8-2017

Secure encrypted data deduplication with ownership proof and user revocation

Wenxiu DING
Xidian University

Zheng YAN
Aalto University

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Citation

DING, Wenxiu; YAN, Zheng; and DENG, Robert H.. Secure encrypted data deduplication with ownership proof and user revocation. (2017). *Algorithms and architectures for parallel processing: 17th International Conference ICA3PP 2017, Helsinki, Finland, August 21-23, Proceedings*. 297-312.
Available at: https://ink.library.smu.edu.sg/sis_research/3788

Secure Encrypted Data Deduplication with Ownership Proof and User Revocation

Wenxiu Ding[1], Zheng Yan[1,2(✉)], and Robert H. Deng[3]

[1] State Key Lab of Integrated Services Networks,
School of Cyber Engineering, Xidian University, Xi'an, China
wenxiuding_1989@126.com, zyan@xidian.edu.cn
[2] Department of Communications and Networking,
Aalto University, Espoo, Finland
[3] School of Information Systems,
Singapore Management University, Singapore, Singapore
robertdeng@edu.smu.sg

Abstract. Cloud storage as one of the most important cloud services enables cloud users to save more data without enlarging its own storage. In order to eliminate repeated data and improve the utilization of storage, deduplication is employed to cloud storage. Due to the concern about data security and user privacy, encryption is introduced, but incurs new challenge to cloud data deduplication. Existing work cannot achieve flexible access control and user revocation. Moreover, few of them can support efficient ownership proof, especially public verifiability of ownership. In this paper, we propose a secure encrypted data deduplication scheme with effective ownership proof and user revocation. We evaluate its performance and prove its security. The simulation results show that our scheme is efficient and effective for potential practical employment.

Keywords: Deduplication · User revocation · Homomorphic encryption · Proxy re-encryption

1 Introduction

Cloud computing provides seemingly unlimited resources as services to cloud users by rearranging various resources. Cloud storage as one of the most popular cloud services enables cloud users to store tremendous amount of data in the cloud, which may exceed their own storage spaces.

In order to improve the storage services, deduplication has become an important technique in cloud storage. Data deduplication can help eliminate multiple copies of same files and improve the utilization of storage. It has proved to achieve high cost savings, such as reducing up to 68% storage for standard file systems [1]. The savings can be passed back to cloud users in many ways, such as reducing storage cost. Thus, efficient deduplication is extremely desired by both cloud service providers and cloud users. Though data deduplication brings many benefits, it also faces some challenges.

First, cloud storage and deduplication management incurs the concern about data privacy and user privacy [2]. Cloud users would lose the full control over their out-sourced personal data, which even may be disclosed by the dishonest cloud service providers. Thus encryption as a popular method is introduced to solve this problem [3]. But the same file encrypted with different encryption schemes would result in different ciphertexts, which makes it difficult to check duplication and conduct deduplication. Second, how to share the duplicated data flexibly to authorized data holders is an issue [4]. Duplicated data would not be stored again, but it should guarantee the access right of data holders. Third, data deletion by users complicates the deduplication management. If some data holders delete their stored data, their access to the data should be completely prevented even when they still hold the previous key for obtaining the data. Thus, deduplication management should support user revocation. Fourth, how to guarantee the ownership proof of data holders is an open issue [5]. In order to reduce the communication cost and computation cost caused by duplicated data upload, file tag is always employed for duplication check. But it is vulnerable to forgery attack and difficult to guarantee the ownership.

In order to solve the above problems, this paper presents a flexible encrypted data deduplication scheme under the cooperation of an Authorized Party (AP) and a cloud service provider (CSP). First, we propose an Additive Homomorphic Re-Encryption (AHRE) algorithm. In our deduplication scheme, uploaded file is encrypted with symmetric encryption while the symmetric key is encrypted with AHRE. It can efficiently update the ciphertexts and revoke those data holders who delete their data at the cloud. Moreover, we propose a scheme to prove the ownership without complicated interactions between data holders and CSP or AP, which can support flexible public verifiability. Different from existing work [6, 7], our scheme can effectively deal with encrypted data deduplication with revocation at the cloud without involvement of data owners or data holders. Specifically, the contributions of this paper are:

- We construct an additive homomorphic re-encryption algorithm, which supports re-encryption and additive homomorphic computation. It lays the basic foundation of our proposed scheme for encrypted data deduplication with effective ownership proof and user revocation.
- We integrate data deduplication with flexible access control based on AHRE, which results in a secure and efficient encrypted data deduplication.
- We design an encrypted data update and user revocation protocol to enhance data security. It can refresh the stored data at any time and support the data holder revocation when it deletes the data. In addition, the above operations do not need any involvement of data holders.
- We design an efficient ownership proof scheme, which does not incur complicated interaction between data holders and CSP and can support public verifiability.
- We prove the security and justify the performance of our scheme through analysis and implementation.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work. System model and security model are introduced in Sect. 3, followed by the detailed design of proposed schemes in Sect. 4. In Sect. 5, security analysis and performance evaluation are given. Finally, we conclude the paper in the last section.

2 Related Work

Cloud storage service providers such as Dropbox [8], Google Drive [9], Mozy [10], perform deduplication to save space by eliminating redundancy in cloud storage and optimizing its utilization. In order to preserve the privacy of data holders, encryption is employed. However, storage savings through deduplication are totally lost if clients conventionally encrypt their data. This is because the encrypted data are saved as different contents by applying different keys, which complicates the deduplication. For example, DeDu [11] - a deduplication system is unable to handle encrypted data.

Encrypted Data Deduplication
Convergent Encryption (CE) as the most prominent manifestation of Message Locked Encryption (MLE) was introduced [12, 13]. In CE, a user employs the hash code of data as the key to encrypt the data, which results in $E_{H(m)}(m)$. Any user with the same data can generate the same ciphertext, thus realizing deduplication. However, CE suffers from offline brute-force dictionary attacks. Moreover, it is hard to support user revocation. Bellare et al. proposed DupLESS to resist the above-mentioned brute-force attacks [14] by introducing a Key Server. But it still cannot control data access of other data users in a flexible way. Wen et al. [15] constructed a session-key-based convergent key management scheme and a convergent key sharing scheme to overcome the problem caused by frequently changed ownership and data blocks. But this work requests all data owners communicate with each other to manage their session key. Liu et al. proposed a secure cross-user deduplication scheme that supports client-side encryption without requiring any additional independent servers by applying a password authenticated key exchange protocol [16]. But this scheme requests that the data owner is always online for data ownership check and deduplication. Thus this approach cannot handle the situation that the data owner is not available, which is very common in practice.

In another work [17], attribute-based encryption (ABE) is applied to realize deduplicated data access control managed by data owners. But it needs the data owners to be online and incurs much computation cost to them. In our previous work [18, 19], we proposed deduplication schemes based on proxy re-encryption (PRE). But the ciphertext update needs the data holders to download the stored file from the cloud and then encrypt it, which incurs higher communication cost and computation overhead. Liu et al. proposed a policy-based deduplication proxy scheme [20], but they did not consider data deletion. Another work [21] even proposed to forfeit deduplication to reduce chunk fragmentation by container capping. But all of the above work does not consider user revocation management. Though the work [7] proposed to solve the deduplication with user revocation, it complicates the key management of Attribute-Based Encryption (ABE) especially during user revocation. Hur et al. proposed a novel server-side deduplication scheme for encrypted data [6]. It allows a cloud server to control access to outsourced data even when data ownership changes dynamically by exploiting randomized convergent encryption and secure ownership group key distribution. This scheme prevents data leakage from revoked users. But it needs all users to upload their encrypted files and data owners should be online for user revocation, which is not efficient.

Data Ownership Verification

Halevi et al. introduced a practical implementation of Proofs of Ownership (PoW) for deduplication [22]. They proposed to use Merkle tree on the pre-processed data to generate verification information. When challenging a prover, a verifier randomly chooses several leaves of the tree and obtains the corresponding sibling-paths of all these leaves. Only when all paths are valid, will the verifier accept the proof. Pietro et al. [23] chose the projection of a file onto some randomly selected bit-positions as proof to realize the PoW. But both schemes above do not consider data privacy. Ng et al. [24] also applied Merkle tree to manage the deduplication of encrypted data. The value of each leaf node is generated from several data blocks, while only one leaf is considered in each interactive proof protocol. Thus, it needs to execute the protocol multiple times to enhance its correctness, which causes high computation overhead.

Yang et al. proposed an efficient scheme to check the ownership [25]. A data holder only needs to access partial and dynamical portions of an original file to generate the proof of possession. In addition, the data holder has no need to upload the file, which can reduce communication and computation costs. In our previous work [19], Elliptic Curve Cryptography (ECC) is employed to verify data ownership by challenging data holders. The scheme presented in this paper simplifies this procedure, which can support public verifiability of ownership when data holders are offline.

3 Problem Statements

3.1 System Model and Security Model

Our proposed scheme mainly consists of four types of entities as shown in Fig. 1:

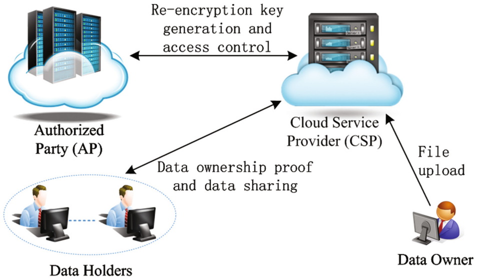

Fig. 1. System model

(1) Cloud Service Provider (CSP) is in charge of data storage and duplication check. CSP is curious about user data, but it follows designed protocols strictly in order to gain commercial benefits from providing storage service to its consumers or users.

(2) Authorized Party (AP) is responsible for access policy check, re-encryption key generation, and user revocation by cooperating with the CSP. It would never collude with the CSP and is fully trusted. AP cannot access the data stored at the CSP.

(3) Data owner is the cloud service consumer and the first data uploder. It encrypts an original file and uploads it to the CSP. The CSP generates one file tag based on the proof message of the data owner.

(4) Data holders are subsequent uploaders, who do not need to encrypt the file but need to pass ownership check in order to obtain the access right of the stored file. If one user deletes its file at the CSP, AP and CSP should revoke its privilege on the file.

We further assume that communication channels among system entities are secure and each system entity can be authenticated based on a unique identifer.

3.2 Preliminary and Notations

Preliminary

A simplified variant scheme [26]: Given two large primes p and q, then $n = p * q$. Let g and h be two elements of maximal order in $\mathbb{G}$, where $\mathbb{G}$ is the cyclic group of quadratic residues modulo n^2.

Key Generation: The public parameters are n, g and $h = g^x \, mod \, n^2$ by randomly choosing a secret value $x \in [1, ord(\mathbb{G})]$.

Encryption (Enc): Given a message $m \in \mathbb{Z}_n$, random number r is chosen in $\mathbb{Z}_n^*$. The ciphertext is computed as $[m]_h = (T, T') = \{h^r(1 + m * n), g^r\} \, (mod \, n^2)$.

Decryption (Dec): Knowing x, m can be obtained as follows: $m = L(T/(T')^x mod \, n^2)$, where $L(u) = (u - 1)/n$.

Notations

Table 1 summarizes the notations used throughout the paper.

Table 1. System notations

Symbols	Description
g	The system generator that is public
n	The system parameter
$[m]$	The ciphertext of data m
$\mathcal{L}(*)$	The bit length of input data
$H()$	The hash function
$e(;)$	The bilinear pairing: $G_1 \times G_1 \rightarrow G_T$
v	The generator in G_1
$(sk_{AP}, pk_{AP}) = (b, v^b)$	The key pair of AP
$(sk_{CSP}, pk_{CSP}) = (a, v^a)$	The key pair of CSP
$(sk_i, pk_i) = (u_j, v^{u_j})$	The key pair of user j
$uKey$	The updating key for ciphertext refresh
$rk_{i \rightarrow j}$	The re-encryption key from user i to user j
$E_k()$	The symmetric encryption under symmetric key k
PrM	The proof message generated by data uploaders

4 Algorithm and Scheme Design

In this section, we first combine PRE and homomorphic encryption to obtain a newly designed AHRE algorithm. Then we propose some schemes to support flexible duplicated data management based on AHRE.

4.1 Additive Homomorphic Re-Encryption (AHRE)

AHRE lays the technical foundation of deduplication, which can support homomorphic processing and re-encryption computation. Its detailed design is described below.
System Setup: Let p, q be two large primes. Due to the property of safe primes, there exist two primes p' and q' that satisfy that $p = 2p' + 1$, $q = 2q' + 1$. We compute $n = p * q$ and choose generator g with order $\lambda = 2p'q'$, which can be chosen by selecting a random number $z \in \mathbb{Z}^*_{n^2}$ and computing $g = -z^{2n}$. The value λ can be used for decryption, but we choose to conceal it and protect it from all parties. In addition, the system chooses two groups G_1 and G_{T} of a prime order with bilinear map e: $G_1 \times G_1 \rightarrow G_T$. The system parameters are random generators $v \in G_1$ and $Z = e(v, v) \in G_T$. A cryptographic hash function: H: $\{0, 1\}^* \rightarrow Z_n$ is also applied.

Key Generation $(KGen)$**:** The CSP and the AP generates their key pairs: $(sk_{CSP}, pk_{CSP}) = (a, v^a)$ and $(sk_{AP}, pk_{AP}) = (b, v^b)$ respectively. User j generates key pair (u_j, v^{u_j}).

Encryption (Enc)**:** Any users upload their data to CSP for storage and deduplication management. User i chooses two random values r_1 and r_2, and then encrypts its raw data m with public keys pk_{AP}. The ciphertext of data m is denoted as: $[m] = \{C_1, C_2, C_3\} = \{(1 + m * n)g^{H(Z^{r_1})*r_2}\, modn^2, g^{r_2}\, modn^2, pk_{AP}^{r_1}\}$.

Re-Encryption Key Generation $(RKGen)$**:** The AP wants to delegate user j by publishing re-encryption key $rk_{AP \rightarrow j} = v^{u_j/b}$.

Re-Encryption $(ReEnc)$**:** The CSP computes $C_3' = e\big(pk_{AP}^{r_1}, rk_{AP \rightarrow j}\big) = Z^{r_1 * u_j}$, and sets $C_2' = C_2$ and $C_1' = C_1$. Finally, the CSP forwards $\{C_1', C_2', C_3'\}$ to user j.

Decryption (Dec)**:** Upon receiving the encrypted data, user j can directly decrypt it to obtain the original data: (1) compute $C_3'' = H((C_3')^{1/u_j}) = H(Z^{r_1})$; (2) decrypt to obtain the raw data $m = L(C_1/(C_2')^{C_3''} modn^2)$ where $L(u) = (u - 1)/n$.

Moreover, it can also support ciphertext refresh and additive homomorphism.
Updating Key Issue (UKI)**:** In case that the CSP wants to update the ciphertext $[m]$, AP can generate an auxiliary parameter for the CSP: $uKey = g^{H(Z^{r_1})} = g^{H(e(v^{1/b}, pk_{AP}^{r_1}))}\, mod\, n^2$.

Ciphertext Refresh $(CipR)$**:** The CSP can update the ciphertext with its own secret key and $uKey$ by: (1) choose random r_3 and compute $\overline{C_1} = C_1 * uKey^{r_3}\, mod\, n^2$; (2) compute $\overline{C_2} = C_2' * g^{r_3}\, mod\, n^2$; (3) $\overline{C_3} = C_3$.

Additive Homomorphism (AH)**:** With the updating key, the CSP can achieve additive homomorphic operation over the ciphertext $[m]$. It simply chooses another data m' and computes $\widetilde{C}_1 = (1 + m' * n) * C_1 \, mod \, n^2$. Finally, it directly calls $CipR$ to update$\left\{\widetilde{C}_1, C_2, C_3\right\}$. As a result, we get the ciphertext of $(m' + m)$.

4.2 Ownership Check

In order to support deduplication, we first present an ownership proof scheme in Fig. 2. If U1 wants to upload its file m, then it generates a proof message $PrM_1 = pk_{CSP}^{H(m)/u_1}$ and forwards it to CSP. Upon receiving the message, the CSP first computes a file tag with the public key of uploader U1: $e(PrM_1, pk_{u_1}) = Z^{a*H(m)}$. It then checks if this tag has been stored. If yes, it means that the corresponding file has been kept and the CSP only needs to record U1 as a data holder. Otherwise, it informs U1 to upload the file. The details will be presented in the next section.

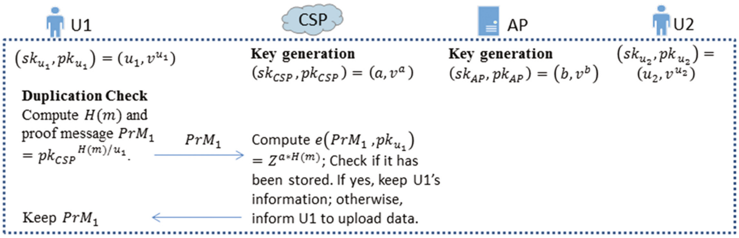
U1
CSP
AP
U2
$(sk_{u_1}, pk_{u_1}) = (u_1, v^{u_1})$
Key generation
$(sk_{CSP}, pk_{CSP}) = (a, v^a)$
Key generation
$(sk_{AP}, pk_{AP}) = (b, v^b)$
$(sk_{u_2}, pk_{u_2}) = (u_2, v^{u_2})$
Duplication Check
Compute $H(m)$ and proof message $PrM_1 = pk_{CSP}{}^{H(m)/u_1}$.
PrM_1
Compute $e(PrM_1, pk_{u_1}) = Z^{a*H(m)}$; Check if it has been stored. If yes, keep U1's information; otherwise, inform U1 to upload data.
Keep PrM_1

Fig. 2. The procedure of ownership check

Specially, the ownership proof can realize public verifiability. AP or any data owner can check the legality of data holders based on proof message and file tag. The ownership check is also applied during re-encryption key generation executed by AP, which can help prevent the collusion of CSP with unauthorized cloud users.

4.3 Data Deduplication Management

We suppose that data holder U1 stores file m at the CSP and later U2 wants to store the same file. If U2 wants to save the same data to the CSP, then the CSP will cooperate with the AP to enable U2 to access file m without uploading file. Figure 3 illustrates a brief protocol of encrypted data deduplication and data retrieve based on the AHRE. The details are presented below (system setup refers to Sect. 4.1):

Step 1 - Original Data Upload: Data holder U1 wants to store file m at the CSP. It follows the ownership check above and sends proof message PrM_1 to CSP. Upon receiving PrM_1, the CSP first computes file tag $Z^{a*H(m)}$ and checks if this file has been stored. If not, the CSP informs U1 to upload its file and related files to CSP for storage.

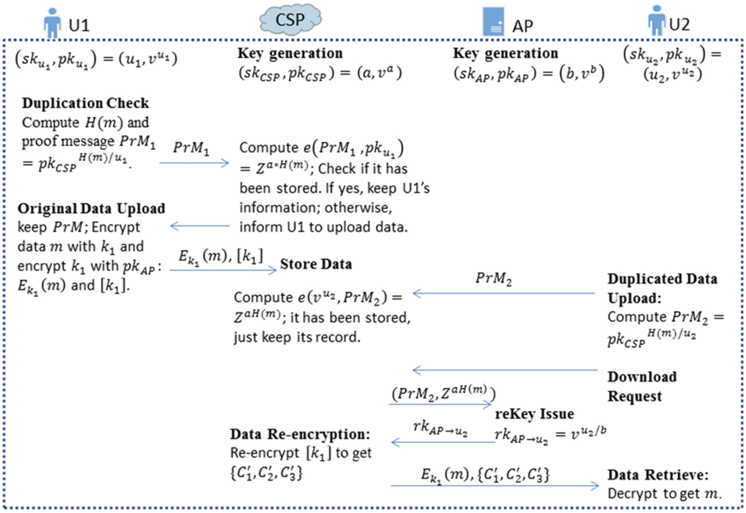
U1
CSP
AP
U2
$(sk_{u_1}, pk_{u_1}) = (u_1, v^{u_1})$
Key generation
$(sk_{CSP}, pk_{CSP}) = (a, v^a)$
Key generation
$(sk_{AP}, pk_{AP}) = (b, v^b)$
$(sk_{u_2}, pk_{u_2}) = (u_2, v^{u_2})$
Duplication Check
Compute $H(m)$ and proof message $PrM_1 = pk_{CSP}{}^{H(m)/u_1}$.
PrM_1
Compute $e(PrM_1, pk_{u_1}) = Z^{a*H(m)}$; Check if it has been stored. If yes, keep U1's information; otherwise, inform U1 to upload data.
Original Data Upload
keep PrM; Encrypt data m with k_1 and encrypt k_1 with pk_{AP}: $E_{k_1}(m)$ and $[k_1]$.
$E_{k_1}(m)$, $[k_1]$
Store Data
PrM_2
Duplicated Data Upload:
Compute $PrM_2 = pk_{CSP}{}^{H(m)/u_2}$
Compute $e(v^{u_2}, PrM_2) = Z^{aH(m)}$; it has been stored, just keep its record.
Download Request
$(PrM_2, Z^{aH(m)})$
reKey Issue
$rk_{AP\to u_2} = v^{u_2/b}$
$rk_{AP\to u_2}$
Data Re-encryption:
Re-encrypt $[k_1]$ to get $\{C_1', C_2', C_3'\}$
$E_{k_1}(m), \{C_1', C_2', C_3'\}$
Data Retrieve:
Decrypt to get m.

Fig. 3. The procedure of data deduplication management

U1 as the first data uploader should encrypt its file with k_1 using symmetric key encryption, call *Enc* to encrypt k_1. The data package $\{E_{k_1}(m),\ [k_1] = \{C_1, C_2, C_3\},\ Z^{a*H(m)}\}$ is kept by the CSP for data storage and duplication check.

When U2 wants to upload the same file, it follows:

Step 2 - Duplicated Data Upload: U2 also generates proof message $PrM_2 = pk_{CSP}^{H(m)/u_2}$ based on its own secret key, while the CSP finds that $e(v^{u_2}, PrM_2) = Z^{a*H(m)}$ has been stored. The CSP directly informs U2 that the data has been stored.

Step 3 - Download Request: The data holder (such as U2) sends its request for accessing file m to the CSP. The CSP contacts the AP for re-encryption key generation.

Step 4 - Re-encryption Key Issue: AP checks the legality of data holders through public verifiability. If it does not pass the check, it rejects to provide the re-encryption key; Otherwise, AP calls *RKGen* to generate a re-encryption key for authorized data holder U2: $rk_{AP\to u_2} = v^{u_2/b}$. Notably, the failed cases of ownership check can be broadcasted, which will obviously reduce the reputation of CSP and its benefits.

Step 5 - Data Re-Encryption: CSP calls *ReEnc* using $rk_{AP\to u_2}$ to encrypt $[k_1]$ to generate new key ciphertext for U2: $\{C_1', C_2', C_3'\}$. Then data packet $\{C_1', C_2', C_3'\}$ and $E_{k_1}(m)$ are sent back to data holder U2.

Step 6 - Data Retrieve: Upon receiving data packet, U2 decrypts $\{C_1', C_2', C_3'\}$ first to get the symmetric key k_1 and then performs decryption to gain the original file m.

4.4 Encrypted Data Update

In some cases, the data holders or owners would like to update the stored data periodically to enhance data security. But in most of existing schemes, the data holders always need to download the file and use new keys to encrypt the original file, which is time-consuming and causes high communication cost. In order to solve this problem, we propose the following scheme to update encrypted data without user interaction.

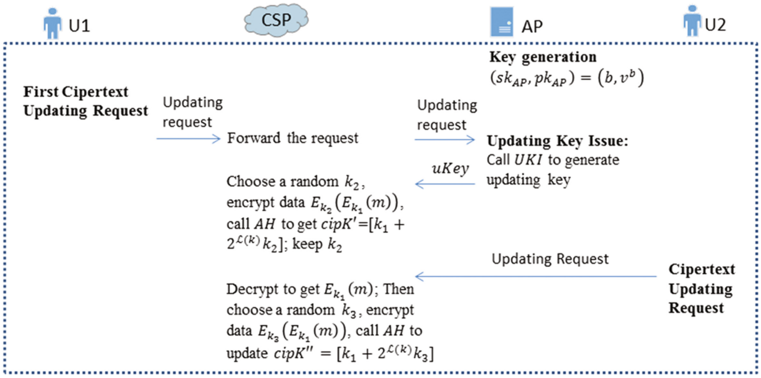
U1
CSP
AP
U2
Key generation
$(sk_{AP}, pk_{AP}) = (b, v^b)$
First Cipertext Updating Request
Updating request
Forward the request
Updating request
Updating Key Issue:
Call UKI to generate updating key
$uKey$
Choose a random k_2, encrypt data $E_{k_2}(E_{k_1}(m))$, call AH to get $cipK'$=$[k_1 + 2^{\mathcal{L}(k)} k_2]$; keep k_2
Updating Request
Cipertext Updating Request
Decrypt to get $E_{k_1}(m)$; Then choose a random k_3, encrypt data $E_{k_3}(E_{k_1}(m))$, call AH to update $cipK'' = [k_1 + 2^{\mathcal{L}(k)} k_3]$

Fig. 4. Procedure of encrypted data updating

The ciphertext update request may occur in two cases (See Fig. 4) and lead to different operating procedures, which are presented as follows:

Scheme 1 - First ciphertext update request: In this case, it is the first update initiated by the CSP or data holders over the stored data $\{E_{k_1}(m), [k_1] = \{C_1, C_2, C_3\}\}$.

The CSP randomly chooses key k_2 to further encrypt the ciphertext of file m. In order to optimize the management of keys, we concatenate two keys in a secure way. The CSP first scales k_2 by $2^{\mathcal{L}(k)}$ where $\mathcal{L}(k)$ is the largest length of symmetric keys, and calls AH to get the ciphertext of $(k_1 + 2^{\mathcal{L}(k)} k_2)$. It keeps the packet $\{E_{k_2}(E_{k_1}(m))$, $cipK' = [k_1 + 2^{\mathcal{L}(k)} k_2]\}$ and keeps k_2 secret in its storage.

Scheme 2 - Follow-up update request: In this case, the ciphertext update is requested after first ciphertext update request, which is over the ciphertext $\{E_{k_2}(E_{k_1}(m)),\ cipK' = [k_1 + 2^{\mathcal{L}(k)} k_2]\}$. The CSP follows the steps below:

(a) Use k_2 to decrypt the ciphertext $E_{k_2}(E_{k_1}(m))$;
(b) Choose a random number k_3, and compute $(n - 2^{\mathcal{L}(k)} k_2 + 2^{\mathcal{L}(k)} k_3)$;
(c) Call AH to get the ciphertext of $(k_1 + 2^{\mathcal{L}(k)} k_3)$: $cipK'' = \left[k_1 + 2^{\mathcal{L}(k)} k_3\right]$;
(d) Update its stored packet with $\{E_{k_3}(E_{k_1}(m)),\ [k_1 + 2^{\mathcal{L}(k)} k_3],\ k_3\}$.

4.5 User Revocation Management

If use i deletes their storage in the cloud, the ciphertext update as described above is not secure enough since it cannot prevent the access of user i if it has kept some previously secret keys. Thus, we further design a secure scheme to really block the access from these revoked users.

Similar to the ciphertext update, the user revocation also falls into two types:
Scheme 3 – Revocation before ciphertext update (See Fig. 5)**:** Some user deletes its storage of file m without any update on its ciphertext $E_{k_1}(m)$. In this case, this revocation can be completed through the following steps:

(a) CSP chooses a random number r_3, and calls AH to get $cipK = [k_1 + r_3]$; finally, $cipK$ is sent to AP;

(b) The AP decrypts $cipK$ to get $(k_1 + r_3)$, calls Enc to encrypt $(k_1 + r_3)$ with newly chosen randoms to get $cipK' = [k_1 + r_3] = \{(1 + (k_1 + r_3) * n)g^{H(Z^{r'_1}) * r'_2} \, mod \, n^2$, $g^{r'_2} \, mod \, n^2$, $Y^{r'_1}\}$; in addition, it calls UKI to generate a new updating key $uKey' = g^{H(e(v^{1/y}, Y^{r'_1}))} = g^{H\left(Z^{r'_1}\right)} \, mod \, n^2$. Then, the data packet $\{cipK', uKey'\}$ is sent back to the CSP.

(c) The CSP directly updates its ciphertext with a new symmetric key k_4 to get the new packet $\{E_{k_4}(E_{k_1}(m)), \, [k_1 + 2^{\mathcal{L}(k)} k_4]\}$.

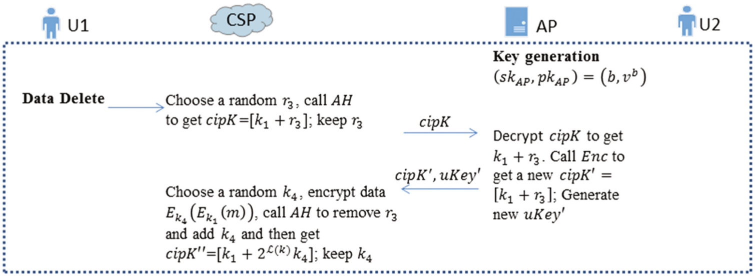
U1
CSP
AP
U2
Key generation
$(sk_{AP}, pk_{AP}) = (b, v^b)$
Data Delete
Choose a random r_3, call AH to get $cipK$=$[k_1 + r_3]$; keep r_3
$cipK$
Decrypt $cipK$ to get $k_1 + r_3$. Call Enc to get a new $cipK' =$ $[k_1 + r_3]$; Generate new $uKey'$
$cipK', uKey'$
Choose a random k_4, encrypt data $E_{k_4}(E_{k_1}(m))$, call AH to remove r_3 and add k_4 and then get $cipK''$=$[k_1 + 2^{\mathcal{L}(k)} k_4]$; keep k_4

Fig. 5. Procedure of user revocation caused by user deletion

Scheme 4 – Revocation after ciphertext update: In some cases, data holder deletes its storage after ciphertext update. Hence, the CSP executes user revocation over ciphertext $\{E_{k_2}(E_{k_1}(m)), [k_1 + 2^{\mathcal{L}(k)} k_2]\}$.

(a) The CSP chooses a random r_4; then it calls AH to get $cipK = [k_1 + r_4 + 2^{\mathcal{L}(k)} k_2]$; finally, $cipK$ is sent to AP.

b) Similar to the step b) above, the AP chooses random numbers r'_1 and r'_2, and then get $\{cipK' = [k_1 + r_4 + 2^{\mathcal{L}(k)} k_2] \, , uKey'\}$.

(c) Upon receiving the data packet, the CSP decrypts $E_{k_2}(E_{k_1}(m))$ with k_2 and chooses random key k_5 to get $E_{k_5}(E_{k_1}(m))$; then it calls AH to remove r_4 and update k_2 with k_5 to get a new ciphertext key $cipK'' = [k_1 + 2^{\mathcal{L}(k)}k_5]$. It updates its storage with new ciphertexts $\{E_{k_5}(E_{k_1}(m)), [k_1 + 2^{\mathcal{L}(k)}k_5]\}$.

Through the schemes above, CSP finally block unauthorized users' access without the need of the intervention of data holders or data owner.

5 Security Analysis and Performance Evaluation

5.1 Security Analysis

Our scheme provides a secure approach to realize the deduplication management. The security of the proposed scheme is guaranteed by the security of the AHRE algorithm. Thus, we mainly concentrate on the security proof of AHRE and the ownership check.

Assumptions

Definition 5.1. Discrete Logarithm (DL) Problem: Given $g \in G$ and $y = g^x$ $(x \in Z_q^*)$, it is hard to get x.

Definition 5.2. Computational Diffie-Hellman (CDH) Problem: Given a group G and group element v, and v^x and v^y, it is hard to compute the value of v^{xy}.

Proposition 1. If the CDH problem holds, then it is hard for adversaries to pass the ownership proof without the original file even when it colludes with CSP.

Proof. We prove it by contraction. For this purpose, we assume that the adversary can pass the ownership proof without the real file. Our goal then is to use $\mathcal{A}$ to construct an algorithm to solve the CDH problem.

Given the challenge public parameters (v, Z, $e(;)$), the adversary $\mathcal{A}$ can construct its own key pair ($u_{\mathcal{A}}$, $v^{u_{\mathcal{A}}}$). If it colludes with CSP, then the adversary can further get $pk_{CSP}^{H(m)/u_1}$, $v^{H(m)/u_1}$ and $Z^{H(m)}$ of real data holder u_1 from CSP. The adversary can compute to get its own proof message $pk_{CSP}^{H(m)/u_{\mathcal{A}}} = v^{aH(m)/u_{\mathcal{A}}}$ on the $pk_{CSP}^{H(m)/u_1} = v^{aH(m)/u_1}$ and v^{u_1}. The adversary can generate real proof message $v^{aH(m)/u_{\mathcal{A}}}$, and easily gain $v^{aH(m)}$ with its own secret key.

Here, we set $v^x = v^{aH(m)/u_1}$ and $v^y = v^{u_1}$, thus it means the adversary gets $v^{xy} = v^{aH(m)}$ and breaks the problem of CDH. Hence, our ownership proof is secure and can guarantee that only real data holders can pass the proof. In addition, anyone can verify the ownership of data holders with proof message and file tag.

Proposition 2. If the DL problem holds in group G_1 and the CDH problem holds in group $Z_{n^2}^*$, then the AHRE is secure.

Proof. Given the AHRE ciphertext of data m under the secret key of CSP: $[m]$ =$\{C_1, C_2, C_3\} = \left\{(1 + m * n)g^{H(Z^{r_1})*r_2} \, mod \, n^2, g^{r_2} \, mod \, n^2, pk_{AP}^{r_1}\right\}$, the adversary $\mathcal{A}$ would like to obtain the original data m.

Due to the difficulty of DL problem, it is hard to get v^{r_1} from $pk_{AP}^{r_1} = v^{b*r_1}$. Hence, the adversary cannot obtain the value of $H(Z^{r_1})$. In the update process, the updating key $g^{H(Z^{r_1})}$ is issued to CSP. If the adversary colludes with CSP, it can get packet

$(\{C_1, C_2, C_3\} = \{(1 + m * n)g^{H(Z^{r_1}) * r_2}\, mod\, n^2, g^{r_2}\, mod\, n^2, pk_{AP}^{r_1}\}, g^{H(Z^{r_1})})$. But due to CDH problem, the adversary cannot get $g^{H(Z^{r_1}) * r_2}$ from the packet and cannot obtain the original data.

Proposition 3. The cooperation of CSP and AP without collusion guarantees that only eligible data holders can access the original file m and the file can be deduplicated securely.

Proof. The adversary has no way to obtain the original file m even when it colludes with CSP as it is always in an encrypted form. The file m is encrypted with symmetric encryption (such as AES) while the symmetric key is encrypted with AHRE. Owing to the security of AHRE, the symmetric key is protected and unauthorized users cannot get it. Moreover, original data confidentiality is guaranteed by symmetric encryption. Hence, only the authorized users can decrypt ciphertext of keys to further obtain the original file.

The ownership proof helps check the legality of cloud users without reducing the confidentiality of original data. The re-encryption key issue further transforms the ciphertext under the secret key of AP to another one under the secret key of authenticated data holders. It helps control the access to the ciphertext. In addition, the ciphertext of original file is kept by the CSP, thus the AP can only control the ciphertext of symmetric key.

In the ciphertext update, a random number is introduced to mask the original symmetric key, which can make sure neither CSP nor AP can get the symmetric key.

5.2 Computation Complexity

The proposed scheme involves four kinds of system roles: data owner, CSP, AP, and data holder. To present the computation complexity in details, we adopt AES and AHRE. As the encryption over the uploaded data is unavoidable, we neglect the symmetric encryption. Due to the limitation of paper length, we analyze the complexity of deduplication presented in Sect. 4.3 as below:

Data Owner: It needs to do one hash, one modular exponentiation in duplication check, and two modular exponentiations and two exponentiations over group G_1 or G_T to upload its data to the CSP. Thus, its computation complexity is $\mathcal{O}(1)$.

CSP: CSP performs one pairing to compute the file tag for duplication check and one pairing for re-encryption with regard to each data holder's data upload. Thus, its computation complexity is $O(N)$, where N is the number of data holders.

AP: It conducts one pairing for duplication check and one exponentiation over group G_1 for re-encryption key generation. Thus, its computation complexity is $\mathcal{O}(N)$.

Data Holder: It also needs one modular exponentiation in duplication check. In order to get the data, it should do one exponentiation over G_T, one hash and one modular exponentiation. Thus, its computation complexity is $\mathcal{O}(1)$.

Besides the computation above, they all need to generate one key pair for themselves, which involves one exponentiation over G_1. Table 2 lists the computation of all entities. We also compare it with our previous work [18, 19]. We can observe that our scheme incurs a little higher computation cost during data upload and data retrieval process but enables more functionalities, such as public verifiability and user revocation. In addition, it simplifies ownership challenge and reduces the communication costs caused by ciphertext update or user revocations. In our scheme, CSP and AP only need to exchange the ciphertexts of keys. The ciphertext of original file is always kept and updated by CSP without any involvement of data owners or data holders, which saves the communication cost, especially for multimedia data.

Table 2. Computation complexity of each entity by comparing with previous work [18, 19]

Entity	Algorithm	Computations [our scheme]	Computations [19]	Computations [18]
Data owner	Setup	1 * Exp	1 * PointMulti + 1 * Exp	1 * ModInv + 1 * ModExp
	Data upload	2 * Ex + 2 * ModExp	2 * ModExp + 1 * PointMulti	3 * ModExp
Csp	Re-encryption	1 * Pair	1 * Pair	1 * Pair
	Duplication check	1 * Pair	–	–
Data holder	System setup	1 * Exp	1 * PointMulti + 1 * ModExp	1 * ModInv + 1 * ModExp
	Duplication check	1 * ModExp	2 * ModExp + 1 * PointMulti	–
	Data upload	–	–	3 * ModExp
	Data retrieval	1 * Exp + 1 * Hash + 1 * ModExp	1 * ModExp	1 * ModExp
AP	System setup	1 * Exp	1 * ModExp	1 * ModExp
	Ownership check and rekey generation	1 * Exp + 1 * Pairing	2 * ModExp + 2 * PointMulti	1 * ModExp

Notes: Pair: Bilinear Pairing; Exp: Exponentiation in G_1 or G_T; ModInv: Modular Inversion; ModExp: Modular Exponentiation; N: Number of data holders; PointMulti: Point multiplication in ECC

5.3 Performance Analysis

System Setting

In this section, we further implemented the proposed schemes and tested their performances to check with our theoretic analysis and prove its correctness. The evaluations are performed on a laptop with Intel Core i5-3337U CPU 1.8 GHz and 8 GB RAM with Java Paring-Based Cryptography library (jPBC). To achieve better accuracy, we tested each algorithm 1000 times and reported the average value of all testing results. We choose AES as the symmetric key encryption. Unless particularly specified, some parameters in our tests are set as default values: (1) $\mathcal{L}(n) = 1024$ bits; (2) bilinear pairing parameters generator - TYPE A; (3) length of random numbers – 500 bits; (4) length of symmetric key – $\mathcal{L}(k) = 128$ bits.

Performance of AHRE

As AHRE is applied to encrypt symmetric key, we only evaluate its performance over data with $\mathcal{L}(m)$ = 128bits. The computation time of each algorithm in AHRE is presented in Table 3. Through tests, one pairing in jPBC library can be computed in approximately 10.1 milliseconds (ms). From the simulation results, we can observe that the encryption is a little time-consuming but the decryption is very efficient, which is acceptable for cloud users as they only need to execute it once. Moreover, if the cloud users are subsequent data holders, they do not need to execute encryption.

Table 3. The computation time of each algorithm in AHRE (Unit: ms)

Algorithm	System Setup	*KGen*	*Enc*	*RKGen*	*ReEnc*	*Dec*	*UKI*	*CipR*	*AH*	Pairing
Time	89.81	13.04	38.82	13.9	10.01	4.08	25.5	13.46	14.02	10.1

Performance of Our Proposed Schemes

In this experiment, we test the performance of our proposed schemes by testing their simulation time over different size of original files: 10 MB, 30 MB and 50 MB. As some basic operations are similar to the operations of AHRE and are not affected by the size of files, we only present the computation time varying with the size of original file, which is shown in Table 4.

Table 4. The computation time of some operations in deduplication schemes (Unit: ms)

Operation			Time		
			File size: 10 MB	File size: 30 MB	File size: 50 MB
Data upload (User)			82.1	317.9	618.1
Decryption before update or revocation (User)			55.78	300.7	589.7
Update	Scheme 1 (CSP)		62.2	298.1	581.8
	Scheme 2 (CSP)		117.6	619.4	1210.1
Revocation	Scheme 3	Step 1 (CSP)	13.7	13.8	13.8
		Step 2 (AP)	80.5	82.5	78.7
		Step 3 (CSP)	67.6	314.7	591.3
	Scheme 4	Step 1 (CSP)	13.6	14.0	12.3
		Step 2 (AP)	79.8	82.7	76.6
		Step 3 (CSP)	124.9	600.7	1202.0
Decryption after update or revocation (User)			104.8	608.7	1182.9

We can observe that the ciphertext update and user revocation almost double the decryption time of users. But it does not involve the data holders in the two operations. In most of existing schemes, the data holders need to download the outsourced file, decrypt the ciphertext and then re-encrypt them, which is not efficient, especially for large files, such as multimedia data. Though revocation and update introduce much computation overhead, they are almost undertaken by the CSP, which is acceptable for a cloud service provider.

6 Conclusion

Data deduplication helps improving the utilization of cloud storage and in turn helps reducing the storage cost of cloud users. In this paper, we proposed a flexible data deduplication scheme with effective ownership proof and user revocation. Our scheme can flexibly support outsourced data update and data sharing among data holders. Moreover, it can support public verifiability of ownership and user revocation without intervention of data owners, which greatly enhances cloud data security and effectively reduces the communication cost caused by ciphertext update, especially significant for large files. Extensive performance analysis and test shows that our scheme is secure and efficient. Although our scheme incurs a little higher computation cost than some existing work, it can provide advanced features. In future work, we will optimize our design and study privacy-preserving and verifiable data deduplication.

Acknowledgment. This work is sponsored by the National Key Research and Development Program of China (grant 2016YFB0800700), the NSFC (grants 61672410 and U1536202), the Project Supported by Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2016ZDJC-06), the 111 project (grants B08038 and B16037), and Academy of Finland (Grant No. 308087).

References

1. Meyer, D.T., Bolosky, W.J.: A study of practical deduplication. ACM Trans. Storage **7**(4), 1–20 (2012)
2. Ali, M., Dhamotharan, R., Khan, E., Khan, S.U., Vasilakos, A.V., Li, K., Zomaya, A.Y.: SeDaSC: secure data sharing in clouds. IEEE Syst. J. **99**, 1–10 (2015)
3. Liu, C., Yang, C., Zhang, X.Y., Chen, J.J.: External integrity verification for outsourced big data in cloud and IoT: a big picture. Future Gener. Comput. Syst. **49**, 58–67 (2015)
4. Puzio, P., Molva, R., Onen, M., Loureiro, S.: ClouDedup: secure deduplication with encrypted data for cloud storage. In: Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science, pp. 363–370. IEEE (2013)
5. Mulazzani, M., Schrittwieser, S., Leithner, M., Huber, M.: Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In: Proceedings of USENIX Security Symposium, p. 5 (2011)
6. Hur, J., Koo, D., Shin, Y., Kang, K.: Secure data deduplication with dynamic ownership management in cloud storage. IEEE Trans. Knowl. Data Eng. **28**(11), 3113–3125 (2016)
7. Kwon, H., Hahn, C., Kim, D., Hur, J.: Secure deduplication for multimedia data with user revocation in cloud storage. Multimedia Tools Appl. **76**(4), 5889–5903 (2017)
8. Dropbox: A file-storage and sharing service. http://www.dropbox.com/
9. Google Drive. http://drive.google.com
10. Mozy, Mozy: a file-storage and sharing service. http://mozy.com/
11. Sun, Z., Shen, J., Yong, J.M.: DeDu: building a deduplication storage system over cloud computing. In: IEEE International Conference on Computer Supported Cooperative Work in Design, pp. 348–355. IEEE (2014)

12. Wallace, G., Douglis, F., Qian, H.W., Shilane, P., Smaldone, S., Chamness, M., Hsu, W.: Characteristics of backup workloads in production systems. In: Proceedings of USENIX Conference on File and Storage Technologies, p. 500 (2012)
13. Wilcox, Z.O.: Convergent encryption reconsidered (2011). http://www.mail-archive.com/cryptography@metzdowd.com/msg08949.html
14. Bellare, M., Keelveedhi, S., Ristenpart, T.: DupLESS: server aided encryption for deduplicated storage. In: Proceedings of 22nd USENIX Conference on Security, pp. 179–194 (2013)
15. Wen, M., Ota, K., Li, H., Lei, J.S., Gu, C.H., Su, Z.: Secure data deduplication with reliable key management for dynamic updates in CPSS. IEEE Trans. Comput. Soc. Syst. **2**(4), 137–147 (2015)
16. Liu, J., Asokan, N., Pinkas, B.: Secure deduplication of encrypted data without additional independent servers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 874–885. ACM (2015)
17. Yan, Z., Wang, M.J., Li, Y.X., Vasilakos, A.V.: Encrypted data management with deduplication in cloud computing. IEEE Cloud Comput. **3**(2), 28–35 (2016)
18. Yan, Z., Ding, W., Zhu, H.: A scheme to manage encrypted data storage with deduplication in cloud. In: Wang, G., Zomaya, A., Perez, G.M., Li, K. (eds.) ICA3PP 2015. LNCS, vol. 9530, pp. 547–561. Springer, Cham (2015). doi:10.1007/978-3-319-27137-8_40
19. Yan, Z., Ding, W.X., Yu, X.X., Zhu, H.Q., Deng, R.H.: Deduplication on encrypted big data in cloud. IEEE Trans. Big Data **2**(2), 138–150 (2016)
20. Liu, C., Liu, X., Wan, L.: Policy-based de-duplication in secure cloud storage. In: Yuan, Y., Wu, X., Lu, Y. (eds.) ISCTCS 2012. CCIS, vol. 320, pp. 250–262. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35795-4_32
21. Lillibridge, M., Eshghi, K., Bhagwat, D.: Improving restore speed for backup systems that use inline chunk-based deduplication. In: Proceedings of USENIX Conference on File and Storae Technologies, pp. 183–198 (2013)
22. Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A.: Proofs of ownership in remote storage systems. In: Proceedings of the 18th ACM conference on Computer and communications security, pp. 491–500. ACM (2011)
23. Pietro, R.D., Sorniotti, A.: Boosting efficiency and security in proof of ownership for deduplication. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, pp. 81–82. ACM (2012)
24. Ng, W.K., Wen, Y., Zhu, H.: Private data deduplication protocols in cloud storage. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 441–446. ACM (2012)
25. Yang, C., Ren, J., Ma, J.F.: Provable ownership of file in de-duplication cloud storage. In: IEEE Global Communications Conference, pp. 695–700. IEEE (2013)
26. Bresson, E., Catalano, D., Pointcheval, D.: A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 37–54. Springer, Heidelberg (2003). doi:10.1007/978-3-540-40061-5_3