Enabling Ciphertext Deduplication for Secure Cloud Storage and Access Control

Heyi Tang
Tsinghua University
tangheyi.09@gmail.com

Yong Cui
Tsinghua University
cuiyong@tsinghua.edu.cn

Chaowen Guan
University at Buffalo
chaoweng@buffalo.edu

Jianping Wu
Tsinghua University
jianping@cernet.edu.cn

Jian Weng
Jinan University
cryptjweng@gmail.com

Kui Ren
University at Buffalo
kuiren@gmail.com

ABSTRACT

To secure cloud storage and enforce access control, data encryption has become essential, given the ever increasing cyber threat everywhere. Attribute-based Encryption (ABE) crypto systems are widely considered as a promising solution under such a context for its security strength, scalability and control flexibility. One major challenge, however, for applying ABE-based techniques in real world applications is its high overhead in various aspects. In this research, we are particularly concerned with the storage size expansion in existing ABE schemes. This combined with the vast-size nature of the cloud data poses an enormous challenge to the effective usage of the cloud data storage space and affects the utility of data deduplication. Normally, data deduplication is carried out based on identifying similar and even identical contents both within and between data files, however, these patterns will be destroyed after performing data encryption using any semantically secure encryption scheme including ABE. In this research, we focus on ciphertexts deduplication under ABE, which to our best knowledge is the first of such an effort. Our fundamental observation stems from the structure of ABE ciphertexts and the possible similarities among different access structures. We show how to design a secure ciphertext deduplication scheme based on a classical CP-ABE scheme by innovatively modifying the construction with a recursive algorithm, eliminating the duplicated secrets and adding additional randomness to some certain ciphertext. We then give a detailed analysis on the proposed scheme with respect to both efficiency and security. To thoroughly assess the performance of the proposed scheme, we also implement a prototype system and conduct comprehensive experiments, which shows that our ciphertext reduplication scheme could reduce up to 80% computation and storage cost in the best case.

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

DOI: http://dx.doi.org/10.1145/2897845.2897846

1. INTRODUCTION

With the rapid growth of Internet users and amount of information, the benefits of cloud computing services become more and more tempting. However, the wide use of cloud poses new threats to security and privacy. Cloud service providers might not be trusted for financial incentives, which indicates the concern that personal data will be compromised. For privacy and security reasons, one method is to outsource data in encrypted form.

One disadvantage of encrypting data is that it severely limits the ability of data owners to selectively share their encrypted data at a fine-grained level. To address this problem, Sahai and Waters [12][10] introduced the notion of attribute-based encryption (ABE). There are two kinds of ABE schemes: key-policy ABE (KP-ABE) [12] and ciphertext-policy ABE (CP-ABE) [10]. In a CP-ABE scheme, every ciphertext is associated with an access policy on attributes, and every user's private key is associated with a set of attributes. A user is able to decrypt a ciphertext only if the set of attributes associated with his/her private key satisfies the access policy associated with the ciphertext.

One of the main drawbacks of existing CP-ABE schemes is that they generate big-size ciphertexts (more details about overhead of existing CP-ABE schemes can be found in section 3.4). Our experiment(in Section 3.4) shows that when we deal with many small files, the ciphertext size will lead to a significant amount of storage. Moreover, the decryption is expensive due to pairing operations[17], and the number of required pairing operations grows with the complexity of the access policy. Thus the computation cost will also be very large when there are many attributes. Some papers tried to solve this problem by transferring some heavy computation to the cloud side[13][14]. But the overall computation overhead is not reduced. Notice that the computation overhead is related to the attributes, and with high probability that there exist duplicated attributes.

To achieve cost-effective storage, cloud service providers will perform deduplication in real life application. The traditional schemes for deduplication are hashing and chunking.[20][25]. Roughly speaking, the duplicated data chunks have the same hash value, therefore, only one copy needs to be stored on the cloud. In a word, these deduplication techniques are carried out based on identifying similar and even identical contents both within and between files. Unfortunately, these patterns will be destroyed after performing encryption over the stored data, because the nature of en-

cryption is to make all content appear random. There exist some works[8][11][24] aiming to address this problem, but none of them provide deduplication and fine-grained access control simultaneously.

In this paper, we focus on the deduplication techniques on ciphertexts under ABE schemes, which to our best knowledge is the first of such an effort. Our contributions are summarized as follows:

- We propose a general ciphertext deduplication technique in access tree model. In a very high level, instead of generating the secrets from the root to leaf, our proposed scheme first generates the secrets of the leaves, then builds the secrets on the non-leaf nodes in a bottom-up manner. By doing so, we can use the same secret on leaves with the same attribute. We also design a one-way function so as to make the secrets satisfy the reconstruction property. To compare the non-leaf nodes' structure and eliminate the duplications, we use Merkle hash tree method and design a tree rebuild algorithm.
- We employ the proposed deduplicaiton techniques to give a duplication-free CP-ABE scheme based on an existing CP-ABE scheme[27]. The ciphertext is carefully designed so that the data with similar policies can share some ciphertexts. We also give detailed analysis showing that the resulting scheme is secure as long as the security of the original CP-ABE scheme is guaranteed. Although our proposed scheme only focuses on the deduplication of ABE ciphertexts, we will show how to combine the scheme with the existing deduplication schemes on files. Both the duplicate encrypted files and duplicate ABE ciphertexts can be eliminated in one system.
- We evaluate the performance of our proposed scheme. Concretely, we first give theoretic analysis of how much the scheme reduce the computation and storage cost. Then we implement a prototype of our scheme and conduct intensive experiments on it. The results demonstrate that the computation overhead and storage consumption is reduced by 80% compared with the original scheme in the best case. In the worst case, the additional cost of our scheme is less than 10%, which will become negligible when the number of ciphertexts dramatically grows.

The rest of this paper is organized as follows. Related work is given in section 2. Section 3 describes the background of the paper. We give the secret reusing technique of our scheme in section 4. In section 5, a detailed construction based on the existing schemes is presented. Security issues of our scheme is analyzed in section 6. Section 7 describes the implementation of our prototype. The evaluation details and the result are shown in section 8. Finally we summarize this paper in section 9.

2. RELATED WORK

Deduplication of encrypted data is an active topic nowadays. In traditional scenarios where the data are not stored in encrypted form, there exist many deduplication schemes to improve the storage efficiency. The main idea behind those schemes is to split the files into chunks and marking them with their hash values. The performance of different chunking algorithms are shown in [18]. But when data is encrypted before being stored, chunking and hashing become difficult because the contents appear random after encryption. Note that encrypting the data of the same contents by different keys will probably give different ciphertexts. To deduplicate this kind of encrypted data, Douceur et al. [11] introduced the notion of convergent encryption, which was extended in [24]. The main idea is using the hashes of file/chunk as the key for encryption. But this kind of methods are vulnerable to brute-force attack, as the attacker could easily recover the files in a knownlist. To avoid this kind of attack, a method depending on an additional independent server was proposed in [8].

Access control in the cloud is more complicated. For privacy and security issues, the data owner will encrypt his data before uploading them. However, it will severely limit the ability of data owners to selectively share their encrypted data at a fine-grained level. To address the problem of enabling flexible access control over encrypted data, Sahai and Waters introduced Attribute-based Encryption [10][12]. There are two kinds of ABE schemes:Ciphertext-Policy ABE(CP-ABE)[10] and Key-Policy ABE(KP-ABE)[12]. In KP-ABE schemes, the ciphertext is associated with a set of attributes while the user's private key is generated based on his corresponding access policy. While in CP-ABE schemes, a user's private key is associated with a set of attributes, and ciphertext is encrypted under a specified access structure. In both types of schemes, a user is able to decrypt a ciphertext only if the attributes associated with the ciphertext/private key satisfies the access policy related to his private key/ciphertext. Some existing access control systems are built based on these schemes[6][5][16][28].

Most ABE schemes[12][10] are constructed based on Bilinear Pairing[17]. Note that the pairing operation is expensive. Moreover, in those ABE schemes, the number of pairing operations during encryption/decryption phase increases linearly with the size of ciphertexts. The computational efficiency will become a bottleneck when the size of ciphertexts grows. Some papers[22][10][26] gave the detailed performance analysis of ABE schemes. To largely eliminate the computation overhead for users, One solution is to split the computation to multiple phases and move the hard part to other places. Green et al[13] introduced a method where the decryption of ABE ciphertexts are outsourced to the cloud server. Qin et al[15] further extended this method by adding a fast verification function to the outsourcd decryption. Another work aiming to reduce the encryption and key generation computational costs was done by Hohenberger and Waters [14]. In their scheme, the computation for these two algorithms are split into two phases, where the offline phase does the vast majority of work and the data owner only computes the cheap part in online phase.

3. PROBLEM FORMULATION

3.1 Preliminaries

3.1.1 Access Structure and Access Tree

Definition 1. *Access Structure[7]*:Let $\{P_1, P_2, .., P_n\}$ be a set of parties. A collection $\mathbb{A} \subset 2^{\{P_1, P_2, .., P_n\}}$ is monotone if $\forall B, C$: if $B \in A$ and $B \subset C$ then $C \in A$. An access

structure is a collection $\mathbb{A} \subset 2^{\{P_1,P_2,...,P_n\}} \cup \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, while the sets not in $\mathbb{A}$ are called the unauthorized sets.

In our paper, the parties are replaced by the attributes. An access structure $\mathbb{A}$ will contain some authorized sets of attributes. So an access policy can be defined as an access structure. Similar to [10][23][14][13], we only consider the monotone access structures. In the following chapters, by an access structure we mean a monotone access structure.

Definition 2. *Access tree* An access structure can be represented by an access tree. There are two kinds of nodes in an access tree: non-leaf node and leaf node. Each non-leaf node represents a threshold gate. A non-leaf node x has n_x children and a threshold value k_x. Specifically, when $k_x = 1$, the threshold gate is an OR gate, when $k_x = n_x$, it is an AND gate. Each leaf node is associated with an attribute, and denote it by $attr(x)$.

Definition 3. *Satisfying an access tree.* If a set of attributes A satisfies a node x, we denote it by $T_x(A) = 1$. If x is a non-leaf node, A satisfies it only if A satisfied at least k_x nodes. If x is a leaf node, A satisfies it if and only if $attr(x) \in A$. A satisifies the access tree if A satisfies the root node.

3.1.2 Ciphertext Policy Attribute Based Encryption

A ciphertext policy encryption in our settings consists of four algorithms: Setup,Encrypt,KeyGen, Decrypt.

Setup(U). The setup algorithm takes security paramerters and attribute universe description as input. It outputs the public parameters PK and a master key MK.

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

KeyGen(MK, S). The key generation algorithm takes as input the master key MK and a set of attributes S that describe the key. It outputs a private key for a specific user.

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

3.1.3 Security Model

We assume that the cloud may send the owner's data to all users. The data owners are fully trusted, while the users are can be malicious. Like the security model in [27], we assume that the adversary is able to query for any private keys that cannot decrypt the challenge ciphertext. Since our scheme just aims to provide the additional property: deduplication. The security model just follows the conventional model as below.

Setup: The setup algorithm is run, the challenger generates the global parameters and public keys ,then sends them to the adversary.

Phase 1: The adversary makes the secret key queries to the challenger. The challenger recieves the sets $S_1, S_2, ..., S_{q_1}$ and runs the KeyGen algorithm to generate corresponding secret keys.

Challenge: The adversary submits two messages of equal length M_0,M_1, then the adversary gives a challenge access structure $\mathbb{A}$ that none of the sets from phase 1 satisfies it. The challenger flips a random coin b,the message M_b and the access structure $\mathbb{A}$ are inputted to the encryption algorithm. The challenger then runs the encryption algorithm and output a ciphertext CT. The ciphertext CT is given to the adversary.

Phase 2: The adversary makes more secret key queries of the attribute sets $S_{q_1+1}...S_q$. None of the sets satisfy $\mathbb{A}$.

Guess: The adversary outputs a guess b' of b.

The advantage of an adversary is defined as

$$Pr[b' = b] - \frac{1}{2}$$

Definition 4. Our scheme is secure if all polynomial time adversaries have at most a negligible advantage in the above security game.

3.2 Access Control in the Cloud

Our ciphertext deduplication scheme is based on CP-ABE schemes. In CP-ABE schemes, the users have some attributes. The data owner defines an access policy for some data. Then he encrypts the data using the attribute public keys according to the access policy and uploads it to cloud. Data user can only decrypt the data if and only if the attributes associated with him satisfy the corresponding access policy. This kind of access control system consists of the following four types of entities.

Authority: The authority manages the attributes and initializes the system. It generates secret/public key pairs for the attributes. Then it generates the user secret keys according to a user's attributes and give the keys to the user. The authority also makes the public keys as public parameters.

Cloud Server: The cloud server stores the encrypted data uploaded by data owners. In this paper, we consider the cloud server is semi-trusted. The cloud will not corrupt data but may read and analyze it. So owners will encrypt the data before uploading.

Data Owner: For each file that he wants to upload, he defines an access policy and encrypts the file based on the access policy. Then the resulting ciphertext and the access policy are uploaded to the server.

Data User: A data user has a private key associated with the set of his own attributes. He connects to the server and download the data he wants. He can only decrypt a specific file when the attributes associated with his private key satisfies the corresponding access policy. Otherwise, he can't learn any extra information from the encrypted data.

3.3 Existing CP-ABE Constructions

There exist two construction models of CP-ABE scheme: LSSS model[7] and access tree model[9]. In this paper, we mainly focus on the improvement on the CP-ABE scheme of access tree model. Specifically, in the access tree model, the value given to a tree node is called the ***secret*** for the node. The secret on the root node is the symmetric key to encrypt the whole file. In the existing schemes, Secret-Sharing Schemes(SSS)[9] are used to divide the secret on the root node among leaves. In encryption phase, the owner generates the secrets for all leaves, then encrypts the secrets with attribute public keys. In decryption phase, a valid

user uses his attribute private keys to decrypt the necessary secrets on leaves, then reconstruct the secret on root node.

In the original access-tree-based schemes, the secrets are generated recursively. The root node r first generates the secrets of its children nodes with its own secret s_r, for each non-leaf node, it then generates the secrets of its children nodes with its own secret. For node x, assume the secret of x is s_x, the secret generation algorithm will choose a $k_x - 1$ degree polynomial $q_x(i)$, where k_x is the threshold value of x and $q_x(0) = s_x$. Denote the n_x children of x by $\{x_i\}$, and set $s_{x_i} = q(i) \forall i = 1, 2, ..., n_x$. Let S be a set of some indexes in $\{1, 2, ..., n_x\}$,$card(S) = k_x$, then we define the Lagrange coefficient $\Delta_{i,S}(t)$ for $i \in S$: $\Delta_{i,S}(t) = \prod_{j \in S, j \neq i} \frac{t-j}{i-j}$. With the coefficients, we can reconstruct s_x by $s_x = q(0) = \sum_{i \in S} \Delta_{i,S}(0)$.

The two special cases are AND gate and OR gate. When $k_x = 1$, the gate is an OR gate, and we have $q_x(i) = s_x, 1 \leq i \leq n_x$, so $s_{x_i} = s_x$, where the value of the children equals to the value of the parent. When $k_x = n_x$, the gate is an AND gate. Then it becomes $S = \{1, 2, ..., n_x\}, s_x = q_x(0) = \sum_{i=1}^{n_x} \Delta_{i,S}(0) s_i$, where all n_x secret values are necessary.

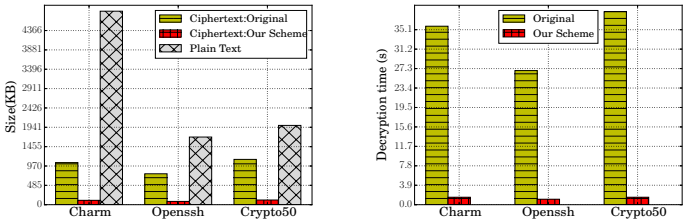
3.4 Overhead of Existing Schemes

In order to see the storage/computation overhead incurred in the original CP-ABE scheme, we run some experiment on the existing ABE scheme [27]. We choose 3 open source projects to encrypt: charm crypto library[4], openssh[3], and crypto50[1]. Those three open source projects include 211 files, 200 files and 293 files, respectively. We encrypt all the files in these projects, where each file corresponds to an access policy with only AND gates on 25 attributes. An example policy is like (GROUPID=25 AND ID=123456 AND STUDENT), where GROUPID is an 8-bit integer, ID is 16-bit integer, and STUDENT/TEACHER are two additional attributes. There are 50 different attributes in total.

The comparison of the storage cost between the plaintext and ABE ciphertext is shown in Fig.1a (The symmetric ciphertext is not included). From Fig.1a, we can see that each source code file in plaintext form is, on average, of size not more than 20KB , but each policy could introduce about more than 4KB ciphertext when using the existing CP-ABE scheme for encryption. Overall, the storage consumption incurred by the generated ciphertext is about hundreds of KBs, which implies that this cost can probably become significant when the number of files dramatically increases. The storage cost of the original scheme is significant. While in our proposed scheme, it only incurs little storage consumption because the scheme reuses the duplicate ciphertexts among files.

As for the comparison of decryption time, which is shown in Fig.1b the original scheme costs about 30 seconds for encryption when dealing with the projects with 200 files. While our scheme only costs hundreds milliseconds and will not increase if more files are added.

Note that the storage/computation overhead will grows as the number of files increases. Imagine that a business company may possess thousands, and even millions of files. Then the storage/computation overhead might become astronomical when using the existing CP-ABE scheme for access control over encrypted data. In this paper, our proposed deduplication scheme can serve as a possible method to alleviate this situation. Next, we will present the details of how to achieve deduplication over CP-ABE ciphertexts.



(a) Ciphertext Size (b) Decryption Time

Figure 1: Comparison among 3 open source projects

4. THE DEDUPLICATION TECHNIQUES

In this section, we propose the novel deduplication techniques that can be applied to access-tree-based CP-ABE schemes. Generally, instead of computing the secrets from root to leaf, we first build the secrets on leaf nodes, then use these secrets to build non-leaf nodes. When building the nodes, the nodes with the same attribute/structure could share the same secret, so we could eliminate the duplications by reusing the same secret.

4.1 Building the Secrets on Leaf Nodes

In the original schemes, the secret values for the nodes are chosen from root node to leaf nodes. Thus even two access tree nodes with the same structure will have different node values, since their values are computed according to each node's parent, respectively.

In order to eliminate the duplicate ciphertexts, the duplicate access structure should not occur multiple times. In contrast to the original scheme, we construct the secrets in a bottom-up manner, the secrets on leaf nodes are determined first and then the secret of the non-leaf nodes will be computed according to their children. That is, for each attribute, we choose a random value $r_i \in Z_p$. Then the leaf node x with the attribute $attr(x)$ will be associated with value $s_x = r_{attr(x)}$. If two leaf nodes are associated with the same attribute, then they are assigned the same values. Mathematically, $s_x = s_y$ if $attr(x) = attr(y)$. Then we can reuse the values and avoid duplications. Next, we will discuss how to build the secrets on the non-leaf nodes.

4.2 Building the Secrets on Non-leaf Nodes

A non-leaf node is a threshold gate. For simplicity, we start with AND gate. Let x be the node of AND gate, we need all n_x values of the children nodes to calculate the value of the node x, and we have $s_x = \sum_{i=1}^{n_x} \Delta_{i,S}(0) s_{x_i}$. Thus the secret value of the AND gate node is determined.

As for a general threshold gate that $k_x < n_x$, it is hard to determine the value of s_x. In the original schemes, those n_x values for the children are the evaluations of an $k_x - 1$ degree polynomial q at 1 to n_x, that is, $s_{x_1} = q(1), .., s_{x_{n_x}} = q(n_x)$. So that each k_x nodes can determine the polynomial q and compute the value $s_x = q(0)$. In our reusing scheme, since we use a bottom-to-top manner, the values of the children nodes are pre-defined. Denotes the secret value of ith child by s_{x_i}, they can be regarded as randomized. It is highly impossible that they are all in one $k_x - 1$ degree polynomial.

Our approach is that we first randomly choose a $k_x - 1$ degree polynomial $q(i)$, then let $s_x = q(0)$. We then define the other n_x values $s'_{x_i} = q(i)$. Thus we can compute s_x with k_x values in the n_x values. Since s'_{x_i} and s_{x_i} are determined respectively, it is obvious that $s'_{x_i} \neq s_{x_i}$, we should have some technique to compute s'_{x_i} from s_{x_i}. We then define

a function f as $f(s_{x_i}, a_i) = s'_{x_i}$, where a_i is the additional information. As a result, if we have k_x children, we first use f to compute s'_{x_i}, then we use s'_{x_i} to compute s_x.

Note that f should be a one-way function such that computing s_x by $f(s_{x_i}, a_i)$ is easy while finding s_{x_i} based on s'_{x_i} and a_i is hard, because the secrets among different trees are the same, if x_i corresponds to an attribute which the user doesn't have, the leakage of s_{x_i} will bring new threats. The design of f is related to the actual encryption/decryption algorithm. Hence, detailed discussion on the one-way function f will be presented in section 5.

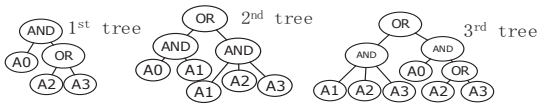
4.3 Reusing Secrets in Complete Access Trees

For the entire access tree, we want to store the relationship between a node structure and its secret value, so that a node with the same structure can reuse the secret value, and the duplicate computation/storage can be eliminated. We use Algorithm 1 to construct the secret in a recursive manner.

Algorithm 1 Secret Construct

Require: A Access Tree Node N. A global table T
Ensure: The secret value s of N.
1: Let $V = FindNodeVal(N, T)$
2: **if** V is not $NULL$ **then**
3: $s = V.secret$
4: **else**
5: **if** $N.type = ATTRIBUTE$ **then**
6: Let $s = s_{attr(N)}$, the secret of the attribute.
7: **else**
8: **for** $i = 1$ to $N.n$ **do**
9: $s_i = SecretConstruct(N.children[i], T)$
10: **end for**
11: **if** $N.k == N.n$ **then**
12: Let $S = \{1, 2, ..N.n\}$
13: Let $s = \sum_{i=1}^{N.n} \Delta_{i,S}(0) \cdot N.children[i].secret$
14: **else**
15: Choose a random $N.k - 1$ degree polynomial q.
16: Let $s = q(0)$
17: **for** $i = 1$ to $N.n$ **do**
18: Let $s'_i = q(i)$
19: Find a_i that $f(s_i, a_i) = s'_i$
20: Store a_i
21: **end for**
22: **end if**
23: **end if**
24: Create a map value V, let $V.secret = s$.
25: Add N, V pair into T
26: **end if**

Algorithm 1 shows how to construct a secret of an arbitrary access tree while eliminating the duplications. In the above description, a node is represents as N, $N.n$ denotes the count of its children, $N.k$ denotes the threshold value of the gate. $N.secret$ denotes the secret value of this node and $N.children[i]$ denotes the ith child of the node. This algorithm first checks if the secret of the node with same structure has been constructed. If so, the secret can be instantly returned, otherwise the algorithm continues. If the input structure is just an attribute, we just reuse the existing secret. Besides, the algorithm recursively computes the secrets of its children, then use the methods in section 4.2 to construct the secret. The global table T is used to avoid computing the secret of an existing structure. After



(a) Access Trees

Phases	Node structure	Secret values
Initial	A0 A1 A2 A3	r_0, r_1, r_2, r_3
1st tree	OR: A2 A3	$r_{O(2,3)}$ $=f(r_2, a_1) = f(r_3, a_2)$
	AND: A0 O(2,3)	$r_{A(0,O(2,3))}$ $=\Delta_{1,\{1,2\}}(0) \cdot r_0 + \Delta_{2,\{1,2\}}(0) \cdot r_{O(2,3)}$
2ndtree	AND: A0 A1	$r_{A(0,1)}$ $=\Delta_{1,\{1,2\}}(0) \cdot r_0 + \Delta_{2,\{1,2\}}(0) \cdot r_1$
	AND: A1 A2 A3	$r_{A(1,2,3)}$ $=\sum_{i=1}^{3} \Delta_{i,\{1,2,3\}}(0) \cdot r_i$
	OR: A(0,1) A(1,2,3)	$r_{O(A(0,1),A(1,2,3))}$ $= f(r_{A(0,1)}, a_1')=f(r_{A(1,2,3)}, a_2')$
3rdtree	OR: A(1,2,3) A(0,O(2,3))	$r_{O(A(1,2,3),A(0,O(2,3)))}$ $=f(r_{A(1,2,3)}, a_1'')=f(r_{A(0,O(2,3))}, a_2'')$

(b) Phases of Algorithm

Figure 2: Example of Secret Construction

the computation of a secret, the node/secret pair will be added to T. All the access policies the data owner uploads share the same T. Thus every kind of structure will be only calculated once, and the duplicate structures can be reused.

Fig.2 shows an example that how to store and reuse the structure. In the initial phase, only the secrets of 4 attributes are added to the table T. The 1st structure consists of an OR gate and an AND gate, During the execution of Algorithm 1, the secret of the OR gate is constructed as $r_{O(2,3)}$, then the secret of the AND gate is constructed. Two pieces of additional information(a_1, a_2) are added in this phase and two (node,secret) pair are added to T. When constructing the secret of 2nd structure, 3 pairs are added and 2 more pieces of additional information are used. As for the 3rd structure, both children of the root OR node are existing structures. Then the algorithm will reuse them, only one pair is added to T and only 2 pieces additional information are produced. Overall these 3 complicated structures need to upload 4 attribute secrets and 6 more pieces of the additional information. While the original schemes need 14 secrets on 14 leaves. On the other hand, although the table T has a certain amount of items, when uploading more files, the secrets in table T can be reused in the newly added files and fewer secrets need to be added. While the original schemes have to construct secrets for all the files.

4.4 Node Comparison

In the secret construction Algorithm 1, an important part is the $FindNodeVal$ function. This function checks whether the current node is in the table T and if so, returns the corresponding data. This function is the key to enabling the reusage of the nodes with the same structure. Below, we give a method to check the existence of a node structure.

4.4.1 Comparison via Merkle Hash Tree

A simple idea to compare two nodes' structure is to compare with every structure in the table one by one. However

this will take $O(n^2)$ time where n is the total count of the nodes. In traditional deduplication schemes, hashing was used on data chunks. In order to employ the similar method, we need to deal with the problem of how to convert a tree structure to a hash key. In our scheme, we use the idea of Merkle tree[19] to calculate the hash key. A Merkle tree is a tree where each node has a hash value. The hash values of leaves are some randomly chosen values. As for non-leaf nodes, the hashes are calculated by hashing their children. Hence, a user can compare two trees by the hash values of the root node instead of traversing the tree structure.

In our scheme, we also define a hash key to represent the structure of a node. For a leaf node, we can simply define the attribute's hash value as its hash key. As for the non-leaf nodes, where each node can be an OR gate, an AND node or a general threshold gate, we just need a hash function H, which require a set of hashes, and output one hash key.

Before running Algorithm 1, we will first run Algorithm 2, the hash key generation algorithm. This algorithm takes a node as input, if the node is an attribute, the output hash value is simply its ID. Besides, the hash functions are adopted. The children's hash values are computed first, in order to keep the hash values being the same, the hash values are sorted to keep in order. Then the node's hash value is computed with H by the node's type and children's hash. Both $N.n$ and $N.k$ are considered since different $N.k$ present different structures. The hash key can be used as the key of T to implement $FindNodeVal$.

When we implement the hash function, we can build a piece of text by linking the hashes with a special mark, then use SHA256 to compute the hash key. For example, the key of policy A1 AND A2 AND A3 could be computed by SHA256("AND-1-2-3"). Besides, we can also choose other hash functions, since the collision probability is negligible, the security of our proposed scheme will not be affected.

Algorithm 2 Hash Key Generation

Require: The Access Tree Node N;
Ensure: The hash key of N:$N.hash$
1: **if** $N.type = ATTRIBUTE$ **then**
2: $N.hash = H_{ATTR}(N.ATTR_{ID})$.
3: **else**
4: **for** $i = 1$ to $N.n$ **do**
5: $h_i = HashKeyGeneration(N.children[i])$
6: $S_h = S_h \cup h_i$
7: **end for**
8: Sort S_h to let $h_1, h_2, ...$ be ascending order.
9: $N.hash = H(N.n, N.k, h_1, h_2, ...h_{N.n})$
10: **end if**

4.4.2 Adjust the Access Tree Structure

Another problem is how to identify the access trees of equivalent boolean functions. For example, the policy ((A AND B) AND C) and the policy (A AND (B AND C)). The two policies have different tree structures but are representing an equivalent boolean function. Note that Algorithm 2 can't help distinguish these two policies. Therefore, we propose Algorithm 3 to build an general access tree that make the policies become the same. In Algorithm 3, if some children nodes have the same type with current node, this node will be removed, and the children of this node will be added as the current node's children. Then both ((A AND B) AND C) and (A AND (B AND C)) will be rebuilt as (A AND B AND C) and will get the same hash value.

The above algorithm deals with the cases that several AND gates/OR gates are continuous. There will be some more complicated cases that two policies express the same ability: (A AND B) OR (A AND C), (A AND (B OR C)). For this kind of cases, a solution is to translate all the policies into CNF/DNF formulas. Thus the structures of the policies with the same ability will be totally the same, and then generate the same hash keys.

Algorithm 3 Rebuild Access Tree

Require: The Access Tree Node N
Ensure: A new access tree node N'
if $N.type == AND$ or $N.type == OR$ **then**
Let $S = N.children$,$N'.children = \emptyset$
while $S \neq \emptyset$ **do**
Pop one node N_0 from S
if $N_0.type == N.type$ **then**
$S = S \cup N_0.children$
else
$N'.children = N'.children \cup N_0$
end if
end while
Let $N'.n = card(N'.children)$,$N'.type = N.type$
if $N.type == AND$ **then**
$N'.k = N'.n$
else if $N.type == OR$ **then**
$N'.k = 1$
end if
else
Let $N'.children = N.children$,$N'.n = N.n$,$N'.k = N.k$
end if
for $i = 1$ to $N.n$ **do**
$N'.children[i] = RebuildAccessTree(N'.children[i])$
end for

In a word, the data owner first converts the access policies to access trees, then generates the hashes of each node with Algorithm 2, finally Algorithm 1 is run so that the secrets are generated.

4.5 Discussion

4.5.1 Dealing with NOT

Note that our proposed construction only considers the monotonic access structures that the policy cannot contain NOT gate. We can extend our proposed scheme to support NOT gate. According to the previous work on ABE scheme which supports non-monotonic access structure[21], first, the non-monotonic access formula is transformed into a monotonic one with negative attributes. Then a broadcast revocation scheme is used to encrypt the secret on the leaf node with negative attribute. Since our construction focuses on the share generating procedure, we can first define the shares of all the negative attributes and use similar method to encrypt it, then we use the same procedure as in Section 4 to build the secret on root. Our scheme is also adapted to the non-monotonic ABEs.

4.5.2 Combining with Outsourced ABE

The recent outsouced ABE schemes[13][15][14] aims at shifting the expensive computation to the cloud side. Since

our scheme only change the secret building method, these approaches can be combined with our scheme. The outsouced schemes Therefore, the efficiency on the client-side can be further improved using this method.

5. THE PROPOSED CP-ABE SCHEME WITH DEDUPLICATION

In this section, we will show how to use the above techniques that enable the reusage of the secrets on nodes to construct a CP-ABE scheme with fewer ciphertexts based on the ABE scheme in [27]. One of the main differences between our proposed construction and that in [27] is that the access structures are divided into multiple small access structures. The small access structures have hash values and the corresponding ciphertexts are stored in the hash table. Thus the ciphertexts of the duplicate access structures can be reused so the duplications are eliminated. Notice that most of the ABE schemes based on the access tree model use the similar method to build secrets, by modifying the computation of the ciphertext in our algorithms, we can simply apply our techniques to other ABE schemes.

5.1 System Initialization

5.1.1 Global Setup

The setup phase is the same as the original scheme. Two multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ are chosen of the same prime order p and the bilinear map e:$\mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Choosing a generator of $\mathbb{G}$ as g, the global parameters are

$$GP = (p, g, H)$$

5.1.2 Authority Setup

Assume the system has U attributes, the authority chooses U random group elements $h_1, .., h_U \in \mathbb{G}$ for the attributes. Then it chooses random exponents α, $a \in \mathbb{Z}_p$, and publishes the public keys as $PK = g, e(g,g)^\alpha, g^a, h_1, ..h_U$. The authority sets $MSK = g^\alpha$ as the master secret key.

5.2 Key Generation

For an arbitrary user with the attribute set S, the authority chooses a random $t \in \mathbb{Z}_p$, generates the private keys for the user as follows.

$$SK = \{K = g^\alpha g^{at}, L = g^t, \forall x \in S : K_x = h_x^t\}$$

5.3 Data Encryption

The owner first chooses the attribute set $\{A_i\}$ which contains m attributes he needs. For each A_i, he chooses random variables s_i as the owner's attribute secret. Then he computes the initial ciphertexts as

$$C_i = g^{as_i} h_i^{-r_i}, D_i = g^{r_i}$$

where r_i is a random variable for decrypting. Then the owner initializes a hash table T, For every file, the owner inputs the root node of the access tree and runs Algorithm 4, the encryption algorithm. The output are some ciphertexts in hash table T and a secret. Finally, for each file, he keeps the corresponding access structure's hash key h and the value V. Let the secret $s = V.secret$, then he encrypts the file m as

$$C = m \cdot e(g,g)^{\alpha s}, C' = g^s$$

Similar to Algorithm 1, the small access structures are stored in the hash table. The hash key is the hash of a structure while the hash value contains the particular access structure, the secret and the ciphertexts.

The secret construction method in Algorithm 4 is the same as Algorithm 1. When constructing a secret for a specific access structure, the algorithm also constructs the corresponding ciphertext according to the type of access structure. If the access structure is an AND gate, in fact, it doesn't return any ciphertext, since the secret of the AND gate is simply built with the secrets of its child nodes. When the access structure is an OR gate or a general threshold gate, the additional information on each child should be added(Line 18). How to design the additional information is one important problem, we will discuss it in the following part.

Algorithm 4 Encrypt Node

Require: The access tree node N. The hash table T_c
Ensure: The ciphertexts C, the secret s

```
1: Run the Rebuild algorithm to rebuild the access tree.
2: Run the Hash algorithm to get hash value.
3: Let V = FindCT(N.hash, T_c)
4: if V is not NULL then
5:    C = V.ct, s = V.secret
6: else
7:    if N.type = ATTRIBUTE then
8:       Let x be the attribute index.
9:       C = C_x, s = s_x
10:   else
11:      for i = 1 to N.n do
12:         V_i = EncryptNode(N.children[i], T)
13:      end for
14:      if N.type = AND then
15:         Let s = sum_i^{N.n} Δ_{i,S}(0) · V_i.secret
16:      else
17:         Choose random N.k − 1 degree polynomial q
18:         Let s = q(0), C = ∅
19:         for i = 1 to N.n do
20:            Let s_i = V_i.secret, s'_i = q(i)
21:            Compute the additional ciphertext c_i by s_i and
               s'_i for decryption.
22:            C+ = {c_i}
23:         end for
24:      end if
25:   end if
26:   Create a value V, let V.secret = s,V.ct = C.
27:   Add the N.hash, V pair into T_c
28: end if
```

The owner removes the secrets from the hash table, and uploads the hash table to the server. Finally for each file, the owner uploads the ciphertext and its access structure (C, C', M) to the server.

5.4 Data Decryption

The data user downloads the encrypted files and the hash table, and each encrypted file has the hash value of its access structure. Then for each encrypted file, he runs the following algorithm, Algorithm 5 to recover the access structure secret. The decryption of the root node in Algorithm 5 is just similar to the scheme in [27]. One important modification is that we use a decrypted ciphertext hash table T_d to store the decrypted nodes.

Algorithm 5 shows how to recursively compute the necessary value on the root node. Similar to Algorithm 1,4, the

calculation depends on the type of the input access structure. If the input is an attribute(see Line 9), we can simply use the decrypting method in [27]. To compute the value of an AND gate, we need to recursively compute the values of all its children. After that, if the values of all its children are successfully computed, according to the secret construction method in section 3.3, the secret value can be simply computed with Lagrange coefficients(see Line 17). The resulting value will be $e(g,g)^{at\sum_i s_i\Delta_{i,S}(0)} = e(g,g)^{atq(0)} = e(g,g)^{ast}$.

Algorithm 5 Decrypt Node

Require: The access tree node N. The ciphertext hash table T_c. The decrypted hash table T_d.
Ensure: The decrypted ciphertext d.
1: Compute the hash of the nodes in access tree.
2: Let $d' = FindDecrypted(N.hash, T_d)$
3: **if** d' is not $NULL$ **then**
4: $d = d'$
5: **else if** The access tree is not satisfied **then**
6: $d = NOT\_SATISFIED$
7: **else**
8: $C = FindCT(N.hash, T_c)$
9: **if** $N.type = ATTRIBUTE$ **then**
10: Let x be the attribute index.
11: $d = e(C_x, L)e(D_x, K_x)$
12: **else if** $N.type = AND$ **then**
13: Let the required children set $S = \{1, 2, ..., N.n\}$
14: **for** $i = 1$ to $N.n$ **do**
15: $d_i = DecryptNode(N.children[i], T_c, T_s)$
16: **end for**
17: $d = \prod_{i=1}^{N.n} d_i^{\Delta_{i,S}(0)}$
18: **else**
19: Find k the valid children index set S
20: **for** $i \in S$ **do**
21: $d_i = DecryptNode(N.children[i], T_c, T_d)$
22: Let c_i be the corresponding information.
23: $d'_i = F(d_i, c_i)$
24: **end for**
25: $d = \prod_{i\in S} d'^{\Delta_{i,S}(0)}_i$
26: **end if**
27: Add the $N.hash, d$ pair into T_d
28: **end if**

As for a general threshold gate (including the OR gate), we need to find $N.k$ indexes S that the corresponding children nodes can be decrypted. Then the value $d_i = e(g,g)^{as_it}$ is decrypted for each $i \in S$, Thus d'_i can be computed with the additional information (we will specify this additional information in the next section) $d'_i = F(e(g,g)^{as_it}, a_i) = e(g,g)^{as'_it}$. We can then compute the secret with $N.k$ Lagrange coefficients and d'_i(see Line 25).

By running the algorithm, for each file, the algorithm will return a decrypted value d. If $d = NOT\_SATISFIED$, then the user doesn't have enough attributes to decrypt the file. Otherwise, assume that the file secret is s, we know $d = e(g,g)^{ast}$. The decryption key can be calculated as

$$e(C', K)/d = e(g^{\alpha+at}, g^s)/e(g,g)^{ast} = e(g,g)^{\alpha s}$$

With this derived decryption key, we can recover the file m by computing $m = C/e(g,g)^{\alpha s}$.

5.5 Building the Additional Ciphertext

In the encryption phase, the additional ciphertext c_i is added for each child node of a general threshold gate. In the decryption phase, a function F is used to compute the secret of the gate with this additional ciphertext. This function need to have the following properties to keep the algorithm correct.

- The additional c_i could be computed by s_i, s and the public parameters.
- The parameters of F should be related to the decrypted secret of the corresponding child nodes. So that we can compute $e(g,g)^{ats}$ from $e(g,g)^{ats_i}$, the secret of the ith child node.
- The calculation of F should be independent of the user parameter t. That is, $F(e(g,g)^{at_1s_i}, c_i) = e(g,g)^{at_1s'_i}$ while $F(e(g,g)^{at_2s_i}, c_i) = e(g,g)^{at_2s'_i}$.

The above properties indicate that the function F should be a linear operation that is independent of t. One simple idea is to compute the value $\Delta s_i = s'_i/s_i$, thus $e(g,g)^{ats'_i} = (e(g,g)^{ats_i})^{\Delta s_i}$.

Clearly, this simple approach satisfies the above three properties. However, it can't provide the necessary security, since it's easy to compute $1/\Delta s$. Hence, for security issues, the following additional requirements should also be guaranteed.

- The additional ciphertext c_i cannot leak the any information about hidden variables(a, s'_i, t).
- The calculation of function F must be one-way. That is, it's hard to find a function F' that $F'(e(g,g)^{ats'_i}, c_i) = e(g,g)^{ats_i}$ so that a malicious user cannot compute A_1's secret when he has computed the secret of A_1 OR A_2.

Go back to the above simple approach. When a user has computed $e(g,g)^{ats'_i}$ with the node value $e(g,g)^{ats}$, he can then calculation $e(g,g)^{ats_i} = (e(g,g)^{ats'_i})^{1/\Delta s_i}$.

To avoid this situation, we need to make the computation of F be hard to invert. A possible solution is to encrypt the additional information by the secret. Similar to the final ciphertext, for a child node with secret $e(g,g)^{ats_i}$, we could let $C'_i = g^{s_i}$. Then we can calculate $e(g^\alpha g^{at}, g^{s_i})/e(g,g)^{ats_i} = e(g,g)^{\alpha s_i}$, and encrypt Δs_i by $e(g,g)^{\alpha s_i}$.

Unfortunately, this method will leak the information of g^{s_i}. Thus, we need to define one more value to overcome this. Precisely, we choose a global random secret s_0, and then the ciphertext $C_0 = g^{as_0}h_0^{-r_0}, D_0 = g^{r_0}$, where h_0 and r_0 are random variables. Similar to the computation on the leaf node, a user could compute a value $d = e(C_0, L)e(D_0, K_0) = e(g,g)^{ats_0}$. For each child node $e(g,g)^{s_i}$, the data owner computes $\Delta s_i = s'_i/(s_i + s_0)$.

After that, the additional information $g^{s_i+s_0}$ is added to the child node. Note that both g^{s_i} and g^{s_0} are not leaked when the user only know $g^{s_i+s_0}$. The value Δs_i is then encrypted by $e(g,g)^{\alpha(s_i+s_0)}$.

In decryption phase, we process the function F as follows: we first compute

$$e(g,g)^{\alpha(s_i+s_0)} = e(g^\alpha g^{at}, g^{s_i+s_0})/(e(g,g)^{ats_i} \cdot e(g,g)^{ats_0})$$

Hence Δs_i is decrypted, then we have

$$e(g,g)^{ats'_i} = (e(g,g)^{ats_i} \cdot e(g,g)^{ats_0})^{\Delta s_i}$$

As a result, we successfully compute $e(g,g)^{ats_i'}$ while the information about s_i is not leaked.

5.6 Combining with Deduplication of Files

Since our scheme focused on the deduplication of ABE ciphertexts, the encrypted files are not changed. When the files are large, deduplicating the encrypted files is also important. In fact, our scheme can simply be combined with the existing data deduplication system such as DupLESS[8]. In the system, a data owner first generates a tag of the encrypted file, the cloud server uses the tag to check whether the file exists to avoid duplication. To combine the system with our scheme, we can encrypt the tag and the AES key by our scheme, then we can enable deduplicatoin of ABE ciphertexts as well as encrypted files.

6. SECURITY ANALYSIS

Note that our scheme is based on the CP-ABE scheme in[27], which was proved to be secure. Also recall that an ABE schemes is said to be secure if no adversary succeeds in the security game defined in Section 3, where none of the queried secret keys can satisfy the access structure $\mathcal{A}$ that is associated to the challenge ciphertext. Therefore, in this section, we mainly focus on the analysis on the modifications in our scheme and show that the resulting scheme is not violating the CP-ABE security definition.

First, we look into the construction of ciphertexts. We use the construction similar to that of the original scheme to build our ciphertexts. The main difference lies in the construction of the secret value which is contained in the ciphertexts. Note that in the original scheme, the secret value related to some certain attribute in a ciphertext is in the form of exponent and won't be leaked during the execution of the system. Thus, in order to remain secure, the modifications to the original scheme should not violate this feature. In particular, only if the user possesses a valid secret key that satisfies a policy can he compute the value of $e(g,g)^{ast}$, which is the key to a decryption.

Below, we give the analyses on the constructions of ciphertexts associated with OR gate and AND gate, respectively.

General Threshold Gate (including OR gate): As for the general threshold gate, the construction of the ciphertext differs from that in the original scheme. We proposed a completely different method to build the child nodes of a general threshold gate's ciphertext. In the original scheme, the n_x children of the OR gate x have the secret values $\{s_i\}$. These values are the evaluations of an $k_x - 1$ degree polynomial q at 1 to n_x, that is, $q(1) = s_1, q(2) = s_2, ..., q(n_x) = s_{n_x}$. So the values satisfy the reconstruct property: let S be any k_x valid indexes, $s_x = q(0) = \sum_{i \in S} \Delta_{i,S}(0) \cdot s_{x_i}$. As for our scheme, the values $\{s_i\}$ do not satisfy the same property, so we use the additional ciphertext and one-way function to enable the user to compute the values $\{s_i'\}$ by using the values $\{s_i\}$, and let $\{s_i'\}$ satisfy the above property. Then the main issue boils down to whether this additional ciphertext renders this scheme insecure. The additional ciphertext contains two parts: $g^{s_i+s_0}$ and the encrypted $\Delta s_i = s_i'/(s_i + s_0)$. On the one hand, by the hardness of the Discrete Logarithm Problem, the value $s_i + s_0$ is not leaked. g^{s_0} is not shared to users so that the value g^{s_i} is not leaked and any information about s_i keeps secure. Also, due to the fact that g^{s_0} is not shared to users, the value g^{s_i} will not be learnt, and no information about s_i will be leaked. Hence, no information about $g^{s_i+s_0}$ is exposed to user. Also, Δs_i can only be computed when knowing all the values of s_i, s_0 and s_i' at the same time. Note that, all of those 3 values are not known to the user during the decryption. As a consequence, Δs_i reveals nothing about any of those 3 values. We point out that our current approach to building reusable secrets for OR gates (or general threshold gate) is limited, because it would downgrade the scheme which is processed with OR gate (or general threshold gate) deduplication operations to a weaker security model. To see that, we use the following example. Consider ciphertext C encrypted under access policy (A OR B), and user Alice with private key associated with attribute A and user Bob with attribute B. Note that either Alice or Bob can decrypt C using his/her own private key. That is, they are able to derive the same secret value s_{or} that is associated with the OR gate. With the relationship between s_{or} and his/her own private key, which results from the processing of OR-gate deduplication, It is possible for Alice and Bob to work together to derive a valid private key that is associated with attribute set {A, B}. Note that this attack can't happen in the original security model.

AND gate: In the original scheme, the secret value of an AND gate x with n_x children should be computed as $s_x = \sum_{i=1}^{n_x} \Delta_{i,S}(0) \cdot s_{x_i}$, where s_{x_i} is the secret of the ith child, $S = 1, 2, ...n_x$. Only when the values of all the child nodes are computed can the user derive the value associated with the AND gate. As for our scheme, the only way to compute the valid AND-gate value is the same as that in the original scheme. Therefore, unlike the OR-gate deduplication case, the scheme resulting from being processed with only AND-gate deduplication technique achieves the same security level as the original scheme does. We also provide a more formal security proof for this type of schems in [2].

As for other algorithms in the proposed scheme including global setup and key generation, we omit the security analysis since they are the same as in the original scheme. Therefore, the security of our proposed scheme follows as long as the security of the original scheme is guaranteed.

7. IMPLEMENTATION

To evaluate the performance of our proposed scheme, we implemented our scheme using the Charm library[4], which is a general framework for cryptosystems written in Python. Our implementation is based on the implementation of the CP-ABE scheme [27] in this library, and the overall implementation is only about 300 lines of Python code. In summary, the modifications are as follows.

- The merkle tree function is added to the data structure of access trees so that the node with the same structure will share the same hash key.
- The ciphertexts is stored in a hash table (using the Python Dict), and the recursive encryption algorithm 4 is implemented to construct this hash table.
- The addtional ciphertext of a general threshold gate is added when dealing with it.
- The recursive decryption algorithm is implemented by picking items from the hash table.

8. EVALUATION

To evaluate the performance of our scheme, we first analyzed how the access policies affect the performance of our scheme, then we conducted a series of experiments to verify the analysis. For better assessment, we ran our implementation and the implementation of the original scheme [27] using

the same input sets, made comparisons in terms of computation cost and storage cost. The experiments are conducted on a Macbook Pro with an Intel Core i5 at 2.4GHz and 8.00 GB RAM. Three key factors will be compared in our experiments: the size of ciphertext, the encryption time, the decryption time. Each experiment will be run 100 times to calculate the average value.

8.1 Theoretic Analysis

Before giving out the analysis, we first define some notations that will be used. Here we only consider the access tree with only AND gates and OR gates, the general threshold gate situation can be derived by the OR case. We assume there are n nodes, m edges in p access policy trees, and there are n_l leaf nodes and n_n non-leaf nodes. Hereafter, we denote our scheme's storage overhead by s and the original scheme's overhead by s'.

According to the property of trees,$n_n + n_l = n, n = m + p$. Assume there are n_{attr} attributes, n_{and} AND gates, n_{or} OR gates. The ith AND gate appears $k_{and,i}$ times, and has $c_{and,i}$ children, The ith OR gate appears $k_{or,i}$ times, and has $c_{or,i}$ children, The ith attribute appears $k_{attr,i}$ times. Since all the leaf nodes are built by attributes and the non-leaf nodes consist of OR gates and AND gates, we have

$$n_l = \sum_i^{n_{attr}} k_{attr,i}, n_n = \sum_i^{n_{and}} k_{and,i} + \sum_i^{n_{or}} k_{or,i} \quad (1)$$

Every edge in the access trees indicates a parent-child relationship in an OR gate or an AND gate, thus

$$m = \sum_i^{n_{and}} k_{and,i} c_{and,i} + \sum_i^{n_{or}} k_{or,i} c_{or,i} \quad (2)$$

Since each AND/OR gate has at least 2 children,$c_{or,i} \geq 2, c_{and,i} \geq 2$. Based on the above equations, we have

$$\sum_i^{n_{attr}} k_{attr,i} + \sum_i^{n_{and}} k_{and,i} + \sum_i^{n_{or}} k_{or,i} \quad (3)$$

$$= \sum_i^{n_{and}} k_{and,i} c_{and,i} + \sum_i^{n_{or}} k_{or,i} c_{or,i} + p \quad (4)$$

Now we analyze the ciphertexts size in the schemes. The ciphertexts in the original scheme are built according to the attributes on the leaves. Thus the size of the ciphertexts in the original scheme is $s' = C_{attr} \cdot n_l = C_{attr} \sum_i^{n_{attr}} k_{attr,i}$, where C_{attr} is the size of each ciphertext generated by an attribute. As for our proposed scheme, the ciphertexts contains two parts. One is the ciphertexts of attributes, and the other is the additional ciphertexts on the OR nodes. The size in our proposed scheme is $s = C_{attr} n_{attr} + C_{OR} \sum_i^{n_{or}} c_{or,i}$, where C_{OR} is the size of one ciphertext added by an OR node. By comparing s with s', we have

$$s/s' = \frac{C_{attr} n_{attr} + C_{OR} \sum_i^{n_{or}} c_{or,i}}{C_{attr} \sum_i^{n_{attr}} k_{attr,i}} \quad (5)$$

Based on the construction in section 5, both the ciphertext of an attribute and the ciphertext of an OR gate contain two group elements. Eq. 5 can be simplifies to

$$s/s' = \frac{n_{attr} + \sum_i^{n_{or}} c_{or,i}}{\sum_i^{n_{attr}} k_{attr,i}} \quad (6)$$

$$= \frac{1}{\bar{k_{attr}}} + \frac{n_{or} \bar{c_{or}}}{n_{attr} \bar{k_{attr}}} \quad (7)$$

We can see that, when $n_{or} = 0$, s/s' reaches the minimum value $\frac{1}{\bar{k_{attr}}}$. That is, our scheme will offer much better performance when there is no OR gates, but only AND gates, where the cost is only $\frac{1}{\bar{k_{attr}}}$ compared with the original scheme. Otherwise, the more OR gates, the larger s/s' is. According to eq. 3, we have

$$\sum_i^{n_{or}} (c_{or,i} - 1) k_{or,i} = \sum_i^{n_{attr}} k_{attr,i} - p - \sum_i^{n_{and}} (c_{and,i} - 1) k_{and,i}$$

In the worst case, $n_{and} = 0, k_{or,i} = 1$, (8)

$$\sum_i^{n_{or}} c_{or,i} = \sum_i^{n_{attr}} k_{attr,i} - p + n_{or} \quad (9)$$

$$s/s' = 1 + \frac{n_{attr} + n_{or} - p}{\sum_i^{n_{attr}} k_{attr,i}} \quad (10)$$

$$= 1 + 1/\bar{k_{attr}} + (n_n - p)/n_l \quad (11)$$

Since every non-leaf node has at least two children, we have $n_n < n_l$, and $n_n/n_l < 1$, $k_{attr,i} \geq 1$, so $0 < 1/\bar{k_{attr}} < 1$.

$$1 < 1 + 1/\bar{k_{attr}} + (n_n - 1)/n_l < 3 \quad (12)$$

Thus, in our scheme, the size of the ciphertexts is at most 3 times bigger than that in the original scheme.

As for the computation cost, since the computation is mainly depending on the ciphertexts, the analysis will be similar and it also gives that the proportion of AND gates determines how good our scheme is. Therefore, we will mainly focus on two extreme case: only AND gate and only OR gate.

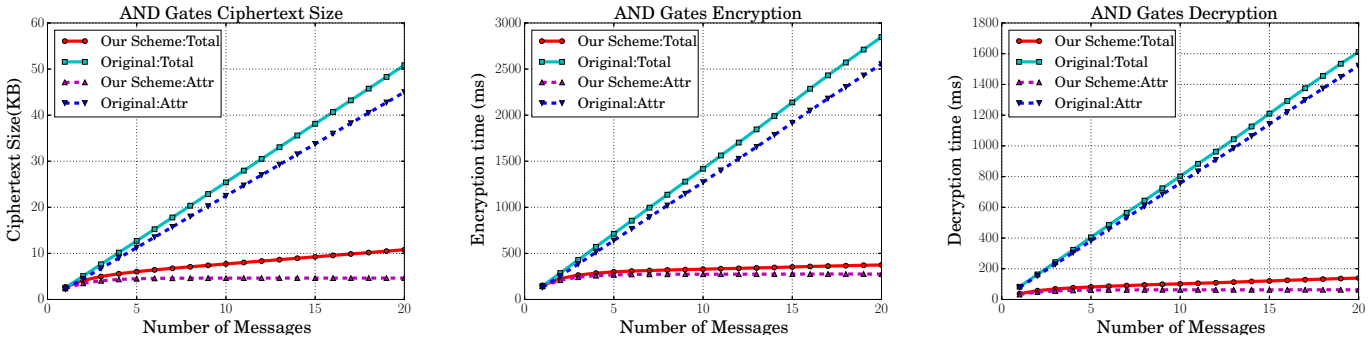
8.2 Only AND gate

The above analysis demonstrates that our scheme will achieve much better performance when there are only AND gates in the access structures. For this reason, we first evaluate our scheme in this only-AND-gate case.

Assume there are U attributes in total and p files to be encrypted. Each file will be encrypted by a session key (we will use session key and message interchangeably hereafter), and each session key is encrypted under an access policy using the CP-ABE schemes. The access policy of each session key's ciphertext is an AND gate with c random attributes. We then run the encryption and decryption algorithm in different numbers of messages. We set $U = 50$,$p = 20$,$c = 16$. The storage cost,encryption cost and the decryption cost with different numbers of messages are as shown in Fig.3.

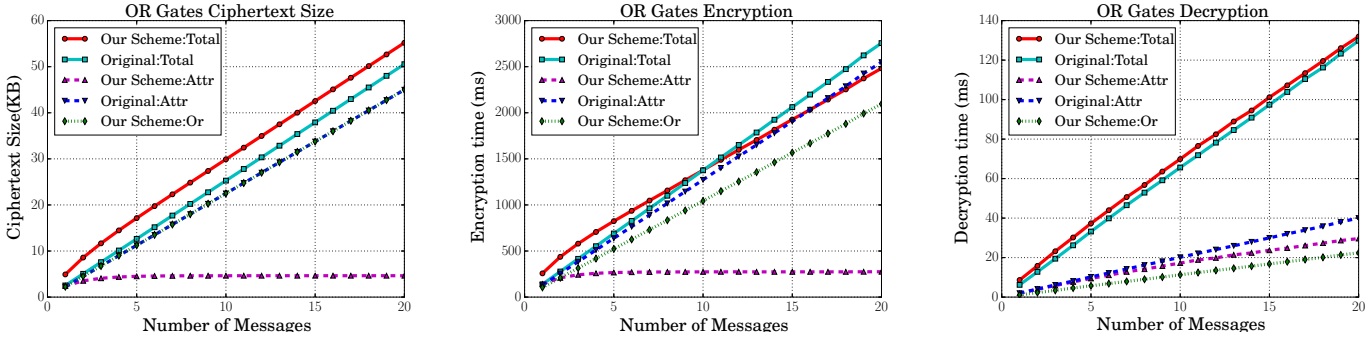
Fig.3a compares ciphertext size between our scheme and the original scheme. When the message number p grows linearly, the benefit of our scheme will also increase. As the number of the messages grows, the ciphertext size of the original scheme grows linearly. We can see that the ciphertext size of the original scheme grows linearly with the amount of messages, while in our proposed scheme, the size doesn't increase. Concretely, in our scheme, as the duplicate attributes are reduced, the size of the ciphertext has a upper bound. The size of the ciphertext generated by attributes is kept under 5KB, which is the size of about 50 attributes. Since each message needs some additional ciphertext for the last step of decryption, the size of the rest of ciphertext in our scheme keeps growing but at a low speed. Compared with the original scheme, the rest of the ciphertext in two schemes are very close. Due to the duplicate ciphertext of the attributes, when the number of the messages reaches 20, the total size of ciphertext in our scheme is about 1/5 of the corresponding size in original scheme.

As for encryption, in Fig.3b, the difference is very similar to the ciphertext size case shown above. The growth in the computation cost to build the ciphertext in the final



(a) Ciphertext Size (b) Encryption Time (c) Decryption Time

Figure 3: Performance Evaluation of AND Gates



(a) Ciphertext Size (b) Encryption Time (c) Decryption Time

Figure 4: Performance Evaluation of OR Gates

step is very small compared to the overall computation cost. Hence the difference between two schemes becomes much more significant. While the total encryption time of our scheme almost becomes stable after it reaches about 400ms, the encryption time of the original scheme goes beyond 2500ms when the number of messages reaches 20.

For decryption time, since the decryption of AND gate requires all its children to be decrypted, the comparison of decryption time, as shown in Fig.3c, is rather similar to that of encryption time, though the total time of decryption is smaller than encryption.

Overall, in the best case, our scheme shows a significant boost in both storage and computation compared to the original scheme.

8.3 Only OR gate

As for the situation with only OR gates, we use the same settings: U attributes, p messages, OR gates with c random attributes, where $U = 50$,$p = 20$,$c = 16$. The results are shown in Fig.4.

In Fig.4a, as we have expected, although the size of attribute ciphertext in our scheme has a upper bound, the additional ciphertext introduced by the OR gates makes the size grow linearly with the message number. Although the size of the ciphertext brought by the OR gates in our scheme is almost the same as the size of the attribute ciphertext in original scheme, our scheme has more fixed-sized attribute ciphertexts. Clearly, the size in our scheme is always larger then the original scheme. But since this size difference between these two schemes is fixed (related to the size of the attribute universe), it is acceptable when the number of the messages grows larger. According to above analysis, the difference between two algorithms is $\frac{1}{\bar{k}_{attr}} + \frac{n_{or}\bar{c_{or}}}{n_{attr}\bar{k}_{attr}}$. In this case, $\frac{n_{or}\bar{c_{or}}}{n_{attr}\bar{k}_{attr}}$ is fixed, the difference mainly depends on $1/\bar{k}_{attr}$. As a result, this difference becomes smaller when the number of messages increases, and further the cost difference between these two schemes become closer.

In Fig.4b, the overall difference on the encryption time between these two schemes is similar to that shown in Fig.4a. Specifically, the time of constructing the ciphertext of an OR gate's child(which is the additional ciphertext) is a little less than constructing the ciphertext of one attribute. This important difference leads to the fact that, Our scheme has a bigger advantage over the original one when the number of messages grows to a high value.

In decryption phase, one attribute can be enough for each OR gate. Concretely, the decryption consists of two steps: computing the secret on root node by decrypting one leaf node, and using the secret to compute the decryption key. Since our scheme only introduces a small amount of additional cost in the first step, the difference between their decryption time is not significant.

The above results show that even in the bad cases, our scheme doesn't introduce too much extra overhead.

9. CONCLUDING REMARKS

Motivated by the fact that there usually exist duplicate access structures in the application of CP-ABE scheme to cloud access control systems, in this paper, we proposed new ciphertext deduplication techniques. Our proposed techniques reorganizes the construction of the access structures and reuses the duplicate structures. We apply our proposed deduplication methods to an existing CP-ABE scheme [27] and give an instantiation of ciphertext-deduplicated CP-ABE scheme. To better assess the performance of our proposed scheme, we implemented the modified CP-ABE scheme and conducted a series of experiments. The experimental results validate our analysis and demonstrate that our proposed scheme achieves good performance. For our future work, we will try to implement a lightweight source code control system with fine-grained access control using

our proposed deduplication techniques. Another interesting open problem is how to enable deduplication on ciphertexts under LSSS model.

10. ACKNOWLEDGMENTS

This work is supported in part by National Natural Science Foundation of China (No. 61422206, 61120106008), Tsinghua National Laboratory for Information Science and Technology (TNList), US National Science Foundation under grant CHN-1262277.

11. REFERENCES

[1] Crypto++ 5.0. http://www.cryptopp.com/.

[2] Enabling ciphertext deduplication for secure cloud storage and access control(full version). https://www.dropbox.com/sh/o5apxgnrli49hr0/AAAIwKMJXWnfXNLW2yIBTuz3a.

[3] Openssh: 2.1.1p4. http://www.openssh.com/.

[4] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[5] J. A. Akinyele, C. U. Lehmann, M. D. Green, M. W. Pagano, Z. N. Peterson, and A. D. Rubin. Self-protecting electronic medical records using attribute-based encryption. 2010.

[6] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 135–146. ACM, 2009.

[7] A. Beimel. *Secure schemes for secret sharing and key distribution*. PhD thesis, Technion-Israel Institute of technology, Faculty of computer science, 1996.

[8] M. Bellare, S. Keelveedhi, and T. Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *Proceedings of the 22nd USENIX conference on security*, pages 179–194. USENIX Association, 2013.

[9] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Proceedings on Advances in cryptology*, pages 27–35. Springer-Verlag New York, Inc., 1990.

[10] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[11] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624. IEEE, 2002.

[12] 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*, pages 89–98. Acm, 2006.

[13] M. Green, S. Hohenberger, and B. Waters. Outsourcing the decryption of abe ciphertexts. In *USENIX Security Symposium*, volume 2011, 2011.

[14] S. Hohenberger and B. Waters. Online/offline attribute-based encryption. In *Public-Key Cryptography–PKC 2014*, pages 293–310. Springer, 2014.

[15] J. Lai, R. H. Deng, C. Guan, and J. Weng. Attribute-based encryption with verifiable outsourced decryption. *Information Forensics and Security, IEEE Transactions on*, 8(8):1343–1354, 2013.

[16] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):131–143, 2013.

[17] B. Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford University, 2007.

[18] D. Meister and A. Brinkmann. Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 8. ACM, 2009.

[19] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology-CRYPTO'87*, pages 369–378. Springer, 1988.

[20] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 174–187. ACM, 2001.

[21] R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM, 2007.

[22] 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*, pages 99–112. ACM, 2006.

[23] Y. Rouselakis and B. Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security*, pages 463–474. ACM, 2013.

[24] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10. ACM, 2008.

[25] D. Teodosiu, N. Bjorner, Y. Gurevich, M. Manasse, and J. Porkka. Optimizing file replication over limited bandwidth networks using remote differential compression. *Microsoft Research TR-2006-157*, 2006.

[26] P. Traynor, K. R. Butler, W. Enck, and P. McDaniel. Realizing massive-scale conditional access systems through attribute-based cryptosystems. In *NDSS*, 2008.

[27] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography–PKC 2011*, pages 53–70. Springer, 2011.

[28] Y. Wu, Z. Wei, and R. H. Deng. Attribute-based access to scalable media in cloud-assisted content sharing networks. *Multimedia, IEEE Transactions on*, 15(4):778–788, 2013.