A Secure Privacy Preserving Deduplication Scheme for Cloud Computing

Yongkai Fan[1,2], Xiaodong Lin[2], Wei Liang[3], Gang Tan[4] ,Priyadarsi Nanda[5]

[1]*Dept. of Computer Science and Technology, ChiFeng University, Chifeng, China*,

[2]*Dept. of Computer Science and Technology, China University of Petroleum, Beijing, China*

[3]*Trusted Computing and Network Provincial Key Laboratory, Hunan University, Changsha,China*

[4]*Dept. of Computer Science and Engineering, Penn State University, PA, USA*

[5]*School of Electrical and Data Engineering, University of Technology, Sydney, Australia*

Abstract. Data deduplication is a key technique to improve storage efficiency in cloud computing. By pointing redundant files to a single copy, cloud service providers greatly reduce their storage space as well as data transfer costs. Despite of the fact that the traditional deduplication approach has been adopted widely, it comes with a high risk of losing data confidentiality because of the data storage models in cloud computing. To deal with this issue in cloud storage, we first propose a TEE (trusted execution environment) based secure deduplication scheme. In our scheme, each cloud user is assigned a privilege set; the deduplication can be performed if and only if the cloud users have the correct privilege. Moreover, our scheme augments the convergent encryption with users' privileges and relies on TEE to provide secure key management, which improves the ability of such cryptosystem to resist chosen plaintext attacks and chosen ciphertext attacks. A security analysis indicates that our scheme is secure enough to support data deduplication and to protect the confidentiality of sensitive data. Furthermore, we implement a prototype of our scheme and evaluate the performance of our prototype, experiments show that the overhead of our scheme is practical in realistic environments.

Keywords: Deduplication, Trusted Execution Environment, Cloud storage, Encryption

1 Introduction

Since Google first put forward the concept of "cloud computing" in 2006, cloud computing has drawn wide attention from industry and academia. With the development of hardware virtualization technology, the availability of high-capacity networks as well as the low-cost of computing resources and storage devices, cloud computing is capable to virtualize the computing infrastructure (computer networks, servers, storage, applications and service) and put them together to form a shared pool of configurable resources, which can provide users with a variety of computing and storage services which are seemingly unlimited capacity. By this paradigm, users can reduce their computing cost, while enjoying rich and convenient cloud services. However, as cloud computing becomes more popular than ever, the data stored in the cloud begins to increase sharply, bringing the difficulty of managing a massive amount of data.

Deduplication [1,26] was introduced to reduce the storage and network communication requirements in cloud computing. It is a specialized data-compression technique that aims to eliminate duplicate copies of data. Each piece of duplicate data is retained and an indexing scheme is designed so that all duplicate copies refer to that single instance. Therefore, if cloud users attempt to upload a file (block) that is already in the system, the cloud system will reuse the old file (block) and mark the users as one of the owners of the file (block).

Although deduplication has proven to be efficient in cloud computing, how to preserve security and privacy is a critical challenge. As data owners upload their data to the cloud, they lose control of the outsourced data: data owners cannot guarantee the secure management of their data in remote storage systems. Therefore, using encryption technology to protect the security of data seems to be an effective solution. Unfortunately, data deduplication is incompatible with traditional encryption methods. The purpose of data deduplication is to prevent the same data from being stored repeatedly; however, traditional encryption encrypts the same data with different keys and outputs different ciphertexts, which effectively disables data deduplication. To balance storage efficiency and data privacy, convergent encryption (CE) was first proposed by Douceur et al. [2] as a workaround. It is a deterministic scheme in which a plaintext m is encrypted by $\{C = E(m,k) \mid k = h(m)\}$, where h is a cryptographic hash function, E is a deterministic symmetric-encryption such as AES or DES, and C is the resulting ciphertext. In this way, identical plaintext will be encrypted into identical ciphertext, which enables the compatibility between data deduplication and encryption. In addition, to achieve effective access control in cloud storage, a protocol called proofs of ownership (PoW) [3] was introduced to verify the identity of data users. Supposing that a user Alice requests for some data stored in the cloud, she must certify her ownership of the data to the cloud service provider (CSP); if PoW is successful, Alice will receive a permission to perform duplicate checking and access the data; otherwise the access request will be rejected.

Both PoW and CE are widely used in deduplication systems [4-6], which rely on the aid of additional independent servers (also referred to as third party). All the systems are based on one assumption: the independent servers are reliable and secure. However, such a strong assumption is very difficult to realize in a commercial context. The trusted execution environment (TEE) [7] is one option for preserving security and privacy. The execution environment can be divided into two worlds as depicted in Fig. 1. The first one is the Rich Execution Environment (REE), which also is called as Normal World Execution Environment. In a real environment, the REE is an ordinary OS responsible for managing hardware and software resources of the device. The application running on REE is referred to as CA. The second world is the TEE in which Secure OS runs. Corresponding to the CA in the REE, we call the application on the TEE as trustlets, which is a secure application signed by the user and the access verification must be performed before loading into TEE. In this two different worlds, while CA requests for cryptographic operations, the key generation or storage assistance

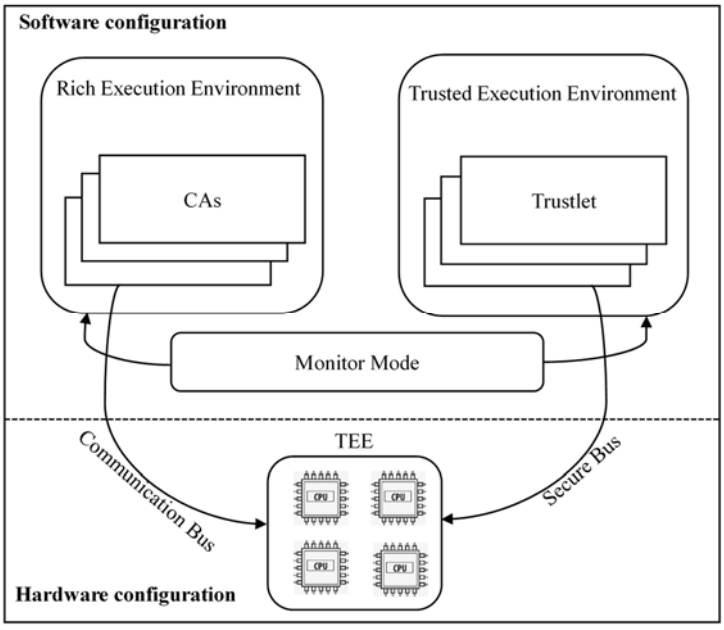


Fig. 1 Overview of TEE components

from trustlets, it must be licensed by the user and with the help of Monitor mode, CA cannot catch any sensitive information from trustlets during the interaction. In the meantime, TEE offers a secure communication side-channel between the external peripheral and the processor, which avoids the malware invading TEE by intercept the input and output.

In this paper, we propose a secure deduplication scheme that resorts to TEE for key management; specifically, TEE is used to replace third party servers. Our scheme restricts cloud users with insufficient privileges from accessing privileged data through a PoW protocol. In our approach, each user is assigned with a set of privileges during the system setup phase. While a user attempts to upload a file to the cloud, he/she needs to generate a privilege-based authentication code with the assistance of TEE. According to the authentication code, CSP determines whether the user has the right to perform a duplicate check or a PoW operation; only the user who proves his ownership of the file can retrieve the file. Such a change makes the scheme significantly more resistant to Man-in-the-Middle attacks and brute-force guessing attacks.

1.2 Contributions

Building on the above insights, our system makes several contributions, stated as follows:

- We present a scheme to achieve secure deduplication. Users can encrypt their data by semantically secure encryption, and the server can check users` access permission of a file on basis of the privileges that users hold.
- We adopt TEE to perform encryption in the client side. Thanks to the special key-management mechanism of TEE, the scheme is in strict compliance with encryption.
- We implement our scheme to show that it improves security and incurs lower overhead compared to previous deduplication schemes.

1.3 Organization

The remainder of this paper is organized as follows. In section 2, we describe related works. In section 3, we propose the system model and introduce the system goal for our scheme. In section 4, the implementation of our scheme is presented. The security analysis and performance evaluation for our scheme are shown respectively in section 5 and section 6. Finally, we conclude in section 7.

2 Related work

2.1 Convergent encryption with deduplication

Convergent encryption was proposed to enable both file-level deduplication [2] and block-level deduplication [8] with encrypted data, it is a special case of message-locked encryption (MLE) [9], which also presents a new security definition called “PRV$-CDA” (PRV and CDA stand for indistinguishability of random numbers and chosen distribution attack respectively) and proved CE is to strictly meet PRV$-CDA. However, most of the storage architectures have their limitations and difficulties in direct adoption, since these programs are craving for third-party assistance.

Puzio et al. [10] presented a secure block-level deduplication scheme called “ClouDedup”, which introduces a third-party server in charge of access control. Data owner first encrypts each data block with CE, then delivers it to the third-party server to perform additional encryption. Stanek et al. proposed a scheme that only deduplicates popular files and introduced an index sever to keep track of popular data [11]. Li et al. considered a hybrid cloud architecture for secure

authorized deduplication, which takes a private cloud as the auditor and key depository [12]. Bellare et al. proposed a system called DupLESS that implants an additional secret into the key generation process of CE [13], which is provided by the third parties. Subsequently, Duan showed an "Encryption with Signature" (EWS) scheme [14] that is similar to DupLESS, which is built upon an RSA-based threshold signature scheme [15-17], and generates the convergent key based on threshold signature with the help of a third-party server or other users. Although the third-party server is "honest-but-curious" in these proposals, if the server is comprised, it may lead a Man-in-the-Middle attack to the users. Recently, Zhang et al. proposed their scheme as HealthDep [31], which aimed at performing the EMRs deduplication and reduce storage cost. Kambo and Sinha also proposed the secure deduplication mechanism based on Rabin CDC and MD5, which allowed to process the data chunks for encrypted data on cloud [32].

2.2 Proof-of-ownership with deduplication

In 2011, Halevi et al. proposed the notion of "proof-of-ownership" (PoW) for secure deduplication system [3], which is somewhat similar to proofs of retrievability (PoRs) [18,19] and proofs of data possession (PDPs) [20] that achieve an opposite result. While the latter two consider the client as the prover rather than the server, the former focuses on proving the ownership of the client to the server. Recently, other proposals consider different implementations are discussed diffusely. Jin et al. combined PoW with proxy re-encryption (PRE), which not only supports deduplication, but also reduces the overhead effectively [21]. Ng et al. proposed the combination of public key cryptography together with PoW [22]. In his scheme, files are encrypted at client side and decryption keys are distributed among a specific group of users. Chen et al. proposed a model of message-locked proof of ownership, which provides secure deduplication and random block accessing etc. [23]. However, the overhead of key management is difficult to meet business requirements.

3 System model

In this section, we describe our scheme. Firstly, we introduce the cryptographic primitive and the notions we defined in this paper. Secondly, we introduce the adversary model. Then, the design goal of our scheme is described. Lastly, we describe our scheme in detail.

3.1 Preliminaries

1) Terminology: In this subsection, we describe the notions we have introduced to this paper, which can be listed as Table 1.

2) Cryptographic primitives: In this subsection, we also present all algorithms of the cryptographic primitive in this paper.

Collision-resistant hash functions: A collision-resistant hash function $H: \{0,1\}^* \rightarrow \{0,1\}^*$ that it is computationally infeasible to find two different values x and y that satisfy $H(x) = H(y)$.

One-way collision-resistant trapdoor function: $\widetilde{\mathcal{FH}}$ is a cryptographic function with trapdoor 1^λ as $\widetilde{\mathcal{FH}}: \{0,1\}^* \times \{0,1\}^\lambda \rightarrow \{0,1\}^*$ that it is computationally infeasible to find two different values x and y that satisfy $\widetilde{\mathcal{FH}}(\mathrm{x}\ ,\ \mathrm{k}) = \widetilde{\mathcal{FH}}(\mathrm{y}\ ,\ \mathrm{k})$, or find two different keys k_1 and k_2 that satisfy $\widetilde{\mathcal{FH}}(\mathrm{x}\ ,\ k_1) = \widetilde{\mathcal{FH}}(\mathrm{x},\ k_2)$.

Keyed-hash message authentication codes: A keyed-hash message authentication code $HMAC_k(x): \{0,1\}^* \times \{0,1\}^\lambda \rightarrow \{0,1\}^*$ is a deterministic hash function that takes an input x as well as a key k, outputs a value y.

Deterministic symmetric encryption functions: A deterministic symmetric encryption function $Enc_k(m): \{0,1\}^* \times \{0,1\}^\lambda \rightarrow \{0,1\}^c$ is a deterministic function that takes a key k as well as a plaintext m, outputs a ciphertext c.

Table 1. The notions in our scheme.

Symbol	Description
CSP	Cloud computing service provider
TEE	Trusted Execution environment
F	Original file belongs to user
k_{CE}	Convergent key to encrypt the file
P_ξ	A privilege set of the user
k_ξ	Privilege key to calculate authentication code

Deterministic symmetric decryption functions: A deterministic symmetric decryption function $Dec_k(c): \{0,1\}^c \div \{0,1\}^\lambda \rightarrow \{0,1\}^*$ is a deterministic function that takes a key k as well as a ciphertext $\boldsymbol{c}$, outputs a plaintext m.

3.2 Adversarial model

In the adversarial model, we consider the potential threats from both internal adversaries and external adversaries. The goal of the malicious adversaries is to get the trust of CSP and access the file stored in the cloud. We assume that the CSP is honest-but-curious, anyone who wants to access the file must follow the protocol negotiated in section 3.4.

Internal adversaries.

1) Rational CSP: We assume that CSP is honest-but-curious. However, in the case of increasing their profits in the system, they may deviate from the predefined protocol. Furthermore, the CSP, as well as some malicious insiders working for the CSP, may violate the confidentiality of the outsourced data to retrieve users' privacy.

External adversaries.

1) Hardware destroyer: The hardware destroyer targets to break the confidentiality of outsourced data by compromising the user's local device remotely. Such adversaries are unable to physically access users' device generally, but in the users' inadvertent, they may achieve their plot by intercepting the I/O operations on users' device.

2) Online adversaries: Such adversaries can eavesdrop on communication between CSP and users. Besides, they may disguise as users and attempt to forge the authentication code to cheat CSP.

For a detail security analysis against these adversaries is discussed in section 5 respectively.

3.3 Our design goals

The design objectives of our scheme can be summarized as follows:

Correctness. Each user can access the outsourced file if and only if he/she has the corresponding files and privileges, an authentication code based on the file and privilege being necessary when the user requests for the file.

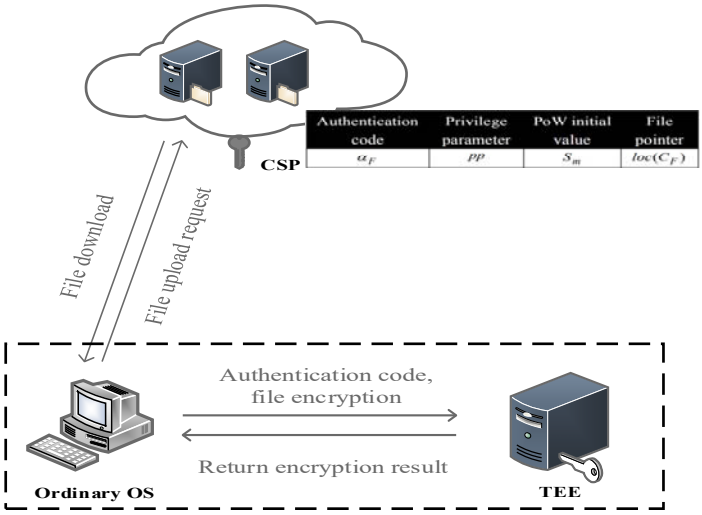


Fig. 2. Architecture of secure data deduplication.

Soundness. A malicious user without the complete file is unable to pass the ownership checking protocol. Moreover, a privileged file with convergent encryption can improve confidentiality of the file and protect it from outside attacks. Certainly, it supports for secure cross-user deduplication.

Secure key management. All the key is derived from TEE and stored in the shared memory, which means that all the key in TEE only can be accessed by authorized members.

3.4 System design

In this subsection, we will illustrate our system prototype in four parts. The whole secure deduplication architecture is depicted in Fig. 2. In order to make the diagram as concise as possible, we split the ordinary OS and TEE into two entities.

1) *System setup:* Assume that a privilege set $P = \{p_1, p_2, \cdots, p_n\}$, $n \in N^*$ is predefined. At the same time, each user is assigned with a privilege subset P_ξ, $P_\xi \subseteq P$ and $m = |P_\xi|$. The symmetric key k_ξ for each privilege $p_i \subseteq P_\xi$ is derived from TEE by a shared protocol. A PoW protocol is also defined, it can be described as $\rho = \{S, \Pi\}$, the proof algorithm is $S \leftarrow Proof$ performed in TEE, while the verify algorithm is $\Pi \leftarrow Verify$ performed in cloud side. An entry table to support data deduplication is also initialized in the cloud.

Algorithm 1: authentication code α_F generation

Input: An outsourced file F, privilege parameter pp, the privilege key k_ξ and the file hash value φ_F

Output: An encrypted file C_F

$H_{pp} \leftarrow HMAC_{k_\xi}(pp)$;

$k_{CE} \leftarrow \widetilde{\mathcal{FH}}(\varphi_F, H_{pp})$;

for

$p \mid\mid = p_i$;

end for

2) *Data outsourcing:* When a user requests for uploading content to the remote data center, he/she first sends the file F and the encryption privilege set P_ξ from CA to TEE. After receiving the file F and P_ξ, TEE computes a hash tag $\varphi_F = H(F)$ based on the file F and generates the privilege key k_ξ according to the received privilege set as $k_\xi = \widetilde{\mathcal{FH}}(p_1 \parallel \cdots \parallel p_i \parallel \cdots \parallel p_m, m_{d_m})$ for $p_i \in P_\xi$, m_{d_m} is a "dummy message" that is predefined corresponding to p_m for achieving secure encryption. Here, it is taken as the trapdoor in this algorithm. Along with these two values, an authentication code $\alpha_F = HMAC_{k_\xi}(\varphi_F)$ is computed and sent back to the user. Upon receipt of α_F, the user then sends it to the cloud. Once the CSP gets the authentication code, it first checks whether the corresponding file exists in its storage. In this context, we consider the authentication code as an access key for data deduplication. If the CSP does not possess the authentication code α_F, the user is seen as an initial uploader. Otherwise, he/she is considered as a subsequent uploader. The encoding algorithm of authentication code generation is shown as algorithm 1.

3) *PoW protocol initialize:* If the CSP cannot find the corresponding α_F in the entry table, the CSP regards the user as an initial uploader. Then the CSP will generate a privilege parameter *pp* puts it into the entry table, after that the CSP transmits *pp* to the user. Then the user sends *pp* with the file F to TEE, who first computes the convergent key $k_{CE} = \widetilde{\mathcal{FH}}(H(F), H(pp, k_\xi))$ and encrypts file F with the key in a deterministic way. A detailed encryption scheme can be shown as algorithm 2. Meanwhile, a PoW checking protocol is also activated. In the setup phase of PoW checking protocol, an initial value $\mathcal{S}_m$ is computed as the algorithm 4, which is generated using P_ξ and φ_F by hash and XOR operation. Later, TEE will destroy the convergent key k_{CE} in local environment and send the encrypted copy C_F and $\mathcal{S}_m$ back to the user. After receiving of C_F and $\mathcal{S}_m$, the user submits them to cloud. $\mathcal{S}_m$ is stored in the entry table as well as a pointer $loc(C_F)$, which is assigned to index the storage path of the cipher file C_F.

4) *Data update:* If the user is identified as a subsequent uploader, the user proceeds to prove his ownership to CSP. A PoW protocol is an interactive algorithm that requires the prover and the verifier to achieve together. In this paper, we consider the user as a prover while the cloud as a verifier. The detail of the checking protocol is shown as algorithm 4. First, the cloud generates a random parameter sp as well as a random $^{m}/_{2}$-element subset $R \leftarrow \{r_1, r_2, \cdots, r_{m/2}\}$ of the set $\leftarrow \{1, 2, \cdots, m\}$, which is the index of the privilege in P_ξ. Then, the cloud sends them to the user. After received the value from the cloud, TEE runs the algorithm $S \leftarrow Proof$ to computing $\mathcal{P}_i = H(p_{r_i}, H(File))$ and $\mathcal{U}_i = H(p_{u_i}, H(File))$,

Algorithm 3: Convergent decryption

Input: An encrypted file C_F, privilege parameter pp, the privilege key k_ξ and the file hash value φ_F

Output: A decrypted file F

$H_{pp} \leftarrow HMAC_{k_\xi}(pp)$;

$k_{CE} \leftarrow \widetilde{\mathcal{FH}}(\varphi_F, H_{pp})$;

$\{r_i, u_i | r_i \in R, u_i \in U - R\}$. Then it computes $\mathcal{P}_m = \bigoplus_{i=1}^{m/2} \mathcal{P}_i \oplus sp$, $\mathcal{U}_m = \bigoplus_{i=1}^{m-(m/2)} \mathcal{U}_i \oplus sp$ and commits them to the cloud. Cloud first computes $\mathcal{V}_m = \mathcal{S}_m \oplus \mathcal{U}_m$ and runs the algorithm $\Pi \leftarrow Verify$ to checking whether $\mathcal{P}_m$ is equal to $\mathcal{V}_m$. If $\mathcal{P}_m$ is unequal to $\mathcal{V}_m$, the updating request will be rejected. Otherwise, the user can access

Fig. 3. The workflow of data deduplication in our scheme.

send $\mathcal{S}_m$ to cloud;

end user

CSP:

chooses $R \leftarrow \{r_1, r_2, \cdots, r_{m/2}\} \subseteq U$, $U \leftarrow \{1, 2, \cdots, m\}$;

generate a secure parameter $sp \leftarrow \{0,1\}^*$;

send R and sp to the user;

end CSP

User:

for $i \leftarrow 1$ **to** $m/2$ **do**

$\mathcal{P}_i = H(p_{r_i}, \varphi_F)$;

end for

$\mathcal{P}_m = \bigoplus_{i=1}^{m/2} \mathcal{P}_i$;

$\mathcal{P}_m \oplus = sp$;

$\bar{R} \leftarrow \{u_1, u, \cdots, u_{n/2}\}$ that satisfies $\bar{R} = U - R$;

for $i \leftarrow 1$ **to** $m/2$ **do**

the file, decrypts the file by k_{CE} as algorithm 3, then update it, perform the phase ***Data outsourcing*** and ***PoW protocol initialize*** again. Because the user only needs to hold φ_F in local side rather than the other private information, this reduces much storage cost while performing the ownership protocol.

5) ***File retrieval:*** When the user wants to access the outsourced file in the cloud, the user firstly sends a retrieval request as well as the file name to the CSP. Upon receiving the file name and the request, the CSP will check whether the user has the permission to access the file. If so, the CSP will send back a corresponding cipher C_F. Thus, the user can decrypt the cipher by his own convergent key k_{CE} to get the original file F. Otherwise, a signal which indicates that the download

request has been rejected will be return to the user.

The whole steps in our deduplication scheme is shown as Fig. 3.

4 Implementation

We turn to describe the detail implementation that we have done for our proposed scheme following the technique described in the above algorithm. Our implementation includes two entities as separate C programs. A storage server is standing for the CSP to provide file storage service and perform data deduplication. A client is made up by an ordinary operating system (interacting with the storage server) and TEE (encrypting files and key management).

The proposed scheme is implemented using Ubuntu 14.04 LTS as an operating system, the cryptographic operations of hashing and encryption are computed by TEE internal API [7] and OpenSSL Library [24]. TEE is built upon a simulation environment Open-TEE [25]. The implementation is shown as the algorithms in section 3.

5 Security Analysis

Our deduplication scheme includes three entities: the privilege-based authentication code, the convergent encryption and PoW protocol. We assume that a privilege-based authentication code is fundamental to performing a duplicate check, while the PoW protocol and convergent encryption is aimed at guaranteeing the data confidentially and storage efficiency. In our assumption, they are secure enough to support our proposed deduplication scheme. Next, we will show that our scheme is secure to resist attacks we discussed in section 3.2.

5.1 Uniqueness of an authentication code

An authentication code is aimed at preventing the malicious user who without any privilege from accessing to the file. As we have shown in section 3.4, if a user attempts to interact with the cloud, an authentication code is required to be yielded through the corresponding file $\underset{\sim}{F}$ and the privilege set P_ξ. Without the file $\underset{\sim}{F}$ or P_ξ, it is impossible to yield a valid authentication code to request for duplicate checking. Next, let's consider such an active attack from two different scenarios: 1) We assume that an adversary who is a previous uploader. 2) We assume that an adversary who covets the outsourcing file but without the information of the file and the encryption privilege. We can formalize above the process by an interactive game between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

1） Assuming as the first scenario, the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ satisfy the following conditions, respectively.

Setup: In the first case, $\mathcal{A}$ is a previous uploader. Thus, $\mathcal{A}$ holds the privilege set P_ξ, which is necessary for calculating the authentication code, and the file F_{origin} that before update. Challenger $\mathcal{C}$ runs the algorithms 1 and 2 to forge the privilege key k_ξ, but $\mathcal{C}$ knows nothing about the trapdoor for generating k_ξ.

Hash query: $\mathcal{A}$ submits the file F_{origin} to challenger $\mathcal{C}$, then $\mathcal{C}$ computes the hash value of F_{origin} as $\varphi_{F_{origin}} = H(F_{origin})$.

Privilege key query: $\mathcal{A}$ submits its privilege set P_ξ to $\mathcal{C}$. According to the privilege set, $\mathcal{C}$ searches for k_ξ. If k_ξ

exists in the query, return k_ξ. Otherwise, returns $\mathcal{C}$ a forged privilege key k'_ξ.

Output: According to the privilege key k'_ξ and the hash value $\varphi_{F_{origin}}$, an authentication code can be forged as $\alpha_{F_{origin}} = HMAC_{k'_\xi}(\varphi_{F_{origin}})$ by $\mathcal{A}$. If $k'_\xi = k_\xi$, $\alpha_{F_{origin}} = \alpha_F$, the adversary $\mathcal{A}$ is said to win the game.

However, as mentioned before in section 4, the privilege key k_ξ is calculated by a one-way collision-resistant trapdoor function, which is collision-resistant. At the same time, an updated file F is modified by the latest uploader that is different from the previous one. Thus, $\alpha_{F_{origin}}$ is impossible to be forged and equal to the correct one α_F with non-negligible probability.

2） In the second scenario, the adversary $\mathcal{A}$ holds the outsourcing file C_F but without the information of the file and the encryption privilege. The challenger $\mathcal{C}$ can run algorithms as above and assign $\mathcal{A}$ with a privilege set P'_ξ.

Hash query: $\mathcal{A}$ submits the file C_F to challenger $\mathcal{C}$, then $\mathcal{C}$ computes a forged hash value of C_F as $\varphi_{F_f} = H(F_f)$.

Privilege key query: $\mathcal{A}$ submits its privilege set P'_ξ to $\mathcal{C}$. According to the privilege set, $\mathcal{C}$ returns a privilege key k''_ξ.

Output: According to the privilege key k''_ξ and the hash value φ_{C_F}, an authentication code can be forged as $\alpha_{F_f} = HMAC_{k''_\xi}(\varphi_{F_f})$ by $\mathcal{A}$. If $k''_\xi = k_\xi, \alpha_{F_f} = \alpha_F$, the adversary $\mathcal{A}$ is said to win the game.

Obviously, the above assumption is invalid since that $\varphi_{F_f} \neq \varphi_F$. Thus, α_{F_f} is impossible to be forged and equal to the correct one α_F.

5.2 Safety of encryption key

In this proposal, different kinds of keys are derived to satisfy different encryption algorithms. The encryption keys we have involved are: the privilege key k_ξ, the convergent key k_{CE}. We suppose such a scenario that there exist a hardware adversary is dedicating to despoiling all private information. If the encryption keys are preserved in local device, it is apparently that these keys are prone to be held by the attacker. In order to solve this conundrum, we resort to TEE to support key management. Since TEE can be viewed as a black-box from the outside world, illegal attacker is unable to learn the keys in TEE. Meanwhile, it is also forbidden to peek the internal implementation of TEE from outside, which means that it is invisible to outside attacker what kind of encryption algorithm we have chosen. Thus, even if the attacker is peeked at the information stored in TEE, the attacker is unable to purloin anything in an illegal way.

5.3 Confidentially of outsourcing data

Any file before it is outsourced to the CSP, the convergent encryption is invoked to support for secure data deduplication in the cloud. However, the data encrypted with such cryptosystem is unable to achieve the goal of semantic security. The notion of semantic security is proposed by Goldwasser and Micali [27]. A cryptosystem is semantically secure if a polynomial-time adversary who is given the ciphertext *Ç* of a message *m* as well as the length L_m, he/she cannot better distinguish any partial information of m than the other polynomial-time adversary who only holds L_m without *Ç*. It is a practical definition because it can be applied to a variety of attack modes such as chosen plaintext attack (CPA), chosen ciphertext attack (CCA1), and adaptive chosen ciphertext attack (CCA2), which can be categorized as brute-force attack. Clearly convergent encryption is not satisfied

with semantically secure as it leaks message equality. Recent research also shows that convergent encryption is suffering from these attacks [29-31]. To overcome these challenges, we improve the convergent encryption in section 3 and name it as privilege-based convergent encryption. Next, we will discuss the confidentiality of data that encrypted with our scheme and how it can withstand brute-force attacks.

The ciphertexts stored in the cloud is encrypted with a privilege-based convergent encryption, which is based on different privilege keys with the privilege parameter. While the traditional convergent encryption just relies on the file's hash value to get a ciphertext, a privilege-based convergent encryption combines the privilege parameter with the file's hash value to compute the convergent key $k_{CE} = \widetilde{\mathcal{FH}}(H(F), H(pp,\ k_{\xi}))$. As we have shown before, k_{CE} is kept in TEE, and it is unknown to the adversary unless the entire set of privilege keys is published. Therefore, we consider that our construction is semantically secure and is effective to against CPA, CCA1 and CCA2 since the adversary cannot obtain the encryption method.

5.4 Correctness of PoW checking protocol

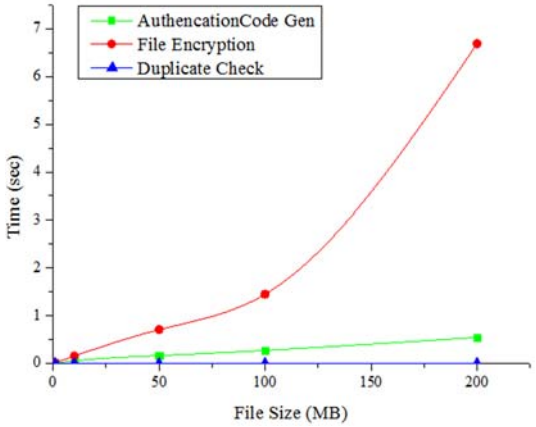
In the PoW checking protocol, we generate the proof value and verify value by randomly choosing the user's privileges and the hash value of the file that he/she holds. Namely, the proof value for performing PoW protocol is always different every time. Simultaneously, since these hash values are XORed with each other, and then XORed with the security parameter again, which makes sure that the adversary from outside world cannot catch any information about the original privilege set and the hash value of the file if the adversary did not capture the information of security parameter. Therefore, any attempt to subvert the pow protocol by forging the proof value will fail.

6 Performance evaluation

In this section, we provide a performance evaluation of our prototype. The overhead of data deduplication is affected by many factors, we chose two most important factors among these factors. One is the impact of file sizes, and the other is the number of stored files. In the following experiments, we evaluate the upload process in different phases: 1) Authentication code generation; 2) File encryption; 3) Duplicate check. At the same time, we compare our PoW proposal with other schemes, and then we show the time-consuming with different privileges and the storage cost of it. All the experiments were performed on a device with Intel(R) Core(TM) i5-3230M CPU processor running at 2.60 GHz and 4 G memory running Ubuntu 14.04 LTS 64-Bit Operation System.

6.1 Time consumption of different file sizes

To evaluate the effect of different file sizes on time consuming on different phases, we choose five different size files (5 for each type), the size of file ranges from 10 MB to 200 MB. We record the time breakdown in different phases with specific file size. The average time consuming of each phase test by different file sizes are plotted in Fig. 4. From the figure, we can see that the time consuming of authentication code generation, file encryption and duplicate checking is increasing linearly with different sizes. Moreover, most of the time consumption is distributed in the file encryption phase. For an outsourced file that its size is 50 MB, it costs less than 1s in encryption while the authentication code generation only need 0.15s. However, if the file size increases to 200MB, the time consuming of file encryption has a sharply growth while it compares to the authentication code generation.



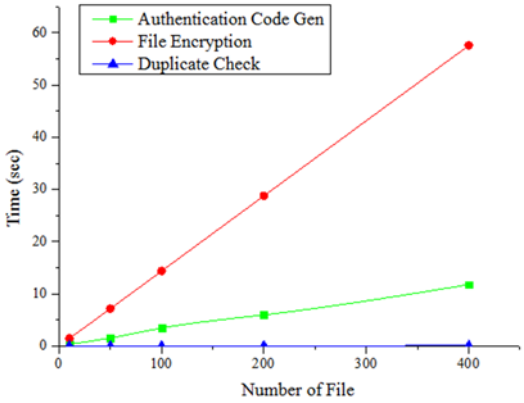


Fig. 4. Time breakdown for different file size

6.2 Time consumption of storing different number of files

To evaluate the effect of the number of files to time consuming on different phase, we choose 400 10MB unique files and upload them to the cloud continuously. Then we record the time breakdown of the whole files in different phases. As is shown in Fig. 5, the time consuming of each phase scales linearly along with the number of files. The time consuming of duplicate checking and authentication code generation are far less than the cost of file encryption.

6.3 Time consumption comparison in different PoW with different privilege set

The number of privileges in the privilege set has a crucial influence on the time-consuming of PoW checking protocol. In this experiment, we choose five different privilege sets that with different number of privileges from 10 to 100. According to the complete PoW protocol, the time consumption can be divided into initial time consumption, proof time consumption and verify time consumption, the experimental results are shown in Table 2. Since the number of elements in our chosen privilege set is only incremented by 10 each time, we can see that the time consuming increasing inconspicuous throughout the experiment, the time consumption in these three phases is always floats up and down in 0.0020s, 3.1s and 0.000016s, respectively.

Table 2. Time cost with different number of privileges.

Number of privileges	Time Cost (Sec)		
	Initial value	Proof	Verify
10	0.0020	3.1889	0.000016
20	0.0021	3.1805	0.000016
50	0.0027	3.1931	0.000012

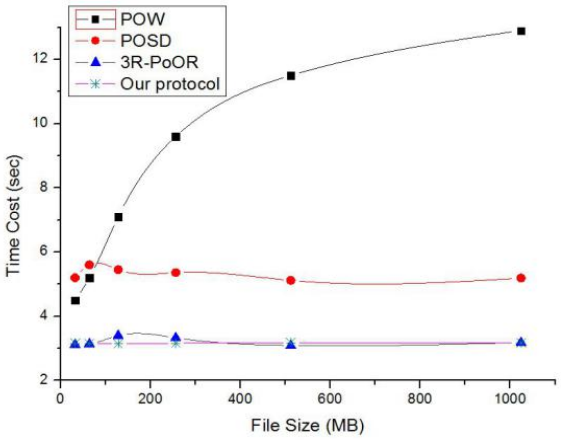


Fig. 6. Time consuming comparison of our scheme, 3R-PoOR [23]

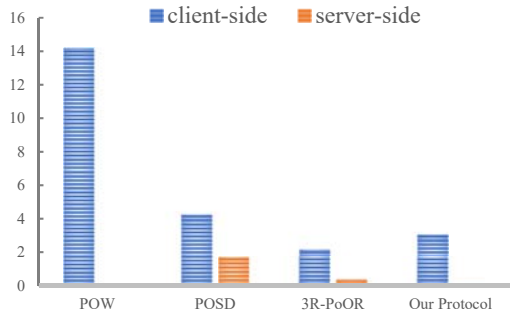


Fig. 7. Time consumption comparison of our PoW protocol, 3R-PoOR [23], POW [3] and POSD [28] with 1 GB file

6.4 Time consumption comparison of different PoW schemes

To show that our proposed PoW scheme is efficient and elastic, we implement other three schemes of Halevi, et al. [3] (POW), Zheng, et al [28] (POSD) and Chen, et al [23] (3R-PoOR) for a comparison. In our implementation, we choose 25 unique files in different sizes: 32MB, 64MB, 128MB, 256MB, 512MB (5 file for each size), and evaluate the average time consuming of each scheme with fixed file size. As is shown in Fig. 6, we present the simulation result of different schemes. From the result, we can see that our proposal is more efficient than the POW [3] and POSD [28] and closed to the 3R-PoOR. Besides, we also take the 1GB file for another comparison. In this experiment, we divide the time consuming into the user-side and the server-side and define that the testing privilege set with 50 different privileges. The complete comparison result is shown in Fig. 7. As we can see, our protocol is obviously better than POW and POSD in both user-side and server-side. Meanwhile, although the time consuming of our protocol is more than 3R-PoOR at the user side, we spend less time on the server side. It costs only about 0.000017s for the 1 GB file, since the value is very small, it cannot be visualized in Fig. 7.

6.5 Storage cost evaluation

For the user side, the storage cost is affected by the number of privilege that user holds yet is foreign to the size of the outsourced file. Regardless of whether the user is the initial uploader or the subsequent uploader, the storage cost is identical. In addition to the consumption of user-owned privilege information, the only storage consumption comes from the hash value of the outsourced file. In our proposal, while users attempt to upload his file to the cloud or update the outsourced file, he/she need to generate the authentication code, the convergent key and prove his ownership to CSP base on the privilege set and hash value of the file. Since that we take the SHA-256 hash algorithm to compute the file's hash value, the hash length is 256 bits, which is 32 bytes of storage consumption. For the privilege $p_i \in P$, it takes 64 bytes to store the encryption trapdoor m_{d_i}. Namely, users that with m privileges will cost about $64m + 32$ bytes for necessary information storage.

6.6 Communication cost evaluation

In our proposal, the transaction between users and the cloud runs throughout the whole process, and the communication overhead is the result of these interactions. In the authentication code generation phase and the file encryption phase, users

generate the authentication code to request for data outsourcing, then CSP return a privilege parameter pp to the users, which can assist the users in completing file encryption. Because these two values are consstant, we can regard the cost of them as $O(1)$. In the PoW checking protocol, the communication overhead is variable. First, the users send an initial value S_m to the cloud, $\mathcal{S}_m$ is constant so we consider the cost as $O(1)$. However, while the users prove his ownership to the cloud, the communication cost changes with the number of the users' privilege. The communication value is $\{\mathcal{P}_m, \mathcal{U}_m, sp, R\}$, the size of $\mathcal{P}_m$, $\mathcal{U}_m$ and sp are constant, so the cost is $O(1)$. While the size of the subset R is changeable, it costs about $O(\frac{m}{2})$ if the users request for ownership proven. Therefore, the total communication cost is $O\left(\frac{m}{2}\right) + 5O(1)$, since that $O(1)$ can be ignored, so the cost is $O(m)$.

7 Conclusion

In this paper, we propose a secure deduplication scheme that contains the operations of duplicate checking, proofs of ownership and convergent encryption. Only the users who hold legal privilege can perform the above operations. Meanwhile, we achieve the end of secure key management by aid of TEE, in which the encryption key cannot be peeped by unauthorized adversary. Hence, the outsourced data and encryption key can be safely stored in both cloud side and client side. To the best of our knowledge, it is the first data deduplication scheme that based on TEE for secure cloud storage. Finally, we perform a security analysis as well as performance evaluation, which shows that our proposal is efficient and feasible in practice.

Acknowledgment

This work was partially supported by CERNET Innovation Project (No. NGII20180406), by Beijing Higher Education Young Elite Teacher Project (No.YETP0683), by Beijing Higher Education Teacher Project (No. 00001149).

References

1. Data Deduplication, http://en.wikipedia.org/wiki/Data deduplication.

2. Douceur J R, Adya A, Bolosky W J, et al. Reclaiming Space from Duplicate Files in a Serverless Distributed File System[C]// International Conference on Distributed Computing Systems. IEEE, 2002.

3. Halevi S, Harnik D, Pinkas B, et al. Proofs of ownership in remote storage systems[C]//Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011: 491-500.

4. Li J, Chen X, Li M, et al. Secure Deduplication with Efficient and Reliable Convergent Key Management[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(6):1615-1625.

5. Jin X, Wei L, Yu M, et al. Anonymous deduplication of encrypted data with proof of ownership in cloud storage[C]// IEEE/CIC International Conference on Communications in China. IEEE, 2013.

6. Anderson P, Zhang L. Fast and secure laptop backups with encrypted de-duplication[C]// International Conference on Large Installation System Administration. USENIX Association, 2010.

7. TEE specifications, http://globalplatform.org/specificationsdevice.asp.

8. Quinlan, Sean, Dorward, et al. Venti: A New Approach to Archival Storage[J]. Proc.usenix Conf.on File & Storage Tech, 2002:89-101.

9. Bellare M, Keelveedhi S. Interactive Message-Locked Encryption and Secure Deduplication[M]// Advances in Cryptology – EUROCRYPT 2013. Springer Berlin Heidelberg, 2013.

10. Puzio P, Molva R, Onen M, et al. ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage[C]// IEEE International Conference on Cloud Computing Technology & Science. IEEE Computer Society, 2013.

11. Stanek J, Sorniotti A, Androulaki E, et al. A Secure Data Deduplication Scheme for Cloud Storage[J]. Ibm Corporation, 2014.

12. Li J, Li Y K, Chen X, et al. A hybrid cloud approach for secure authorized deduplication[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(5): 1206-1216.

13. Bellare M, Keelveedhi S, Ristenpart T. DupLESS: Server-aided encryption for deduplicated storage[C]// Proceedings of the 22nd USENIX conference on Security. USENIX Association, 2013.

14. Duan Y. Distributed Key Generation for Encrypted Deduplication: Achieving the Strongest Privacy[C]// ACM, 2014.

15. Bellare, Namprempre, Pointcheval, et al. The One-More-RSA-Inversion Problems and the Security of Chaum\"s Blind Signature Scheme[J]. Journal of Cryptology, 2003, 16(3):185-215.

16. Camenisch J, Neven G, Shelat A. Simulatable Adaptive Oblivious Transfer[J]. 2007.

17. Chaum D. Blind Signatures for Untraceable Payments[M]// Advances in Cryptology. 1983.

18. Juels A, Kaliski B S. Pors:proofs of retrievability for large files[C]// Acm Conference on Computer & Communications Security. 2007.

19. Hovav Shacham, Waters B. Compact Proofs of Retrievability[J]. Journal of Cryptology, 2013, 26(3):442-483.

20. Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores[C]//Proceedings of the 14th ACM conference on Computer and communications security. Acm, 2007: 598-609.

21. Jin X, Wei L, Yu M, et al. Anonymous deduplication of encrypted data with proof of ownership in cloud storage[C]// IEEE/CIC International Conference on Communications in China. IEEE, 2013.

22. Ng W K, Wen Y, Zhu H. Private data deduplication protocols in cloud storage[C]//Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012: 441-446.

23. Chen J, Zhang L, He K, et al. Message-locked proof of ownership and retrievability with remote repairing in cloud[J]. Security & Communication Networks, 2016, 9(16):3452-3466.

24. OpenSSL Project. http://www.openssl.org/.

25 Mcgillion B, Dettenborn T, Nyman T, et al. Open-TEE -- An Open Virtual Trusted Execution Environment[C]// 2015 IEEE Trustcom/BigDataSE/ISPA. IEEE, 2015.

26. Storer M W, Greenan K, Long D D E, et al. Secure data deduplication[C]// Acm International Workshop on Storage Security & Survivability. 2008.

27. Goldwasser S. Probabilistic encryption and how to play mental poker, keeping secret all partial information[J]. 14th STOC, 1982.

28. Zheng Q, Xu S. Secure and efficient proof of storage with deduplication[C]//Proceedings of the second ACM conference on Data and Application Security and Privacy. ACM, 2012: 1-12.

29. Pettitt. J.: Hash of plaintext as key? http://cypherpunks.venona.com/date/1996/02/ msg02013.html.

30. Perttula: Attacks on convergent encryption. http://bit.ly/yQxyvl.

31. Zhang Y, Xu C, Li H, et al. HealthDep: An Efficient and Secure Deduplication Scheme for Cloud-Assisted eHealth Systems[J]. IEEE Transactions on Industrial Informatics, 2018:1-1.

32. Kambo H, Sinha B. Secure data deduplication mechanism based on Rabin CDC and MD5 in cloud computing environment[C]// IEEE International Conference on Recent Trends in Electronics. IEEE, 2018.