Attribute-Based Content Distribution with Hidden Policy

Shucheng Yu†, Kui Ren⋆, and Wenjing Lou†

†Department of ECE, Worcester Polytechnic Institute, MA 01609
{yscheng, wjlou}@wpi.edu
⋆Department of ECE, Illinois Institute of Technology, IL 60616
kren@ece.iit.edu

***Abstract*—Access control in content distribution networks (CDNs) is a long-standing problem and has attracted extensive research. Traditional centralized access control approaches, such as reference monitor based approach, do not suit for CDNs as such networks are of large scale and geographically distributed in nature. Current CDNs usually resort to cryptographic-based distributed approaches for better fulfilling the goal of access control. Hence, it is highly critical to design and adapt appropriate cryptographic primitives for such purpose. In this paper, we propose a novel distributed access control approach for CDNs by exploiting a new cryptographic primitive called Ciphertext Policy Attributed-Based Encryption (CP-ABE). Our approach provides flexible yet fine-grained access control (per file level) so that the contents are available only to the authorized users. We further consider the protection of user privacy and enhance the current design of CP-ABE so that not only the contents themselves but also the access policies, which could lead to the revelation of sensitive user information, are well protected.**

I. Introduction

The emergence of content distribution networks (CDNs) has greatly bridged the gap between the content publishers and the end users. A typical CDN often consists of a large number of nodes deployed in geographically distributed locations across the global Internet. To optimize the delivery process, a CDN usually replicates content among its nodes. End users' requests for content are then algorithmically directed to nodes that meet certain predefined criteria such as minimal delay, minimal number of hops, etc. Among other flavors, non-profitable CDNs [1] exploit the bandwidth of volunteers to balance the workload among network nodes in contrast to the commercial CDNs based on dedicated servers. When such a model makes content publish and access more efficient than ever, it also introduces particular challenges for access control of private contents restricted to only authorized users. Because CDN nodes are highly distributed, it is not possible to employ the traditional centralized access control approaches, such as reference monitor approach [2]. And the fact that individual CDN nodes themselves may not be trustable makes it even harder to enforce reliable access control.

Current solutions [3]–[5] usually follow the cryptographic-based approach where the contents are encrypted before being stored in CDNs, and the decryption keys are distributed only to respective authorized users. In the cryptographic-based approach, flexibility and granularity of content access control heavily relies on the underlying cryptographic primitives being used. Solutions based on conventional pairwise key or group key primitives repeatedly exhibit difficulties in resolving the tension between the complexity of key management and granularity of access control. For example, in [4], to achieve per file access control, a symmetric key and an asymmetric key-pair are generated for each new file created. And the corresponding decryption keys are then distributed to respective authorized users. Obviously, the number of keys that has to be managed by each user is linear to that of the total files. While such an approach can achieve fine grained access control, its applicability in large scale is poor given the high complexity of key management. In [3], files are categorized into multiple $filegroup$, and the files of the same filegroup are all protected with the same key. This approach reduces the key management complexity but can only offer coarse-grained access control as it requires filegroups to be relatively static and their number to be small for avoiding heavy communication load for key requests due to filegroup dynamics. Hence, it remains to find a solution which provides both fine-grained access control and lightweight key management simultaneously.

In this paper, we propose a PKC-based solution aiming at resolving the above problem. To provide fine-grained access control, we explore a novel cryptographic primitive called ciphertext policy attribute-based encryption (CP-ABE). In our solution, each user is associated with a set of attributes based on which the user's secret key is generated. Files are encrypted under CP-ABE such that only those whose attributes match the access policy are able to decrypt it. Using CP-ABE, our protocol is able to provide not only per file grained access control, but also sophisticated access control semantics such as "$(manager \wedge (work\ experience > 5y) \wedge know\ Spanish) \vee ((position > manager) \wedge HR\ department)$". Furthermore, key management in the proposed solution is kept simple as users are only required to keep track of a single system-wide secret key information. To this end, a $key\ chain$ is designed to record the evolution of the system secret so that users can always get re-synchronized even after being off-line for a long period. Another salient feature is that our solution also protects user privacy by hiding files' access policies, which is known only to the file owner himself.

The rest of this paper is organized as follows. Section II

introduces the required preliminaries. Section III presents our scheme. Section IV briefly analyzes security and efficiency performance of our protocol. Section V concludes this paper.

II. Preliminaries

This section briefly introduces CP-ABE as well as the user revocation strategy used by our protocol.

A. Ciphertext Policy Attribute-Based Encryption

In CP-ABE [6], each user is associated with a set of attributes and her secret key is generated based on these attributes. When encrypting a message, the encryptor specifies the threshold access structure for her interested attributes. Message is then encrypted based on this access structure such that only those whose attributes satisfy the access structure can decrypt it. Unintended users, however, are not able to decrypt the cihpertext even if they collude. In current CP-ABE schemes, the access structure is sent in plaintext. A CP-ABE scheme consists of four algorithms:

Setup This algorithm takes a security parameter κ as input and generates the public key *PK* and a system master secret key *MK*. *PK* is used for encryption. *MK* is used to generate user secret keys[1]. It is known only to the central authority.

Encrypt This algorithm takes as input the public key *PK*, a message *M*, and an access structure *P*. It outputs the cihphertext *CT*.

KeyGen This algorithm takes as input a set of attributes *S* associated with the user and output a secret key *SK* that identifies with *S*.

Decrypt This algorithm takes as input the ciphertext *CT* and a secret key *SK* for an attributes set *S*. If only *S* satisfies the access structure, does it return the message *M*.

We refer to [6], [7] for more details.

B. Member Revocation

Member revocation is an important yet difficult issue in identity-based encryption and related schemes. In current construction of CP-ABE [6], member revocation is realized by associating each user's secret key SK with an expiration date, say X. Message is encrypted on some date Y such that the user are able to decrypt if only $X \geq Y$. This solution has a salient property that the access privilege of any user is automatically revoked after the expiration date. One drawback of this approach, on the other hand, is that it alone is not able to deal with early revocation (i.e., revoke the user before his key expires) in case of malicious behavior detected or at the user's will. To enable early user revocation, we need to update the system master secret key SK[2] for the remaining users while preventing the leaving user from being updated. As future content is encrypted under the new PK (and hence SK), the leaving users is not able to decrypt.

In the following sections, we use a symbol K_A to represent the CP-ABE public key and system master secret key. When we say "update K_A", we actually mean "update both the system master secret key and the public key". Encrypting using CP-ABE is simply denoted by encrypting using K_A. Figure 1 lists our definition of main notations used in this paper.

[1] *MK* is also used when generating *PK*.

[2] *PK* is updated accordingly as well.

notation	meaning
PK	CP-ABE public key.
SK	CP-ABE user secret key.
MK	CP-ABE system master secret key which is used to generate both *PK* and *SK*.
P,p	Access policy/structure.
K_A	A notation to represent *PK* and *MK*.
Δ	Incremental of K_A
ck	*Content encrypt key* which is used to encrypt files.
kek	*File-lockbox key* which is used to protect each file's content encrypt key *ck*.
$\{m\}_k$	Encrypt message *m* using asymmetric key *k*.
$[m]_k$	Encrypt message *m* using symmetric key *k*.
kek_ab	*kek* of version *b* for access policy *a*.
KA_i	K_A of version *i*.

Fig. 1. Main notations and their meanings

III. Our Construction

This section introduces our design. We start with introducing models and goals used in our design. Next we present our key management scheme independently since it is critical to our design. Then we describe the execution of our protocol. Finally, we give an enhanced design in which access policy of some critical content can be hidden at the content publisher's choice.

A. Models and Goals

Our network model, security goals, and adversary model are defined as follows.

Network Model Three parties are involved in our protocol:the content provider, CDN nodes, and end users. The content provider is the party who provides various kinds of content services such as file download, multimedia broadcast, and etc. They could be commercial or non-commercial parties. Each of them has at least the same computational power than a modern commercial server. CDN nodes are the parties who are responsible for efficiently delivering contents to content consumers. They could be either commercial servers or volunteers. End users are parties who consume contents provided by CDNs. They each has at least the same computational power than a modern PC. Network connection is assumed to be broadband. We do not make security assumptions on communication channels.

Security Goals The main security goal of our protocol is to prevent content from being accessed by unauthorized users. In particular, revoked users could not be allowed to access contents published after they leave. Content integrity is not our interest though it is another important security requirement in CDNs. We resort to some existing techniques to address it.

Adversary Model In our protocol, adversaries could be unauthorized users or malicious CDN nodes. The main goal of the adversaries is to get access to contents that they are not authorized to. They may work independently or collude.

B. Key Management Scheme

To fulfill the task of fine-grained access control, our scheme relies on the following key management mechanisms.

Hierarchical Key Our protocol encrypts each file[3] in a hierarchical way as follows:

$$\{kek\}_{K_A}, \quad [ck]_{kek}, \quad [file]_{ck}$$

In this three layer hierarchy, each file is encrypted with a unique content encrypt key ck. Each ck is then encrypted with the $file{-}lockbox$ key [3] kek (both ck and kek are symmetric keys). kek is finally encrypted using CP-ABE such that only those having the intended attributes are able to decrypt it.

If there were no member revocation, ck's for files under the same access policy could be encrypted with the same kek. However, the following two factors should be considered in practice: On the one hand, when creating new files, system sometime needs to update kek to prevent them from being accessed by revoked users. On the other hand, to avoid frequent re-encrypting ck's of files, we have to adopt a methodology called $lazy\ revocation$ [3]. Under this methodology, member revocation does not require immediate re-encrypting ck's of old files, following the logic that the revoked member may have already accessed the old files accessible to her. This causes the fact that ck's of old files and new files may be encrypted under different kek's. To efficiently manage kek's under the same access policy, we adopt the $key\ regression$ technique [5]. Using this technique, users are able to derive old keys from new keys, while only the content provider herself is able to calculate new keys given old keys. In our key hierarchy, we only encrypt the latest kek with CP-ABE for each access policy.

The advantages of using our key hierarchy can be summarized as follows: First, the number of keys each user needs to keep track of is minimized. As shown above, each user only needs to keep her CP-ABE secret key. Second, the vulnerability to known plaintext and known ciphertext attacks is further decreased because each file is encrypted with its unique ck. Third, it is computationally efficient since K_A and ck are not directly correlated: On the one hand, K_A update does not require re-encrypting ck's; On the other hand, changing ck does not involve expensive CP-ABE operations.

Key Chain As mentioned in Section II, CP-ABE supports member revocation by broadcasting a K_A update message. However, using broadcast not only causes heavy communication load in large scale systems, but also requires users to be always online. Those whose secret keys are not updated in time have to contact the content provider individually afterward, which turns out to cause extra communication load and demand the content provider to be always online. It is therefore desirable to adopt a technique such that users can get updated without contacting the content provider even after they have been off-line for a long period.

[3]For convenience of expression, we use a file distribution system as an example to describe our protocol for CDNs.

To address this problem, we employ a $key\ chain$ to record the update history of K_A as shown in Figure 2. This key chain is stored in a public directory on the CDN nodes. Whenever users access the files on the CDN nodes, they update their secret keys to the correct version with the help of this key chain. When the content provider updates K_A, she always updates the key chain files on the CDN nodes in a timely manner.

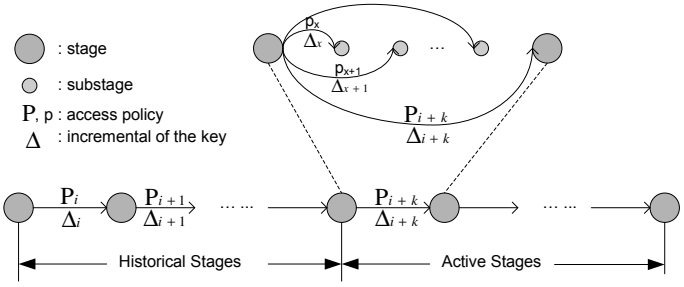

Fig. 2. Key Chain for K_A

In a key chain, we define $stages$ for K_A. On each stage, we record the incremental of K_A (denoted by Δ) between the current stage and the last stage as well as the access policy (denoted by P or p) under which this incremental is encrypted. Between two consecutive stages, we also define $substages$ which record the change history of the key in more detail. K_A on any substage can be derived from the immediate previous stage via one step as shown in the Figure 2.

We also define $active\ stages$ and $historical\ stages$. The term $active\ stages$ refers to the range inside of which K_A's are active, i.e., existing files are encrypted under some of these K_A's . The definition of $historical\ stages$ is similar except that the K_A's in historical stages are obsolete. The content provider use historical stages to help the users who have not updated their secret key for a long period to update their secret keys. In historical stages, substages are removed. We use the term $length$ to represent the number of stages in active stages or historical stages. The lengths of active stages and historical stages are fixed except for protocol setup phrase. Usually, the length of active stages is short while that of historical stages is long. For example, if one stage represents a natural day, we can create a key chain having active stages for 10 days and historical stages for three months.

The starting points of active stages and historical stages slide forward by one stage each time. If we assume stage to be a natural day, the starting points change everyday. When the starting points is sliding forward, three operations are executed on the key chain file: First, files encrypted under K_A's between the oldest and the second oldest active stages are updated with the latest K_A. Next, remove the substages between the oldest and the second oldest active stages and set the second oldest active stage as the new starting point of active stages. Finally, set the second oldest historical stage as the new starting point of historical stages and remove the oldest one. Because the "window" of the key chain keeps sliding forward, users who

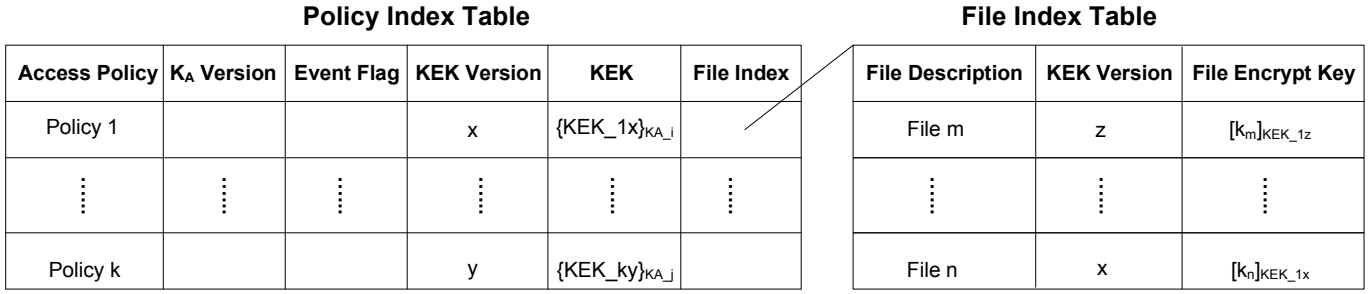

Fig. 3. Index Table for Private File Management

have not accessed the CDN for extremely long period may lose synchronization. In this case, they need to contact the content provider to get re-synchronized.

Another concern is how to create a substage. Ideally, once a member revocation event occurs, the content provider should update K_A and create a substage. However, this way may cause frequent update to the key chain if the member revocation event happens very frequently. The key chain file may expand to be extremely large as well. An appropriate way is to aggregate member revocation events and update the key chain in a reasonable interval, e.g., update the key chain every 30 minutes if any event happens. Once a new substage is created, system records the event such that new files can use this record to decide if they need to be encrypted under the latest K_A. We will discuss it in detail in the next section.

C. Protocol Description

This part describes the execution of the protocol. In particular, we discuss the protocol setup process and how the protocol reacts under events such as file access, file create, file delete, member join, member revocation. Before introducing the protocol itself, we discuss an important data structure of our system called $index\ table$.

Index Table As shown in Figure 3, this data structure contains two type of tables: $policy\ index\ table$ and $file\ index\ tables$. The system has a unique policy index table. This table records all the access policies used by the files. Because the number of access policies used could not be larger than the number of the files, the size of this table is at the most comparable to that of the system directory file. Each item in this table has six fields: policy description, K_A version, event flag, KEK version, KEK and file index. K_A version records the version number of K_A (version number is represented by stage and substage) used by current access policy. The event flag field records if any new subtage has been created since the latest upload of file under current access policy. Every time when a new substage is created, system marks the event flag field for each access policy. KEK version field records the latest kek's version for current access policy. KEK field records the corresponding credential of the latest version kek. File index field records the link to current access policy's $file\ index\ table$, which remembers all the files encrypted under current policy. Each item of the file index table has two columns: file description and the credential for current file's ck.

Now we discuss the execution of the protocol.

Protocol Setup Before providing content service, the content provider runs the setup algorithm to configure the system. This setup algorithm is as follows:

• First, it defines a set of attributes for users and runs the CP-ABE setup algorithm.

• Next, it creates empty index tables and a key chain file containing a start stage. Then, it uploads these files to CDN nodes.

• Now the content provider is ready to serve the users.

File Access When a file is being accessed, system first gets the access policy of this file and then looks up the policy index table for this policy. Using the content contained in the policy index table, the user can decrypt the kek if she owns required attributes. Then the user reads the KEK version field of this file from file index table. If KEK version is older than that in the policy index table, the user derives the correct version kek from the one she decrypted using key regression method. With correct kek, the user can decrypt ck and get access to the file.

File Create When a file is being uploaded, system first generates a ck and encrypts this file with this ck. Then, system checks the policy index table to see if the access policy of this file has already been recorded. If not, system does the following: (1)create a new kek and encrypt it with the latest K_A under the file's access policy; (2) create a new item in the policy index table and update all the fields of this new item. In particular, the event flag field is set to unmarked status; (3) create an empty file index table and link it to the file index field in policy index table; (4) encrypt current file's ck with kek and insert the credential to the file index table together with other information. If the file's policy is already in the policy index table, system first checks the event flag field of current policy. If its status is unmarked, system just creates a new item and inserts it to the file index table for current file. Otherwise, system does the following: (1)generate a new kek based on current kek; (2)encrypt the new kek with the latest K_A; (3) updates other fields for this access policy accordingly; (4)encrypt the new file's ck with the new kek and insert it to the file index table together with other information. Uploading files in batch is more efficient than doing that one by one.

File Delete To delete a file, system just removes the corresponding item from the file index table. If current file is the last one in the file index table, system also deletes the corresponding item from the policy index table.

Member Join When a new user is to join, she contacts the content provider to obtain her attributes and secret key. The content provider gives this new user her secret key on the oldest active stage. The user can calculate her keys on the remaining stages by herself. In this paper, we assume the new user can contact the content provider via an out-of-band channel and run the CP-ABE algorithm correctly.

Member Revocation When our protocol is to revoke a user, it just records the event. Member revocation events are aggregated and processed in batch. When processing the member revocation events, our protocol updates K_A, creates a new substage, and sets the event flag of all the items in policy index table as we discussed previously.

D. Access Control with Hidden Policy

This section considers applications in which access policies for some files should be hidden. Here is an example: Company C wants to offer a promotion to consumers of particular attributes. The promotion code file is published on its server and encrypted such that only the intended consumers can decrypt it. If the access policy is disclosed to others, they may know to whom the promotion is aimed and thus the scale of this activity. To prevent competing companies from knowing this information, company C wants to hides the promotion code file's access policy. Similar examples are not hard to find in our life.

Actually, if we have a CP-ABE scheme that supports hidden policy, we can embed aforementioned functionality in our protocol: In the policy index table, we just replace the policy description field with a pseudo policy ID while keeping other fields the same as before. Because the new CP-ABE algorithm can decrypt ciphertext without knowing the access policy, it can decrypt kek correctly and thus get access to the file. Unfortunately, current constructions of CP-ABE [6], [7] do not support hidden policy.

In this section, we will re-design [7] to support hidden policy in CP-ABE. For brevity, we do not present the original scheme in this paper. We refer to the original paper for details.

Security Goals Intuitively, our security goal is to prevent users, intended or not, from knowing under which access policy the message is encrypted. They should not be able to obtain this information even if they collude.

Our scheme achieves hidden policy based on the Symmetric External Diffie-Hellman (SXDH) [8] assumption between paired elliptic curve groups.

Definition (SXDH) We say that the SXDH assumption holds if, given values $y, y_1, y_2, y_3 \in \mathbb{G}_1$, it is not computationally feasible to decide if there is an integer $a \in \mathbb{Z}_p$ such that $y_1 = y^a$ and $y_3 = y_2^a$, i.e., $\mathbb{G}_1$ is a DDH-hard group. The same requirement must hold for $\mathbb{G}_2$, i.e., it is also a DDH-hard group.

CP-ABE with Hidden Policy We follow the same definition on the notations as [7]. For completeness, we present them here: The set of attributes are defined as $N := 1, \ldots, n$ for some natural number n. Attribute i and their negations $\neg i$ are referred to as $literals$. Let I denote the set of attributes that are needed for decryption. The scheme considers access structures that consist of a single AND gate whose inputs are literals, denoted by $\bigwedge_{i \in I} \underline{i}$, where every $\underline{i}$ is a literal (i.e., i or $\neg i$).

SETUP. This algorithm selects bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p with generator g_1 and g_2 respectively. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is defined on them. Next, it chooses random exponents $y, t_1, \ldots, t_{2n} \in \mathbb{Z}_p$. The public key is published as:

$$PK = (e, g_1, g_2, Y, T_1, \ldots, T_{2n})$$

where $Y = e(g_1, g_2)^y$, $\forall i \in \mathbb{Z}_{2n} : \ T_i = g_1^{t_i}$. The master secret key is $MK = (y, t_1, \ldots, t_{2n})$.

In our construction, each attribution only has two occurrences: positive and negative. $don't\ care$ element is discarded, while it is a key element in the original construction.

ENCRYPT. Given a message $M \in \mathbb{G}_T$ and an AND gate $W = \bigwedge_{i \in I} \underline{i}$, the ciphertext is output as $CT = (\tilde{C}, \hat{C}, \{C_{i,0}, C_{i,1} | i \in N\})$, where $\tilde{C} = M \cdot Y^s$, $\hat{C} = g^s$, and s is a random number in $\mathbb{Z}_p$.

For each $i \in I$, $C_{i,0}$ and $C_{i,1}$ are computed as follows.

- if $\underline{i} = i$, $C_{i,0} = T_i^s$, $C_{i,1} = T_{n+i}^x$.
- if $\underline{i} = \neg i$, $C_{i,0} = T_i^x$, $C_{i,1} = T_{n+i}^s$.

x is a random number in $\mathbb{Z}_p$.

For each $i \notin I$, $C_{i,0} = T_i^s$ and $C_{i,1} = T_{n+i}^s$.

KEYGEN. Let S denote the input attribute set. Every $i \notin S$ is considered a negative attribute. The secret key is defined as $SK = (\hat{D}, \{D_i | i \in N\})$, where $\hat{D} = g_2^{y-r}$, $r = \sum_{i=1}^n r_i$, r_i is randomly selected from $\mathbb{Z}_p$. For each $i \in N$, $D_i = g_2^{\frac{r_i}{t_i}}$ if $i \in S$; otherwise, $D_i = g_2^{\frac{r_i}{t_{n+i}}}$.

DECRYPT. Suppose the input ciphertext is of form $CT = (\tilde{C}, \hat{C}, \{C_{i,0}, C_{i,1} | i \in N\})$. Let $SK = (\hat{D}, \{D_i | i \in N\})$. For each $i \in N$, if the user's attribute is positive, then

$$F_i = e(C_{i,0}, D_i) = e(g_1^{t_i \cdot s}, g_2^{\frac{r_i}{t_i}} = e(g_1, g_2)^{r_i \cdot s})$$

If the user's attribute is negative, then

$$F_i = e(C_{i,1}, D_i) = e(g_1^{t_{n+i} \cdot s}, g_2^{\frac{r_i}{t_{n+i}}} = e(g_1, g_2)^{r_i \cdot s})$$

Decrypt finishes as follows: $M = \frac{\tilde{C}}{\hat{Y}^s} = \frac{\tilde{C}}{e(g_1, g_2)^{y \cdot s}}$, where

$$e(g_1, g_2)^{y \cdot s} = e(g_1^s, g_2^{y-r}) \cdot e(g_1, g_2)^{r \cdot s} = e(\hat{C}, \hat{D}) \cdot \prod_{i=1}^{n} F_i.$$

Above equations demonstrate how an intended user can decrypt the ciphertext. If the user is not the intended recipient, there is at least one attribute for which the user gets F_i with the form $e(g_1, g_2)^{r_i \cdot x}$. Therefore, she can not calculate $e(g_1, g_2)^{y \cdot s}$ as shown in the above equation.

Security Intuition As is shown above, ciphertext in our construction does not include the access policy. In decrypt algorithm, the user uses all her attributes to decrypt the ciphertext. If the user's i^{th} attribute is positive while $C_{i,0}$ has the form T_i^x, this user can not decrypt the ciphertext. However, because the user can not distinguish between T_i^x and T_i^s according to SXDH assumption, she is not able to know which attributes are desired by the encryptor. Therefore, she can not derive any information about the access policy. For the same reason, an intended user only knows if she can decrypt the ciphertext while not knowing which attributes grant her the access. Therefore, the access policy is hidden to all the users.

IV. Discussion

A. Security Analysis

In our protocol, each file is encrypted by a three-layered hierarchical key. To access a file, the adversary first needs to break the CP-ABE scheme to harvest kek, which is hard since current CP-ABE [7] is proven to be secure even under CCA-2 attacks. Moreover, known plaintext and known ciphertext attacks to ck's and files are difficult since each of them is encrypted under a unique random key. Our enhanced CP-ABE with hidden policy is secure under SXDH assumption as shown above.

B. Performance Analysis

The main concern in terms of performance of our scheme is the efficient implementation of file encryption and user revocation using CP-ABE. In this paper, we introduce the hierarchical key and an index table to reduce the number of the expensive CP-ABE operations. With these techniques, file re-encryption does not involve CP-ABE encryption operations. Encryption of files under the same access policy just involves one CP-ABE encryption in total. To facilitate user revocation, we attach an expiration date to each user's secret key. In this way, regular users can be automatically revoked after the expiration date. To further support early revocation, we update the system master secret key using a key chain. For this purpose, we can define two attributes for each bit of the user ID as is proposed by [9], [10]. According to the experimental result in [10], the average message complexity for multiple user revocation in this kind of solution is $O(logM)$, where M is the total number of users. The constant factor is about 3.5.

C. Related Work

Using ABE, Traynor et. al. [11] proposed a conditional access scheme for massive-scale systems. This scheme is aimed for massive-scale content distribution systems where content encryption keys are frequently changed. The authors proposed a novel tired construction in which users are divided into groups. Each group is assigned a group attribute under which the content encrypt key is encrypted. When this construction improves the performance of key distribution, the inherent flexibility of ABE is greatly sacrificed since it mandatorily divide the user access privilege by the group. In addition, as this paper is based on a previous work on threshold ABE [12], which is vulnerable to collusion attacks as is addressed in the paper, its security strength is weakened.

V. Conclusion

In this paper, we propose a PKC-based distributed access control protocol for CDNs. To provide fine-grained access control, we explore a novel cryptographic primitive called CP-ABE. Each user is associated with a set of attributes. Each file is then encrypted under CP-ABE such that only those whose attributes match the access policy are able to decrypt it. Besides per file grained access control, sophisticated access control semantics can be achieved in our protocol. Our protocol is also design with key management efficiency in mind. In our protocol, users are only required to keep track of a single system-wide secret key information. To further protect user privacy, our protocol also provides an option for content providers to encrypt files with hidden policy. An important future work is to implement our protocol and evaluate its practical performance.

Acknowledgment

This work was supported in part by the US National Science Foundation under grants CNS-0626601, CNS-0716306, and CNS-0831963.

References

[1] MU, "Existing cdns." [Online]. Available: http://www.cs.mu.oz.au/ apathan/CDNs.html

[2] J.P.Anderson, "Computer security planning study," *Technical Report 73-51, Air Force Electronic System Division*, 1972.

[3] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *USENIX'02*, San Francisco, CA, Dec. 2002.

[4] A. Harrington and C. Jensen, "Cryptographic access control in a distributed file system," in *SACMAT '03*. New York, NY, USA: ACM, 2003, pp. 158–165.

[5] K. Fu, "Integrity and access control in untrusted content distribution networks," Ph.D. dissertation, MIT, September 2005. [Online]. Available: http://prisms.cs.umass.edu/ kevinfu/papers/fu-phd-thesis.pdf

[6] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 321–334.

[7] L. Cheung and C. Newport, "Provably secure ciphertext policy abe," in *CCS '07*. New York, NY, USA: ACM, 2007, pp. 456–465.

[8] L. Ballard, M. Green, B. de Medeiros, and F. Monrose, "Correlation-resistant storage via keyword-searchable encryption," Cryptology ePrint Archive, Report 2005/417, Nov. 2005, http://eprint.iacr.org/2005/417.

[9] S. Yu, K. Ren, and W. Lou, "Attribute-based on-demand multicast group setup with membership anonymity," in *SecureComm'08*, Istanbul, Turkey, Sep. 2008.

[10] L. Cheung, J. A. Cooley, R. Khazan, and C. Newport, "Collusion-resistant group key management using attribute-based encryption," Cryptology ePrint Archive, Report 2007/161, 2007, http://eprint.iacr.org/.

[11] P. Traynor, K. Butler, W. Enck, and P. McDaniel, "Realizing massive-scale conditional access systems through attribute-based cryptosystems," in *NDSS'08*, San Diego, CA, 2008.

[12] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters, "Secure attribute-based systems," in *CCS'06*, Alexandria, VA, Oct-Nov. 2006.