A Secure Hierarchical Deduplication System in Cloud Storage

Xin Yao*†, Yaping Lin*, Qin Liu*, Yanchao Zhang†
*College of Computer Science and Electronic Engineering, Hunan University,
Changsha, Hunan, 410082, China
Email: {xinyao, yplin, gracelq628}@hnu.edu.cn
†School of Electrical, Computer and Energy Engineering, Ira A. Fulton Schools of Engineering,
Arizona State University, Tempe, AZ 85287-5706, USA.
Email: yczhang@asu.edu

***Abstract*—Data deduplication is commonly adopted in cloud storage services to improve storage utilization and reduce transmission bandwidth. It, however, conflicts with the requirement for data confidentiality offered by data encryption. Hierarchical authorized deduplication alleviates the tension between data deduplication and confidentiality and allows a cloud user to perform privilege-based duplicate checks before uploading the data. Existing hierarchical authorized deduplication systems permit the cloud server to profile cloud users according to their privileges. In this paper, we propose a secure hierarchical deduplication system to support privilege-based duplicate checks and also prevent privilege-based user profiling by the cloud server. Our system also supports dynamic privilege changes. Detailed theoretical analysis and experimental studies confirm the security and high efficiency of our system.**

I. Introduction

Cloud storage services are increasingly used for backing up enterprise and personal data. One critical challenge for cloud storage services is the sheer amount of duplicate data. Specifically, there are nearly 68% duplicate data on standard file systems and up to 90-95% on backup applications [1]. To improve storage utilization, cloud storage services commonly resort to *data deduplication* techniques which allow the cloud to store only a single copy of duplicate data and provide links to that copy whenever needed. Data deduplication can be on the file level or block level. The former refers to storing a single copy of each file [2], while the latter means segmenting files into blocks and storing only a copy of each block [3], [4]. Block-level deduplication can thus eliminate duplicate data blocks in non-identical files. Apart from improving storage efficiency, data deduplication can save tremendous bandwidth when outsourcing (retrieving) data to (from) the cloud storage. For simplicity, we focus on file-level deduplication in this paper, but our schemes can be easily extended to support block-level deduplication.

Data deduplication unfortunately leads to the tension between storage efficiency and data confidentiality. Specifically, cloud users have long been worrying about the confidentiality of their sensitive data hosted by cloud storage servers which cannot be completely trusted. A straightforward remedy to data confidentiality concerns is to let each cloud user outsource encrypted data to the cloud server which does not know the decryption key. Traditional encryption, however, requires each user to use a different key for data encryption. As a result, the same data at different users will appear as totally different ciphertexts. Data deduplication techniques, however, require the cloud server to identify identical data. There is thus a natural conflict between data deduplication for storage efficiency and data encryption for data confidentiality.

There has been some effort to alleviate the above conflict. Douceur *et al.* [2] proposed Convergent Encryption (CE), in which the encryption/decryption key is derived as the cryptographic fingerprint of the data content. CE can enable cross-user data deduplication because the ciphertexts of the same data at different users are now the same. In a typical cloud storage system with both data deduplication and CE enabled [5]–[7], a user first sends the fingerprint of the data to the cloud server for duplicate check. If the cloud server does not find the fingerprint in the system, it asks for the entire copy of the data, and the user uploads the ciphertext generated under CE and retains the key in a safe place. If the same fingerprint is found in the system, the server returns to the user a pointer to the corresponding ciphertext that has been uploaded by this user or someone else. If the same data are needed later, the user uses the pointer to retrieve the ciphertext from the server and decrypt it with the saved CE key. To prevent unauthorized data retrieval, a Proof-of-Ownership protocol [10] was later proposed to require that a user prove his ownership of the data he requests downloading from the server.

The seminal work in [11] supports CE-based deduplication with hierarchical privileges. In their system, each user is assigned a set of privileges (e.g., CEO, Department Head, Project Lead, and Engineer). Each file on the storage server is encrypted with the CE key and is also associated with a set of cryptographic file tokens which together specify the privileges a user must possess to perform a duplicate check for this file. The storage server only answers the duplicate checks from the users who can provide the right file tokens for the requested files. Their scheme, however, allows the storage server to infer which users have higher privileges: the higher privilege a user has, the more file tokens contained in his duplicate-check request. The disclosure of such privilege information is highly undesirable in many scenarios and may lead to more targeted

attacks on high-profile users.

In this paper, we present a novel deduplication system to prevent the above privilege-based profiling attack. Our system targets a typical enterprise which uses the cloud storage service. Each employee in the enterprise is assigned a privilege on the level of 1 to ℓ, where privilege i is higher than privilege $i+1$. In our system, each data file is also encrypted with the CE-based key and accompanied by a cryptographic file token corresponding to a privilege level $j \in [1, \ell]$ specified by the data owner. Any duplicate-check request must contain one and only one file token of some privilege, and the storage server ignores all the duplicate-check requests which do not contain a valid query token with privilege $i \leq j$. Since only one file token is included in any duplicate-check request, the users with higher privileges can no longer be identified in contrast to the seminal work [11].

The contributions of this paper are summarized as follows.

- We propose a secure hierarchical deduplication system to support privilege-based duplicate checks and also prevent privilege-based user profiling by the storage server. Our system is based on a novel Hierarchial Privilege-Based Predicate Encryption (HPBPE) scheme, which is derived from hierarchical predicate encryption [12]. HPBPE is used to generate the cryptographic file tokens for duplicate checks without disclosing the users' privilege information to the storage server.
- We propose an enhanced scheme called HPBPE-R to support dynamic privilege changes, e.g., promotion and demotion commonly seen in an enterprise environment.
- We conduct rigorous security analysis about HPBPE and HPBPE-R and confirm their high efficiency by extensive experiments on a real data set (the Nursery data set).

The rest of this paper is organized as follows. Section II gives the problem formulation. Section III discusses the cryptographic preliminaries underlying our system. Section IV outlines HPBPE. V details the construction of HPBPE and analyzes its security and performance. Section VI presents the construction of HPBPE-R as well as its performance and security analysis. Section VII evaluates HPBPE and HPBPE-R by extensive experiments. Section VIII reviews the related work. Section IX concludes this paper.

II. Problem Formulation

In this section, we formally define the system model and threat model as well as our design goals.

A. System Model

In the secure hierarchical deduplication system, there are three entities (shown in Fig. 1): *Enterprise*, *Deduplication Provider*, and *Cloud Storage*. The enterprise is an entity that demands its employees to outsource encrypted data files to the cloud storage, and it wants to employ file-level data deduplication to save its bandwidth and storage expenses. In contrast to traditional CE-based deduplication systems, our system employs a deduplication provider to facilitate privilege-based duplicate checks, which is also used and called the private cloud in [11]. The cloud storage stores encrypted data and returns them to authorized users.

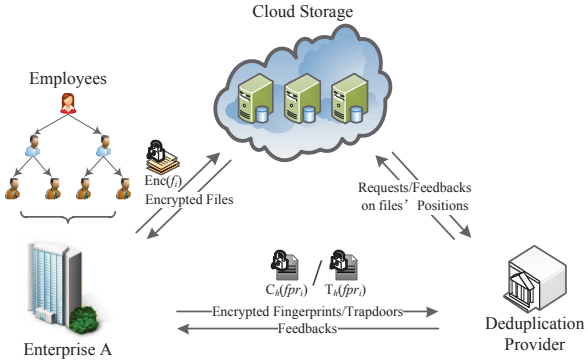


Fig. 1: System model.

Each system user is an enterprise employee and is assigned a privilege on the level of 1 to ℓ, where privilege i is higher than privilege $i+1$. For example, the privileges can be defined according to job positions (e.g., CEO, Department Head, Project Manager, Engineer). Each encrypted file on the cloud storage is associated with a cryptographic file token corresponding to a privilege level $j \in [1, \ell]$ specified by the data owner who uploaded the entire file. A valid duplicate-check request for the same file must contain one file token of privilege $i \leq j$ (i.e., a higher or equal privilege).

We use an example to outline the system operations. Assume that Alice is of privilege $i \in [1, \ell]$ and wants to back up a file to the cloud storage. To avoid uploading redundant data to the file server, Alice generates the fingerprint (i.e., the hash value) of the file and then a file token with our HPBPE (or HPBPE-R) algorithm which takes the file fingerprint and Alice's privilege-based private key as input. Subsequently, Alice submits a duplicate-check request containing the file token to the deduplication provider. With our HPBPE (or HPBPE-R) algorithm, the deduplication provider can verify whether the submitted token matches an existing token in its repository for the same file (i.e., same fingerprint) and with privilege $i \leq j$. If the duplicate check succeeds, the deduplication provider requests a data pointer to the encrypted file from the cloud storage, asks Alice not to upload the file, and finally give her the data pointer. If the file is needed later on, Alice can use the data pointer to download the encrypted file from the cloud storage and do the decryption. If the duplicate check fails, Alice uploads the encrypted file to the cloud storage and also the encrypted file fingerprint to the deduplication server to enable subsequent duplicate checks for the same file by herself or other users. The communication channels among Alice, the deduplication server, and the cloud storage can be secured and authenticated via traditional TLS-like schemes to prevent illegitimate access to the overall system.

B. Threat Model

We assume that the cloud storage is untrusted (e.g., due to a compromised cloud employee) and may want to know the data content. So the users must encrypt their files before uploading them to the cloud storage. In addition, the cloud storage may try to inflate the bandwidth and storage charges to the enterprise by not performing proper duplicate checks. Therefore, we cannot rely on the cloud storage performing duplicate checks, which can also prevent the cloud storage from knowing the privileges of different users.

The deduplication provider is assumed to be a semi-trusted honest-but-curious entity. In particular, it runs our system operations faithfully, but it may be interested in inferring the file content, the file fingerprint, which file (fingerprint) a duplicate-check request is for, and the requesting user's privilege for each duplicate-check request.

Finally, some users may be malicious and try to access the data beyond their privileges. They may also try to fake duplicate-check requests of higher privileges so as to infer whether some files have been stored in the cloud storage. We also assume that the deduplication provider does not collude with malicious cloud users.

C. Design Goals

Our system is designed with the following objectives.

- File privacy: Each file stored on the cloud storage must be encrypted and can only be decrypted by the authorized users knowing the correct key.
- Fingerprint privacy: Each file fingerprint must be encrypted with a privilege-based key before being submitted to the deduplication server, and it cannot be decrypted by the deduplication server or any user whose privilege is below the specified one.
- Query privacy: The deduplication provider cannot infer which ciphertext fingerprint any duplicate-check request is for or the requesting user's privilege information.
- Authorized search: No user can fake a duplicate-check request higher than his privilege, and the deduplication server can detect and ignore all unauthorized duplicate-check requests.
- Dynamic privileges: Our system should have secure and efficient support for the users' privileges, e.g., in case of promotion, demotion, and employment termination.
- Efficiency: Our system should be highly efficient.

III. Preliminaries

In this section, we present some cryptographic definitions and assumptions, which closely follow those in [12]. The main notation used in this paper are listed in Table I.

Let ($\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$) be three cyclic groups of prime order q. Assume that g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is defined as a non-degenerate bilinear pairing operation, and $e(g_1, g_2)$=$g_T \neq 1$. Notice that the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are multiplication, and this construction also supports *symmetric* pairing groups, i.e., $\mathbb{G}_1$=$\mathbb{G}_2$. Some definitions are described as follows:

TABLE I: Main Notation

Symbol	Definition
φ, χ	the number of the files and the employees
q	A large prime
$\mathbb{G}_i$, i=$\{1,2,3\}$	Three cyclic groups of order q
g_i, i=$\{1,2,T\}$	The group element of $\mathbb{G}_i$, i=$\{1,2,T\}$
$\mathbb{V}$ and $\mathbb{V}^*$	Two vector spaces
$\mathbf{x}$ and $\mathbf{y}$	Two vectors $(g_1^{x_1}, \cdots, g_1^{x_n})$ and $(g_2^{y_1}, \cdots, g_2^{y_n})$
$\mathbb{A}$ and $\mathbb{A}^*$	Two canonical bases of $\mathbb{V}$ and $\mathbb{V}^*$
$\vec{x}$/$\vec{v}$	Plaintext/Predicate vector
α and β	The size of elements in $\mathbb{G}_i$, i=$\{1,2,T\}$ and q

Definition 1. (*Vector Spaces $\mathbb{V}$ and $\mathbb{V}^*$*). *Vector Spaces* $\mathbb{V} = \underbrace{\mathbb{G}_1 \times \cdots \times \mathbb{G}_1}_{N}$ *and* $\mathbb{V}^* = \underbrace{\mathbb{G}_2 \times \cdots \times \mathbb{G}_2}_{N}$, $\mathbb{G}_i$ $(i = 1, 2)$ *are both including N-dimensional vectors. For vectors* $\boldsymbol{x} \in \mathbb{V}$ *and* $\boldsymbol{y} \in \mathbb{V}^*$, $\boldsymbol{x}$ *is expressed as* $(g_1^{x_1}, \cdots, g_1^{x_N})$ *and* $\boldsymbol{y}$ *is denoted as* $(g_2^{y_1}, \cdots, g_2^{y_N})$, *where* $x_i, y_i \in \mathbb{F}_q$ *for* $i \in [1, N]$.

Definition 2. (*Canonical Bases $\mathbb{A}$ and $\mathbb{A}^*$*). $\mathbb{A} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_N)$ *and* $\mathbb{A}^* = (\boldsymbol{a}_1^*, \cdots, \boldsymbol{a}_N^*)$ *are canonical bases of* $\mathbb{V}$ *and* $\mathbb{V}^*$, *respectively. Therein,* $\boldsymbol{a}_1 = (g_1, 1, \cdots, 1)$, $\boldsymbol{a}_2 = (1, g_1, \cdots, 1)$, $\cdots$, $\boldsymbol{a}_N = (1, \cdots, 1, g_1)$, *while* $\boldsymbol{a}_1^* = (g_2, 1, \cdots, 1)$, $\boldsymbol{a}_2^* = (1, g_2, \cdots, 1)$, $\cdots$, $\boldsymbol{a}_N^* = (1, \cdots, 1, g_2)$.

Definition 3. (*Pairing Operation*). *For* $\boldsymbol{x} \in \mathbb{V}$ *and* $\boldsymbol{y} \in \mathbb{V}^*$, *the pairing operation* $e(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{N} e(g_1^{x_i}, g_2^{y_i}) = e(g_1, g_2)^{\sum_{i=1}^{N} x_i \cdot y_i} = g_T^{\vec{x} \cdot \vec{y}} \in \mathbb{G}_T$.

Definition 4. (*Dual Pairing Vector Spaces* (*DPVS*)). *Each element of the tuple* $(q, \mathbb{V}, \mathbb{V}^*, \mathbb{G}_T, \mathbb{A}, \mathbb{A}^*)$ *is of a prime order q. Vector spaces* $\mathbb{V}$ *and* $\mathbb{V}^*$ *over* $\mathbb{F}_q$ *are both N-dimensional.* $\mathbb{G}_T$ *is a cyclic group of a prime order q. These elements and canonical bases* $\mathbb{A}$ *and* $\mathbb{A}^*$ *satisfy the following conditions:*

1) *Non-degenerate bilinear pairing: It can polynomial-time compute nondegenerate bilinear pairing* $e : \mathbb{V} \times \mathbb{V}^* \to \mathbb{G}_T$, *i.e.,* $e(a\boldsymbol{x}, b\boldsymbol{y}) = e(\boldsymbol{x}, \boldsymbol{y})^{ab}$ *and if* $e(\boldsymbol{x}, \boldsymbol{y}) = 1$ *for all* $\boldsymbol{x} \in \mathbb{V}$, *then* $\boldsymbol{y}$=$\boldsymbol{0}$.
2) *Dual orthonormal bases: For* $\boldsymbol{a}_i \in \mathbb{A}$ *and* $\boldsymbol{a}_j^* \in \mathbb{A}^*$, *the non-degenerate bilinear pairing e satisfies* $e(\boldsymbol{a}_i, \boldsymbol{a}_j^*) = g_T^{\delta_{i,j}}$ *for* $\forall i, j \in [1, n]$. *If i=j,* $\delta_{i,j}$*=1, otherwise 0, and* $g_T \neq 1 \in \mathbb{G}_T$.
3) *Distortion maps: There exist polynomial-time computable endomorphisms* $\phi_{i,j} \in \mathbb{V}$ *and* $\phi_{i,j}^* \in \mathbb{V}^*$ *satisfying* $\phi_{i,j}(\boldsymbol{a}_j) = \boldsymbol{a}_i$, $\phi_{i,j}(\boldsymbol{a}_k) = \boldsymbol{0}$ *and* $\phi_{i,j}(\boldsymbol{a}_j^*)$=$\boldsymbol{a}_i^*$, $\phi_{i,j}(\boldsymbol{a}_k^*) = \boldsymbol{0}$ *if k* $\neq$ *j.* $\phi_{i,j}$ *and* $\phi_{i,j}^*$ *are referred to as distortion maps.*

Definition 5. (*Hierarchical Privilege-Based Predicate* (*HPBP*)). *For a positive integers tuple* $\vec{\psi}$=$(n, \ell; \psi_1, \cdots, \psi_\ell)$ $(\psi_0 = 0 < \psi_1 < \psi_2 < \cdots < \psi_\ell = n)$, $\sum_t = \mathbb{F}_q^{\psi_\ell - \psi_{\ell-1}} \setminus \{\vec{0}\}$ $(t = 1, \cdots, \ell)$ *is denoted as the set of privileges, and* $\sum = \cup_{t=1}^{\ell} (\sum_1 \times \cdots \times \sum_t)$ *means the hierarchical privileges* $(\sum_i \cap \sum_j \neq \phi$, *iff* $i = j)$. *Then, the hierarchical privilege-based predicate is defined as* $f_{(\vec{v}_1, \cdots, \vec{v}_l)}$ *for a hierarchical privilege* $(\vec{x}_1, \cdots, \vec{x}_h) \in \sum$ *as* $f_{(\vec{v}_1, \cdots, \vec{v}_h)}(\vec{x}_1, \cdots, \vec{x}_\ell) = 1$ *iff* $h \leq \ell$ *and* $\vec{x}_i \cdot \vec{v}_i = 0$ *for* $1 \leq i \leq \ell$.

IV. Outline of Hierarchical Privilege-Based Predicate Encryption (HPBPE)

A. Overview

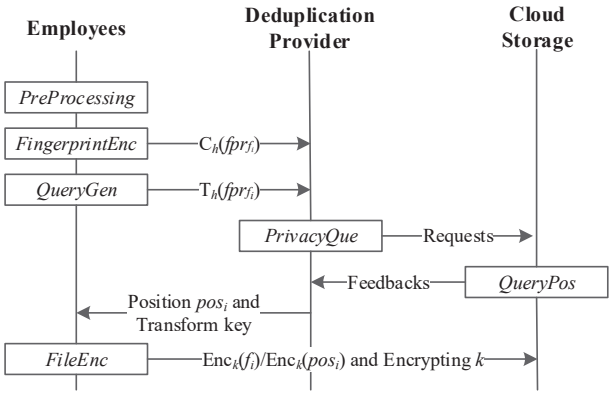


Fig. 2: The working process of HPBPE scheme.

We assume that the enterprise has a hierarchical privilege structure consisting of ℓ levels from 1 to ℓ. A larger privilege level corresponds to a lower privilege; i.e., the employees on the *i*-th level have higher privileges than those on (*i*+1)-th level. Figure 2 depicts the basic operations in HPBPE. Assume that a user (e.g., Alice) is of privilege $h \in [1, \ell]$ has w files to upload, denoted by $\{f_1, \ldots, f_w\}$. In the PreProcessing module, Alice generates a fingerprint for each file, denoted by (fpr_{f_1},$\cdots$,fpr_{f_w}) = (*hash*(f_1),$\cdots$,*hash*(f_w)), where *hash*($\cdot$) refers to any cryptographic hash function such as SHA-1 [17] or SHA-2 [18]. In the FingerprintEnc module, Alice uses a special key corresponding to privilege h to encrypt each ciphertext, denoted by ($C_h(fpr_{f_1})$,$\cdots$,$C_h(fpr_{f_w})$). In the QueryGen module, Alice uses another key corresponding to privilege h to generate a trapdoor for each fingerprint, denoted by ($T_h(fpr_{f_1})$,$\cdots$,$T_h(fpr_{f_w})$). To perform a duplicate check, Alice submits a query to the deduplication provider, which contains the trapdoors ($T_h(fpr_{f_1})$,$\cdots$,$T_h(fpr_{f_w})$). Each trapdoor corresponds to a file token related to both the file content and the user's privilege. In the PrivacyQue module, the deduplication provider checks whether each of the w trapdoors matches any ciphertext fingerprint in its database, which stores ciphertext fingerprints.

If a trapdoor does not match any existing ciphertext fingerprint, the deduplication provider instructs Alice to upload the corresponding file to the cloud storage. To do so, Alice generates a random symmetric key whereby to encrypt the file with a deterministic symmetric encryption algorithm. Alice also uses her public key to encrypt the symmetric key and then uploads the ciphertext key and file together to the cloud storage. Then Alice records the file position on the cloud storage and also the symmetric key in a safe place. Finally, Alice uploads the corresponding ciphertext fingerprint to the deduplication provider to enable subsequent duplicate checks for the same file.

If a trapdoor matches an existing ciphertext fingerprint, the deduplication provider contacts the cloud storage for the corresponding file position via the Querypos module. It then returns to Alice the file position as well as a transform key generated under the Proxy Re-Encryption scheme [19]. Then Alice cancels the submission of the corresponding file, and she also records the file position and the transform key. If Alice needs the file later, she can use the file position to download the encrypted data file and symmetric-key file from the cloud storage. The transform key allows Alice to decrypt the symmetric-key file with her own private key even if the symmetric key was encrypted under someone else's public key.

B. Hierarchical Privilege-Based Predicate Encryption

The HPBPE scheme is based on dual pairing vector spaces and comprises six polynomial time algorithms as follows.

1) **Setup**($1^\lambda, \vec{\mu}$): On input the security parameter 1^λ and format of hierarchy $\vec{\mu}$, it outputs the master key pair (*pk*,*sk*).
2) **GenKey**((*pk*, *sk*),$\vec{V}$): On input the master key pair (*pk*,*sk*) and predicate vectors $\vec{V} = (\vec{v}_1, \cdots, \vec{v}_h)$, and returns a corresponding secret key $sk_{(\vec{v}_1,\cdots,\vec{v}_h)}$.
3) **Delegate**(*pk*,$sk_{(\vec{v}_1,\cdots,\vec{v}_h)}$,$\vec{v}_{h+1}$): It takes the public key *pk*, the security key $sk_{(\vec{v}_1,\cdots,\vec{v}_h)}$ for privilege k, and the (*h*+1)-th predicate vector $\vec{v}_{h+1}$ as input, and outputs the security key $sk_{(\vec{v}_1,\cdots,\vec{v}_h,\vec{v}_{h+1})}$ for privilege *h*+1.
4) **FPREnc**(*pk*,$\vec{X}_h$,*fpr*): For a user with privilege h, it takes the master public key *pk*, privilege vectors $\vec{X}_h = (\vec{x}_1, \cdots, \vec{x}_h, \vec{x}_{h+1}^+, \cdots, \vec{x}_\ell^+)$ ($\vec{x}_i^+ \xleftarrow{U} \mathbb{F}_q$ and *h*+1$\leq i \leq \ell$) and the fingerprint *fpr* as the input, and returns ciphertext C_h(*fpr*).
5) **TCompute**($sk_{(\vec{v}_1,\cdots,\vec{v}_{h'})}$,*fpr*′): It takes the secret key $sk_{(\vec{v}_1,\cdots,\vec{v}_{h'})}$ ($1 \leq h' \leq \ell$) and the tested fingerprint *fpr*′ as the input, and output the trapdoor $T_{h'}$(*fpr*′).
6) **FPRTest**(C_h(*fpr*),$T_{h'}$(*fpr*′)): It takes the C_h(*fpr*) and $T_{h'}$(*fpr*′) as the input and tests whether *fpr* = *fpr*′ and $h' \leq h$.

We have the following remarks. (1) For the security parameter 1^λ and the privilege hierarchy $\vec{\mu}$=($n, d; \mu_1, \cdots, \mu_d$), the enterprise runs the **Setup** algorithm to obtain the master key pair (*pk*,*sk*). (2) To generate the private key for the employee with the *h*-th privilege level, the **GenKey** algorithm takes the master key pair (*pk*,*sk*) and the predicate vector $\vec{V}$ as the inputs, and outputs the secret key $sk_{(\vec{v}_1,\cdots,\vec{v}_h)}$. (3) To reduce the burden of the enterprise generating the secret key for each employee, the **Delegate** algorithm permits each user of privilege h to generate the private key for each user of privilege $h+1$. (4) To encrypt the fingerprint *fpr*, the user with the public key *pk* and the privilege vector $\vec{X}$ runs the **FPREnc** algorithm and outputs the ciphertext C_h(*fpr*). (5) For verifying whether the deduplication provider has a matching ciphertext fingerprint for *fpr*′ or not, the **TCompute** algorithm takes *fpr*′ and the private key of the current user $sk_{(\vec{v}_1,\cdots,\vec{v}_{h'})}$ and outputs the trapdoor $T_{h'}$(*fpr*′). (6) After receiving the trapdoor $T_{h'}$(*fpr*′), the deduplication provider tests it against

stored ciphertext fingerprints with the **FPRTest** algorithm. If the output is false, the deduplication server returns *null* to the user; otherwise, the file position on the cloud storage and the corresponding transform key are returned to the user.

V. Detailed Construction and Analysis of HPBPE

A. Detailed Construction of HPBPE

In this section, we detail the construction of HPBPE using dual pairing vector spaces (DPVS). Let $\mathbb{B}$=($\mathbf{b}_1,\cdots,\mathbf{b}_{n+3}$) and $\mathbb{B}^*$=($\mathbf{b}^*_1,\cdots,\mathbf{b}^*_{n+3}$) be two vector spaces in DPVS, where $\mathbb{B}$ and $\mathbb{B}^*$ are both n+3 dimensional spaces. The public parameters for DPVS are ($\mathbf{b}_1,\cdots,\mathbf{b}_n,\mathbf{b}_{n+1}+\mathbf{b}_{n+2},\mathbf{b}_{n+3}$). Each algorithm of HPBPE is detailed as follows.

1) **Setup**(1^λ,$\vec{\mu}$=(n,d;$\mu_1,\cdots,\mu_d$)). It firstly takes 1^λ and n+3 as the input of $\mathcal{G}_{ob}$, and randomly chooses the following parameters *param*, $\mathbb{B}$ and $\mathbb{B}^*$ (($param$,$\mathbb{B}$,$\mathbb{B}^*$)$\xleftarrow{R}$ $\mathcal{G}_{ob}(1^\lambda,n{+}3)$). Then the master private key *sk* is ($\vec{X}$,$\mathbb{B}^*$), and the master public key *pk* is (*param*,$\widehat{\mathbb{B}}$). Here $\widehat{\mathbb{B}}$ is ($\mathbf{b}_1,\cdots,\mathbf{b}_n,\mathbf{b}_{n+1}{+}\mathbf{b}_{n+2},\mathbf{b}_{n+3}$), and $\vec{X}$ consists of ($\vec{x}_1,\cdots,\vec{x}_\ell$), i.e., (($x_1,\cdots,x_{\mu_1}$),$\cdots$,($x_{\mu_{\ell-1}+1},\cdots,x_{\mu_\ell}$)).
2) **GenKey**((*pk*,*sk*),($\vec{v}_1,\cdots,\vec{v}_h$)). On input the predicate vectors (($v_1,\cdots,v_{\mu_1}$),$\cdots$,($v_{\mu_{h-1}+1},\cdots,v_{\mu_h}$)), *pk* and *sk*, it uniformly chooses σ_i and η from $\mathbb{F}_q$ for $\forall i \in (1,\cdots,h)$, and generates the private key $sk_{(\vec{v}_1,\cdots,\vec{v}_h)}$=$\mathbf{k}^*_h$ on the h-th privilege level as follows,

$$\mathbf{k}^*_h = \sum_{t=1}^{h}\sigma_t\Big(\sum_{i=\mu_{t-1}+1}^{\mu_t} v_i\mathbf{b}^*_i\Big) + \eta\mathbf{b}^*_{n+1} + (1-\eta)\mathbf{b}^*_{n+2}\ . \tag{1}$$

3) **Delegate**(pk,$sk_{(\vec{v}_1,\cdots,\vec{v}_h)}$,$\vec{v}_{h+1}$=($v_{\mu_h+1},\cdots,v_{\mu_{h+1}}$)). The HPBPE-delegate is the same as the HPE-delegate [12].
4) **FPREnc**(pk,$\vec{X}_\ell$,*fpr*). Given a fingerprint *fpr*, the user with privilege h chooses a random ζ from $\mathbb{F}_q$ and generates the ciphertext C_h(*fpr*)=(C_1,C_2) as follows.

$$\begin{aligned} C_1 =&\Big(\sum_{t=1}^{\ell}\delta_t\Big(\sum_{i=\mu_{t-1}+1}^{\mu_t} x_i\mathbf{b}_i\Big) + \zeta(\mathbf{b}_{n+1}+\mathbf{b}_{n+2}) \\ &+ \delta_{n+3}\mathbf{b}_{n+3}\Big)\cdot \mathit{fpr} \\ C_2 =&\, g_T^{\zeta} \end{aligned} \tag{2}$$

Besides, $\vec{X}_\ell$ is formed by concatenating ℓ-h random vectors ($\vec{x}^+_{h+1},\cdots,\vec{x}^+_\ell$) and $\vec{X}_h$; i.e., $\vec{X}_\ell$ is expressed as ($\vec{x}_1,\cdots,\vec{x}_h,\vec{x}^+_{h+1},\cdots,\vec{x}^+_\ell$). The ciphertext C_h(*fpr*)=(C_1,C_2) is uploaded to the deduplication provider.

5) **TCompute**($sk_{(\vec{v}_1,\cdots,\vec{v}_{h'})}$,*fpr'*). To verify whether the ciphertexts containing the information *fpr'* and corresponding to a privilege level no larger than *h'*, it firstly randomly chooses r from $\mathbb{F}_q$ and computes $T_{h'}$(*fpr'*) as follows,

$$\begin{aligned} T_{h'}(\mathit{fpr}') =&(\mathbf{k}^*_{h'} + r\cdot\mathbf{b}^*_{n+1} + (-r)\cdot\mathbf{b}^*_{n+2})\cdot(\frac{1}{\mathit{fpr}'}) \\ =&(\sum_{t=1}^{h'}\sigma_t(\sum_{i=\mu_{t-1}+1}^{\mu_t} v_i\mathbf{b}^*_i) + (\eta+r)\cdot\mathbf{b}^*_{n+1} \\ &+ (1-\eta-r)\mathbf{b}^*_{n+2})\cdot(\frac{1}{\mathit{fpr}'})\ . \end{aligned} \tag{3}$$

6) **FPRTest**(C_h(*fpr*),$T_{h'}$(*fpr'*)). It takes the ciphertext C_1 and the trapdoor $T_{h'}$(*fpr'*) as the input. The given ciphertext fingerprint contains the information *fpr'* and $h' \le h$ if the following equation holds,

$$e(C_1, T_{h'}(\mathit{fpr}')) = C_2\ . \tag{4}$$

Next, we prove the correctness of Equation 4. Let the ciphertext C_h(*fpr*) and the trapdoor $T_{h'}$(*fpr'*) be the outputs of *FPREnc*(*pk*,$\vec{X}_h$,*fpr*) and *TCompute*($sk_{(\vec{v}_1,\cdots,\vec{v}_{h'})}$,*fpr'*), respectively. The proof is described as follows.

Proof. 1) If $h' > h$:

$$\begin{aligned} e(C_1, T_{h'}(\mathit{fpr}')) =& g_T^{\sum_{1\le i\le h}\delta_i\sigma_i\vec{x}_i\cdot\vec{v}_i\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})}\cdot g_T^{\zeta\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})} \\ &\cdot g_T^{\sum_{h+1\le i\le h'}\delta_i\sigma_i\vec{x}^+_i\cdot\vec{v}_i\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})} \end{aligned} \tag{5}$$

$\because \forall i \in (h+1, h')$, $\vec{x}^+_i\cdot\vec{v}_i \neq 0$
$\therefore$ Whether $\mathit{fpr}' = \mathit{fpr}$ or $\mathit{fpr}' \neq \mathit{fpr}$

$$e(C_1, T_{h'}(fpr')) = g_T^{(\sum_{h+1\le i\le h'}\vec{x}^+_i\cdot\vec{v}_i+\zeta)\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})} \neq g_T^{\zeta} = C_2 \tag{6}$$

2) If $h' \le h$:

$$e(C_1, T_{h'}(\mathit{fpr}')) = g_T^{\sum_{1\le i\le h'}\delta_i\sigma_i\vec{x}_i\cdot\vec{v}_i\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})} \tag{7}$$

$\because \forall i \in (1, h')$, $\vec{x}_i\cdot\vec{v}_i = 0$

$\therefore$ When $\mathit{fpr}' = \mathit{fpr}$,

$$e(C_1, T_{h'}(\mathit{fpr}')) = g_T^{(\sum_{1\le i\le h'}\vec{x}^+_i\cdot\vec{v}_i+\zeta)\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})} = g_T^{\zeta} = C_2 \tag{8}$$

When $\mathit{fpr}' \neq \mathit{fpr}$,

$$e(C_1, T_{h'}(\mathit{fpr}')) = g_T^{(\sum_{1\le i\le h'}\vec{x}^+_i\cdot\vec{v}_i+\zeta)\cdot(\frac{\mathit{fpr}}{\mathit{fpr}'})} \neq g_T^{\zeta} = C_2 \tag{9}$$

□

B. Performance Analysis of HPBPE

In HPBPE, the lengths of the ciphertext, the secret key and the trapdoor are all $n+3$ group elements for n dimensional vectors. Next, we briefly analyze the computational and memory overhead of the following algorithms: **Setup**, **GenKey**, **Delegate**, **FPREnc**, **TCompute** and **FPRTest**.

Setup, **GenKey** and **Delegate**. In **Setup** and **GenKey** algorithms, the major computation is generating $\widehat{\mathbb{B}}$ and $\mathbb{B}^*$, which each contains $O(n_0^2)$=$O(n^2)$ point multiplications (exponentiations), where n_0=$n+3$ is the dimensional of the ECC

vector spaces in HPBPE and n is equivalent to the dimensional of $\vec{x}$ and $\vec{v}$, i.e., n=$|\vec{x}|$=$|\vec{v}|$. For the memory overhead, we assume that each element in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ is expressed by αB and that q is βB. Therefore, the memory overhead of pk is $\alpha[n_0(n_0\text{-}1)+3]$B, while that of sk is $n_0^2(\alpha+\beta)$B. For the **Delegate** algorithm, the computational overhead is the same with that in **GenKey** algorithm, i.e., ($O(n_0^2)$).

FPREnc and **TCompute**. In the **FPREnc** algorithm, the ciphertext C_1 is the result of the point multiplication between pk and the fingerprint fpr, and the ciphertext C_2 is also a point multiplication result. Thus, the encryption time is $O(n_0)$. For memory consumption, each ciphertext is of $\alpha(n_0+1)$B. In the **TCompute** algorithm, the process of generating a trapdoor is also a point multiplication between sk and the fingerprint fpr', and sk has the same dimensional of pk. So the computation overhead of **TCompute** is the same as that of **FPREnc** (i.e., ($O(n_0^2)$)), and the memory overhead is of αn_0B.

FPRTest. According to Equation 4, we can conclude that **FPRTest** needs to execute $n+3$ pairing operations. Thus, the computation overhead of **FPRTest** is $O(n+3)$.

C. Security Analysis of HPBPE

Definition 6. (*Decisional Subspace Problem with Irrelevant Dual Vector Tuples* (*IDSP*)). *For all security parameter* $\lambda \in \mathbb{N}$, IDSP *advantage of a probabilistic machine* $\mathcal{B}$ *is defined as follows:*

$$\begin{aligned}\mathrm{Adv}_{\mathcal{B}}^{\mathrm{IDSP}}(\lambda) =&|\Pr[\mathcal{B}(1^\lambda,\rho)\to 1|\rho \xleftarrow{R} \mathcal{G}_0^{\mathrm{IDSP}}(1^\lambda,n)]\\ &-\Pr[\mathcal{B}(1^\lambda,\rho)\to 1|\rho \xleftarrow{R} \mathcal{G}_1^{\mathrm{IDSP}}(1^\lambda,n)]|.\end{aligned} \tag{10}$$

The IDSP *assumption is that for any probabilistic polynomial-time adversary* $\mathcal{B}$, $Adv_{\mathcal{B}}^{\mathrm{IDSP}}(\lambda)$ *is negligible in* λ.

Definition 7. (*Decisional Subspace Problem with Relevant Dual Vector Tuples* (*RDSP*)). *The* RDSP *advantage of* $\mathcal{B}$, $\mathrm{Adv}_{\mathcal{B}}^{\mathrm{RDSP}}(\lambda)$, *and the* RDSP *assumption are defined similarly as in Definition 6.*

In this section, we prove that HPBPE satisfies the privacy requirements for the fingerprints, the duplicate-check queries, and the authorized search. In HPBPE scheme, the cloud storage can obtain the ciphertext of the data files encrypted with public-key encryption schemes, whose security depends on the previous public-key encryption schemes. While, deduplication provider mainly focuses the duplicate-check queries, and they can obtain the ciphertext of the fingerprint, and the trapdoor. Recall that the ciphertext for the fingerprint in HPBPE is generated by the algorithm **FPREnc**, the generation of the trapdoor relies on the algorithm **TCompute**, and the query depends on the algorithm **FPRTest**. Thus, all the privacy requirements can be satisfied if HPBPE is provably secure. We have the following theorem. Before describing the Theorem 1, we first introduce two assumptions as well as that in [12], i.e., Def. 6 and Def. 7

Theorem 1. *The proposed HPBPE scheme is selectively privilege-hiding against* CPA *under the RDSP and IDSP assumptions. For any adversary* $\mathcal{A}$, *there exist probabilistic machines* $\mathcal{B}_1$ *and* $\mathcal{B}_2$, *whose running times are essentially the same as that of* $\mathcal{A}$, *such that for any security parameter* λ,

$$Adv_{\mathcal{A}}^{\mathrm{HPBPE,IH}}(\lambda) \le Adv_{\mathcal{B}_1}^{\mathrm{RDSP}}(\lambda) + Adv_{\mathcal{B}_2}^{\mathrm{IDSP}}(\lambda) + 3\nu/q \tag{11}$$

where ν *is the number of adversary's queries.*

Proof. To prove Theorem 1, we deploy the following five games derived from [12], from *Game* 0 to *Game* 4. *Game* 0 is the original selectively privilege-hiding game. *Game* 1 is an extension of the concept of *Game* 0. The **Genkey** (instead of **Delegate**) algorithm in *Game* 2 presents the query of a delegated key. *Game* 3 means that the plaintext part of the target ciphertext and trapdoor is randomized. The randomness of the ciphertext part of the target ciphertext and trapdoor is in *Game* 4. According to the security analysis in [12], the gap between Games 1 and 2 is bounded by $3\nu/q$.

- *Game* 0: Primitive game [12].
- *Game* 1: Apart from the following condition, *Game* 1 is defined as the same with *Game* 1 in [12]. The adversary $\mathcal{A}$ gives the challenge plaintexts $fpr^{(0)}$ and $fpr^{(1)}$ to challenger $\mathcal{C}$, who computes the ciphertext (C_1,C_2) and the trapdoor $T_h(fpr)$ as follows and returns it to $\mathcal{A}$.

$$\begin{aligned}C_1 =&(\sum_{i=1}^{n} x_i^+\mathbf{b}_i + \zeta(\mathbf{b}_{n+1}+\mathbf{b}_{n+2}) + \delta_{n+3}\mathbf{b}_{n+3})\cdot fpr\\ C_2 =&g_T^{\zeta}\end{aligned} \tag{12}$$

$$\begin{aligned}T_h(fpr) =&(\sum_{i=1}^{h} v_i^+\mathbf{b}_i^*) + (\eta+r)\cdot\mathbf{b}_{n+1}^* + (1-\eta-r)\\ &\mathbf{b}_{n+2}^*)\cdot(\frac{1}{fpr})\end{aligned} \tag{13}$$

  where $\zeta, \delta_{n+3}, \eta$ and $r \xleftarrow{U} \mathbb{F}_q$.
- *Game* 2: It is defined as the same with that in [12].
- *Game* 3: *Game* 3 is the same as *Game* 2 except the ciphertext $C_n(fpr)$ consisting of C_1 and C_2 and the trapdoor $T_h(fpr)$ defined as follows.

$$\begin{aligned}C_1 =&(\sum_{t=1}^{n} x_i^+\mathbf{b}_i + \zeta_1\mathbf{b}_{n+1} + \zeta_2\mathbf{b}_{n+2} + \delta_{n+3}\mathbf{b}_{n+3})\cdot fpr\\ C_2 =&g_T^{\zeta}\end{aligned} \tag{14}$$

$$\begin{aligned}T_h(fpr) =&(\sum_{i=1}^{h} v_i^+\mathbf{b}_i^* + (\eta+r_1)\cdot\mathbf{b}_{n+1}^* + (1-\eta-r_2)\\ &\mathbf{b}_{n+2}^*)\cdot(\frac{1}{fpr})\end{aligned} \tag{15}$$

  where $\zeta, \zeta_1, \zeta_2, \delta_{n+3}, r_1, r_2, \eta \xleftarrow{U} \mathbb{F}_q$.
- *Game* 4: *Game* 4 and *Game* 3 are indistinguishable except the ciphertext C_1 and C_2 defined as follows.

$$\begin{aligned}C_1 =&(\sum_{t=1}^{n} u_i\mathbf{b}_i + \zeta_1\mathbf{b}_{n+1} + \zeta_2\mathbf{b}_{n+2} + \delta_{n+3}\mathbf{b}_{n+3})\cdot fpr\\ C_2 =&g_T^{\zeta}\end{aligned} \tag{16}$$

$$T_h(\mathit{fpr}) = (\sum_{i=1}^{n} w_i^{+}\mathbf{b}_i^{*} + (\eta + r_1)\cdot\mathbf{b}_{n+1}^{*} + (1-\eta-r_2)\,\mathbf{b}_{n+2}^{*})\cdot(\frac{1}{\mathit{fpr}}) \tag{17}$$

where $\zeta, \zeta_1, \zeta_2, \delta_{n+3}, r_1, r_2 \xleftarrow{U} \mathbb{F}_q$ and $\vec{u}$=$(u_1, \cdots, u_n)$ $\vec{w}$=$(w_1, \cdots, w_n) \xleftarrow{U} \mathbb{F}_q^n \setminus \{\vec{0}\}$.

□

We assume that $Adv_{\mathcal{A}}^{HPBPE,IH}$ for *Game* 0 is $Adv_{\mathcal{A}}^{(0)}$, and the advantage of $\mathcal{A}$ for *Game i* corresponds to $Adv_{\mathcal{A}}^{(i)}$, where $i \in [1,4]$. Since *Game* 1 is an extension of the concept of *Game* 0, $Adv_{\mathcal{A}}^{(0)}(\lambda)$ is the same with $Adv_{\mathcal{A}}^{(1)}(\lambda)$. Besides, $Adv_{\mathcal{A}}^{(4)}$ is equal to 0 by Lemma 4 in [12].

According to the analysis of evaluating the gaps between pairs of $Adv_{\mathcal{A}}^{(i)}(\lambda)$. We conclude that $Adv_{\mathcal{A}}^{HPBPE,IH}(\lambda)$=$Adv_{\mathcal{A}}^{(0)}(\lambda)$= $Adv_{\mathcal{A}}^{(1)}(\lambda) \leq \sum_{i=1}^{3} |Adv_{\mathcal{A}}^{(i)}(\lambda) - Adv_{\mathcal{A}}^{(i+1)}(\lambda)| + Adv_{\mathcal{A}}^{(4)}(\lambda) \leq Adv_{\mathcal{A}}^{RDSP}(\lambda) + Adv_{\mathcal{A}}^{IDSP}(\lambda) + 3\nu/q$.

VI. Hierarchical Privilege-Based Predicate Encryption with Revocation (HPBPE-R)

HPBPE only considers static privileges. In practice, an enterprise user may experience promotion, demotion, or even employment termination, in which case his data access privilege will change. In this section, we present a Hierarchical Privilege-Based Predicate Encryption with Revocation (HPBPE-R) scheme to support dynamic privilege changes.

A. Overview of HPBPE-R

HPBPE-R considers privilege promotions and also demotions. Recall that HPBPE allows a user of a higher privilege to perform duplicate checks for and retrieve the files of the same privilege or lower. If a user is promoted, HPBPE-R only needs to issue the public and private keys of the promoted privilege to the user, and nothing else needs to be done. For example, if *Alice* is currently on the privilege level h and is promoted to the privilege level $h-1$. In HPBPE-R, Alice just needs to be issued the public and private keys of privilege $h-1$, i.e., pk_{h-1},sk_{h-1}). In contrast, the privilege demotion is far more complicated, as it involves all the users and ciphertext on the same privilege level or lower. For instance, assume that Alice is demoted from the privilege level h to $h+1$. In HPBPE-R, all the users on the privilege levels from h to ℓ must be issued new privilege-based public and private keys, and all the ciphertext data files and fingerprints on the privilege levels from h to ℓ also need to be updated. In what follows, we detail the HPBPE-R scheme for privilege demotion.

B. Construction of HPBPE-R for Privilege Demotion

The HPBPE-R algorithm for privilege demotion consists of four algorithms: **UpdatePrivilege**, **CreateUpdateKey**, **ReEncrypt** and **UpdateSK**. After receiving a demotion order, the enterprise needs to update the privilege vectors on and below the demoted privilege level. For example, if the original privilege vector is $\vec{X}$=$(\vec{x}_1,\cdots,\vec{x}_\ell)$ and the demoted employee is of privilege *h*, the updated privilege vector $\vec{X}'$ is $(\vec{x}_1,\cdots,\vec{x}_{h-1},\vec{x}'_h,\cdots,\vec{x}'_\ell)$. Then the enterprise needs to run the **CreateUpdateKey** algorithm to create the updated keys for $\vec{X}'$ and upload these updated keys to the deduplication provider. After receiving these updated keys, the deduplication provider runs the **ReEncrypt** algorithm to update fingerprint ciphertexts. These algorithms are detailed as follows.

1) **UpdatePrivilege**: Suppose that the demoted user is on the *h*-th privilege level. To demote the user's privilege, the enterprise first updates the privilege vector $\vec{X}$=$(\vec{x}_1,\cdots,\vec{x}_\ell)$ to the latest privilege vector $\vec{X}'$=$(\vec{x}_1,\cdots,\vec{x}_{h-1},\vec{x'}_h,\cdots,\vec{x'}_\ell)$. As a result, the user privilege vectors $\vec{X}_h,\cdots,\vec{X}_\ell$ change to $\vec{X}'_h,\cdots,\vec{X}'_\ell$ such that $\vec{X}'_i$=$(\vec{x}_1,\cdots,\vec{x}_{i-1},\vec{x'}_i,\vec{x'}^{+}_{i+1},\cdots,\vec{x'}^{+}_\ell)$ for $i \in$[h,ℓ].
2) **CreateUpdateKey**: To generate an update key for updating pk_h to the latest public key pk'_h, the enterprise sets the update key $UpdateKey_{h,1}$ as pk'_h/pk_h and $UpdateKey_{h,2}$=$g_T^{\zeta'}$. Where $pk_h = \sum_{t=1}^{\ell}\delta_t \cdot (\sum_{i=\mu_{t-1}+1}^{\mu_t} \cdot x_i \cdot \mathbf{b}_i) + \zeta\cdot(\mathbf{b}_{n+1}+\mathbf{b}_{n+2}) + \delta_{n+3}\cdot\mathbf{b}_{n+3}$ and $pk'_h = \sum_{t=1}^{\ell}\delta'_t \cdot (\sum_{i=\mu_{t-1}+1}^{\mu_t} \cdot x_i \cdot \mathbf{b}_i) + \zeta'\cdot(\mathbf{b}_{n+1}+\mathbf{b}_{n+2}) + \delta'_{n+3}\cdot\mathbf{b}_{n+3}$.
3) **ReEncrypt**: Assume that the fingerprint ciphertext of the *h*-th privilege level is $C_h(\mathit{fpr}) = (C_1, C_2)$. The deduplication provider re-encrypts the ciphertext by setting C_2 to $UpdateKey_{h,2}$ and then computing $C'_1 = C_1 \cdot UpdateKey_{h,1}$, i.e., $C'_1 = C_1 \cdot pk'_h/pk_h$. The new ciphertext $C'_h(\mathit{fpr})$ = (C'_1,$UpdateKey_{h,2}$).
4) **UpdateSK**: $UpdateKey_i$ is the update key for updating pk_i to pk'_i, where $i \in [h,\ell]$. The enterprise also needs to update the private keys for the employees on the *h*-th and lower privilege levels. The latest predicate vector $\vec{V}'$ is generated by the latest privilege vector $\vec{X}'$, i.e., $\vec{V}'$ = $(\vec{v}_1,\cdots,\vec{v}_{h-1},\vec{v'}_h,\cdots,\vec{v'}_\ell)$. Thus, the employees' private keys are ($sk_{(\vec{v}_1,\cdots,\vec{v}_{h-1},\vec{v'}_h)}$,$\cdots$,$sk_{(\vec{v}_1,\cdots,\vec{v'}_\ell)}$)

Therefore, the ciphertext re-encrypted by the **ReEncrypt** algorithm under pk'_i/pk_i is the same as the ciphertext encrypted by the **FPREnc** algorithm under pk'_i ($i \in [h,\ell]$). Next, we prove the correctness of HPBPE-R for privilege demotion.

Proof. For a ciphertext $C_h(\mathit{fpr})$ consisting of C_1 and C_2, the deduplication provider receives the update key ($UpdateKey_1$,$UpdateKey_2$) (pk'_h/pk_h,$g_T^{\zeta'}$) from the enterprise. Then the deduplication provider computes $C_1 \cdot UpdateKey_1$, i.e., $C_1 \cdot pk'_h/pk_h$. In fact, C_1 is a $|\vec{X}|+3$ dimensional vector, and each element is the result of point multiplication of elements in pk_h and fingerprint *fpr*. So the result of C_1/pk_h is a vector consisting of ($|\vec{X}|+3$) *fpr*. Therefore, C'_1 generated by $C_1 \cdot pk'_h/pk_h$ is equivalent to that generated by the **GenKey** algorithm with the public key pk'_h. Finally, the deduplication provider assigns $UpdateKey_2$ to C_2. For the **FPRTest** algorithm, although the enterprise updates the privilege vector $\vec{X}$, it also updates the predicate vector $\vec{V}$ satisfying $\vec{x}_i \cdot \vec{v}_i$=0 ($i \in [1,\ell]$). Thus, the privacy-preserving duplicate-request query after revocation can also be run on the aforementioned **FPRTest** algorithm whose correctness has been proved in Section V. □

C. Performance Analysis of HPBPE-R

In HPBPE-R, the efficiency of **UpdatePrivilege** is rather straightforward. Thus, we mainly analyze the computational and memory overhead of the **CreateUpdateKey**, **ReEncrypt** and **UpdateSK** algorithms.

CreateUpdateKey. To generate the update key $UpdateKey_{h,1}$, the enterprise first generates the latest public key pk' whose computational overhead is equivalent to that in generating a pk ($O(n_0^2)$). Then the enterprise outputs the $UpdateKey_{h,1}$ by computing pk'/pk, for which the computational overhead is $O(n_0)$. Finally, the enterprise takes $O(1)$ to generate $UpdateKey_{h,2}$=$g_T^{\zeta'}$. Thus, generating *UpdateKey* needs $O(n_0^2+n_0+1)$ computations, where n_0=n+3. In terms of the memory overhead, $UpdateKey_{h,1}$ has the same memory overhead as the public key pk in ($\alpha[n_0(n_0-1)+3]$)B, and $UpdateKey_{h,2}$ cosumes αB. Thus, the memory overhead of **CreateUpdateKey** is $(\alpha[n_0(n_0-1)+3]+\alpha)$B.

ReEncrypt. To re-encrypt the ciphertext with the *UpdateKey*, the deduplication provider computes $C_{h,fpr} \cdot pk'/pk$ and assigns $UpdateKey_2$ to C_2. The computational overhead for $C_h(fpr)\cdot pk'/pk$ is $O(n+3)$, and the assignment overhead is $O(1)$. Thus, the computational overhead of **ReEncrypt** is $O(n+4)$.

UpdateSK. The enterprise just needs to generate a new private key sk' for the public key pk'. Therefore, the computational and memory overhead of **UpdateSK** are the same as those of the **GenKey** algorithm in HPBPE.

D. Security Analysis of HPBPE-R

Recall the update keys $UpdateKey_{h,1}$ and $UpdateKey_{h,2}$ generation in Section VI-B, we conclude that the security of the update keys is the same with that of the secret key in HPBPE scheme. Beside, the updating processing also follows the Equ. 2, so we enable deduce the security of **ReEncrypt** to that of **FPREnc**, which obeying the Proof V-C.

VII. Experiments

In this section, we evaluate the efficiency of HPBPE and HPBPE-R with six metrics: Setup Time, KeyGen Time, Encryption Time, Delegation Time, Search Time, and Update Time. We have implemented HPBPE and HPBPE-R with the Pairing-Based Cryptography (PBC) Library [20]. All experiments are carried out on a server running Linux with a Intel Core i7-3770 CPU @3.4GHz with 16GB of random access memory. Besides, type-A elliptic curve parameters are adopted, where the group order q is of 160 bits, providing equivalently 80-bit security strength.

A. Experimental Setup

In our experiments, we apply HPBPE and HPBPE-R to the Nursery data set downloaded from the UCI Machine Learning Repository [21] for a proof-of-concept performance demonstration. This data set, containing 12,960 instances with eight categories, has always been used in the previous research on searchable encryption [22], [23]. In our experimental, each instance is randomly defined as a h-th privilege, where the front h categories are utilized to generate the vectors of the secret/public keys for the corresponding h-th privilege, and the rest categories are viewed as the random vectors. Here, the category values are converted into elements in $\mathbb{F}_q$ using SHA-1 hash algorithm. To evaluate the efficiency of our schemes for various instances' number, we divide the data set into ten subsets, and each contains 1296 instances. In the evaluation of the encryption and query efficiency, we utilize multiple subsets, from one to ten, to test the encryption and query time.

B. Experimental Results

Now we report the experimental results for the computation and storage overhead of HPBPE and HPBPE-R.

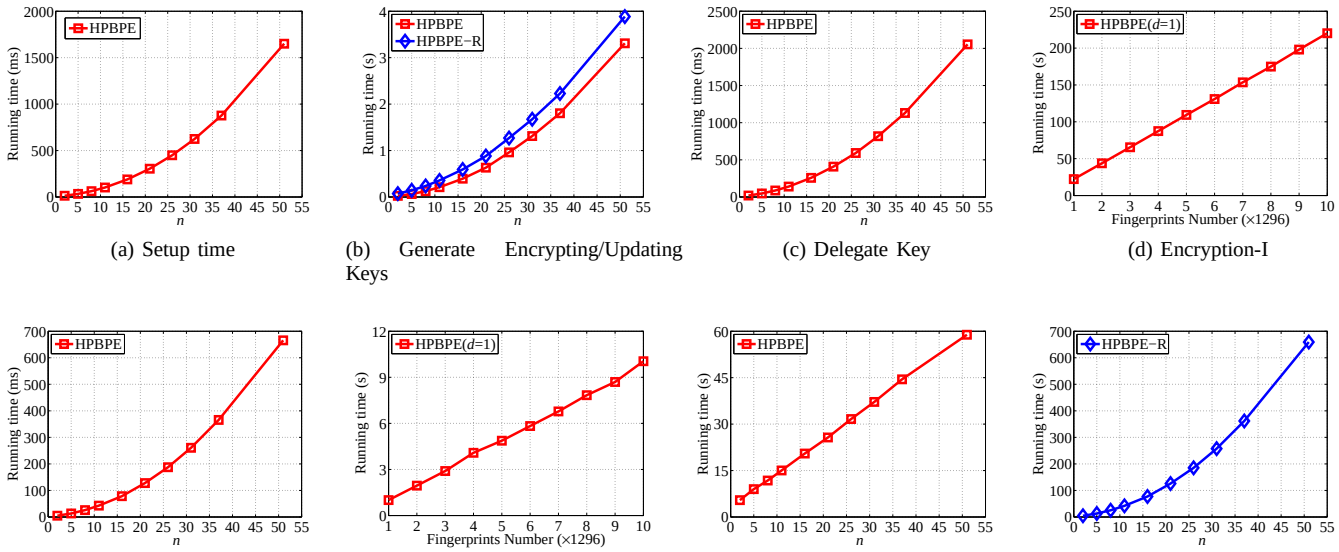
Setup. The main setup overhead is to build the base $\hat{\mathbb{B}}$ and $\mathbb{B}^*$, each involving $O(n_0^2)$=$O(n^2)$ exponentiations. Here n_0 is equivalent to n+3 (i.e., $\sum_{i=1}^{m'} d_i$+4) and denotes the dimension of the ECC vector spaces in HPE, and n is the length of $\vec{x}$, $\vec{v}$. Fig. 3a presents the average setup time with regard to n. As we can see, the setup time is about 624ms for n=31, which is a one-time affordable cost. For the storage overhead, the size of the base field for $\mathbb{G}_1$ and $\mathbb{G}_2$ is of 512 bits, but the elements in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ can be represented in 65B, while q is of 20B. Thus the total size of PK is of 65[n_0(n_0-1)+3]B, and that of MSK is of n_0^2(65+20)=85n_0^2B. When n=31, PK and MSK are of 71KB and 96KB, respectively.

Key Generation and Delegation. To test the performance for key generation and delegation, the experiments mainly assume that there are no subfields for each privilege (k=1) and the query contains all m' privileges. In this case, we choose d ($1 \leq d \leq 8$) privileges from the privilege universe in each privilege to form a query. That is, the vector $\vec{v}$ does not have element $0 \in \mathbb{F}_q$. According to Fig. 3b and Fig. 3c, key generation/updating consumes relatively long time, while the delegation consumes less time. The reason is that the former is processed by the TA[1] because it is usually a one-time operation; while the latter is experienced by Level-2 LTA[2] and users under a Level-1 LTA. Notice that the key generation/updating time and the delegation time both scale as $O(n_0^2)$.

Fingerprint Encryption Time. To evaluate the performance for encrypting file fingerprints, we run the experiments with two changing variables: the number of file fingerprints and n (the length of $\vec{x}$, $\vec{v}$). Assume that d_i=d, $\forall 1 \leq i \leq m'$. We first fix d (e.g., 1,$\cdots$,5) and vary the number of fingerprints from 1,296 to 12,960. Fig 3d only shows an instance of the result for d=1. As we can see, the average fingerprint encryption time increases linearly with the number of file fingerprints. We also fix m'=8 and vary d from 1 to 5, or fix d=1 and vary m' from 8 to 64. Our results show that the average fingerprint encryption time depends on $m'd$ (i.e., n). Fig. 3e shows the average time for encrypting a fingerprint, which scales as $O(n_0^2)$. When n=46, encrypting a

[1]A trusted authority, called TA, always need to be online, and assumes the responsibility of authorization at the same time.

[2]A authorization of local trusted authority, named LTA, issues the search capabilities for the employees of the current department.



(a) Setup time (b) Generate Encrypting/Updating Keys (c) Delegate Key (d) Encryption-I

(e) Encryption-II (f) Query-I (g) Query-II (h) Re-encrypting

Fig. 3: Fig. 3a presents the setup time for variable n. Fig. 3b shows the time to generate encrypting and updating keys in HPBPE and HPBPE-R, respectively. Fig. 3c depicts the delegation time. Fig. 3d and Fig. 3e show the encryption time. Fig. 3f and Fig. 3g show the search time for duplicate-check queries. Fig. 3h shows the re-encryption time in HPBPE-R.

fingerprint takes about 652ms. Since each data user generates the fingerprints for his/her own documents, this computational overhead is quite acceptable in practice. For the memory overhead, an encrypted fingerprint needs $65(n_0+1)$B, which equals merely 3.2KB when n=46.

Search Time. To evaluate the average search time for fingerprints at the deduplication provider, we also carry out the experiments by varying two variables: the number of file fingerprints and n (the length of $\vec{x}$, $\vec{v}$). The results are shown in Fig. 3f and Fig. 3g, respectively. We can see that the search time linearly increases with the number of fingerprints or n when the variable d is fixed. In addition, the search is much faster than encrypting a fingerprint because only n+3 pairing operations are involved. In our experiments, a single pairing operation with type-A elliptic curve parameters takes about 1.37ms without preprocessing and 0.62ms with preprocessing. According to Fig. 3g, when n=31, the search time for 12,960 fingerprints takes about 37.14s. Although it seems time-consuming, we argue that this is still acceptable for practical use on a much more powerful deduplication server.

HPBPE-R. Since privilege promotion is a fairly simple operation, we only report the performance of HPBPE-R for privilege demotion, which involves generating update keys and re-encryption. Fig. 3b shows the generation time for update keys, which increases with n. When n=31, the generating time is 1.66s. The re-encryption time is shown in Fig. 3h, which is also acceptable in practice as long as the privilege demotions do not happen very frequently in the enterprise.

VIII. Related Work

Douceur *et al.* was the first to notice the conflict between data confidentiality and deduplication in cloud storage systems [2]. To remedy this conflict, they proposed a novel solution named CE. For a message M, a user generates a key K=*Hash*(M) and encrypts M as C=*Enc*(K,M)=*Enc*(*Hash*(M),M). Then, the ciphertext C is uploaded to the cloud server, and the user retains K. If another user owns the same message M, he produces the same ciphertext C because the same key K can be derived from the same message M. In this case, the cloud server can easily perform deduplication for the identical ciphertext C. The CE technique has been widely adopted in [5]–[7], [13]. However, CE is deterministic and keyless, so it has inherent vulnerability to offline brute-force dictionary attacks [8]. To resist offline brute-force attack, Bellare *et al.* [8] proposed a secure deduplicated storage system called *DupLess*, which can provide strong security guarantees. In addition, Liu *et al.* [9] proposed a secure cross-user deduplication scheme without requiring any additional independent server. The work in [11] was the first deduplication system considering hierarchical access privileges, but this system allows the cloud server to profile the users according to their privileges.

There is also some work to prevent illegitimate users from exploiting the deduplication feature to retrieve the files they do not own. Halevi *et al.* introduced a security protocol called PoW, which permitted the server to test whether a user possesses the ownership for the requested file [10]. Besides, Halevi *et al.* [10] proposed several PoW constructions based on the Merkle-Hash Tree [14] to implement client-side

deduplication. Another PoW scheme was also proposed in [15], and its security relies on combinatory instead of computational assumptions. None of these PoW schemes address data confidentiality. Most recently, Ng *et al.* [16] extended PoW to encrypted files, but they do not consider hierarchical privileges.

IX. Conclusion

Data deduplication is commonly adopted in cloud storage services to improve storage utilization and reduce transmission bandwidth. It, however, conflicts with the requirement for data confidentiality offered by data encryption. Hierarchical authorized deduplication alleviates the tension between data deduplication and confidentiality and allows a cloud user to perform privilege-based duplicate checks before uploading the data. Existing hierarchical authorized deduplication systems permit the cloud server to profile cloud users according to their privileges. In this paper, we proposed a secure hierarchical deduplication system to support privilege-based duplicate checks and also prevent privilege-based user profiling by the cloud server. Our system also supports dynamic privilege changes. Detailed theoretical analysis and experimental studies confirmed the security and high efficiency of our system.

Acknowledgment

The authors would like to thank Ming Li of [22] for sharing their code for HPE. This work was supported in part by the National Natural Science Foundation of China (Grant No. 61472125, 61402161).

References

[1] "How Dropbox sacrifices user privacy for cost savings," `http://paranoia.dubfire.net/2011/04/how-dropbox-sacrifices-user-privacy-for.html`, [Online].

[2] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *ICDCS'02*, Vienna, Austria, Jul. 2002.

[3] S. Quinlan, and S. Dorward, "Venti: a new approach to archival storage," in *FAST'02*, Monterey, CA, Jan. 2002.

[4] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 174–187, Oct. 2001.

[5] L. Marques, and C. J. Costa, "Secure deduplication on mobile devices," in *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, Lisboa, Portugal, Jul. 2011.

[6] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Advances in CryptologyCEUROCRYPT'13*, Athens, Greece, Jan. 2013.

[7] A. Rahumed, H. Chen, Y. Tang, P. Lee, and J. Lui, "A secure cloud backup system with assured deletion and version control," in *ICPPW'11*, Taipei, Taiwan, Sep. 2011.

[8] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: server-aided encryption for deduplicated storage," in *USENIX Security' 13*, Washington, D.C., Aug. 2013.

[9] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *CCS'15*, Denver, Colorado, Oct. 2015.

[10] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *CCS'11*, Chicago, IL, Oct. 2011.

[11] J. Li, Y. Li, X. Chen, P. P. Lee, and W. Lou, "A Hybrid Cloud Approach for Secure Authorized Deduplication," in *IEEE Transactions on Parallel and Distributed Systems*, vol 26, no. 5, pp. 1206–1216, May 2014.

[12] T. Okamoto, and K. Takashima, "Hierarchical predicate encryption for inner-products," in *Advances in Cryptology-ASIACRYPT'09*, Tokyo, Japan, Dec. 2009.

[13] J. Xu, E. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *SIGSAC'13*, Hangzhou, China, May 2013.

[14] R. Zhang, J. Sun, Y. Zhang, and C. Zhang, "Secure spatial top-k query processing via untrusted location-based service providers," in *IEEE Transactions on Dependable and Secure Computing*, vol 12, no. 1, pp. 111-124, Jan. 2015.

[15] R. Pietro, and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *ASIACCS'12*, Seoul, Republic of Korea, May 2012.

[16] W. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *SAC'12*, Riva del Garda, Italy, Mar. 2012.

[17] "SHA-1," `https://en.wikipedia.org/wiki/SHA-1`, [Online]

[18] J. Sun, R. Zhang, and Y. Zhang, "Privacy-preserving spatiotemporal matching," in *INFOCOM'13*, Turin, Italy, Apr. 2013.

[19] M. Green, and G. Ateniese, "Identity-based proxy re-encryption," in *ACNS'07*, Zhuhai, China, Jun. 2007.

[20] B. Lynn. The pbc library. `http://crypto.stanford.edu/pbc/`

[21] A. Frank and A. Asuncion, "UCI machine learning repository," 2010.

[22] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *ICDCS'11*, Minneapolis, Minnesota, Jun. 2011.

[23] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *CCS'06*, Alexandria, VA, Nov. 2006.