Dynamic Attributes Design in Attribute Based Encryption

Nanxi Chen, Mario Gerla
Department of Computer Science
University of California, Los Angeles
{chnx, gerla}@cs.ucla.edu

Abstract—In the past, sending encrypted messages to a group of users is not easy since it requires a shared group key in advance. Ciphertext-policy attribute based encryption (CPABE) achieves true that anyone who is expected to see the message content can decrypt messages in the absence of group keys. Say, a sender encrypts messages with an access control policy tree which is composed of attributes; a receiver is able to decrypt messages as long as his/her attributes satisfy the policy tree. CPABE saves the cost to distribute group keys; however, it does require users to keep their attributes up to date. In CPABE, updating attributes is not so efficient because if any attribute changes, the whole Private Key, which contains all the attributes, has to be updated. In this paper, by introducing fading function, we make attributes “dynamic” and allow attributes to be updated separately. We study how choosing different time unit of fading function affects the efficiency and security, and also compare our design with CPABE under certain circumstances. In the final, we present attribute delegation in non-trusted environment.

I. INTRODUCTION

As our society becomes more and more informationized, protecting private and confidential information gets important. There has been dozens of encryption systems developed to protect secret information and some of them were very popular in the past years.

However, it appears that people put a lot emphasis on protecting one-to-one communication while the security of group communication is usually overlooked. Using a shared group key is the most common way that people do secure group communication. But this method fails to deal with some situations as followed: A taxi driver working for a small company A wants to tell his/her colleagues that several guests are waiting for taxis at a particular segment of Westwood Blvd. To ensure the business secret, this message should be encrypted. This driver also hopes that only a limited number of drivers, say drivers who are currently at Westwood Blvd. or nearby blocks, are able to decrypt this message. To handle this situation, there should be a group key shared by company A’s drivers who are at Westwood Blvd. at that specific time. The difficulty is that members in this group change very fast and the group key should be changed and reissued whenever the group is reorganized. To be worse, there may be many such kind of temporary groups, e.g., a group for company A’s drivers who are at Pico Blvd..

Sahai et al. [2], [3]’s work on Attribute Based Encryption (ABE) makes it true to proceed group communication in the absence of a group key. In ABE, attributes describe certain properties, e.g., color, size, occupation and time etc.. A sender encrypts a message with an access control policy tree which is composed of attributes; a receiver can decrypt the message as long as his/her private key, which is issued by Key Master and contains a set of attributes, satisfies the policy tree. Later, based on ABE, Xiaoyan Hong et al. [4] developed Situation Aware Trust (SAT), giving a very good application of ABE. Locations and time are encoded into attributes in SAT. Therefore, for our example, this taxi driver can send a message encrypted with a policy tree, which may be described as company A AND Westwood Blvd. AND 10 – 11am. As a result, only company A’s taxi drivers who are at Westwood Blvd. during 10 – 11am can decrypt the message.

For fast changing group, ABE shows better adaptability than group key method. Users who share any set of attributes (non-empty set) form a group; those who do not have a specific attribute are automatically excluded from groups which are defined on that attribute. Thereby, ABE saves the trouble to issue a group key in advance for group communication. However, ABE does require users to keep their private keys (attributes) up-to-date. Attributes describe properties; users should update their private keys whenever their properties change. Say, if locations are attributes, when one moves from Westwood Blvd. to Pico Blvd., he/she should update his/her private key by replacing the old attribute which stands for Westwood Blvd. with a new attribute which stands for Pico Blvd.. Nevertheless, updating private keys/attributes is not simple for the reason that even there is only one attribute needed to be updated, the whole private key, which contains all the attributes, has to be updated. In case that a user’s private key has hundreds of attributes, updating causes a lot of overhead.

Allowing single attribute update, which means changing one attribute in the private key does not affect other attributes, may bring some benefits: Key Master generates less attributes by avoiding regeneration; less updates also mean less bandwidth cost and shorter contact time, which is quite useful to applications like SAT since contact time between vehicles or between road-side units and vehicles in VANET is expected to be very short. To achieve single attribute update, in this paper, we introduce fading function to each attribute, making attributes “independent” and “dynamic” (dynamic attributes). We present how to design and use fading function in ABE as well as how choosing different time unit values for fading function.
The rest of paper is organized as follows: Section II briefly talks about some background knowledge such as ABE and SAT, elaborates the problems with ABE and defines design goals. Section III introduces the design of dynamic attributes. Section IV talks about simulations. Section V presents the design of attribute delegation in non-trust environment. The last section draws a conclusion.

II. BACKGROUND

A. Attribute Based Encryption

Sahai and Waters et al. [2], [3] introduced Attribute Based Encryption (ABE) as a new mechanism for encrypted access control. There are several versions of ABE; the one that we will talk about is so-called Cipher-Policy Attribute Based Encryption (CPABE) [1].

CPABE utilizes identity based encryption [6], [7] and threshold secret sharing scheme [5]. To some extent, it is an extension of conventional asymmetric encryption system. In CPABE system, a user’s private key is associated with an arbitrary number of attributes. On the other hand, a party uses an access control policy tree, which is consisted of attributes, to encrypt a message. Only users whose attributes satisfy the policy tree are able to decrypt the message.

Fig. 1 gives an example of CPABE. The policy tree is composed of three different attributes while each private key has three attributes in it. In this example, Sarah can decrypt the message encrypted with this policy tree while Kevin cannot.

B. Situation Aware Trust

Xiaoyan Hong et al. [4] developed Situation Aware Trust (SAT) to provide adaptive and proactive security in various Vehicular Network (VNET) situations. SAT is a trust built on CPABE providing “data-centric trust” [8]. Attributes in SAT identify a group of entities (e.g., taxes associated with a company, police cars in a city), a type of events (e.g., accidents, congestions), or the property of events (location-based services, road traffic updates).

The above taxi driver example is a typical case in SAT. In this example, users who have attributes CompanyA, WashingtonSt. and 10 – 11am in their private keys are satisfied to decrypted the message. That means users that fulfill a set of descriptive attributes form a group. The group boundary is not clearly defined, whoever satisfy the policy tree can join in the group. This feature allows users in SAT set up trust proactively.

C. Scalability Concern

As pointed out in [1], key revocation is a problem with CPABE. To prevent a user keeping a private key for good, time is encoded into attributes. Time restriction attributes are put both in policy tree and the user’s private private key as showed in Fig. 1. There are two problems with time restriction attributes. First of all, the whole private key has a single time restriction. Secondly, there should be infinite number of time restriction attributes since time values are infinite even time is divided into slots.

We now investigate the first problem. As the dependency of all other attributes in the private key, the time restriction attribute makes it very difficult for private key to scale up in the number of attributes. On the one hand, there is only a time restriction for a private key but no separate time restriction for each attribute, so we do have private key revocation but we do not have single attribute revocation. That is the reason that updating the whole private key is allowed while separate attribute update is forbidden. On the other hand, single time restriction tends to increase the frequency of updating private key. Attributes represent properties; whenever a user’s properties change, the attributes that stand for those properties should be updated. Frequency of property change depends on properties. To achieve the best security, time restriction should be set to ensure that users’ properties do not change before time restriction expires. Although this rule may not be strictly obeyed in practice, it is commonly seen that small time restriction value is set to ensure the security.

Due to the above two reasons, the more attributes are there in the private key, the more costly per update will be and the more frequently private key updates.

The above problem really hurts because in CPABE, number of attributes in the private key is the main factor that affects the speed of key generation process. Cpabe toolkit, the only existing implementation of CPABE so far [1], runs in time

---

1Notice that although the terminology “attribute” appears in both sender’s and receiver’s side, it has different meanings. On the sender’s side, an attribute is public known information used to consist policy tree. On the receiver’s side, an attribute is secret information used to decrypt the message that encrypted with the corresponding attribute on the sender’s side. Because whenever “attribute” appears, we always mention it is on sender’s side or receiver’s side, or used to encrypt or decrypt, we use “attribute” to refer both in the following section.

2Here security means a user should not have the attribute that he/she does not have the corresponding property.
precedes precisely linear in the number of attributes associated with the key it is issuing. It takes around 1 second to generate a private key with 30 attributes. Key Master regenerates all the attributes in the private key when private key updates, resulting in a huge waste of Key Master’s CPU and sometimes making Key Master the bottleneck. Costly and frequent updates also wastes bandwidth, which is highly undesired in the application such as SAT.

Therefore, we hope to improve the key update mechanism in CPABE to meet the following demands:

- Attributes should automatically expire after a pre-defined time. Expiration of one attribute does not affect other attributes in the same private key. To renew an attribute, a user should request that attribute from Key Master again.
- Adding attributes into a private should be easy. That is, attribute add can be done without replacing the whole private key.
- Attributes can be delegated. But the same time, delegation must not allow user to “lend” or “borrow” attributes to or from other users.

III. DYNAMIC ATTRIBUTES

A. Design

To be efficient, we should allow single attribute update. That is, one attribute operation does not affect the rest of attributes in the same private key. However, attributes in the same private key must not be completely independent from each other. Otherwise, one user could “borrow” attributes from others.

As pointed out above, the main factor prevents private key from scaling up is the single time restriction. If each attribute has its own time restriction, a user can easily replace one attribute with another, add new attributes or delete old attributes. We achieve this by attaching a fading function to each attribute.

Because different users may have different description on the same property, Key Master is responsible to maintain the set of attributes. Key Master publishes all the attributes in form of attribute description (expressed in string) to the public. Each attribute represents a unique entity, item or property. A user chooses attributes which he/she wants to encrypt a message with from this public known set. In addition, Key Master publishes a fading function for each attribute. Time is the input of fading function. Fading function can be any kind of form as long as it outputs different values for different inputs. There is also a published operating function associated with each attribute. Operating function takes attribute description for attribute \( a_1 \) and the output of fading function at time \( t_1 \) as input; it outputs \( o_{a_1,t_1} \) as the attribute value for attribute \( a_1 \) at time \( t_1 \). The attribute value is the real “attribute” used to encrypt and generate private keys. Fig. 2 gives an example of how an attribute \( a_1 \) behaves during a period of time. An attribute’s value changes by time in our design, so we name it with “Dynamic Attribute”.

Key Master should be careful when choosing fading functions and operating functions to avoid collision. Collision may cause problems such that \( o_{a_1,t_1} \) may be the same as \( o_{a_2,t_2} \). We suggest Key Master use functions like non-descending functions and change fading functions and operating functions after a period of time for security consideration. To get better performance, simple operations such as add, minus and multiply are considered in priority when choosing operating functions.

1) Auto Expiration: Dynamic attribute makes old attribute invalid automatically. Say, when someone wants to encrypt a message with attribute \( a_1 \) at time \( t_2 \), he/she uses \( o_{a_1,t_2} \). If a receiver gets an attribute \( a_1 \) at time \( t_1 \), the attribute value of \( a_1 \) in this receiver’s private key is \( o_{a_1,t_1} \). Although the sender encrypts with attribute \( a_2 \) and the receiver has attribute \( a_1 \), their attribute value does not match, preventing this receiver from being able to decrypt the message.

2) Time Unit: The value of Time Unit for fading function is very critical. As showed in Fig. 2, Time Unit controls the fading speed of an attribute, that is how fast an attribute is to be invalid. If Time Unit is too small, an attribute in the private key becomes invalid very soon and thus one needs to update attributes frequently. Too big Time Unit is also not expected since sometimes “fading” is very important for security consideration. A proper value of Time Unit is desired. The general question on what is the best Time Unit value is out of the scope of this paper. But we will have an analysis on how Time Unit affects system security and efficiency (see Section IV).

The good news is not all the attributes have to share the same Time Unit value. Different attributes can have different Time Unit values since they use different fading functions. So a general rule to determine Time Unit value is: for attributes supposed to be changed frequently, e.g., local location attributes (street, neighborhood), Time Unit value should be relatively small; attributes supposed to be changed less frequently, such as big locations (state, country), may have a big value.

B. Mathematical Construction

In the following part we present Dynamic Attributes from mathematical perspective. We use Bethencourt et al. [1] ’s construction of CPABE. Attribute \( j \) is represented by \( \text{att}(j) \) in CPABE. We replace \( \text{att}(j) \) by time-dependent value...
F(att(j), f(t)), where F is an operating function and f is a fading function.

In addition, define $G_0$ as a bilinear group of prime order $p$, $g$ is the generator of $G_0$, $e : G_0 \times G_0 \rightarrow G_1$ denotes a bilinear map. Define the Lagrange coefficient $\Delta_i,S$ for $i \in \mathbb{Z}_p$ and a set, $S$, of elements in $\mathbb{Z}_p$: $\Delta_i,S(x) = \prod_{j \in S, j \neq i} \frac{x - j}{i - j}$. Define a hash function $H : \{0, 1\}^* \rightarrow G_0$ as a random oracle. $S$ describes the set of attributes.

**Setup.** Key Master chooses a bilinear group $G_0$ of prime order $p$ with generator $g$ and two random exponents $\alpha, \beta \in G_0$. Public Key $PK = G_0, g, h = g^\beta, e(g, g)^\alpha$ is published. Key Master keeps Master Key $MK = \beta, \alpha$ as secret.

**Encryption**(PK, M, T). A party first chooses a polynomial $q_x$ for each node $x$ in the policy tree $T$. Set the degree $d_x$ of the polynomial $q_x$ to be one less than the threshold value $k_x$ of that node (for OR, threshold is 1; for AND, threshold is the number of children). Starting with the root node $R$, choose a random $s \in \mathbb{Z}_p$ and sets $q_R(0) = s$. Then randomly choose $d_R$ other points of the polynomial $q_x$ to define $q_x$ completely. For any other node $x$, it sets $q_x(0) = q_{\text{parent}(x)}(\text{index}(x))$. (In $\mathbb{T}$, every children node are numbered starting from 1. Index(x) returns such a number associated with node x), and chooses $d_x$ other points randomly to completely define $q_x$.

The ciphertext is given as follows:

$$CT = (T, \tilde{C} = Me(g, g)^{s_1}, C = h^s; \forall j \in S : C_j = g^{t(0)}, C_j' = H(F(\text{att}(j), f(t)))^{q_j(0)}$$

**KeyGen**(MK, S). Key Master chooses a random $r \in \mathbb{Z}_p$ and random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. Then compute Private Key as

$$SK = (D = g^{\frac{rs}{(x)}}, \forall j \in S : D_j = g^rH(F(j, f(t_i)))^r_j, D'_j = g^{r_j})$$

**Decrypt**(CT, SK). A receiver uses his/her own private key to decrypt an encrypted message. He/she first uses a recursive algorithm DecryptNode(CT, SK, x).

If the $x$ is a leaf node for time $t$, let $i = \text{att}(x)$. When $x$ is not in a user’s private key, DecryptNode(CT, SK, x) = NULL. If a user has attribute $x$ in his/her private key, then:

$$\text{DecryptNode}(CT, SK, x) = e(D_i, C_i)$$

$$= e(g^rH(F(x, f(t_i)))^r_j, g^{q_y(0)})$$

$$= e(g^r, H(F(x, f(t_i)))^{q_y(0)})$$

$$= e(g^r, H(F(x, f(t_i)))^{q_y(0)})$$

When $x$ is a non-leaf node, for all nodes $z$ that are children of $x$, it calls DecryptNode(CT, SK, z) and stores the output as $F_z$. If all the children have valid returns (not NULL), compute

$$F_x = \prod_{z \in S_x} F_z^{\Delta_i, S_z(0)}$$

where $i = \text{index}(z)$, $S'_z = \{\text{index}(z) : z \in S_x\}$

If a user has all the attributes to satisfy the policy tree, he/she can successfully obtain $A = \text{DecryptNode}(CT, SK, R) = e(g, g)^{q_{\text{null}(0)}}$ by recursively calling DecryptNode function from leaf node to root node. Finally, decrypt message by

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

**C. Analysis**

Our design is as secure as the original CPABE because our modifications do not destroy any existing security design in the original CPABE.

For private key holders, sometimes dynamic attributes may cause unnecessary updates since attributes automatically expire after predefined time. A user has to request the attribute that has expired again if he/she is still eligible to get this attribute. But dynamic attribute also avoids private key update, which is supposed to be much more costly. In most cases, overhead produced by private key updates outweighs the unnecessary updates in our scheme. Our simulation results in Section IV support our claim.

Fading functions and operating functions do not bring much overhead. The set of available attributes are maintained by Key Master, so, before encrypting, a sender has to check which attributes (get attribute descriptions) can be used for at least once. Sender can get corresponding fading functions and operating functions when getting the attribute description from Key Master. If a user wants to reuse the same attribute, he/she could calculate the attribute value by himself/herself using the two functions.

Is it necessary for a sender to use the up-to-date attribute value when encrypting? Actually a sender can choose to use the old value. An observation is a party encrypting data is always the one who wants to protect data. So he/she will choose the appropriate attribute value to encrypt. In most cases, appropriate means most recent.

**IV. Simulation**

We simulate dynamic attributes using a scenario similar to SAT. Locations are encoded into attributes. Instead of using street/road names as described in [4], we use Quadrant Location System to represent a vehicle’s location.

**A. Choosing Time Unit**

1) Quadrant Location System: In Quadrant Location System (QLS), an area is divided into four squares. Each square represents a quadrant, which is labelled from 1 to 4. Then each quadrant is recursively subdivided into four smaller quadrants until granularity requirement is met. Quadrants which have the same size are in the same layer. As Fig. 3 shows, four quadrants are in layer 1 and sixteen quadrants are in layer 2. The quadrangular number is used to encode the location. In Fig. 3,
A locates in the third quadrant in layer 1 and the first quadrant in layer 2, so A has encoded quadrant location “3, 1”.

In our simulation, each quadrant represents an attribute. That means here A needs two attributes to indicate his/her location, one for layer 1 and the other for layer 2. Ideally, when one gets into a new quadrant, he/she should get an attribute for the new quadrant and the attribute for the old quadrant that he/she previously stays should be dropped. We define update as the behavior that a user gets a new attribute from Key Master and adds it into his/her private key. Generally speaking, it takes longer to pass a lower layer quadrant (with smaller number and larger size) than a higher layer quadrant, meaning a lower layer attribute updates less frequently than a higher layer attribute. Accordingly, higher layer attributes are supposed to have smaller Time Unit value.

2) Efficiency and Security: First we study the impact of choosing Time Unit for fading functions. We care about two aspects: How efficient our scheme works and how secure it is. To measure efficiency and security, we define two metrics.

Multiple Keys: In one Time Unit time, a vehicle may pass several quadrants in the same layer. As we defined above, it will get an attribute for each quadrant. Attribute value does not change in one Time Unit, so all these multiple attributes are valid. We claim this is not secure because a user should not have valid attributes for the locations where a user is not staying. The number of unexpected attributes, which is equal to the number of quadrants that a vehicle passes in one Time Unit minus one, is defined as Multiple Keys.

Update Frequency: If one stays in one quadrant for a long time, he/she has to get a new attribute for the same quadrant since every Time Unit the old attribute gets expired. For example, if a vehicle takes 60s to pass a quadrant, it needs 4 updates in that quadrant if the Time Unit is 15s. We certainly want to update only once in one quadrant. So Update Frequency reflects efficiency.

3) Setup: We proceed simulation using VanetMobiSim. All the nodes move in a 2400m × 2400m area. Each run of simulation lasts for 3600 seconds. We put 10 nodes (vehicles) in the simulation. Nodes have maximum speed of 30m/s and minimum speed of 14m/s. To be close to the real situation, we use TIGER maps [9] and Intelligence Driving Model [10], [11]. Two TIGER maps are used, one is Los Angeles(+34063690, −118415289), and the other is Washington D.C. (+38890000, −77020000). We define a 3-layer QLS. That is, every vehicle uses 3 attributes to identify its location. Layer 1 quadrant has edge length of 1200m, 600m for layer 2 and 300m for layer 3. A different random seed is chosen for each run (random seed affects initial location, path selection and speed). We run 10 times to get average statistics.

4) Results for Time Unit: Fig. 4 shows that Multiple Keys grows linearly with Time Unit value while Update Frequency decreases when Time Unit increases (appears to be \(\text{UpdateFrequency} \times \text{TimeUnit} = \text{Constant}\)). Such results are expected because when Time Unit gets bigger, in most cases, a vehicle moves longer distance thus passing more quadrants in one Time Unit and spending less Time Unit to pass a quadrant.

However, the above curves cannot completely reflect all the truth of attribute update due to the impact of average. Fig. 5(a) shows as Time Unit decreases, most users tend to have less Multiple Keys. Small Time Unit value also improves worst situation, e.g., users have 1 or 0 Multiple Keys during most time when Time Unit is 60s while they have more than 1 Multiple Keys for nearly half of simulation time when Time Unit is 200s. Update Frequency is the opposite. Fig. 5(b) indicates that the increase of the Time Unit improves worst situation. That is, some users may have to update 10 times or more in one quadrant if Time Unit is 30s; while the worst situation for Time Unit of 200s is updating twice in one quadrant. Based on the above analysis, it is quite obvious that there is a tradeoff between Multiple Keys and Update Frequency. This sounds reasonable since as known to all, tradeoff exists between security and efficiency.

B. Performance

1) Setup: We further analyze how well our scheme works in SAT comparing with CPABE. We still use location attributes in 3-layer QLS. Instead of using a single QLS, we use a set of 3-layer QLSs. Each QLS has a vertical and/or horizontal shift from original QLS. That is, if A has location of “1, 1, 4”
in one QLS, it may have location of “1, 2, 4” in another QLS. If there are $n$ QLSs, a user can have up to $3n$ attributes. We use 40s, 75s and 160s as Time Unit for each layer, meaning Multiple Keys is 0.5 and Update Frequency is 2.

For CPABE, all the attributes in the private key rely on single time restriction. We assume time restriction predicts so accurately that a user only needs to update private key when necessary (enter a new quadrant). However, CPABE has to drop the old private key as well as all the attributes in it and get a new private key when updating. For dynamic attributes, private key simply “absorbs” new attribute. But our scheme has additional cost as mentioned in Section III-C.

We increase the size of private key from 1 attribute to 50 and measure how many attributes are transmitted from Key Master to vehicles when updating. Attributes are choosed from 50 QLSs (150 attributes in total); each QLS provides one attribute. Whether an attribute from one QLS is layer 1, layer 2 or layer 3 is randomly decided.

2) Results for Comparison with CPABE: Fig. 6 shows the number of attributes needed to be transmitted (equal to the number of attributes that Key Master has to generate) significantly increases when the number of attributes in private key grows. It illustrates that private key in CPABE is not scalable. Although our scheme has a little more overhead when the number of attributes in private key is very small, it does have good performance when there are many attributes in the private key. It is worthwhile to point out that because we assume CPABE only updates when necessary, the performance of CPABE is a little overestimated.

V. ATTRIBUTE DELEGATION

In some cases, a user may not be able to receive attributes from Key Master. For example, when vehicles are moving in group pattern, they tend to change locations almost the same time and update location attribute simultaneously. Key Master may not be able to handle so many requests in a very short time, thus some vehicles may fail to get requested attributes. Or in a case that a user temporarily loses the connection to Key Master and it needs others to request attributes from Key Master. So delegation is desired.

CPABE is designed against collusion, which means a user cannot use others’ attributes. Therefore, attribute delegation should also take this property into consideration. We modify our design to adapt attribute delegation.

For users’ private key, Key Master chooses random $r, w \in \mathbb{Z}_p$ for each user and random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. $u \in \mathbb{Z}_p$ and $h = g^u$. $u$ is part of Master Key, which is kept as secret by Key Master. Private key looks like:

$$SK = (D = g^{\frac{u+w}{r}}; SS = g^w h^{r+w}$$

$\forall j \in S: D_j = g^r H(F(j, f(t)))^{r_j}, D'_j = g^{r_j})$

Key Master stores $r + w$ for each user.

A group of users first select a group leader. Selection method is user-defined. The only thing should be guaranteed is that the group leader knows all the members in the group. Notice that we don’t need a social trust inside the group. For instance, strangers who want to update the same location attribute can form a temporary group. It is possible that some groups are very large. In that case, there can be several group leaders and each leader is responsible for part of the large group. Group leader sends attribute request to Key Master. In addition, group leader also tells Key Master who are in his/her group.

After verifying request from group leader, Key Master randomly chooses $r_{sample}, r_j \in \mathbb{Z}_p$ (we use $r_s$ to represent $r_{sample}$ afterwards) and generates a sample attribute:

$$ATT_{j,sample} = (D_{j,sample} = g^{r_s h^{r + w}} H(F(j, f(t)))^{r_j}, D'_{j,sample} = g^{r_j})$$

Key Master also generates some extra information to limit the owner of this attribute. $U$ defines all the group members.
Extra information is
\[ m \in U, E_m = \frac{r_m + w_m}{r_k} \]

Key Master sends sample attribute and extra information back to the group leader. Group leader is responsible to distribute the sample attribute and extra information \( E_k \) to each member \( (k) \). To get an attribute from sample:

\[ D_{j, Kevin} = D_{j, sample}/SS_{Kevin} \]
\[ = (g^{r_j}h^{r_j}H(F(j, f(t)))^{v_j})/(g^{r_j}h^{r_j}w_k) \]
\[ = g^{r_k}H(F(j, f(t)))^{v_j}/r_k \]
\[ D'_{j, Kevin} = D'_{j, sample} \]
\[ = (g^{r_j})^{r_j}/r_k \]
\[ = g^{r_j}_{j, Kevin} \]
\[ ATT_{j, Kevin} = (D_{j, Kevin}, D'_{j, Kevin}) \]

Attribute delegation should be secure enough to prevent group leader generating new attributes, and prevent group members derivate attributes from the group leader. Here, group leader (say Kevin) has no way to know secret information \( g^{r_{Kevin}} \) (random \( r \) for Kevin), thus is not able to generate new attributes by himself. Only the attributes issued by Key Master can be derivated while the group leader’s original attributes cannot be “lent” to his/her group members (say Sarah). This is because users never have chances to know \( g^{r_{Kevin}−r_{Sarah}} \) (or \( g^{r_{Sarah}−r_{Kevin}} \)). Since there is no social trust in the group, it is worthwhile for group members to verify the correctness of the attributes obtained from the group leader. This can be done easily by encrypting with that attribute and checking whether the encrypted message can be correctly decrypted using that attribute.

During the whole process, Key Master only generates one copy of attribute as well as a little additional low-cost extra information. Nearly all the high-cost computation and distribution task are done either by group leader or by group members themselves. Therefore, delegation saves a lot of resources for Key Master and reduce the possibility of Key Master being the bottleneck.

VI. Conclusion

CPABE encrypts a message with a access control policy tree, which is a description of a logical composition of a set of attributes. Users whose attributes in their private key satisfy the policy tree can decrypt the message. Therefore, CPABE allows users to do encrypted group communication or multicast in the absence of shared group key. However, all the attributes in CPABE’s private key rely on the single time restriction, which makes it difficult for CPABE to contain a number of attributes in the private key when attributes tend to change frequently. We introduce a fading function to CPABE, allowing attributes “dynamic”. Therefore, updating attributes becomes independent from updating the private key, which makes key management (including key generation, key transmission and private key management) much more efficient.

Our main contribution is the design of dynamic attributes. Dynamic attributes provides a mechanism of attribute revocation. The study of the tradeoff between Multiple Keys and Update Frequency guilds to choose appropriate Time Unit value for fading function. We also provides attribute delegation in non-trust environment, which is quite useful for applications such as SAT.

ACKNOWLEDGMENT

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES