Efficient Sharing of Secure Cloud Storage Services

Qin Liu[†], Guojun Wang[†*], and Jie Wu[‡]

[†]School of Information Science and Engineering
Central South University
Changsha 410083, P. R. China
[*] Correspondence to: csgjwang@mail.csu.edu.cn

[‡]Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122, USA

***Abstract*—Suppose Bob, the boss in *Company A*, pays a secure cloud storage service and authorizes all the employees in that company to share such a service. There exists a user hierarchy: Bob is the user at the upper level and all the employees in the company are the users at the lower level. In this paper, we design and construct a scheme, which enables the user at the upper level to efficiently share the secure cloud storage services with all the users at the lower level. A sender can specify several users at the lower level as the recipients for a file by taking the number and public keys of the recipients as inputs of a hierarchical identity-based encryption algorithm, which enables only the user at the upper level, as well as the intended recipients, to decrypt the file using their own private keys. Using our scheme, the sender needs to encrypt a file only once, and store only one copy of the corresponding ciphertext in a "cloud" communicating with none of the recipients while encrypting a file to multiple recipients. To our best knowledge, this paper is the first to realize the efficient sharing of the secure storage services in cloud computing.**

***Index Terms*—secure storage; cloud computing; hierarchical identity-based encryption; multiple recipients**

I. Introduction

Cloud computing, as one of the 2010 Top 10 Strategic Technologies, enables users to consume computing and storage resources as services, and pay a *cloud service provider* (CSP) only for what they use. With the rapid development of cloud computing, users can enjoy more comprehensive, scalable, and secure services. We consider the following application scenario: Bob, who is the boss in *Company A*, pays a CSP for a secure cloud storage service, and authorizes all the employees in the company to share such a service.

On Monday: Bob has a confidential file f and wants his employees, Alice and Clark, to make a cooperative report on f in two days. He stores f on the CSP, and later Alice retrieves it for making a draft report $f^{'}$.

On Tuesday: Alice stores a draft report f' on the CSP, and later Clark retrieves f and f' for making a final report $f^{''}$.

On Wednesday: Clark stores a final report $f^{''}$ on the CSP, and later Bob retrieves f, $f^{'}$, and $f^{''}$ for inspecting the works collaborated by Alice and Clark.

For the sake of not leaking any information about f, it is essential to ensure that f, and its relevant reports $f^{'}$ and $f^{''}$, can only be read by Bob and employees Alice and Clark, who participate in making the cooperative report. Therefore, Bob may wish to enjoy the following services: (1) Although some important files of *Company A* are stored in the cloud, the CSP has no idea of any information regarding these files. (2) He is able to read all the files stored in the cloud, whereas an employee can only read the files that he is qualified to read.

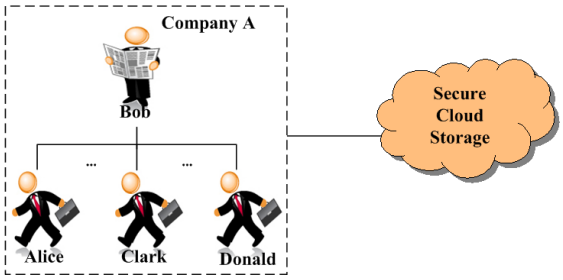

Fig. 1. User hierarchy in the sharing of the secure cloud storage services

The natural way to satisfy the above service demands is to encrypt the files before storing them in a cloud. In the above application scenario, a confidential file involves more than one recipients. Therefore, the adopted encryption system should be "one-to-many", in which one encrypted file can be decrypted by multiple intended recipients.

There has been a lot of research conducted on this field, the most promising and interesting of which is *attribute-based encryption* (ABE) schemes. The first construction for a ABE system was proposed by Sahai et al [14], in which a sender encrypts a message, specifying an attribute set and a number d, so that only a recipient who has at least d attributes of the given attributes can decrypt the message. Based on their work, Goyal et al [15] proposed a fine-grained access control ABE scheme, which supports any monotonic access formula consisting of AND, OR, or threshold gates. Their scheme is characterized as *key-policy* ABE (KP-ABE) since the access structure is specified in the private key, while the attributes are used to describe the ciphertexts. Bethencourt et al [16] introduced a *ciphertext-policy* ABE (CP-ABE) scheme, in which the roles of the ciphertexts and keys are reversed in contrast with the KP-ABE scheme: The access structure is specified in the ciphertext, while the private key is simply created with respect to an attributes set. In recent work, Chase [17] provided a construction for a multi-authority ABE system, where each authority would administer a different domain of attributes. Chase et al [18] provided a more practice-oriented multi-authority ABE system, which removes the trusted central authority while preserving user privacy. Among others, Yu

et al [19] exploited and uniquely combined techniques of KP-ABE, *proxy re-encryption* (PRE), and *lazy re-encryption* (LRE) to delegate most of the computation tasks involved in user revocation to untrusted CSPs without disclosing the underlying data contents, which may make a KP-ABE system more applicable in a cloud environment.

The existing ABE schemes can enable a sender to encrypt a message to multiple recipients in an efficient way, and in consequence, they can be applied to access control in encrypted file systems for securely sharing the information. However, an ABE system may not be applicable in the sharing of a cloud storage service, largely because it cannot embody the user hierarchy in such an environment (see Fig. 1). In an ABE system, there are only two parties: *attribute authorities* (AA) and users, where all the users at the same level request their secret keys from a single AA or multiple AAs at the same level, and can only decrypt the files that they are qualified to read.

In the above application scenario, Bob can not only authorize all the employees in *Company A* to share such a service, but also read all the files stored on the CSP. It is natural that Bob also generates the secret keys for all the employees in that company at the time of authorization. Obviously, there exists a user hierarchy, in which Bob is the user at the upper level (from here-on referred to as the upper-level user), whereas all the employees are the users at the lower level (from here-on referred to as the lower-level users). Therefore, the adopted encryption system in such an environment not only should be "one-to-many", but also it should be "hierarchical".

Based on the above mentioned analysis, we propose an efficient sharing of the secure cloud storage services (ESC) scheme. Our contributions are fourfold:

1) The problem with efficient sharing of the secure cloud storage services is introduced for the first time. A user will enjoy a more comprehensive, scalable, and secure service as the resolution of such a problem comes out.
2) A hierarchical identity-based architecture in cloud computing is proposed to embody the user hierarchy in the sharing of the secure cloud storage services. In this architecture, the root *private key generator* (PKG) delegates the upper-level user as the lower-level PKG to generate the secret keys for all the lower-level users. The upper-level user, and all the lower-level users, compose a domain, in which authentication and secret key transmission can be carried out, locally.
3) The proposed ESC scheme applies and tailors the hierarchical identity-based encryption to a cloud environment. Using our scheme, the sender needs to encrypt a file only once, and store only one copy of the corresponding ciphertext in a cloud communicating with none of the recipients. But, the upper-level user and all the intended recipients can successfully recover the file using their own private keys.
4) The proposed ESC scheme is collusion resistant, which has semantical security against adaptive chosen plaintext attacks in the random oracle model under the *Bilinear Diffie-Hellman* (BDH) assumption [1].

The rest of this paper is organized as follows: We review some related work in Section II, and introduce some preliminary definitions and technologies in Section III. We provide an overview of our proposed scheme in Section IV, and construct the proposed scheme based on the bilinear map in Section V. We analyze the performance and security of the proposed scheme in Section VI, and conclude this paper in Section VII.

II. Related Work

The first construction for a *hierarchical identity-based encryption* (HIBE) system was proposed by Gentry et al [2], which has chosen ciphertext security in the random oracle model under the BDH assumption. In their scheme, a root PKG needs only to generate private keys for domain-level PKGs, who in turn generate private keys for users in their domains in the next level. Authentication and private key transmission can be carried out locally. A subsequent construction by Boneh et al [4] provides a HIBE system with secure identity security under BDH assumption without random oracles. In both constructions, the length of ciphertexts and private keys, as well as the time needed for decryption and encryption, grows linearly with the depth of a recipient in the hierarchy. For better performance, Boneh et al [5] provide an efficient HIBE system with a constant length of ciphertexts and a constant number of bilinear map operation in decryption. In recent work, Gentry et al [6] proposed a fully secure HIBE scheme by using identity-based broadcast encryption with key randomization; Waters [7] achieved full security in systems under simple assumption by using dual system encryption.

In this paper, we borrow ideas from Gentry et al [2]. In their scheme, a user public key is an ID-tuple, which consists of the user's *identity* (ID) and the IDs of the user's ancestors; each PKG uses its secret keys (including a master key and a private key) and a user public key to generate secret keys for each user in its domain; the encrypter uses system parameters and a user public key to encrypt a file, so that only the intended recipient whose public key is on input of the encryption process can decrypt the file using his private key. The limitations of the G-HIBE include: (1) Both the computational cost for decryption, and the length of ciphertexts grow linearly with the depth of the recipient in the hierarchy. Although they also proposed a Dual-HIBE to shorten the length of the ciphertexts, it may not work well if the sender and the recipient didn't share a common ancestor at a lower level. (2) Any of the recipient's ancestors can use his own private key to decrypt the file.

The insights behind the second limitation is that a user can recover a file using his own private key, in case that his public key is as input during encryption, due to the construction of user public key, user private key, and encryption algorithm. However, we consider this limitation as a kind of valuable property that can enable a sender to encrypt a file to multiple recipients (the recipient and all his ancestors). Getting the insights behind such a property, we propose the ESC scheme with full collusion resistance, in which any lower-level user

who is not an intended recipient, is unaware of any information in regards to the encrypted file, even if all of them collude.

III. Preliminaries

We introduce some related definitions and complexity assumptions, which closely follow those in Boneh et al [1].

Definition 3.1 (Bilinear map): Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of some large prime order q, where $\mathbb{G}_1$ is an additive group and $\mathbb{G}_2$ is a multiplicative group. A bilinear map, $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, satisfies the following properties:
(1) Computable: There is a polynomial time algorithm to compute $\hat{e}(P, Q) \in \mathbb{G}_2$, for any $P, Q \in \mathbb{G}_1$.
(2) Bilinear: $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}_q^*$.
(3) Non-degenerate: The map does not send all pairs in $\mathbb{G}_1 \times \mathbb{G}_1$ to the identity in $\mathbb{G}_2$.

Definition 3.2 (BDH Parameter Generator): A randomized algorithm $\mathcal{IG}$ is called a BDH parameter generator if $\mathcal{IG}$ takes a sufficiently large security parameter $K > 0$ as input, runs in polynomial time in K, and outputs a prime number q, the description of two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q, and the description of a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.

Definition 3.3 (BDH Problem): Given a random element $P \in \mathbb{G}_1$, as well as aP, bP, and cP, for some $a, b, c \in \mathbb{Z}_q^*$, compute $\hat{e}(P, P)^{abc} \in \mathbb{G}_2$.

Definition 3.4 (BDH Assumption): If $\mathcal{IG}$ is a BDH parameter generator, the advantage $Adv_{\mathcal{IG}}(\mathcal{B})$ that an algorithm $\mathcal{B}$ has in solving the BDH problem is defined as the probability that $\mathcal{B}$ outputs $\hat{e}(P, P)^{abc}$ on inputs q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\hat{e}$, P, aP, bP, cP, where $< q, \mathbb{G}_1, \mathbb{G}_2, \hat{e} >$ are the outputs of $\mathcal{IG}$ for a sufficiently large security parameter K, P is a random element $\in \mathbb{G}_1$, and a, b, c are random elements of $\mathbb{Z}_q^*$. The BDH assumption is that $Adv_{\mathcal{IG}}(\mathcal{B})$ is negligible for any efficient algorithm $\mathcal{B}$.

IV. Overview of the Proposed Scheme

A. Hierarchical Identity-Based Architecture

To embody the user hierarchy in the sharing of the secure cloud storage services, we propose the hierarchical identity-based architecture in cloud computing (see Fig. 2).

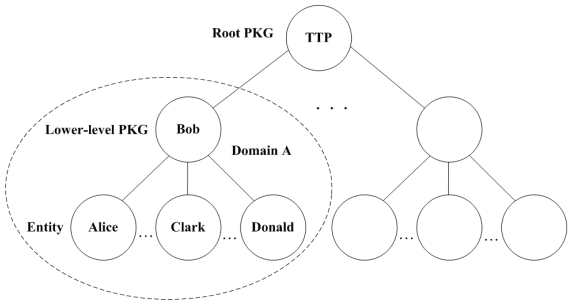

Fig. 2. Hierarchical identity-based architecture in cloud computing

From Fig. 2, we can see that the proposed architecture consists of a root PKG and multiple domains. The root PKG at the top level is a *trusted third party* (TTP). The members in a domain include a lower-level PKG and multiple entities, where the lower-level PKG at the second level is the upper-level user, and all the entities at the bottom level are the lower-level users. In this hierarchical architecture, the root PKG generates the system parameters for the hierarchical identity-based encryption system and the secret keys for the lower-level PKGs, which, in turn generate the secret keys for the entities in their domains at the bottom level. Therefore, authentication and secret key transmission can be carried out, locally, in a domain.

In a domain, the lower-level PKG's public key is an ID-tuple consisting of his own ID, whereas the entity's public key is an ID-tuple consisting of his own ID and the lower-level PKG's ID. Therefore, the member's public key can denote his position in the hierarchical architecture. For example, Bob and all the employees in *Company A* constitute a domain, denoted Dom_A, where Bob is the lower-level PKG, and all the employees are entities. Bob requests the secret keys for Dom_A from the root PKG, whereas the employees request their secret keys from Bob. When the email address is used as an ID, we have: "*Bob@CompanyA.com*" and "*Alice@CompanyA.com*" are the IDs of Bob and Alice, respectively, whereas ("*Bob@CompanyA.com*") and ("*Bob@CompanyA.com*","*Alice@CompanyA.com*") are the public keys of Bob and Alice, respectively.

In a domain, a sender can specify multiple entities as the recipients for an encrypted file, so that only the lower-level PKG, as well as the intend recipients, can decrypt the file using their own private keys. Therefore, the upper-level user can efficiently share the secure cloud storage services with all the lower-level users in a domain.

B. Definition of the ESC Scheme

Based on the proposed architecture, we provide the formal definition of the ESC scheme. Let Bob and all the employees in *Company A* constitute a domain, denoted Dom_A. Suppose there are M employees $\text{E}_1, \ldots, \text{E}_M$ in Dom_A, whose public keys are denoted as $\text{ID-tuple}_i = (\text{ID}_{Bob}, \text{ID}_i)$ for $1 \leq i \leq M$.

In Dom_A, when the sender *X* wants to encrypt a file to N employees $\text{E}_1, \ldots, \text{E}_N$ ($1 \leq N \leq M$), he sends the following message to the CSP:

$$MSG_{X2CSP} = One2ManyEnc(params, N, \text{ID-tuple}_1, \ldots, \text{ID-tuple}_N, f),$$

where $params$ are the system parameters, N is the number of the intended recipients, $\text{ID-tuple}_1, \ldots, \text{ID-tuple}_N$ are the ID-tuples of $\text{E}_1, \ldots, \text{E}_N$, respectively, and f is the file. *One2ManyEnc* is a hierarchical identity-based encryption algorithm with properties discussed below. For the rest of this paper, we use the above application scenario as our sample application.

Our goal is to enable the sender *X* to take the number and public keys of the recipients as inputs of the *One2ManyEnc*

algorithm that will enable Bob, as well as each recipient, to recover f using his own private key. Any other employee in *Company A*, who is outside $\mathrm{E}_1, \ldots, \mathrm{E}_N$, has no idea of any information regarding f, even if all of them collude.

Definition 4.1 (The ESC Scheme): The ESC scheme consists of five randomized polynomial time algorithms as follows:

1) *RootSetup:* The root PKG takes a sufficiently large security parameter K as input to generate the system parameters $params$ and a root master key mk_0. We write $RootSetup(K) = (params, mk_0)$. The system parameters include a description of a finite plaintext space F, and a description of a finite ciphertext space C. The system parameters $params$ will be publicly available, while the root master key mk_0 is only known by the root PKG.
2) *DomSetup:* The root PKG generates the secret keys for the lower-level PKGs, which, in turn generate the secret keys for the entities in their domains at the bottom level. A PKG (either the root PKG or the lower-level PKG in Dom_A) takes $params$, its private key SK_{PKG}, its master key mk_{PKG}, and ID-tuple$_i$ as inputs to generate a private key SK_i and a master key mk_i for any member with ID-tuple$_i$ in Dom_A. We write $DomSetup(SK_{PKG}, mk_{PKG}, \text{ID-tuple}_i) = (SK_i, mk_i)$.
3) *One2ManyEnc:* The sender *X* takes $params$, $f \in F$, $N \in \{1, \ldots, M\}$, and the ID-tuples of the N recipients as inputs, and outputs a ciphertext $C_f \in C$. We write $One2ManyEnc(params, N, \text{ID-tuple}_1, \ldots, \text{ID-tuple}_N, f) = C_f$.
4) *UserDec:* Bob takes $params$, a private key SK_{Bob}, and the ciphertext C_f as inputs to recover the plaintext f. We write $UserDec(params, SK_{Bob}, C_f) = f$.
5) *RecipientsDec:* The intended recipient E_i for $1 \leq i \leq N$ takes $params$, a private key SK_i, a master key mk_i, ID-tuple$_i$, and the ciphertext C_f as inputs to recover the plaintext f. We write $RecipientsDec(params, SK_i, mk_i, \text{ID-tuple}_i, C_f) = f$.

Using the ESC scheme, the sender *X* needs to encrypt a file f only once, and store only one copy of the corresponding ciphertext in a cloud communicating with none of the recipients, but Bob, as well as all the intended recipients can respectively decrypt C_f using their own private keys. It shows that the ESC scheme can enable the upper-level user to share the storage services with all the lower-level users, with efficiency.

Our goal is to realize the efficient sharing of the "secure" storage services, which means that we still need to prove that the ESC scheme is collusion resistant, in which any employee who is outside $\mathrm{E}_1, \ldots, \mathrm{E}_N$ is unaware of any information in regards to f, even if all of them collude. Next, we will define the security for the ESC scheme in the sense of semantic security.

C. Semantic Security of the ESC Scheme

We define the security for the ESC scheme in the sense of semantic security. Semantic security captures our insight that given a ciphertext, the adversary learns nothing about the corresponding plaintext, thus we also say that a semantically secure scheme is IND-CPA secure [1].

As described in Haclgiimfi et al [8], there are two main attacks under such a circumstance, i.e., outer attacks initiated by unauthorized outsiders, and inner attacks initiated by an untrustworthy CSP, as well as some "curious" employees in *Company A*. Here, we call an employee who is outside $\mathrm{E}_1, \ldots, \mathrm{E}_N$, but still wants to read f as a "curious" employee. Therefore, we consider the adversary targeting the ESC scheme has both passive and active capabilities, which can eavesdrop all the communication traffics in a cloud environment and obtain all the encrypted files of *Company A* stored on the CSP, and may collude to compromise the encrypted files.

As those in Boneh et al [1], we strengthen the standard definition of semantic security for a hierarchical identity-based encryption by allowing an adversary to obtain the private key associated with the ID-tuple of any employee in *Company A* of his choice, except for Bob's private key. We refer to such queries as private key extraction queries. Also, we allow an adversary to choose the value of N and the ID-tuples of $\mathrm{E}_1, \ldots, \mathrm{E}_N$ on which it wishes to be challenged. Even under such an attack, the adversary should not be able to distinguish an encryption of a file f_0 from an encryption of a file f_1 for which he did not obtain any of the private keys corresponding to ID-tuple$_{Bob}$, as well as ID-tuple$_1$, …, ID-tuple$_N$.

We say that the ESC scheme is semantically secure against adaptive chosen plaintext attacks if no polynomially bounded adversary $\mathcal{A}$ has a non-negligible advantage against a challenger in the following game:

Setup: The challenger runs the *RootSetup* algorithm when inputting a sufficiently large security parameter K to generate the system parameters $params$ and a root master key. It gives the adversary $params$, but keeps the root master key to itself.

Phase 1: The adversary issues private key extraction queries at a point ID-tuple$_i$ (which is the ID-tuple of an employee in *Company A*). The challenger responds by running the *DomSetup* algorithm to generate the private key SK_i corresponding to ID-tuple$_i$. It sends SK_i to the adversary. These queries may be asked adaptively.

Challenge: Once the adversary decides that Phase 1 is over, it outputs N, N recipients' ID-tuples: ID-tuple$_1$, …, ID-tuple$_N$, and two equal length plaintexts $f_0, f_1 \in F$ on which it wishes to be challenged. The only constraint is that none of the ID-tuples appear in any private key query in Phase 1. The challenger picks a random bit $b \in \{0, 1\}$, sets $C_{f_b} = One2ManyEnc(params, N, \text{ID-tuple}_1, \ldots, \text{ID-tuple}_N, f_b)$, and sends C_{f_b} as the challenge to the adversary.

Phase 2: The adversary issues more private extraction queries at a point ID-tuple$_i$ (which is the ID-tuple of an employee in *Company A*). The only constraint is that ID-tuple$_i$ is outside ID-tuple$_1$, …, ID-tuple$_N$. The challenger responds as in Phase 1.

Guess: The adversary outputs a guess $b' \in \{0, 1\}$, and wins the game if $b = b'$. We define $\mathcal{A}$'s advantage in breaking the ESC scheme to be $Adv_{\mathcal{A}}(K) = |Pr[b = b'] - 1/2|$.

The game above models an attack where the adversary may collude to try and expose an encrypted file. The adversary targeting the ESC scheme is adaptive, who cannot only choose the value N and the recipients set $\mathrm{E}_1, \ldots, \mathrm{E}_N$, but also request private keys for employees in *Company A*, adaptively.

Definition 4.2 (Semantic Security of the ESC Scheme): We say that the ESC scheme is semantically secure if for any polynomial time adversary $\mathcal{A}$, the function $Adv_{\mathcal{A}}(K)$ is negligible.

V. Construction of the ESC Scheme

In this section, we construct the ESC scheme using the bilinear map. Let $\mathcal{IG}$ be a BDH parameter generator. We present the ESC scheme by describing the following five randomized polynomial time algorithms:

RootSetup: Given a sufficiently large security parameter $K \in \mathbb{Z}^+$, the root PKG:

1) runs $\mathcal{IG}$ on input K to generate a prime number q, two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q, and a bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.
2) chooses a random generator $P_0 \in \mathbb{G}_1$.
3) chooses two cryptography hash functions $H_1: \{0,1\}^* \rightarrow \mathbb{G}_1^*$ and $H_2: \mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n, where H_1 and H_2 are random oracles.
4) picks a random element $mk_0 \in \mathbb{Z}_q$.
5) sets $Q_0 = mk_0 P_0 \in \mathbb{G}_1$.

The plaintext space is $F \in \{0,1\}^n$. The ciphertext space is $C = \mathbb{G}_1^N \times \{0,1\}^n$, where N is the number of the recipients. The system parameters $params =< q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P_0, Q_0, H_1, H_2 >$ will be publicly available, while the root master key mk_0 is only known by the root PKG.

DomSetup: Given Bob's public key (ID_{Bob}), the root PKG generates the secret keys for Bob as follows:

1) Set $P_{Bob} = H_1(\mathrm{ID}_{Bob}) \in \mathbb{G}_1$.
2) Set S_0 to be the identity element of $\mathbb{G}_1$.
3) Set $S_{Bob} = S_0 + mk_0 P_{Bob} \in \mathbb{G}_1$.
4) Pick a random element $mk_{Bob} \in \mathbb{Z}_q$.
5) Set $Q_{Bob} = mk_{Bob} P_0$, Q-tuple$_{Bob} = (Q_{Bob})$, and $SK_{Bob} = (S_{Bob}, \text{Q-tuple}_{Bob})$.

Bob has two secret keys: a private key SK_{Bob} and a master key mk_{Bob}, where SK_{Bob} consists of a secret point S_{Bob} and a Q-tuple in the form of (Q_{Bob}). Bob uses SK_{Bob} to decrypt all the files stored in the cloud, and uses SK_{Bob} together with mk_{Bob} to generate the private keys for all the employees in *Company A*.

Given E_i's public key $(\mathrm{ID}_{Bob}, \mathrm{ID}_i)$ for $1 \leq i \leq M$, Bob generates the secret keys for E_i as follows:

1) Set $P_i = H_1(\mathrm{ID}_{Bob}, \mathrm{ID}_i) \in \mathbb{G}_1$.
2) Set $S_i = S_{Bob} + mk_{Bob} P_i \in \mathbb{G}_1$.
3) Pick a random element $mk_i \in \mathbb{Z}_q$.
4) Set $Q_i = mk_i P_0$, Q-tuple$_i = (Q_{Bob}, Q_i)$, and $SK_i = (S_i, \text{Q-tuple}_i)$.

E_i $(1 \leq i \leq M)$ has two secret keys: a private key SK_i and a master key mk_i, where SK_i consists of a secret point S_i and a Q-tuple in the form of (Q_{Bob}, Q_i). E_i uses SK_i and mk_i to decrypt the files that he is qualified to read. It is worth noticing that E_i's public key $(\mathrm{ID}_{Bob}, \mathrm{ID}_i)$ can be concatenated into a string "$\mathrm{ID}_{Bob}||\mathrm{ID}_i$" ("||" denotes string concatenation), which is a standard input of the H_1 hash function.

One2ManyEnc: Given N, E_i's public key $(\mathrm{ID}_{Bob}, \mathrm{ID}_i)$ for $1 \leq i \leq N$, and a file f, the sender *X*:

1) computes $P_{Bob} = H_1(\mathrm{ID}_{Bob}) \in \mathbb{G}_1$.
2) computes $P_i = H_1(\mathrm{ID}_{Bob}, \mathrm{ID}_i) \in \mathbb{G}_1$ for $1 \leq i \leq N$.
3) picks a random element $r \in \mathbb{Z}_q^*$ and computes $U_0 = rP_0$, $U_1 = rP_1$, $\ldots$, $U_N = rP_N$, and $V = f \oplus H_2(\hat{e}(Q_0, P_{Bob})^r)$.
4) sets the ciphertext $C_f = [U_0, U_1, \ldots, U_N, V]$.

To encrypt a file f to N employees $\mathrm{E}_1, \ldots, \mathrm{E}_N$ in *Company A*, the sender *X* runs the *One2ManyEnc* algorithm when inputting the number and public keys of the recipients, and sends the corresponding ciphertext C_f to the CSP. It is worth noticing that the sender has no need to input the public keys of the recipients orderly, for the reason that all the recipients are at the same level in the user hierarchy.

UserDec: Given the ciphertext $C_f = [U_0, U_1, \ldots, U_N, V]$, Bob computes $V \oplus H_2(\hat{e}(U_0, S_{Bob}))$ to recover f.
Observe that:

$$\begin{aligned} & V \oplus H_2(\hat{e}(U_0, S_{Bob})) \\ = & V \oplus H_2(\hat{e}(rP_0, mk_0 P_{Bob})) \\ = & V \oplus H_2(\hat{e}(mk_0 P_0, rP_{Bob})) \\ = & V \oplus H_2(\hat{e}(Q_0, P_{Bob})^r) = f \end{aligned}$$

as required.

RecipientsDec: Given the ciphertext $C_f = [U_0, U_1, \ldots, U_N, V]$, the intended recipient E_i for $1 \leq i \leq N$:

1) computes $P_i = H_1(\mathrm{ID}_{Bob}, \mathrm{ID}_i) \in \mathbb{G}_1$.
2) tests whether $\hat{e}(U_j, Q_i) = \hat{e}(mk_i P_i, U_0)$ for $1 \leq j \leq N$. If so, E_i sets $U = U_j$. Otherwise, E_i terminates the algorithm.
3) computes $V \oplus H_2(\hat{e}(U_0, S_i)/\hat{e}(Q_{Bob}, U))$ to recover f.

Observe that:

$$\begin{aligned} \hat{e}(mk_i P_i, U_0) &= \hat{e}(mk_i P_i, rP_0) \\ &= \hat{e}(rP_i, mk_i P_0) \\ &= \hat{e}(rP_i, Q_i) \end{aligned}$$

If the *RecipientsDec* algorithm does not terminate in the third step, then there exists U_j $(1 \leq j \leq N)$ in the ciphertext $C_f = [U_0, U_1, \ldots, U_N, V]$ satisfying the equations $\hat{e}(U_j, Q_i) = \hat{e}(mk_i P_i, U_0)$. Therefore, we have $U = U_j = rP_i$. Using the part of the ciphertext rP_i, we have:

$$\begin{aligned} & V \oplus H_2(\hat{e}(U_0, S_i)/\hat{e}(Q_{Bob}, U)) \\ = & V \oplus H_2(\hat{e}(rP_0, mk_0 P_{Bob} + mk_{Bob} P_i)/\hat{e}(Q_{Bob}, rP_i)) \\ = & V \oplus H_2(\hat{e}(rP_0, mk_0 P_{Bob})\hat{e}(rP_0, mk_{Bob} P_i)/\hat{e}(Q_{Bob}, rP_i)) \\ = & V \oplus H_2(\hat{e}(mk_0 P_0, rP_{Bob})\hat{e}(mk_{Bob} P_0, rP_i)/\hat{e}(Q_{Bob}, rP_i)) \\ = & V \oplus H_2(\hat{e}(mk_0 P_0, rP_{Bob})\hat{e}(Q_{Bob}, rP_i)/\hat{e}(Q_{Bob}, rP_i)) \\ = & V \oplus H_2(\hat{e}(mk_0 P_0, rP_{Bob})) \\ = & V \oplus H_2(\hat{e}(Q_0, P_{Bob})^r) = f \end{aligned}$$

as required.

There are many files of *Company A* stored on the CSP. Bob can decrypt all the files, whereas an employee in the

company can only decrypt the files that he is qualified to read. Therefore, when an employee runs the *RecipientsDec* algorithm to decrypt a file, the algorithm first tests whether the given file belongs to the files that the employee can read. If so, the algorithm will output a part of the ciphertext, and then uses the corresponding result, and the employee's private key to decrypt the file. Otherwise, the algorithm will terminate. This concludes the description of the ESC scheme.

VI. Discussion

A. Performance

In the ESC scheme, the most expensive computation is the computation for the bilinear map. Recall that the algorithms *RootSetup* and *DomSetup* require no bilinear map computation. Both the length of the system parameters, and the length of a user's private key fix within a small size. To encrypt a file f to N employees $\mathrm{E}_1, \ldots, \mathrm{E}_N$ in *Company A*, *X* runs the *One2ManyEnc* algorithm, which needs to compute one bilinear pairing of Q_0 and P_{Bob} and N point multiplication operations. Note that the computation for the bilinear map is independent of the file to be encrypted, and hence can be done once and for all. Given the ciphertext C_f, Bob runs the *UserDec* algorithm, which needs to execute one bilinear map operation, whereas E_i for $1 \leq i \leq N$ runs the *RecipientsDec* algorithm, which needs to execute the bilinear map operations for almost $\frac{N+1}{2} + 3$ times to recover f ($\frac{N+1}{2} + 1$ times for finding out rP_i). The length of the ciphertexts grows linearly with the number of the recipients (It will increase a element $\in \mathbb{G}_1$ as the number of the recipients increasing).

B. Security

We first provide an intuitive security argument. Recall that a confidential file f is encrypted in the form of $C_f = [U_0, U_1, \ldots, U_N, V] = [rP_0, rP_1, \ldots, rP_N, f \oplus H_2(\hat{e}(Q_0, rP_{Bob}))]$. Clearly, an adversary $\mathcal{A}$ needs to construct $\hat{e}(Q_0, rP_{Bob})$ to decrypt C_f.

Although Q_0 and P_{Bob} can be publicly available, due to the DHP assumption [1], $\mathcal{A}$ is unaware of the value of random mask r. In other words, $\mathcal{A}$ cannot construct $\hat{e}(Q_0, rP_{Bob})$ directly. However, due to the properties of the bilinear map, we have:

$$\hat{e}(Q_0, rP_{Bob}) = \hat{e}(mk_0P_0, rP_{Bob}) = \hat{e}(rP_0, mk_0P_{Bob}) = \hat{e}(U_0, S_{Bob})$$

That is to say, $\mathcal{A}$ can construct $\hat{e}(U_0, S_{Bob})$ instead of $\hat{e}(Q_0, rP_{Bob})$ to decrypt C_f. To create such a pairing, the adversary can only use private keys that were obtained in a security game, as defined in Section IV. According to the constraints in the security game, the adversary cannot request private keys of any employee in $\mathrm{E}_1, \ldots, \mathrm{E}_N$.

In the security game, the only occurrence of S_{Bob} is in the user private key, so the adversary has to use private keys requested for E_i outside $\mathrm{E}_1, \ldots, E_N$ for the pairing, yielding

$$\begin{aligned}
&\hat{e}(rP_0, S_i + \alpha) \\
&= \hat{e}(rP_0, S_{Bob} + mk_iP_i + \alpha) \\
&= \hat{e}(rP_0, S_{Bob})\hat{e}(rP_0, mk_iP_i))\hat{e}(rP_0, \alpha) \\
&= \hat{e}(U_0, S_{Bob})\hat{e}(mk_iP_0, rP_i))\hat{e}(rP_0, \alpha) \\
&= \hat{e}(U_0, S_{Bob})\hat{e}(Q_i, rP_i)\hat{e}(rP_0, \alpha)
\end{aligned}$$

for some α. To obtained $\hat{e}(U_0, S_{Bob})$, the value of $\hat{e}(Q_i, rP_i)$ has to be eliminated. However, the values of rP_i are unknown to the adversary, and cannot be constructed by the adversary also due to the DHP assumption. Therefore, the adversary cannot compromise the ciphertext. In Appendix A, we will provide a more thorough security proof of our construction under the BDH assumption.

VII. Conclusions

Cloud computing is one of the current most important and promising technologies. For the sake of enjoying a more comprehensive, scalable, and secure service, we proposed a scheme to enable the upper-level user to efficiently share the secure cloud storage services with multiple lower-level users. Using our scheme, a sender needs to encrypt a file only once, and store only one copy of the corresponding ciphertext in a cloud communicating with none of the recipients. But, the upper-level user and all the intended recipients can successfully recover the file using their own private keys. The proposed scheme is collusion resistant, in which any lower-level user, who is not an intended recipient, is unaware of any information in regards to the confidential file, even if all of them collude.

As discussed in previous sections, a lower-level user needs to execute the bilinear map operations for almost $\frac{N+1}{2} + 3$ times to decrypt a file ($\frac{N+1}{2} + 1$ times for finding out a part of the ciphertext that can be used to decrypt the file), which will cost too much computational power. One of the biggest advantages of cloud computing is that a user can retrieve the files stored in a cloud anytime and anywhere. If a lower-level user wants to retrieve the files when he is using a PDA with limited bandwidth, CPU, and memory, our scheme may not work well. We have an idea that enables a lower-level user E_i to send a short trapdoor to a CSP before retrieving files. The trapdoor can enable the CSP to find out rP_i without leaking any information about the file. If so, a lower-level user needs only to execute the bilinear map operations two times to decrypt the file, which will largely reduce the computational cost for decryption. In our future work, we will concentrate on solving this problem.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant Nos. 90718034 & 60773013, and the Hunan Provincial Natural Science Foundation of China for Distinguished Young Scholars under Grant No. 07JJ1010.

REFERENCES

[1] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of CRYPTO 2001*, volume 2139 of *LNCS*, pages 213-229.

[2] C. Gentry and A. Silverberg. Hierarchical ID-based cryptography. In *Proceedings of ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 548-566.

[3] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *Proceedings of EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 466-481.

[4] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *Proceedings of EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440-456.

[5] D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Proceedings of EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223-38.

[6] C. Gentry and S. Halevi. Hierarchical identity base encryption with polynomially many levels. In *Proceedings of TCC 2009*, volume 5444 of *LNCS*, pages.437-456.

[7] B. Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Proceedings of CRYPTO 2009*, volume 5677 of *LNCS*, pages 619-636.

[8] H. Haclgiimfi, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in database-service-provider model. In *Proceedings of ACM SIGMOD 2002*, pages 216-227.

[9] A. Weiss. Computing in the clouds. netWorker, 11(4): 16-25, Dec. 2007.

[10] Gartner identifies the top 10 strategic technologies for 2009, http://www.gartner.com/it/page.jsp?id=777212.

[11] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In *Proceedings of IEEE HPCC 2008*, pages 5-13.

[12] M. A. Vouk. Cloud computing-issues, research and implementations. In *Proceedings of ITI 2008*, pages 31-40.

[13] Q. Liu, G. Wang and J. Wu. An efficient privacy preserving keyword search scheme in cloud computing. In *Proceedings of IEEE CSE 2009/TrustCom 2009*, pages 715-720.

[14] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Proceedings of EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457-473.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of ACM CCS 2006*, pages 89-98.

[16] J . Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of IEEE ISSP 2007*, pages 321-334.

[17] M. Chase. Multi-authority attribute based encryption. In *Proceedings of TCC 2007*, volume 4392 of *LNCS*, pages 515-534.

[18] M. Chase and S. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of ACM CCS 2009*, pages 121-130.

[19] S. Yu, C. Wang, K. Ren, and W. Lou, Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings of IEEE INFOCOM 2010*, pages 15-19.

[20] S. Yu, K. Ren, and W. Lou. FDAC: Toward fine-grained distributed data access control in wireless sensor networks. In *Proceedings of IEEE INFOCOM 2009*, pages 19-25.

APPENDIX A
SECURITY PROOF

To analyze the security of the proposed scheme, we provide the following theorem, which shows that the ESC scheme is semantically secure if the BDH problem is assumed to be hard.

Theorem A.1: Suppose that algorithm $\mathcal{A}$ is an adversary that has the advantage ϵ of successfully attacking the ESC scheme. Suppose algorithm $\mathcal{A}$ specifies N employees as the recipients, and makes at most $q_{H_2} > 0$ hash queries to H_2 and at most $q_E > 0$ private key extraction queries. Then, there is an adversary $\mathcal{B}$ that breaks the BDH problem with the advantage at least $\epsilon' = 2\epsilon N^N / q_{H_2}(e(q_E + N))^N$, and a running time $O(time(\mathcal{A}))$. Here $e \approx 2.71$ is the base of the natural logarithm.

Proof: Let H_1 be a random oracle from $\{0,1\}^*$ to $\mathbb{G}_1^*$, and H_2 be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$. Algorithm $\mathcal{B}$ is given q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\hat{e}$, P_0, $\mu_0 = \alpha P_0$, $\mu_1 = \beta P_0$, and $\mu_2 = \gamma P_0$, where $< q, \mathbb{G}_1, \mathbb{G}_2, \hat{e} >$ are the outputs of a BDH parameter generator for a sufficiently large security parameter, P_0 is a generator of $\mathbb{G}_1$, α, β, and γ are random elements of $\mathbb{Z}_q^*$. Its goal is to output $D = e(g,g)^{\alpha\beta\gamma} \in \mathbb{G}_2$. Let D be the solution to the BDH problem. Algorithm $\mathcal{B}$ finds D by interacting with algorithm $\mathcal{A}$ as follows:

Setup: Algorithm $\mathcal{B}$ sets $Q_0 = \mu_0 = \alpha P_0$ and gives algorithm $\mathcal{A}$ $< q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P_0, Q_0, H_1, H_2 >$ as the system parameters, where H_1 and H_2 are controlled by algorithm $\mathcal{B}$ as described below.

H_1-Queries: Algorithm $\mathcal{B}$ maintains a list of tuples called H_1-List, in which each entry is a tuple of the form $(\text{ID-tuple}_j, \text{P-tuple}_j, \text{b-tuple}_j, \text{mk-tuple}_j, \text{c-tuple}_j)$. H_1-List is initially empty. Algorithm $\mathcal{B}$ first adds $\text{ID-tuple}_{Bob} = (\text{ID}_{Bob})$ to H_1-List as follows:

1) Pick a random element $mk_{Bob} \in \mathbb{Z}_q$.
2) Pick a random element $b_{Bob} \in \mathbb{Z}_q$.
3) Set $c_{Bob} = 1$.
4) Set $P_{Bob} = b_{Bob}\mu_1 = b_{Bob}\beta P_0$.
5) Put $((\text{ID}_{Bob}), (P_{Bob}), (b_{Bob}), (mk_{Bob}), (c_{Bob}))$ in H_1-List.

When algorithm $\mathcal{A}$ queries H_1 at a point ID-tuple_i, algorithm $\mathcal{B}$ responds as follows:

1) If $\text{ID-tuple}_i = \text{ID-tuple}_j$ where ID-tuple_j already appears on H_1-List in the form of $(\text{ID-tuple}_j, \text{P-tuple}_j, \text{b-tuple}_j, \text{mk-tuple}_j, \text{c-tuple}_j)$, then algorithm $\mathcal{B}$ returns $H_1(\text{ID-tuple}_i) = \text{P-tuple}_j \in \mathbb{G}_1$ to algorithm $\mathcal{A}$.
2) Otherwise, algorithm $\mathcal{B}$ picks two random elements mk_i and $b_i \in \mathbb{Z}_q$.
3) Pick a random coin $c_i \in \{0,1\}$ so that $Pr[c_i = 0] = \delta$ for some δ that will be determined later.
4) If $c_i = 1$, algorithm $\mathcal{B}$ sets $P_i = b_i P_0$.
5) If $c_i = 0$, algorithm $\mathcal{B}$ sets $P_i = b_i P_0 - mk_{Bob}^{-1} P_{Bob}$, where mk_{Bob}^{-1} is the inverse of mk_{Bob} modulo q.
6) Put $((\text{ID}_{Bob}, \text{ID}_i), (P_{Bob}, P_i), (b_{Bob}, b_i), (mk_{Bob}, mk_i), (c_{Bob}, c_i))$ in H_1-List and return $H_1(\text{ID-tuple}_i) = (P_{Bob}, P_i) \in \mathbb{G}_1$ to algorithm $\mathcal{A}$.

Note that these values are always chosen uniformly in $\mathbb{G}_1$, and are independent of algorithm $\mathcal{A}$'s view as required.

H_2-Queries: Algorithm $\mathcal{B}$ maintains a list of tuples called H_2-List, in which each entry is a tuple of the form (T_j, V_j). The list is initially empty. When algorithm $\mathcal{A}$ queries H_2 at a point of T_i, algorithm $\mathcal{B}$ checks if $T_i = T_j$, where T_j already appears on H_2-List in the form of (T_j, V_j). If so, algorithm $\mathcal{B}$ responds to algorithm $\mathcal{A}$ with $H_2(T_i) = V_j$. Otherwise, algorithm $\mathcal{B}$ picks a random string $V_i \in \{0,1\}^n$, adds the tuple (T_i, V_i) to H_2-List, and responds to algorithm $\mathcal{A}$ with $H_2(T_i) = V_i$.

Phase 1: At any time, algorithm $\mathcal{A}$ may make a private key extraction query on any ID-tuple_i except for ID-tuple_{Bob}.

Algorithm $\mathcal{B}$ responds to this query as follows:

1) Run the H_1-Queries algorithm to obtain the appropriate tuple (ID-tuple$_i$, P-tuple$_i$, b-tuple$_i$, mk-tuple$_i$, c-tuple$_i$) in H_1-List.
2) If $c_i = 1$, then algorithm $\mathcal{B}$ reports failure to algorithm $\mathcal{A}$ and terminates the interaction. Otherwise, we know that $P_i = b_i P_0 - mk_{Bob}^{-1} P_{Bob}$ where mk_{Bob}^{-1} is the inverse of mk_{Bob} modulo q.
3) Set Q-tuple$_i$ $= (Q_{Bob}, Q_i)$, where $Q_{Bob} = mk_{Bob}\mu_0 = mk_{Bob}\alpha P_0$ and $Q_i = mk_i P_0$.
4) Set $S_i = mk_{Bob} b_i \mu_0 = mk_{Bob} b_i \alpha P_0$ and return (S_i, Q-tuple$_i$) as the private key corresponding to ID-tuple$_i$ to algorithm $\mathcal{A}$.

Observe that: $Q_0 = \mu_0 = \alpha P_0$ and $Q_{Bob} = mk_{Bob}\mu_0 = mk_{Bob}\alpha P_0$, so the root master key is α, whereas Bob's master key is $mk_{Bob}\alpha$. Although algorithm $\mathcal{B}$ does not know the values of α and $mk_{Bob}\alpha$, it can output a correct private key for ID-tuple$_i$.

By definition, the value of S_i should be $\alpha P_{Bob} + \alpha mk_{Bob} P_i$.

$$\begin{aligned} S_i &= \alpha P_{Bob} + \alpha mk_{Bob} P_i \\ &= \alpha P_{Bob} + \alpha mk_{Bob}(b_i P_0 - mk_{Bob}{}^{-1} P_{Bob}) \\ &= \alpha P_{Bob} + \alpha mk_{Bob} b_i P_0 - \alpha P_{Bob} \\ &= mk_{Bob} b_i \mu_0 \end{aligned}$$

Therefore, (S_i, Q-tuple$_i$) is the correct private key for ID-tuple$_i$.

Challenge: Once algorithm $\mathcal{A}$ decides that Phase 1 is over, it outputs N, ID-tuples: ID-tuple$_1$, ..., ID-tuple$_N$, and two plaintext f_0, f_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ responds as follows:

1) Run the H_1-Queries algorithm to obtain the appropriate tuples (ID-tuple$_i$, P-tuple$_i$, b-tuple$_i$, mk-tuple$_i$, c-tuple$_i$) in H_1-List, for $1 \le i \le N$.
2) If $c_i = 0$, then algorithm $\mathcal{B}$ reports failure to algorithm $\mathcal{A}$ and terminates the interaction. Otherwise, we know that $P_{Bob} = b_{Bob}\beta P_0$ and $P_i = b_i P_0$, for $1 \le i \le N$.
3) Pick a random $b \in \{0,1\}$ and a random string $J \in \{0,1\}^n$, and give the ciphertext $C = (\mu_2, b_1\mu_2, \ldots, b_N\mu_2, J)$ to algorithm $\mathcal{A}$.

Note that this challenge implicitly defines $J = f_b \oplus H_2(\hat{e}(\gamma\mu_0, P_{Bob}))$. In other words:

$$\begin{aligned} J &= f_b \oplus H_2(\hat{e}(\gamma\alpha P_0, b_{Bob}\beta P_0)) \\ &= f_b \oplus H_2(\hat{e}(P_0, P_0)^{\alpha\gamma\beta b_{Bob}}) \end{aligned}$$

By definition, Bob's private key is (S_{Bob}, Q-tuple$_{Bob}$), where $S_{Bob} = \alpha P_{Bob}$ and Q-tuple$_{Bob}$ $= (Q_{Bob}) = (mk_{Bob}\mu_0)$ $= (mk_{Bob}\alpha P_0)$. He computes $J \oplus H_2(\hat{e}(\mu_2, S_{Bob}))$ to recover the file f_b. Observe that:

$$\begin{aligned} J \oplus H_2(\hat{e}(\mu_2, S_{Bob})) &= J \oplus H_2(\hat{e}(\gamma P_0, \alpha P_{Bob})) \\ &= J \oplus H_2(\hat{e}(\alpha\gamma P_0, P_{Bob})) \\ &= J \oplus H_2(\hat{e}(\gamma\mu_0, P_{Bob})) = f_b \end{aligned}$$

By definition, E$_i$'s private key is (S_i, Q-tuple$_i$), where $S_i = \alpha P_{Bob} + mk_{Bob}\alpha P_i$ and Q-tuple$_i$ $= (Q_{Bob}, Q_i)$ for $1 \le i \le N$. He computes $J \oplus H_2(\hat{e}(\mu_2, S_i)/\hat{e}(Q_{Bob}, b_i\mu_2))$ to recover the file f_b. Observe that:

$$\begin{aligned} & J \oplus H_2(\hat{e}(\mu_2, S_i)/\hat{e}(Q_{Bob}, b_i\mu_2)) \\ &= J \oplus H_2(\hat{e}(\mu_2, \alpha P_{Bob} + mk_{Bob}\alpha P_i)/\hat{e}(Q_{Bob}, b_i\mu_2)) \\ &= J \oplus H_2(\hat{e}(\mu_2, \alpha P_{Bob})\hat{e}(\mu_2, mk_{Bob}\alpha P_i)/\hat{e}(Q_{Bob}, b_i\mu_2)) \\ &= J \oplus H_2(\hat{e}(\gamma P_0, \alpha P_{Bob})\hat{e}(Q_{Bob}, b_i\gamma P_0)/\hat{e}(Q_{Bob}, b_i\gamma P_0)) \\ &= J \oplus H_2(\hat{e}(\gamma\alpha P_0, P_{Bob})) \\ &= J \oplus H_2(\hat{e}(\gamma\mu_0, P_{Bob})) = f_b \end{aligned}$$

Hence, C is a valid ciphertext for f_b as required.

Phase 2. Algorithm $\mathcal{A}$ can continue issuing more private key extraction queries other than ID-tuple$_1$, ..., ID-tuple$_N$. Algorithm $\mathcal{B}$ responds as in Phase 1.

Guess: Algorithm $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b. At this point, algorithm $\mathcal{B}$ picks a random pair (T_i, V_i) from H_2-List, and outputs $\sqrt[b_{Bob}]{T_i}$ as the solution to D.

To complete the proof of *Theorem B.1*, we now show that algorithm $\mathcal{B}$ correctly outputs D with the probability at least $2\epsilon N^N / q_{H_2}(e(q_E + N))^N$. In the first place, we calculate the probability that algorithm $\mathcal{B}$ does not abort during the simulation. Suppose algorithm $\mathcal{A}$ makes a total of q_E private key extraction queries. Then, the probability that algorithm $\mathcal{B}$ does not abort in Phase 1 or 2 is δ^{q_E}. And the probability that it does not abort during the challenge step is $(1-\delta)^N$. Therefore, the probability that algorithm $\mathcal{B}$ does not abort during the simulation is $\delta^{q_E} \cdot (1-\delta)^N$. This value is maximized at $\delta_{opt} = 1 - N/(q_E + N)$. Using δ_{opt}, the probability that algorithm $\mathcal{B}$ does not abort is at least $(N/e(q_E + N))^N$. In the second place, we calculate the probability that algorithm $\mathcal{B}$ outputs the correct result in case algorithm $\mathcal{B}$ does not abort. Let Q be the event that algorithm $\mathcal{A}$ issues a query for V. If $\neg Q$, we know that the decryption of the ciphertext is independent of algorithm $\mathcal{A}$'s view. Let $Pr[b = b']$ be the probability that algorithm $\mathcal{A}$ outputs the correct result, therefore, in the real attack $Pr[b = b' | \neg Q] = 1/2$. Since algorithm $\mathcal{A}$ has the advantage ϵ, $|Pr[b = b' | \neg Q] - 1/2| \ge \epsilon$. According to the following formulae, we know $Pr[Q] \ge 2\epsilon$.

$$\begin{aligned} Pr[b = b'] &= Pr[b = b' | \neg Q] Pr[\neg Q] \\ &+ Pr[b = b' | Q] Pr[Q] \\ &\le 1/2 Pr[\neg Q] + Pr[Q] \\ &= 1/2 + 1/2 Pr[Q] \\ Pr[b = b'] &\ge Pr[b = b' | \neg Q] Pr[\neg Q] \\ &= 1/2 Pr[\neg Q] \\ &= 1/2 - 1/2 Pr[Q] \end{aligned}$$

Therefore, we have that $Pr[Q] \ge 2\epsilon$ in the real attack. Now we know that algorithm $\mathcal{A}$ will issue a query for V with the probability at least 2ϵ. That is to say, the probability that V appears in some pair on H_2-List is at least 2ϵ. Algorithm $\mathcal{B}$ will choose the correct pair with the probability at least $1/q_{H_2}$, thus algorithm $\mathcal{B}$ produces the correct answer with the probability at least $2\epsilon/q_{H_2}$. Since algorithm $\mathcal{B}$ does not abort with the probability at least $(N/e(q_E + N))^N$, we see that algorithm $\mathcal{B}$'s success probability is at least $\epsilon' = 2\epsilon N^N / q_{H_2}(e(q_E + N))^N$, as required. ■