Only two multicast messages are necessary for the refresh operation: a new DEK and a conversion factor, which is a single bilinear group element, are both sent encrypted with the current DEK. New secret keys can be derived locally using one multiplication.

Finally, the use of CP-ABE reduces storage overhead for the GC. Instead of storing all KEKs in the system, the GC stores only the public parameters and master secret of CP-ABE. For a fixed security parameter, these are of constant size, independent of the size of the multicast group.

Related Work The flat table key management scheme was proposed in [3, 6]. Despite its simplicity and good overall performance, flat table has been largely dismissed due to collusion threats. In contrast, tree-based rekey algorithms have gained popularity, including, notably, Logical Key Hierarchy (LKH) [7, 8], One-Way Function Tree (OFT) [2], One-way Function Chain Tree [9], Hierarchical a-ary Tree with Clustering [10] and Efficient Large-Group Key (ELK) [4]. These algorithms provide different tradeoffs among storage, computation and communication overheads. We give some basic comparisons in Sections 6 and B, and we refer to [1] for detailed comparisons based on various performance measures.

In [11], Boneh et al. propose a collusion-resistant broadcast encryption scheme in which both ciphertexts and secret keys are of constant size, but the public key vector grows linearly with N, the size of the user ID space. The authors also outline a secure mailing list application—a problem that is very similar to secure multicast. In that approach, users are represented individually (e.g., two bilinear group elements per user in the public key), therefore linear complexity is inherent in both public key size and in encryption and decryption time[2]. Even with the proposal of parallel instances, the complexities are still linear in the size of each instance (e.g., $\sqrt{N}$). In comparison, attribute-based encryption is more flexible, allowing efficient representations (e.g., $O(\log(N))$) of many large sets. We leave it as interesting future work to fully develop a group key management scheme using broadcast encryption, and to provide detailed comparisons.

Overview Sections 2 and 3 describe the basics of flat table key management and ciphertext-policy attribute-based encryption, respectively. Section 4 presents the specifics of our new group key management scheme, with a decentralized version outlined in Section 5. Section 6 discusses performance issues and presents some simulation results. Concluding remarks follow in Section 7.

[2] The time complexity of encryption and decryption can be reduced by caching previously computed bilinear group elements. However, if large sets of members join and leave the group frequently, the benefit of caching diminishes.

2 Flat Table Key Management

In this section, we review the basic setup of the flat table group key management scheme [3, 6]. Assume that all potential GMs have unique bit-string identifiers of length n. A typical ID is denoted $X_{n-1}X_{n-2}\ldots X_0$, where each X_i is either 0 or 1. The maximum size of the group is thus $N := 2^n$. In addition to a DEK, denoted K, the GC maintains $2n$ KEKs: $\{k_{i,b}|i \in \mathbb{Z}_n, b \in \mathbb{Z}_2\}$. Each GM holds $n+1$ keys: the DEK K and exactly half of the KEKs. More precisely, a GM with ID $X_{n-1}X_{n-2}\ldots X_0$ holds the KEKs $\{k_{i,X_i}|i \in \mathbb{Z}_n\}$.

Join Suppose the joining ID is $X_{n-1}X_{n-2}\ldots X_0$. For group backward secrecy, $n+1$ keys are refreshed: $\{k_{i,X_i}|i \in \mathbb{Z}_n\}$ and K. The GC multicasts the new keys, each encrypted with the corresponding old key: $\{k'_{n-1,X_{n-1}}\}_{k_{n-1,X_{n-1}}}, \ldots,$ $\{k'_{0,X_0}\}_{k_{0,X_0}}, \{K'\}_K$. (Here $\{M\}_k$ denotes the result of encrypting M with symmetric key k.) Finally, the joining GM is given $\{k'_{i,X_i}|i \in \mathbb{Z}_n\}$ and K' via secure unicast messages.

Leave Suppose the leaving ID is $X_{n-1}X_{n-2}\ldots X_0$. For group forward secrecy, all $n+1$ keys held by this GM are refreshed. The GC multicasts the new DEK, encrypted once with each of the n KEKs *not* held by the leaving GM: $\{K'\}_{k_{n-1,\bar{X}_{n-1}}}, \ldots, \{K'\}_{k_{0,\bar{X}_0}}$. (Here $\bar{X}_i$ denotes the complement of X_i.) Clearly, the leaving GM cannot decrypt any of these messages. On the other hand, since every other ID differs from the leaving ID by at least one bit, every remaining GM should be able to decrypt at least one of these messages, thus obtaining K'.

The GC then multicasts new KEKs, each encrypted with both the new DEK and the corresponding old KEK: $\{k'_{n-1,X_{n-1}}\}_{K',k_{n-1,X_{n-1}}}, \ldots, \{k'_{0,X_0}\}_{K',k_{0,X_0}}$. (Here $\{M\}_{k_1,k_2}$ denotes the result of double encryption: first encrypt with k_1; then the resulting ciphertext is encrypted with k_2.) Again, all remaining GMs can decrypt and update the appropriate KEKs, while the leaving GM cannot decrypt at all because he does not have K'.

Multiple Leaves In principle, the procedure above can be repeated to remove multiple members from the group. A more efficient strategy is proposed in [3], using Boolean function minimization (BFM). The main idea is to associate a Boolean function m with each rekey event. This function takes n bits as inputs and returns one bit, satisfying:

- $m(X_{n-1}, \ldots, X_0) = 0$, if the ID $X_{n-1}\ldots X_0$ is leaving;
- $m(X_{n-1}, \ldots, X_0) = 1$, if the ID $X_{n-1}\ldots X_0$ remains in the group.

For any other ID $X_{n-1}\ldots X_0$, $m(X_{n-1}, \ldots, X_0)$ may be either 0 or 1. The GC runs the Quine-McCluskey algorithm [3, 12] to reduce m to a *sum of products expression (SOPE)* with (i) the smallest possible number of **OR**s, and (ii) given the number of **OR**s, the smallest possible number of literals.

For example, suppose 011 and 101 are to be removed from a group of seven members, with all but 000 currently in the group. The membership function $m(X_2, X_1, X_0)$ can be reduced to the minimal expression $X_2X_1 + \bar{X}_2\bar{X}_1 + \bar{X}_0$. To

update the DEK, it is then sufficient to multicast three messages: $\{K'\}_{k_{2,1},k_{1,1}}$, $\{K'\}_{k_{2,0},k_{1,0}}$ and $\{K'\}_{k_{0,0}}$. In general, the number of summands in the minimal expression corresponds to the number of messages, while the literals in each summand indicate which keys should be used to encrypt the new DEK. We refer to [3] for further details.

Collusion Attacks Although the flat table scheme is simple and efficient, it is susceptible to collusion attacks. In the example above, if 011 and 101 collude, then they can decrypt two of the three rekey messages: $\{K'\}_{k_{2,1},k_{1,1}}$ and $\{K'\}_{k_{2,0},k_{1,0}}$. In fact, if a set of leaving GMs collectively hold all $2n$ KEKs (e.g., two GMs with complementary bits in their IDs), then they may collude to obtain the new DEK, no matter how it is encrypted with KEKs.

The new scheme we present Section 4 avoids collusion attacks by replacing actual KEKs with abstract attributes: a GM with ID $X_{n-1} \dots X_0$ is said to possess the attributes $\{A_{i,X_i} | i \in \mathbb{Z}_n\}$. Upon joining, this GM receives a secret key SK_S associated with the attribute set $S := \{A_{i,X_i} | i \in \mathbb{Z}_n\}$ in a ciphertext-policy attribute-based encryption system. These secrete keys are generated in such a way that two keys SK_{S_1} and SK_{S_2} *cannot* be combined to obtain $SK_{S'}$, where $S' \not\subseteq S1$ and $S' \not\subseteq S2$. For example, even if 011 and 101 collude, they cannot obtain a secret key for the attribute set $\{A_{2,1}, A_{1,1}\}$. Essentially, each secret key has its own randomization factors, making it impossible to combine two keys into one coherent new key.

3 Ciphertext-Policy Attribute-Based Encryption

We now describe a simplified version of the BSW scheme [5], sufficient for our purposes. In CP-ABE, each user is associated with a set of attributes (expressed as bit strings) and he receives a secret key based on that set. For each encryption, the encryptor must specify a threshold access structure over attributes and send it as part of the ciphertext. A user can decrypt if and only if his attribute set satisfies this access structure.

In [5], an access structure is a tree in which every leaf y is labeled by an attribute $\mathsf{att}(y)$ and every non-leaf node represents a threshold gate[3]. For our purposes, it is sufficient to consider only depth-1 trees; in particular, **AND** gates (n-of-n) and **OR** gates (1-of-n). Moreover, each node is assumed to have a total ordering of its children: if y is not the root, $\mathsf{index}(y)$ denotes the index of y as a child of its parent.

An CP-ABE scheme consists of four fundamental algorithms: Setup, Encrypt, KeyGen and Decrypt. An optional algorithm, Delegate, is also provided in [5].

Setup This algorithm takes as input the security parameter κ and returns a public key PK and a master secret key MK. It selects:

- a bilinear group G of prime order p, with bilinear map $e : G \times G \to G_1$,
- a generator g of G, two random elements α and β in $\mathbb{Z}_p$, and

[3] A t-of-n threshold gate is satisfied if and only if at least t of the n inputs are satisfied.

– a hash function H that maps bit-string attributes into G.

The public key is $PK := \langle G, g, g^\beta, g^{\frac{1}{\beta}}, e(g,g)^\alpha, H\rangle$. The master secret key is $MK := \langle \beta, g^\alpha\rangle$.

Encrypt This algorithm takes as input the public key PK, a message $M \in G_1$, and an access tree T, and returns a ciphertext CT. For simplicity, we only describe the encryption procedure for depth-1 trees. This is sufficient for our group key management scheme.

Let t_T denote the threshold value for T and let $d_T := t_T - 1$. The algorithm selects a degree-d_T polynomial q_T as follows: (i) choose random $s \in \mathbb{Z}_p$ and set $q_T(0) := s$; (ii) choose d_T other points independently at random. Let Y denote the set of leaf nodes in T and, for every $y \in Y$, set q_y to be the constant polynomial $q_T(\mathsf{index}(y))$. The ciphertext CT is:

$$\langle T,\ M \cdot e(g,g)^{\alpha s},\ g^{\beta s},\ \{g^{q_y(0)} | y \in Y\},\ \{H(\mathsf{att}(y))^{q_y(0)} | y \in Y\}\rangle.$$

KeyGen This algorithm takes as input a set S of attributes and returns a secret key SK associated with S. First, it selects from $\mathbb{Z}_p$ a random r and random r_j for each $j \in S$. (These are the aforementioned *randomization factors*.) The secret key SK is: $\langle g^{\frac{\alpha+r}{\beta}},\ \{g^r \cdot H(j)^{r_j} | j \in S\},\ \{g^{r_j} | j \in S\}\rangle$.

Decrypt This algorithm takes as input a ciphertext CT and a secret key SK for an attribute set S, and returns the message M if S satisfies the access tree T in CT. Due to space considerations, the full description is given in Appendix A.

Delegate This algorithm takes as input a secret key SK for a set S of attributes, and a set $\tilde{S} \subseteq S$, and returns a secret key $\tilde{SK}$ for $\tilde{S}$. First, it selects from $\mathbb{Z}_p$ a random $\tilde{r}$ and random $\tilde{r}_j$ for each $j \in \tilde{S}$. (These are the *re-randomization factors*.) Suppose SK is of the form $\langle D, \{D_j^0 | j \in S\}, \{D_j^1 | j \in S\}\rangle$. The new secret key $\tilde{SK}$ for $\tilde{S}$ is: $\langle D \cdot g^{\frac{\tilde{r}}{\beta}},\ \{D_j^0 \cdot g^{\tilde{r}} \cdot H(j)^{\tilde{r}_j} | j \in \tilde{S}\},\ \{D_j^1 \cdot g^{\tilde{r}_j} | j \in \tilde{S}\}\rangle$.

Security Game CP-ABE security is defined as the statement that all probabilistic polynomial-time adversaries have at most negligible advantage in the following game. The full security proof can be found in [5].

Setup The challenger runs the Setup algorithm and gives the adversary PK.

Phase 1 The adversary chooses sets $S_1, \ldots, S_q$ of attributes and makes KeyGen queries on these sets.

Challenge The adversary submits two messages M_0 and M_1 of equal length, as well as an access structure T such that T is *not* satisfied by any of the sets $S_1, \ldots, S_q$. The challenger chooses $b \in \{0,1\}$ at random and encrypts M_b with T. The resulting ciphertext CT is given to the adversary.

Phase 2 The adversary chooses sets $S_{q+1}, \ldots, S_{q'}$ of attributes and makes KeyGen queries on these sets, provided again none of these sets satisfy T.

Guess The adversary outputs a guess b' of b.

Notice, collusion resistance follows from the fact the adversary may make multiple secret key queries both before and after selecting challenge plaintexts.

4 New Scheme for Group Key Management

This section presents our new proposal for group key management. The underlying concept is flat table, but we rely on CP-ABE to carry out rekey operations. We achieve collusion resistance and some performance improvements, such as constant-size messages for join and periodic refresh.

We assume that each potential GM is uniquely identified by a bit string of length n: $X_{n-1}X_{n-2}\ldots X_0$. Depending on the application, this ID may be a hash value of the GM's public key, or it may be assigned by the GC when the GM joins the group for the first time[4]. In either case, it is desirable to have actual IDs sparsely distributed in the space of all length-n bit strings. This helps to reduce the average size of BFM outputs and hence the message complexity for rekeying.

The GC plays the role of the central authority in CP-ABE. To initialize the group, he runs Setup and obtains $PK = \langle G, g, g^\beta, g^{\frac{1}{\beta}}, e(g,g)^\alpha, H\rangle$ and $MK = \langle \beta, g^\alpha\rangle$. Then he selects random $K \in G_1$ as the DEK[5]. The GC does *not* hold any other keys; in particular, *no* KEKs.

The attributes in our system are of the following form: "The i-th bit in my ID is b," where $i \in \mathbb{Z}_n$ and $b \in \{0,1\}$. This is denoted $A_{i,b}$. Thus, a member with ID $X_{n-1}X_{n-2}\ldots X_0$ possesses the attributes $\{A_{i,X_i} | i \in \mathbb{Z}_n\}$.

Join The joining GM first establishes a secure unicast connection with the GC, who checks whether the member is authorized to join. If the checks succeed, the joining ID is added to a set $\mathcal{C}$ of current IDs. The GC then selects a new DEK $K' \in G_1$ at random and multicasts $\{K'\}_K$. Current GMs decrypt and update the DEK. The GC then runs KeyGen with the attribute set $S := \{A_{i,X_i} | i \in \mathbb{Z}_n\}$. The joining GM receives, by secure unicast, the resulting secret key SK and the new DEK K'.

Leave Our scheme treats a single leaving GM and multiple leaving GMs in exactly the same way. Once the set of leaving GMs is determined, the GC sets $\mathcal{C}'$ to be the set of remaining IDs. Let m denote the Boolean function such that an ID evaluates to 1 if and only if it is in $\mathcal{C}'$. The GC runs the Quine-McCluskey algorithm to reduce m to a minimal SOPE $E = E_0 + \ldots + E_L$. If the GC maintains a list of all former members, then m can be relaxed to allow "don't care" values on unused IDs (i.e., those for which a secret key was never issued.) This reduces the size of E, because the set of IDs to be covered is smaller. Indeed, we maintain such a list in our experimental implementation (see Section 6).

For each $l \in \{0,\ldots,L\}$, let T_l denote any access tree satisfying: the root is an **AND** gate and the leaves are labeled by exactly those literals occurring in E_l. For example, if E_l is $\bar{X}_2\bar{X}_1X_0$, then T_l has exactly three leaf nodes and they are labeled with $A_{2,0}$, $A_{1,0}$ and $A_{0,1}$, respectively.

[4] Assigned IDs may be a better option, because the GC can use the Huffman technique for ID generation, which improves BFM calculations [13, 14].

[5] Alternatively, we may use K as a seed for generating the DEK. The generator function must be known to all GMs.

The GC selects at random a new DEK $K' \in G_1$. For each l, he runs Encrypt on K' and T_l, obtaining CT_l. Then he multicasts $CT_0, \ldots, CT_L$. By construction, every GM with ID in $\mathcal{C}'$ satisfies at least one E_l, therefore it can decrypt at least one of these messages. Similarly, a GM with ID *not* in $\mathcal{C}'$ does not satisfy any E_l, therefore he cannot decrypt any of these messages on his own. Since the BSW scheme is collusion resistant, this implies former and leaving GMs cannot recover K' even if they all collude (e.g., by sharing their secret keys).

Periodic Refresh For perfect forward secrecy, we perform a system-wide refresh at regular intervals. The GC chooses random $K' \in G_1$ and $\alpha' \in \mathbb{Z}_p$. The public and master keys are updated as: $PK' := \langle G, g, g^\beta, g^{\frac{1}{\beta}}, e(g,g)^{\alpha'}, H \rangle$ and $MK' := \langle \beta, g^{\alpha'} \rangle$. Then the GC multicasts two messages: $\{K'\}_K$ and $\{g^{\frac{\alpha'-\alpha}{\beta}}\}_K$.

Each current GM can decrypt both messages because he knows K. The DEK is updated to K'. The second message, $g^{\frac{\alpha'-\alpha}{\beta}}$, is called the *conversion factor* and is used to update the secret key. Let $SK = \langle D, \{D_j^0 | j \in S\}, \{D_j^1 | j \in S\} \rangle$ denote the old secret key held by a GM. The new secret key is calculated as $SK' := \langle D \cdot g^{\frac{\alpha'-\alpha}{\beta}}, \{D_j^0 | j \in S\}, \{D_j^1 | j \in S\} \rangle$. The old key SK is securely erased.

Note that only the first component of SK is changed, and $D \cdot g^{\frac{\alpha'-\alpha}{\beta}} = g^{\frac{\alpha+r}{\beta} + \frac{\alpha'-\alpha}{\beta}} = g^{\frac{\alpha'+r}{\beta}}$, therefore SK' is a valid secret key in the new system for the same attribute set. Moreover, for $\alpha_0 \neq \alpha'$, SK' can be rewritten as $\langle g^{\frac{\alpha_0 + (r+\alpha'-\alpha_0)}{\beta}}, \{g^r \cdot H(j)^{r_j}\}_{j \in S}, \{g^{r_j}\}_{j \in S} \rangle$. This is not a valid secret key in the system parameterized by α_0. Thus, SK' cannot be used to decrypt past rekey messages.

Security Properties In the refresh algorithm above, K' and α' are generated randomly, therefore they do not reveal information about older versions of K and α. Moreover, we have argued that current secret keys are not valid in older systems with different α parameters. Therefore, perfect forward secrecy is satisfied.

Note that, during every rekey operation, the new DEK K' is
- either encrypted with the old DEK K, using symmetric encryption, or
- encrypted to the set $\mathcal{C}'$ of remaining IDs, using CP-ABE.

Assuming the security of symmetric encryption and CP-ABE, a GM joining for the first time cannot decrypt either type of messages from the past, hence group backward secrecy is satisfied.

For collusion resistance and group forward secrecy, we have seen that a leaving GM cannot recover the new DEK K', even if he colludes with other former/leaving GMs (cf. the leave algorithm). Assuming the security of symmetric encryption and CP-ABE, the same holds until this GM rejoins.

5 Decentralized Management

In the scheme we just described, a single entity (namely, the GC) manages the whole group. This imposes many requirements on the GC, including storage

(e.g., maintaining a list of active group members), computation (e.g., signature verification, key generation and BFM) and communication (e.g., unicast messages during join and leave). In this section, we show that some of this workload can be distributed to trusted members who act as *subgroup controllers (SGCs)*.

We assume the number of SGCs is relatively small, therefore secure unicast communication can be used from each SGC to the GC. Multicast is used for the other direction. SGCs may be added or removed dynamically. They maintain precise membership of their own subgroups, and accept join/leave requests on behalf of the GC. Thus, the GC behaves as a simple key server, generating DEKs and α parameters whenever they are requested or for periodic rekeying. SGCs use the Delegate algorithm to issue secret keys *without* knowing the system master key MK. This allows the group to recover quickly from the compromise of an SGC, using the Remove SGC algorithm below.

We also stress the fact that BFM computations are now performed by SGCs, each within his own subgroup. Moreover, the leave operation applies only to a single subgroup, therefore we add a global revoke operation.

Below we define the various operations in the new decentralized context. First, we augment the encryption system with new attributes $\bigcup_{gid \in \mathcal{V}} \{B_{gid}, C_{gid}\}$, where $\mathcal{V}$ is some appropriate name space. Given a subgroup identified by gid, every member of the subgroup possesses the attribute B_{gid}, while only the SGC possesses the attribute C_{gid}. The GC can now encrypt messages to the SGCs only, and an SGC can encrypt messages to members of his subgroup only.

Add SGC Suppose a trusted member is authorized and willing to become an SGC. The GC assigns a subgroup ID gid and gives the new SGC a secret key generated with the attribute set $\{A_{i,b} | i \in \mathbb{Z}_n, b \in \{0,1\}\} \cup \{B_{gid}, C_{gid}\}$. This subgroup ID is added to the list of active subgroups. The SGC also receives the appropriate policy information on who may join his subgroup[6] and how IDs are obtained (e.g., derived from member public keys, or assigned by the SGC from some ID space).

Join The joining GM contacts a SGC, who verifies the requests and sends a unicast message to the GC to signal a join. The GC multicasts $\{K'\}_K$ to the whole group. The SGC then runs the Delegate algorithm with its own secret key and the attribute set $\{A_{i,X_i} | i \in \mathbb{Z}_n\} \cup \{B_{gid}\}$, where $X_{n-1} \ldots X_0$ is the joining ID. The resulting secret key and the new DEK K' are given to the joining GM by secure unicast.

Leave The leaving GM contacts the SGC from whom he received his secret key[7]. The SGC verifies the request and sends a unicast message to the GC to signal a leave. The GC multicasts K' to SGCs *only*, by encrypting the new key using CP-ABE and the C attributes. Each SGC performs a BFM computation

[6] This determines whether a member could join multiple subgroups.

[7] If the leaving GM received multiple secret keys from multiple SGCs, he must contact all of them.

within his own subgroup[8] and multicasts rekey messages with the additional attribute B_{gid}, so that his rekey messages cannot be decrypted by members of other subgroups.

Global Revoke When the GC makes a revocation decision (e.g., because it detects malicious behavior from a particular GM), it multicasts the decision and a new DEK K' to the SGCs *only*, using CP-ABE and the C attributes. Each SGC proceeds with BFM and multicasts rekey messages, as in the leave algorithm.

Periodic Refresh This is done by the GC as in Section 4.

Remove SGC Suppose the GC decides to remove an SGC with subgroup ID gid_0, or the SGC decides to leave the group. The GC multicasts the new DEK K', encrypted using CP-ABE with the attribute set $\{C_{gid} | gid$ active and $gid \neq gid_0\}$. Each remaining SGC proceeds with BFM and multicasts rekey messages, as in the leave algorithm. Note that members of subgroup gid_0 are now effectively removed from the group, because they cannot decrypt rekey messages from other SGCs. These members must now rejoin by contacting another SGC.

6 Performance

In this section we compare our rekey scheme against the original flat table (FT) scheme [3, 6], with and without the BFM optimization. We also compare against tree-based schemes (e.g., OFT [2], LKH [7, 8] and ELK [4]). The comparisons are in terms of storage and communication.

Recall that n is the length of IDs and $N = 2^n$ is the size of ID space. Let M denote the number of current GMs, m denote $\log(M)$, and L denote the number of leaving GMs.

Scheme	Storage		Communication		
	GC	GM	Join	Single Leave	Multiple Leaves
FT (CP-ABE)	$\Theta(M)$	$\Theta(n)$	$\Theta(1)$	$O(n)$	$\approx O(m)$
FT (KEKs)	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(L \cdot n)$
FT/BFM (KEKs)	$\Theta(M+n)$	$\Theta(n)$	$\Theta(n)$	$O(n)$	$\approx O(m)$
Tree-Based	$\Theta(M)$	$\Theta(m)$	$\Theta(1)$	$\Theta(m)$	$O(L \cdot m)$

Fig. 1. Comparison of Storage and Communication Complexities.

For FT(CP-ABE) and FT/BFM (KEKs), the message complexity for leave varies, depending on which IDs are leaving and which IDs remain the group. For single leave, the worst case requires all complementary bits of the leaving

[8] If the leave event is entirely outside his subgroup, the SGC may skip the BFM computation and multicast using the last BFM output.

ID, hence $O(n)$. For multiple leaves, our experimental results suggest the complexity is $O(m)$, although the number of bytes communicated is large in our case due to the size of bilinear group elements (cf. Section 6.1 below). Barring the difference between m and n, our scheme is asymptotically comparable with tree-based schemes, and superior to original flat table. Further explanations of these complexity results are provided in Appendix B.

6.1 Experimental Evaluation

We simulate our group key management scheme using pre-existing software for BSW encryption [5] and for the Quine-McCluskey BFM algorithm [15]. Since the message complexity is constant for both join and refresh, we focus on the message complexity for simultaneous leaves. This case is difficult to analyze due to its dependence on BFM.

Our implementation is driven by an *event file*, which describes the size of the ID space, the IDs of current GMs, and the IDs of leaving GMs. Given an event file, our software generates the corresponding multicast traffic. We test four different group sizes: 16, 32, 64, and 128. For each group size, we test four different scenarios: 10%, 25%, 50%, and 75% members leaving. For each of these sixteen combinations, we run 10 independent trials. The group membership and the leaving members are chosen randomly in each trial, and the size of ID space for all trials is 8-bits[9].

Figure 2 describes the results of these experiments. Each entry describes the average number of messages and the average number of bytes generated from the given combination of group size and percentage of members leaving. We observe, empirically, that a CP-ABE message encrypted on a single attribute requires roughly 630 bytes. Each additional attribute adds roughly 250 to 300 bytes.

Extrapolating from these figures, the message complexity for multiple leaves is $O(\log M)$, where M is the current group size. The constant factor is roughly 3.5. It is also interesting to note that traffic peaks at 50% leaving. This conforms with the reflective nature of BFM; intuitively, 75% leaving and 25% leaving present roughly the same level of difficulty.

On the other hand, the byte overhead of our scheme is large. For these small group sizes, even a naive $O(M)$ scheme has competitive performance. For example, one can encrypt the new DEK to each remaining GM individually, using 32-byte symmetric keys. However, if our logarithmic behavior indeed extrapolates, such advantages would dissipate as the group size grows. These initial results indicate that an important next step is to develop new CP-ABE schemes with short ciphertexts.

[9] Clearly, a larger ID space is needed for large-scale deployment. Our naive implementation of Quine-McCluskey makes larger ID spaces computationally impractical. We believe larger ID spaces can be handled using optimized BFM algorithms, such as those commonly used in the digital logic community.

Group Size	Percentage of Members Leaving			
	10%	25%	50%	75%
16	4 / 2286	4 / 3058	4 / 4405	3 / 3000
32	6 / 5510	6 / 6604	6 / 8105	4 / 5616
64	9 / 9662	10 / 13771	12 / 18128	8 / 13319
128	14 / 18588	20 / 30705	24 / 42711	18 / 33053

Fig. 2. Average Number of Msgs/Bytes Per Leave Event in 8-Bit ID Space.

7 Conclusions and Future Work

In this paper, we propose a group key management scheme based on ciphertext-policy attribute-based encryption (CP-ABE). In this approach, each group member is identified by a set of attributes, and is given a secret key in the CP-ABE system that corresponds to his attribute set. During a leave event, the group controller uses CP-ABE to encrypt the new data key to all (and only) remaining members. We also provide efficient mechanisms for periodic system-wide refresh and for subgroup management.

This approach is conceptually simple and has many good features. For example, user secret keys are independent, even if certain attributes are shared. Thus the compromise of one secret key does not affect other secret keys. Moreover, users cannot collude to decrypt messages that they cannot decrypt individually.

The main challenge is to define attributes in such a way that the set of legitimate members can be distinguished efficiently, using a small combination of attributes. Following the flat table concept, we define attributes based on bits in the IDs of group members, and Boolean function minimization is used to determine which attributes should be included in the encryption of rekey messages. Our experimental results suggest, while BFM itself is computationally intensive, it provides good reduction in the number and size of rekey messages. An interesting direction of future research is to find other definitions of attributes for which rekey messages can be calculated more efficiently. For example, we may return to a tree-based setting, where attributes correspond to nodes in a tree.

Our experimental results show that BFM is computationally impractical in its naive implementation. (The problem is NP-hard.) However, efficient approximations are well known and widely used within the digital logic community. It would be interesting to simulate our rekey algorithm with an optimized BFM algorithm and to determine the maximum ID length for which BFM computations are feasible. Also, the difficulty of BFM is mitigated to some extent in our decentralized scheme, where SGCs perform BFM calculations locally.

Our experiments also show that BSW encryption produces very large ciphertexts. This is because a ciphertext contains roughly two bilinear group elements per attribute, and these elements are large (128 bytes each). We are currently investigating a modification to the BSW scheme, which reduces the ciphertext size at the cost of increased public key size. Large public keys are easier to handle in

the setting of group key management, because they can be obtained out-of-band, for example, pre-placed in hardware.

Acknowledgments We thank Ran Canetti, Ron Rivest and Eran Tromer for helpful discussions and suggestions.

References

1. Rafaeli, S., Hutchison, D.: A survey of key management for secure group communication. ACM Computing Surveys **35**(3) (2003) 309–329
2. McGrew, D., Sherman, A.: Key establishment in large dynamic groups using one-way function trees. Technical Report 0755, TIS Labs at Network Associates, Inc. (1998)
3. Chang, I., Engel, R., Kandlur, D., Pendarakis, D., Saha, D.: Key management for secure internet multicast using Boolean function minimization techniques. In: Proceedings IEEE Infocomm'99. (1999)
4. Perrig, A., Song, D., Tygar, J.: ELK: a new protocol for efficient large-group key distribution. In: Proceedings of the IEEE Symposium on Security and Privacy (Oakland). (2001)
5. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings of the 28th IEEE Symposium on Security and Privacy (Oakland). (2007) To appear.
6. Waldvogel, M., Caronni, G., Sun, D., Weiler, N., Plattner, B.: The VersaKey framework: versatile group keey management. IEEE Journal on Selected Areas in Communications **17**(9) (1999) 1614–1631 Special Issue on Middleware.
7. Wong, C., Gouda, M., Lam, S.: Secure group communications using key graphs. In: Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. (1998)
8. Wallner, D., Harder, E., Agee, R.: Key management for multicast: issues and architectures. (RFC 2627)
9. Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast security: a taxonomy and some efficient constructions. In: Proceedings of the IEEE INFOCOM'99. (1999)
10. Canetti, R., Malkin, T., Nissim, K.: Efficient communication-storage tradeoffs for multicast encryption. Lecture Notes in Computer Science **1592** (1999) Advances in Crytology – EUROCRYPT'99.
11. Boneh, D., Gentry, C., Waters, B.: Collusion resistant broadcast encryption with short ciphertexts and private keys. Lecture Notes in Computer Science **3621** (2005) Advances in Crytology – CRYPTO'05.
12. C.H. Roth, J.: Fundamentals of Logical Design. 5 edn. Thomson Engineering (2003)
13. Ilango, S., Thomas, J.: Group key management utilizing Huffman and Petrick based approaches. In: Proc. of the Int. Conf. on Information Technology: Coding and Computing (ITCC'04). (2004)
14. Srinivasan, T., Sathish, S., Kumar, R.V., Vijayender, M.: A hybrid scalable group key management approach for large dynamic multicast networks. In: Proc. of the Sixth IEEE Int. Conf. on Computer and Information Technology (CIT'06). (2006)
15. : Quine-mccluskey logic simplifier. (`http://sourceforge.net/projects/qmls/`)

A CP-ABE Decryption

The decryption algorithm takes as input a ciphertext CT and a secret key SK for an attribute set S, and returns the message M if S satisfies the access tree T in CT. Suppose SK is of the form $\langle D, \{D_j^0\}_{j\in S}, \{D_j^1\}_{j\in S}\rangle$ and CT is of the form $\langle T, \tilde{C}, C, \{C_y^0\}_{y\in Y}, \{C_y^1\}_{y\in Y}\rangle$, where Y is the set of leaf nodes in T.

Let $y \in Y$ be given and suppose $\mathsf{att}(y) \in S$. Using bilinearity of e, we can compute $e(g,g)^{r\cdot q_y(0)}$ as $\frac{e(D^0_{\mathsf{att}(y)}, C^0_y)}{e(D^1_{\mathsf{att}(y)}, C^1_y)}$. Indeed, the latter equals

$$\frac{e(g^r \cdot H(\mathsf{att}(y))^{r_{\mathsf{att}(y)}}, g^{q_y(0)})}{e(g^{r_{\mathsf{att}(y)}}, H(\mathsf{att}(y))^{q_y(0)})} = \frac{e(g^{r+m\cdot r_{\mathsf{att}(y)}}, g^{q_y(0)})}{e(g^{r_{\mathsf{att}(y)}}, g^{m\cdot q_y(0)})} = \frac{e(g,g)^{(r+m\cdot r_{\mathsf{att}(y)})\cdot q_y(0)}}{e(g,g)^{q_y(0)\cdot m\cdot r_{\mathsf{att}(y)}}},$$

where m is any exponent satisfying $g^m = H(\mathsf{att}(y))$.

By assumption, S satisfies T. Thus, there are at least t_T many leaf nodes y with $\mathsf{att}(y) \in S$. We select exactly t_T of such y's and compute $e(g,g)^{r\cdot q_y(0)}$ for each of them. Now we can compute $e(g,g)^{r\cdot q_T(0)}$ as $\prod_y e(g,g)^{r\cdot q_y(0)\cdot \Delta_y}$, where Δ_y is the appropriate Lagrange coefficient. (We refer to [5] for more details.)

Finally, the message M is computed as follows.

$$\frac{\tilde{C}\cdot e(g,g)^{r\cdot q_T(0)}}{e(C,D)} = \frac{M\cdot e(g,g)^{\alpha\cdot s}\cdot e(g,g)^{r\cdot q_T(0)}}{e(g^{\beta\cdot s}, g^{\frac{\alpha+r}{\beta}})} = \frac{M\cdot e(g,g)^{(\alpha+r)\cdot s}}{e(g,g)^{(\alpha+r)\cdot s}}$$

B Performance Comparisons

Storage (GC): In FT, the GC need not keep track of membership, although exact membership is needed to use the BFM optimization. Thus, the GC stores at least one DEK and $2n$ KEKs. In our scheme, the GC stores (i) the DEK, (ii) public and master keys of the CP-ABE system and (iii) the list of current GMs. Since the public and master keys for CP-ABE are of constant size, the GC's storage is dominated by M, the current group size. In a tree-based scheme, a GC stores all the KEKs described by the key tree. Typically, this amounts to $2M - 1$ KEKs.

Storage (GM) In FT, each GM stores one DEK and n KEKs. In our scheme, each GM stores its secret key in the CP-ABE system, which consists of $2n + 1$ group elements. (Note that bilinear group elements are quite large, 128 bytes each.) In a tree-based scheme, a GM stores the KEKs on its path to the root. Since the tree height is m, the GM stores m KEKs.

Communication (Join) In FT, the DEK and exactly n KEKs are refreshed during a join event. Hence $n + 1$ messages are necessary. Our scheme requires only one multicast message: the new DEK encrypted with the old DEK. In a tree-based scheme, the message complexity varies. If new KEKs are derived from old ones using one-way functions, only one message is necessary to signal the join event. In ELK, for example, this is further reduced to 0, because the joining members must wait until the next periodic refresh [4]. If new KEKs on the GM's path to the root are sent by the GC, then m messages are necessary, one for each affected key.

Communication (Single Leave) Without BFM, FT requires $2n$ messages, each containing one symmetric key. With BFM, the number of messages can be reduced, depending on the particular GM leaving and the specific composition of the remaining group. In our scheme, we generate one message for each summand in the BFM output. In the worst case, we would have n messages, each describing one attribute. Depending on the result of BFM, fewer messages may be sufficient.

In a tree-based scheme, at least $\Theta(m)$ messages are required: one for each affected KEK on the path from the leaving GM to the root. The actual complexity (e.g., the constant factors) varies depending on the particular scheme.

Communication (Multiple Leaves) Let L denote the number of leaving GMs. Without BFM, FT requires $2n$ messages per leaving GM, thus $2L \cdot n$ messages in total. Each message contains one symmetric key. With BFM, the number of messages can be reduced, depending on the particular GMs leaving and the specific composition of the remaining group. Similarly for our scheme. In our experiments (cf. Section 6), $O(m)$ messages were generated on average, regardless of the number of leaving members.

In a tree-based scheme, $L \cdot m$ messages are needed if members are removed in succession. Some optimization is possible by taking advantage of the relative positions of leaving GMs. For example, if the leaving GMs share a common ancestor deep in the tree, then the number of messages required is small. If instead they are well seperated, the number of messages approaches $L \cdot m$.