Efficient CP-ABE Attribute/Key Management for IoT Applications

Lyes Touati
Sorbonne universités
Université de Technologie de Compiègne,
CNRS, Heudiasyc UMR 7253,
CS 60 319, 60 203 Compiègne cedex.
Email: lyes.touati@hds.utc.fr

Yacine Challal
Centre de Recherche sur l'Information Scientifique et Technique
CERIST, 05 Rue des Frères Aissou, Ben Aknoun
Algiers, Algeria
Email: ychallal@cerist.dz

***Abstract*—Ciphertext-Policy Attribute-Based Encryption (CP-ABE) is a promising cryptographic mechanism for fine-grained access control to shared data. Attribute/Key management is a keystone issue in CP-APE because of low efficiency of existing attribute revocation techniques. Indeed, existing solutions induce great side effect after each attribute revocation. The side effect induces rekeying and/or re-assignment of attributes to all users.**

In this paper, we propose a solution which does not require extra entities like proxies to re-encrypt data after every access policy change. Moreover, our solution does not imply latencies following access grants and revocations. We compare our solution with the batch-based CP-ABE attribute management technique and we show that our solution outperforms existing rekeying/revocation techniques in terms of overhead.

***Keywords*-CP-ABE; Internet of Things; Access Control; Attribute revocation;**

I. Introduction

The Internet of Things (IoT) is an enabling technology for Cyber-Physical Systems or Systems of Systems. Indeed, the Internet is evolving from a network of personal computers and servers toward a huge network interconnecting billions of smart communicating objects. It is expected that more than 50 billions devices will be connected to the Internet by 2020 (sensors, smart-phones, laptops, cars, clothes, wristwatches, etc.). These objects will be integrated into complex systems and use sensors and actuators to observe and interact with their physical environment, and hence allowing interaction among autonomous systems [1].

Internet of Things applications are ranging from military (enemy territories exploration, soldiers monitoring, ...), to e-health (monitoring elder-lies, remote diagnosis, ...), smart cities, smart grid, smart vehicles and transportation (traffic jam management), etc. The challenge of securing Internet of Things applications is a tricky issue as these latter are very sensitive to attacks and great damages may be caused in both systems and their users in the case of a possible security attack. Therefore, fine-grained access control becomes a crucial security service to prevent attacks against those sensitive applications.

Attribute-Based Encryption (ABE) [2], [3], [4] is a promising mechanism which allows implementing efficiently fine-grained access control in IoT applications. However, IoT environments are usually dynamic systems that evolve through time, therefore, attribute-based encryption must be combined with an efficient attribute management mechanism. The latter is known to be a tricky issue in ABE, because an attribute could be shared with many users at the same time. Thus, revoking that attribute from a user has inevitably impact on other users sharing the same attribute i.e. their secret keys must be updated, and therefore, this could decrease considerably system performances.

In this paper we propose a solution to implement an attribute revocation mechanism with CP-ABE without requiring data re-encryption after every access policy change. Our solution eliminates the overhead due to re-encryption and renaming attributes and does not require proxies to achieve attribute revocation. Moreover, our solution reduces to the minimum the number of parts in generated secret keys.

The rest of the paper is organized as follows. We begin with a presentation of some preliminaries in section II. Then, we provide an overview and a construction of our solution in sections III and IV, respectively. Next, we analyze the performance of our scheme and we compare it against another Batch-based attribute mangement scheme in section V. Finally, we discuss related works in section VI, and conclude the paper in section VII.

II. Background

In this section we review some basic concepts and notions related to CP-ABE scheme [3].

Ciphertext-policy Attribute-Based Encryption is an asymmetric encryption mechanism that allows to implement cryptographic fine-grained access control. Each user is associated with a list of attributes that reflect her/his role in the system. A special entity called *Attribute Authority* (AA)generates a public key PK which is shared with all system entities, and generates also users' private keys SK from their lists of attributes.

Y. Challal is associate professor at Ecole Nationale Supérieure d'Informatique (ESI, Algiers, Algeria). He is member of Systems Design Methods lab. (LMCS)

An entity that wishes to encrypt a message will specify an access policy in a form of an access tree. Attributes list of a user who wants decrypting an ciphertext must satisfy the access policy in order to be able to decrypt the message.

Access tree.

An access tree is used to describe access policy of an encrypted message. For instance, access policy shown in Figure 1 can be expressed differently as follows: ((*"Student"* **OR** *"Ph.d Student"* **OR** *"researcher"*) **AND** (*"Physics"* **OR** *"Biology"*)).

Each non-leaf node of the access tree represents a threshold gate, described by its children and a threshold value. If num_x is the number of children of a non-leaf node x and k_x is its threshold value, then $0 < k_x \leq num_x$. Two particular cases are **"AND"** and **"OR"** gates: *"AND"* gate has $k_x = num_x$ and *"OR"* gate has $k_x = 1$.

Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$.

Some functions are defined to facilitate working with access trees:

- *parent(x)*: denotes the parent of the node x in the tree.
- *att(x)*: is defined only if x is a leaf node, and denotes the attribute associated with the leaf node x in the tree.
- *index(x)*: denotes the order of the node x between its brothers. The nodes are randomly numbered from 1 to num.

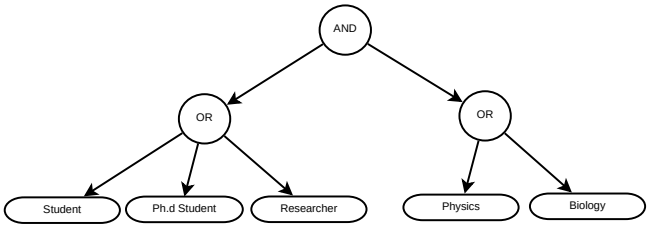

Figure 1: Example of an access tree

Satisfying an access tree. Let γ be an access tree with root r. γ_x denotes the sub-tree of γ rooted at the node x. Hence γ is the same as γ_r. If a set of attributes A satisfies the access tree γ_x, we denote it as $\gamma_x(A) = 1$. We compute $\gamma_x(A)$ recursively as follows:

If x is a non-leaf node, evaluate $\gamma_{x'}(A)$ for all children x' of node x. $\gamma_x(A)$ returns 1 if and only if at least k_x children return 1.

If x is a leaf node, then $\gamma_x(A)$ returns 1 if and only if $att(x) \in A$.

For example, the attribute set $S_1 = \{"Student","Physics"\}$ satisfies the access policy defined above, but not $S_2 = \{"Ph.d\ Student","Computer\ Science"\}$.

III. Overview

In this section we present an overview of our solution to implement an attribute revocation mechanism for CP-ABE.

A. System Model

Let us consider a set of users U_i where $1 \leq i \leq N$ and a set of attributes A. Each user U_i holds a subset of attributes $A_i \subseteq A$.

In our solution, we target IoT applications where all start dates and duration of attributes validity are known beforehand. This is common in institutions where users' roles and hence attributes do not evolve rapidly. For instance, this can be applied in health or education institutions where physicians, nurses, interns, students, professors, etc. and relating objects hold attributes reflecting their positions, roles and functions for a known period starting from a known date.

Thus, the Attribute Authority begins collecting all attributes validity periods. Then, the Attribute Authority determines for each attribute, separately, the series of time slots with variable durations as shown in figure 2. Then, The Attribute Authority assigns an identifier to each time slot and determines the number of secret key parts to generate and send to each user according to their attribute validity periods.

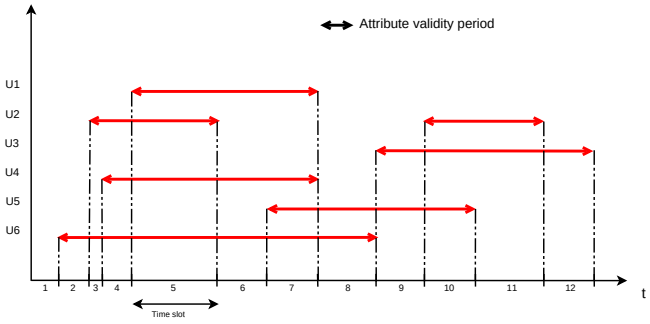

Figure 2: Example of creating time slots with variable durations

In figure 2, axis of ordinates contains different users $(U1, U2, ..., U6)$, and axis of abscissa represents time which is split into different time slots with different durations. Vertical projections of time events [1] upon axis abscissa are shown with dotted lines. These projections will determine beginnings and ends of time slots.

For example, user $U1$ has a validity period that extends over three time slots (5, 6 and 7). Table I shows the number of time slots and their assignment by the Attribute Authority to each user.

When an attribute related event occurs (attribute validity period starts or ends), the Attribute Authority increments the

[1]We mean by event any attribute validity period beginning or ending.

Table I: Example.

	Number of time slots	Corresponding time slots
U1	3	5,6,7
U2	5	3, 4, 5, 10, 11
U3	4	9, 10, 11, 12
U4	4	4, 5, 6, 7
U5	4	7, 8, 9, 10
U6	7	2, 3, 4, 5, 6, 7, 8

time slot identifier T_{att} related to that attribute and informs all entities in the system. We assume that the system implements a synchronization protocol that allows all entities to know when to move to another time slot number for a given attribute. This mechanism can be achieved by broadcasting a signal for example, or synchronize all nodes' clocks and acquaint all entities of security parameters changing dates.

In our solution, secret key parts (SKP) associated to an attribute are generated so that a user shifts easily to the new secret key associated to the current time slot without being able to generate secret keys for time slots outside the attribute validity scope for the user.

B. Security Requirements

Our solution guarantees the following security services:

- **Data Confidentiality:** Unauthorized users who do not have the required attributes satisfying the access policy of a ciphertext must be prevented from accessing the plaintext of the data.
- **Backward secrecy:** A user gaining new attributes should not have any access to previous unauthorized encrypted data even if her/his new list of attributes satisfies the access policy of the encrypted data.
- **Forward secrecy:** When some attributes are revoked to a user, she/he should have no access to current and future encrypted data if her/his new list of attributes does not satisfy the access policy of the encrypted data.
- **Collusion freedom:** Collusion resistance is a required property of any ABE system. Even if many users not satisfying the access policy collude, they can obtain no information about the plaintext of the encrypted data. This property means that private keys could not be used together in order to gain more access rights than it would be if they are used separately.

IV. Our Solution

A. Notation

We use the following notations to describe our solution achieving attribute revocation with Ciphrtext-Policy Attribute-Based Encryption.

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:

Table II: Summary of notations.

Notation	Description
att	An attribute
PK	Public Key generated by the Attribute Authority
SK	Secret Key generated by the Attribute Authority for each user from her/his attributes list
SKP	Secret Key Parts
num_x	Number of children of a non-leaf node x in an access tree
k_x	Threshold value assigned to each non-leaf node in an access tree
γ	Access tree defining access policy for a ciphertext
T_{att}	Current time slot identifier related to the attribute att
TSL	Time Slots identifier List
TSL_{att}	Time Slot identifiers List representing Validity period for the attribute att for a specific user
$TSL\,(att)$	Time slot identifier corresponding to the attribute att in TSL list
M	Plaintext of the message to be encrypted
CT	Ciphertext representing the encrypted message
Y	Leaf nodes set of an access tree

1) Bilinearity: for all $u, v \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$, we have $e\left(u^a, v^b\right) = e\left(u, v\right)^{ab}$.
2) Non-degeneracy: $e\left(g, g\right) \neq 1$.

We say that $\mathbb{G}_0$ is a bilinear group if the group operation in $\mathbb{G}_0$ and the bilinear map e are both efficiently computable.

B. Solution Concept

In this work, we aim to develop an attribute revocation for CP-ABE scheme without renaming attributes or introducing any delay between the real attribute validity period and the validity period associated to the generated secret key parts list.

The idea of our solution is to split time axis into time slots with variable durations. These time slots are determined according to users attributes validity periods as explained in section III-A. Instead of renaming attributes for each time slot, we introduce a new one way hash function that returns a different result for each time slot.

We define the new one way hash function as follows:

$$\begin{array}{rccc} H: & \mathbb{A} \times \mathbb{N} & \longrightarrow & \mathbb{G}_0 \\ & (att, i) & \longmapsto & H\left(att, i\right) \end{array}$$

$\mathbb{A}$ denotes the set of all attributes used by the Attribute Authority in the system. $\mathbb{G}_0$ is a bilinear group of prime order p.

The hash function we use in our approach takes two parameters. The first parameter is an element from the set of attributes maintained by the Attribute Authority, the second one is an integer that represents a time slot queuing ticket.

The probability of collision existence in the one-way hash function defined above should be infinitely small, which means:

$$\forall att_i, att_j \in \mathbb{A}, \forall k, l \in \mathbb{N} : (att_i, k) \neq (att_j, l) \Rightarrow P\left(H\left(att_i, k\right) = H\left(att_j, l\right)\right) \approx 0 \quad (1)$$

By using our new hash function, a secret key part related to an attribute is valid during only one time slot which is the one whose identifier is given to the hash function. This way, the Attribute Authority has not to rename attributes in order to revoke them from some users, and has not also to regenerate all secret keys for all users every attribute revocation, it generates only parts of the secret key related to an attribute.

C. Primitives

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 : denote the bilinear map.

There are four cryptographic primitives:

Setup. The setup algorithm is run by the Attribute Authority at the bootstrap phase. It takes no input other than the implicit security parameter. It outputs the public parameters PK which is shared with all the entities of the system and a master key MK kept secret.

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

$$PK = \left(\mathbb{G}_0, g, h = g^{\beta}, f = g^{1/\beta}, e\left(g,g\right)^{\alpha}\right) \tag{2}$$

and the master key is:

$$MK = (\beta, g^{\alpha}) \tag{3}$$

KeyGen*(MK, S)*. The KeyGen primitive is run by the Attribute Authority for each user joining the system. It takes as input the master key MK, a set of couples S. Each element of the set S contains two parts: the first one is an attribute $att \in A$, and the second one is a list of time slots numbers TSL_{att} defining the validity period of the attribute att.

We can write the set S as:

$$S = \left\{\left(att, TSL_{att}\right), \forall att \in A\right\} \tag{4}$$

The Key generation algorithm begins by choosing a random $r \in Z_p$, and then a random $r_j \in Z_p$ for each attribute $j \in A$. Then, it computes the key as follows:

$$SK = \Big(D = g^{(\alpha+r)/\beta}, \forall j \in A, \forall k \in TSL_j : D_{j,k} = g^{r} \cdot H\left(j,k\right)^{r_j}, D'_j = g^{r_j}\Big) \tag{5}$$

Note here that the parameter $D_{j,k}$ is related to the attribute j for the time slot number k.

In formula 5, SK represents a user *global* secret key throughout the lifetime of the system; it contains all the subkeys that are used to decrypt ciphertexts. At a specific time, the user uses one of these subkeys to decrypt data. The subkeys are extracted from the global secret key SK by keeping only the elements related to the current time slot number for each attribute in A. Let TSL be a list of time slots identifiers representing the current time slots identifiers of all attributes in A. The subkey related to TSL is noted SK_{TSL} and is computed as following:

$$SK_{TSL} = \left(D, \forall j \in A : D_{j,TSL(j)}, D'_j\right) \tag{6}$$

The writing $TSL(j)$ means the element of TSL which is related to the attribute j, it represents the current time slot identifier of the attribute j.

Encrypt*(PK, M, γ, TSL)*. The encryption algorithm takes as input the public parameters PK, a message M, an access structure γ over the universe of attributes and a time slots list TSL containing a list of current time slots numbers associated with the attributes of the access structure leaf nodes. The algorithm will encrypt M and produce a ciphertext CT such that only a user that possesses a set of attributes, during their corresponding time slots in TSL, that satisfies the access structure will be able to decrypt the message. We will assume that the ciphertext implicitly contains γ and TSL.

It operate in the same manner as the standard version defined in [3] except in using our hash function defined in IV-B. Each attribute in leaf nodes of the access tree γ has its corresponding time slot number in TSL. The algorithm first chooses a polynomial q_x for each node x in the access tree γ. These polynomials are chosen in top-down manner, starting from the root node R down to leaf nodes. For each node x in the tree, the degree of the polynomial q_x is set to be one less than the threshold value k_x of that node: $d_x = k_x - 1$.

The algorithm chooses a random $s \in Z_p$ and sets $q_R\left(0\right) = s$. Then, chooses d_R other points of the polynomial q_R randomly to define it comletely. For any other node x, it sets $q_x\left(0\right) = q_{parent(x)}\left(index\left(x\right)\right)$ and chooses d_x other points randomly to define q_x.

Let, Y be the set of leaf nodes in γ. Y and TSL have the same size, and every element $y \in Y$ has its corresponding element $TSL\left(y\right) \in TSL$. The ciphertext is the constracted by giving the tree access structure γ, the current time slot number for each element in Y, and computing:

$$CT = \Big(\gamma, \tilde{C} = Me(g,g)^{\alpha s}, C = h^{s}, \forall y \in Y : TSL\left(y\right), C_y = g^{q_y(0)}, C'_y = H\left(att\left(y\right), TSL\left(y\right)\right)^{q_y(0)}\Big) \tag{7}$$

Decrypt*(CT, SK_{TSL})*. The decryption algorithm takes as input a ciphertext CT, which contains an access policy γ, and a private key SK_{TSL} constructed from a list A of attributes associated to the time slots list TSL. The time slots list TSL used here is the same as the one used for

constructing the ciphertext CT. If the set A associated to a time slots list TSL of attributes satisfies the access structure γ then the algorithm will be able to decrypt the ciphertext and return a message M.

The decryption primitive is pretty similar to the one defined in [3] except in using our hash function defined in section IV-B. We first define $DecryptNode(CT, SK_{TSL}, x)$ which is a recursive function. It takes a ciphertext $CT = \left(\gamma, \tilde{C}, C, \forall y \in Y : TSL(y), C_y, C'_y\right)$, a private key $SK_{TSL} = \left(D, \forall j \in A : D_{j,TSL(j)}, D'_j\right)$ which is associated with a set A of attributes, and a node x from γ.

Case 1: The node x is a leaf node, then we let $i = att(x)$. If $i \in A$, then

$$\begin{aligned} DecryptNode(CT, SK_{TSL}, x) &= \frac{e(D_{i,T}, C_x)}{e(D'_i, C'x)} \\ &= \frac{e\left(g^r \cdot H(i, TSL(i))^{r_i}, g^{q_x(0)}\right)}{e\left(g^{r_i}, H(i, TSL(i))^{q_x(0)}\right)} \\ &= e(g,g)^{rq_x(0)}. \end{aligned}$$

and if $i \notin A$, then $DecryptNode(CT, SK_{TSL}, x) = \perp$.

Case 2: The node x is a not leaf node.

The algorithm proceeds as follows: For all nodes z that are children of x, it calls $DecryptNode(CT, SK_{TSL}, z)$ and stores the output as F_z.

Otherwise, we compute

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}; \\ &= \prod_{z \in S_x} \left(e(g,g)^{r \cdot q_z(0)}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e(g,g)^{r \cdot q_{parent(z)}(index(z))}\right)^{\Delta_{i,S'_x}(0)} \\ &= \prod_{z \in S_x} e(g,g)^{r \cdot q_x(i) \cdot \Delta_{i,S'_x}(0)} \\ &= e(g,g)^{r \cdot q_x(0)} \text{ (Using polynomial interpolation)} \end{aligned}$$

Where $i = index(z), S'_x = \{index(z) : z \in S_x\}$.

We recall that $\Delta_{i,S}(x)$ is the Lagrange coefficient defined as follows:

$$\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} (x - j) / (i - j).$$

where i be an element in $\mathbb{Z}_p$, and S a set of elements in $\mathbb{Z}_p$.

After describing the function $DecryptNode$, we can now write the decryption algorithm. The algorithm begins by calling the function $DecryptNode$ on the root node R of the tree γ. If the tree is satisfied by the attributes set A we set $B = DecryptNode(CT, SK_{TSL}, r) = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$. The algorithm now decrypts by computing

$$\begin{aligned} \tilde{C} / \left(e(C,D)/B\right) &= \tilde{C} / \left(e\left(h^s, g^{(\alpha+r)/\beta}\right) / e(g,g)^{rs}\right) \\ &= Me(g,g)^{\alpha s} / \left(e(g,g)^{s(\alpha+r)} / e(g,g)^{rs}\right) \\ &= M. \end{aligned}$$

We recall that if the attribute list A that the user possesses during the time slots in TSL does not satisfy the access policy γ, the decryption primitive could not decrypt the ciphertext.

V. Performance Evaluation

A. Simulation Model

For the sake of simplicity and without loss of generality, we consider a group of users which ask gaining the access right to one attribute. Results can be easily extrapolated when considering multiple independent attributes. We modeled users' requests, which represent starting dates of attribute validity periods, by Poisson process with parameter λ. The attribute validity periods durations for all users follow exponential distribution with parameter μ. The following simulations are made considering a system with one thousand (1000) entities.

We are interested in evaluating the overhead in terms of generated secret key parts (SKP) for an attribute[2]. This metric is very important since it determines the size of generated secret key sent by the Attribute Authority to system entities. SKP reflects closely system performances. Indeed, the less is the number of secret key parts SKP, better is the solution.

B. Performance Analysis

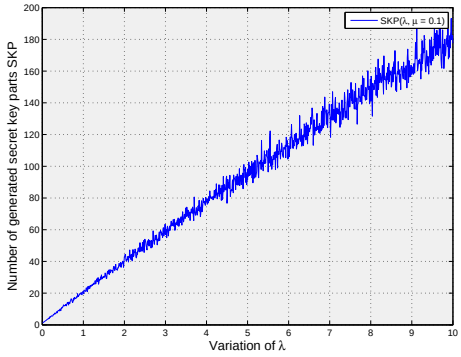

Figure 3: Number of generated secret key parts SKP with respect to λ

[2]We mean by secret key part the element $D_{j,k}$ in the secret key SK (see section IV-C)

Figure 3 shows the variation of the secret key parts SKP needed to be sent with respect to Poisson process parameter λ. The value fixed for μ is 0.1. The number of generated secret key parts SKP increases almost linearly with the λ. our solution shows better results in applications cases where λ is small.

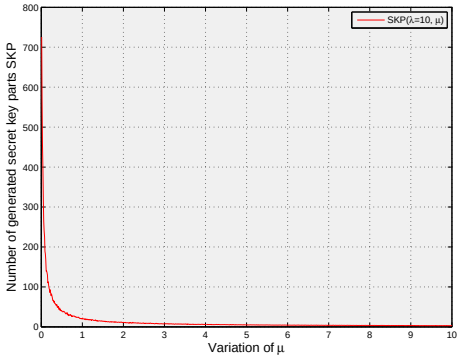

Figure 4: Number of generated secret key parts SKP with respect to μ

Figure 4 shows the variation of the secret key parts needed to be sent with respect to the exponential distribution parameter μ. We fixed $\lambda = 10$.

According to the two figures 3 and 4 we approximate $SKP(\lambda, \mu)$ by the following formulas:

$$SKP(\lambda, \mu) \approx 2 * \lambda/\mu + 1 \quad (8)$$

Figure 5 shows the variation of the number of generated secret key part SKP with respect to both λ and μ.

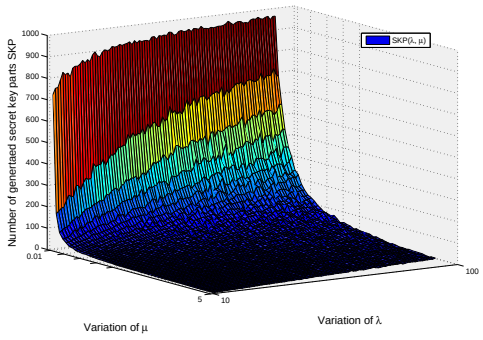

Figure 5: Number of generated secret key parts SKP with respect to both λ and μ

C. Comparisons

We compare our solution to a Batch-based rekeying technique (BB-CP-ABE) [5]. In BB-CP-ABE, all time slots have the same duration Δt and if attribute validity period beginning and end occur during a time slot, they are delayed until the next time slot begins.

We have conducted simulations to compare them and show the advantages of our new solution. We considered a system with one thousand (1000) users, where attribute starting dates follow Poisson process with parameter λ and attribute validity periods have an exponential distribution with parameter μ.

We choose three values for time slot duration Δt of Batch-Based CP-ABE solution: $\Delta t1 = 1$, $\Delta t2 = 2$ and $\Delta t3 = 4$. These three cases correspond to three possible configurations of BB-CP-ABE.

In the first simulation, we set $\mu = 0.1$ and we computed the variation of the numbers of generated secret key parts in the four cases with respect to λ. Simulation results are shown in figure 6.

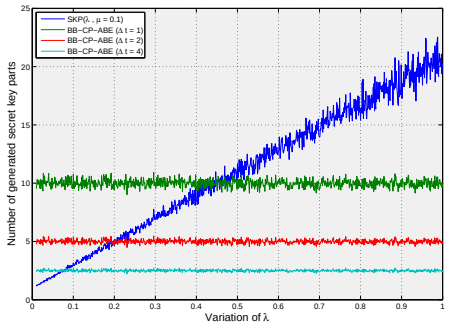

Figure 6: Comparison with respect to λ

In the case of our solution, the number of elements to be sent increases linearly with respect to λ, this can be explained by the fact that the higher is λ, the less the time between two incoming users (attribute validity periods); therefore, the more frequent are overlaps between attribute validity periods. The curves in the three cases of BB-CP-ABE are almost constant, this is because the number of elements to be sent does not depend on λ, it depends only on μ and time slot duration. For small values of λ, our solution shows better performance than BB-CP-ABE. It is important to remember here that our solution does not induce any delay, The requested validity period is the same as the delivered validity period. but in BB-CP-ABE and according to the time slot duration we have an average delay equal to 1/2, 1, 2 respectively in the case of $\Delta t1$, $\Delta t2$ and $\Delta t3$.

In the second simulation, we set $\lambda = 0.1$ and we computed the variation of the numbers of generated secret key parts in the four cases with respect to μ. Simulation results are shown in figure 7.

The four curves are inversely proportional to μ. For small values of μ, our solution shown with dark blue ink gives best results then the three others. But with larger values of

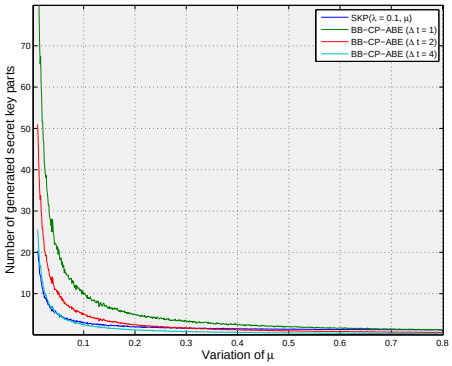

Figure 7: Comparison with respect to μ

μ, our solution becomes less efficient then the three others. We also recall here that our solution contrary to others does not induce any delay.

VI. Related Works

Attribute-based encryption (ABE) is a public key encryption mechanism that allows users to encrypt and decrypt messages based on descriptive user attributes. There are mainly two variants of ABE: Ciphertext-Policy Attribute-Based Encryption [3] and Key-Policy Attribute-Based Encryption [4]. In KP-ABE, attributes are used to describe the encrypted data and policies are built into user's keys; while in CP-ABE, the attributes are used to describe a user's private key, and an encryptor determines a policy on who can decrypt the data and include it into the encrypted data. Therefore, CP-ABE is considered as a promising solution to resolve the issue of fine-grained cryptographic access control on shared data. However, CP-ABE suffers from some drawbacks such as the nonexistence of solutions implementing efficiently attribute revocation.

We can find some solutions in the literature for attribute revocation in ABE systems. M. Pirretti et al. proposed in [6] an idea on how attribute revocation could be implemented with CP-ABE. The principle of their idea is to renaming attributes by concatenating them with their corresponding expiration dates. Once an attribute expiration date comes, the Attribute Authority renames that attribute and broadcasts it to all entities in the system, then, it regenerates all secret keys to the non-revoked users (the revocation is materialized by not receiving a new secret key including the renamed attribute). This solution induces a heavy overhead as long as all entities will be affected by the revocation. Another solution proposed in [3] by J. Bethencourt et al. which consists on expressing the revocation condition in the access tree and including it into the access tree. This is possible by transforming numerical attributes to non-numerical ones. This solution transforms access trees bigger and more complex than before, and therefore, the overhead considerably increases.

Another kind of solutions consist on using a proxy re-encryption mechanism (PRE [7]) such as [8], [9], [10] and [11]. Proxies are provided in the network to absorb the overhead due to the re-encryption. In [8], Z. Xu et al. addressed user revocation and key refreshing issue for CP-ABE sheme in data-owner-centric environments like those for cloud storage. Their solution named *DURKR* uses the proxy re-encryption mechanism and considers only user revocation. It requires a cloud storage provider to re-encrypt data for every user request. Y. Cheng et al. considered in [9] a data storage and delivery system. They solution consists on combining proxy re-encryption (PRE) and (n, n) threshold scheme known as Secret Sharing Schemes (SSS). In [11], Yu et al. tried to resolve the challenging issue of key revocation in CP-ABE by considering practical scenarios like data sharing in which semi-trusted on-line proxy servers are available. Their solution integrates Proxy Re-Encryption (PRE [7]) technique with CP-ABE and enables the authority to revoke user attributes and to delegate laborious tasks to proxy servers. This solution requires to regenerate all users secret keys and re-encrypting data after every change occurred in the access policy.

In [10], S. Jahid et al. developed a proxy-based revocation solution for attribute based encryption called *PIRATTE*. The revocation mechanism is based on polynomial secret sharing which allows to do up to t revocations, where t is the degree of the polynomial in the mechanism. Their solution requires a proxy that participates to the decryption process. Although the scheme achieves dynamic user/attribute revocation without regenerating users keys, it can only revoke up to a predefined numbers of users/attributes.

In [12] and [13], Wang et al. combined Hierarchical identity-based encryption (HIBE) [14] system and CP-ABE to propose a Hierarchical Attribute Based Encryption (HABE) with with full delegation. The attribute revocation is achieved by re-encrypting data and updating secret keys. Authors proposed to use proxy re-encryption (PRE) [7] and lazy re-encryption to enhance system performances.

In [5], a batch-based solution is proposed to ensure attribute revocation in CP-ABE scheme. Authors proposed to split time axis into intervals of fixed duration called *time slots*, and all attribute-based access policy changes that occurred during a given time slot are delayed until it ends. The challenge of this solution is to find the appropriate time slot duration that optimizes system performances which strongly depends on the type of the application.

VII. Conclusion and Future Work

In this paper we addressed an important issue which is attribute revocation for attribute based encryption schemes. In particular, we considered practical application scenarios in which the Attribute Authority knows beforehand start

dates and durations of all attributes validity periods, and proposed a scheme supporting attribute revocation. One nice property of our proposed scheme is that it doesn't require extra entities in the network like proxies and does not require re-encrypting data to achieve the revocation. The solution we proposed here induces zero delay and a minimum of generated secret key parts.

ACKNOWLEDGMENT

This work was carried out and funded in the framework of the Labex MS2T. It was supported by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.05.010

[2] L. Pang, J. Yang, and Z. Jiang, "A survey of research progress and development tendency of attribute-based encryption," *The Scientific World Journal*, vol. 2014, 2014.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007, pp. 321–334.

[4] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 89–98.

[5] L. Touati and Y. Challal, "Batch-Based CP-ABE with attribute revocation mechanism for the internet of things," in *2015 International Conference on Computing, Networking and Communications, Wireless Networks Symposium (ICNC'15 WN)*, Anaheim, USA, Feb. 2015.

[6] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters, "Secure attribute-based systems," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 99–112. [Online]. Available: http://doi.acm.org/10.1145/1180405.1180419

[7] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *In EUROCRYPT*. Springer-Verlag, 1998, pp. 127–144.

[8] Z. Xu and K. Martin, "Dynamic user revocation and key refreshing for attribute-based encryption in cloud storage," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, June 2012, pp. 844–849.

[9] Y. Cheng, Z.-y. Wang, J. Ma, J.-j. Wu, S.-z. Mei, and J.-c. Ren, "Efficient revocation in ciphertext-policy attribute-based encryption based cryptographic cloud storage," *Journal of Zhejiang University SCIENCE C*, vol. 14, no. 2, pp. 85–97, 2012. [Online]. Available: http://dx.doi.org/10.1631/jzus.C1200240

[10] S. Jahid and N. Borisov, "Piratte: Proxy-based immediate revocation of attribute-based encryption," *arXiv preprint arXiv:1208.4877*, 2012.

[11] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 261–270. [Online]. Available: http://doi.acm.org/10.1145/1755688.1755720

[12] G. Wang, Q. Liu, and J. Wu, "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 735–737.

[13] G. Wang, Q. Liu, J. Wu, and M. Guo, "Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers," *Computers & Security*, vol. 30, no. 5, pp. 320 – 331, 2011, advances in network and system security.

[14] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Advances in Cryptology—EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, vol. 3494. Berlin: Springer-Verlag, 2005, pp. 440–456, available at http://www.cs.stanford.edu/~xb/eurocrypt05a/.