IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 10, 2020, accepted February 27, 2020, date of publication March 2, 2020, date of current version March 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2977721

Lightweight Cloud Storage Auditing With Deduplication Supporting Strong Privacy Protection

WENTING SHEN[1], YE SU[2], AND RONG HAO[1]
[1]College of Computer Science and Technology, Qingdao University, Qingdao 266071, China
[2]School of Mathematics, Shandong University, Jinan 250100, China

Corresponding author: Rong Hao (hr@qdu.edu.cn)

This work was supported in part by the National Development Foundation of Cryptography under Grant MMJJ20170118, and in part by the Key Research and Development Project of Shandong Province under Grant 2019GGX101051.

ABSTRACT The cloud storage auditing with deduplication is able to verify the integrity of data stored in the cloud while the cloud needs to keep only a single copy of duplicated file. To the best of our knowledge, all of the existing cloud storage auditing schemes with deduplication are vulnerable to brute-force dictionary attacks, which incurs the leakage of user privacy. In this paper, we focus on a new aspect of being against brute-force dictionary attacks on cloud storage auditing. We propose a cloud storage auditing scheme with deduplication supporting strong privacy protection, in which the privacy of user's file would not be disclosed to the cloud and other parties when this user's file is predictable or from a small space. In the proposed scheme, we design a novel method to generate the file index for duplicate check, and use a new strategy to generate the key for file encryption. In addition, the user only needs to perform lightweight computation to generate data authenticators, verify cloud data integrity, and retrieve the file from the cloud. The security proof and the performance evaluation demonstrate that the proposed scheme achieves desirable security and efficiency.

INDEX TERMS Cloud storage auditing, deduplication, strong privacy protection, data security, cloud storage.

I. INTRODUCTION

With the rapid development of cloud computing, cloud storage has been widely accepted by individuals and enterprises for its advantages of universal access, low costs and on-demand service. Users can outsource complex computations to the cloud to reduce their computational burden [40], [41]. In addition, users also can outsource their large-scale data to the cloud to release their local storage burden [7], [30]. Under such a trend, it becomes urgent to guarantee the quality of data storage services for the users and the cloud. On one hand, the outsourced data might be corrupted or lost due to the inevitable operation errors or software/hardware failures in the cloud [27]. Thus, it is critical to develop cloud storage auditing, by which users can verify the integrity of cloud data without downloading the whole data from the cloud. On the other hand, lots of data stored in the cloud are duplicated. Based on the survey by EMC, 75% of cloud data are duplicated copies [8]. In order to improve the storage efficiency of the cloud, it is necessary to perform data deduplication [12], [33], where the cloud keeps only a single copy of the duplicated file and makes a link to the file for the users.

Users usually encrypt their data before outsourcing them to the cloud since they would not like to disclose their sensitive data to the cloud and other parties [9], [42]. In order to realize deduplication over encrypted data, the convergent encryption (CE) [6] was proposed to encrypt data. A convergent encryption algorithm encrypts data with a key deterministically derived from the data (e.g., the file's hash value). Thus, the same file will produce the same ciphertext. It means that the deduplication over ciphertexts is feasible. However, directly using CE is not secure in some situations. For example, when the file is predictable or from a small space [16], CE cannot resist brute-force dictionary attacks, in which the malicious cloud can recover the entire file with a number of guesses. In order to deal with this problem, Li *et al.* [18] proposed a secure auditing and data deduplication scheme

The associate editor coordinating the review of this manuscript and approving it for publication was Weizhi Meng.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

by introducing a key server to help user generate the convergent key. In this scheme, the cloud cannot deduce or derive the convergent key from the content of file since a secret ''seed'' is embedded in the convergent key. Unfortunately, the key server is able to guess or derive the file's content from the file's hash value sent from the user by launching the brute-force dictionary attacks. Therefore, this scheme cannot fully prevent the brute-force dictionary attacks. In addition, all users who want to upload file to the cloud need to generate a file index and send it to the cloud for duplicate check. With the file index, the cloud can verify whether the file uploaded by the user is duplicated or not. If the file index has been kept by the cloud, then the subsequent users do not need to upload data to the cloud any more. Most of deduplication schemes [14], [28], [33] set the hash value of the file as the file index. It will result in the data privacy leakage because the malicious cloud or other parties might guess or derive the content of file by performing the brute-force dictionary attacks. Thus, how to realize deduplication supporting strong privacy protection in cloud storage auditing is very important and valuable. Unfortunately, previous schemes are weak in privacy protection because they cannot fully defend against the brute-force dictionary attacks.

Our main contributions can be summarized as below:

In this paper, we investigate how to fully resist the brute-force dictionary attacks and realize deduplication with strong privacy protection in cloud storage auditing, and propose a concrete scheme satisfying this property. In order to realize deduplication with strong privacy protection, we design a novel method to generate the file index, and employ a new strategy to generate the key for file encryption. In the detailed design, the file index is generated with the help of an Agency Server (AS) instead of directly being produced by the hash value of file. The key for file encryption is generated with the file and the file label. The file label is kept by the user secretly. In this way, the privacy of the user's file is protected against the cloud and the AS. In order to improve the storage efficiency, the users, who own the same file, are able to generate the same ciphertext and the same authenticators. The proposed scheme effectively achieves data deduplication and authenticator deduplication. Furthermore, to reduce the computation burden on the user side, the user only needs to perform lightweight computation to generate data authenticators, verify the integrity of the cloud data, and retrieve his file from the cloud.

We give the security analysis of the proposed scheme, showing that the proposed scheme satisfies correctness, soundness and strong privacy protection. We also justify the performance by concrete implementations. The result shows that the proposed scheme is efficient.

A. RELATED WORKS

1) SECURE DEDUPLICATION

Deduplication is a popular technique in cloud storage, where the cloud keeps only a single copy of redundant data, regardless of how many users want to upload this file. In order to support deduplication over encrypted data, convergent encryption (CE) was proposed by Douceur *et al.* [6], which encrypts data under a message-dependent key (e.g., the hash of the file). It means that the users keeping the identical file can produce the identical ciphertext. Subsequently, Bellare *et al.* [3] formalized CE under the notion of Message-Locked Encryption (MLE). Keelveedhi *et al.* [16] presented the DupLESS by introducing a key server. In DupLESS, users encrypt their data with the MLE keys generated by the key server. Li *et al.* [17] designed distributed deduplication systems with high reliability. In this scheme, data privacy can be protected by using the secret splitting technique. To restrict the side channel information leakage, Halevi *et al.* [11] designed a Proof of Ownership (PoW) scheme, in which the users who want to store data to the cloud can make the cloud convince that they exactly own this file. Zheng *et al.* [42] designed a secure system toward encrypted cloud media by combining deduplication with video coding techniques. In [26], Singh *et al.* focused on the problems of fault tolerance and key management in deduplication, and presented a concrete scheme to address these problems. Yan *et al.* [33] designed a heterogeneous data management scheme supporting both deduplication and access control simultaneously. By constructing a role authorized tree, Xiong *et al.* [31] designed a secure encrypted data deduplication scheme, in which only the authorized user is able to access the specific file. Cui *et al.* [4] proposed an attribute-based encrypted data deduplication scheme in the cloud. In this scheme, users can share their data with other users by specifying access policies instead of sharing decryption keys.

2) CLOUD STORAGE AUDITING

In order to guarantee the integrity of data stored in the cloud, a number of cloud storage auditing schemes [2], [13], [15], [19], [22], [23], [32], [38] were proposed. Ateniese *et al.* [2] firstly proposed a notion of ''Provable Data Possession'' (PDP) and designed a publicly verifiable PDP scheme by utilizing homomorphic authenticators and random sample technique. In this scheme, an auditor is allowed to verify the integrity of cloud data without downloading the entire data from the cloud. Juels and Kaliski [15] constructed a model of ''Proof of Retrievability'' (PoR) and proposed a concrete scheme, which combines error-correcting codes and spot-checking technique to guarantee the retrievability and integrity of cloud data. Later, two PoR schemes with private verifiability and public verifiability were designed by Shacham and Waters [23] based on pseudorandom function and BLS signature. In order to support data dynamic, Yang *et al.* [34] designed a data integrity auditing scheme supporting dynamic data updates based on skip list structure. In this scheme, user is able to modify, insert, or delete his data blocks after uploading his data to the cloud. With the random masking technique, Wang *et al.* [29] proposed a privacy preserving cloud storage auditing scheme. To reduce user's computation overhead, Ding *et al.* [5] constructed a

TABLE 1. Notations.

Notation	Meaning
p	One large prime
G_1	Multiplicative cyclic group with order p
g, u	Two generators of group G_1
Z_p^*	A prime field with nonzero elements
H_1	A cryptographic hash function: $H_1 : \{0,1\}^* \rightarrow G_1$
H_2	A cryptographic hash function: $H_2 : G_1 \rightarrow \{0,1\}^\lambda$
h_1	A cryptographic hash function: $h_1 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$
h_2	A cryptographic hash function: $h_2 : \{0,1\}^* \rightarrow Z_p^*$
f	A pseudo-random function $f : \{0,1\}^* \times K_{prf} \rightarrow Z_p^*$
(x, y)	AS's private-public key pair
β_1	The file index
β_2	The file lable
sk_{enc}	The symmetric encryption key used to encrypt the file
C_F	The ciphertext stored in the cloud
n	The number of data blocks of ciphertext C_F
$sk = (k_{enc}, k_{mac})$	The user's private key
ε	The file tag
$\Phi = \{\sigma_i\}_{1 \le i \le n}$	The authenticator set of ciphertext C_F
c	The number of challenged data blocks
$PoW.Chall = \{v_i\}_{i \in I}$	The PoW challenge from the cloud
$PoW.proof = \sum_{i \in I} v_i c_i$	The PoW proof from the user
$Auditing.Chall = \{i, v_i\}_{i \in I}$	The auditing challenge from the user
$Auditing.proof = \{\mu, \sigma\}$	The auditing proof from the cloud

lightweight cloud storage auditing scheme with data privacy protection, which employs edge server to compute data authenticators and verify data integrity. Guo *et al.* [10] designed a cloud storage auditing scheme, which is able to execute and verify multiple update operations at once. Shen *et al.* [25] proposed a data integrity auditing scheme without private key storage. In this scheme, the biometric data, such as fingerprint and iris scan, is considered as the user's fuzzy private key to generate data authenticators, which avoids the use of the hardware token. By utilizing key update technique, Yu *et al.* [35], [36], Yu and Wang [37] proposed cloud storage auditing schemes with key-exposure resilience. Shen *et al.* [24] took the problem of data sharing with sensitive information hiding into account and designed a concrete identity-based cloud storage auditing scheme. In order to improve the cloud storage efficiency and save network bandwidth, Yuan and Yu [39] designed a public and constant cost cloud storage integrity auditing scheme with deduplication based on polynomial-based authentication tags and homomorphic linear authenticators. Li *et al.* [18] proposed a cloud storage auditing scheme with both data and authenticator deduplication by introducing a key server and an auditor. Liu *et al.* [20] presented another cloud storage auditing scheme with deduplication. In this scheme, in order to achieve authenticator deduplication, the initial user utilizes the file's hash value as the private key for computing data authenticators. Hou *et al.* [14] considered the problem of files' security levels according to data popularity in cloud storage auditing with deduplication, and proposed a cloud storage auditing scheme with deduplication supporting different security levels based on data popularity.

However, all of the above schemes cannot achieve the deduplication supporting strong private protection for encrypted data in cloud storage auditing. It means that the useful information of user's file might be disclosed to the malicious cloud or other parties when this user's file is predictable or from a small space.

B. ORGANIZATION

The rest of the paper is organized as follows. In Section II, we introduce the notions and system model. Section III presents design goals and definition. Section IV gives the proposed scheme. The security analysis and the performance evaluation are presented respectively in Section V and Section VI. In Section VII, we conclude the whole paper.

II. NOTIONS AND SYSTEM MODEL

A. NOTIONS

In Table 1, we show some notations used in the description of our scheme.

B. SYSTEM MODEL

The system model consists of three types of entities: the cloud, the user, and the Agency Server (AS), as illustrated in Fig.1.

(1) Cloud: The cloud has enormous storage space, and supplies storage services and downloading services for users. In order to improve storage efficiency, the cloud performs deduplication for duplicated files. In other words, the cloud keeps only a single copy of any duplicated file and its corresponding authenticators, and provides user with a link to the corresponding file.

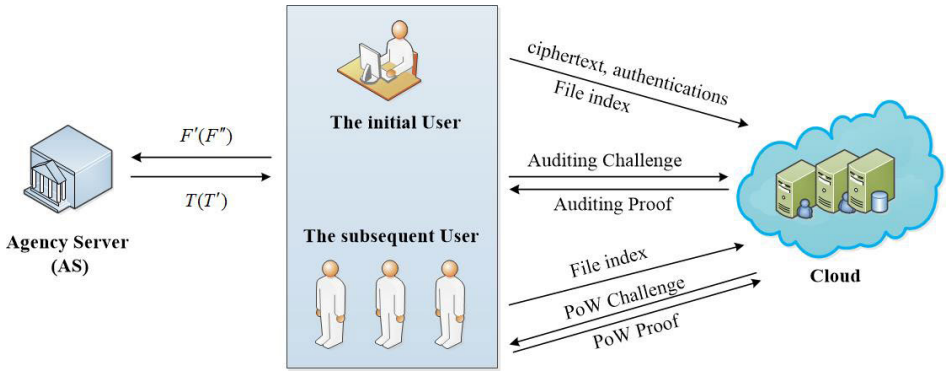


FIGURE 1. The system model.

(2) User: The user is divided into two categories. One is the initial user who uploads files that did not exist in the cloud previously. The other one is the subsequent users who upload files that the cloud has kept. The initial user generates the authenticators for each encrypted file, then uploads the encrypted file, its corresponding authenticators and the file tag to the cloud. The subsequent user does not need to generate the data authenticators and upload the above messages to the cloud. Later, both the initial user and the subsequent user can recover their data after downloading the data from the cloud. In addition, users are able to verify the integrity of the cloud data by executing the cloud storage auditing protocol with the cloud.

(3) AS: The AS is responsible for helping users generate the file index and the file label with his private key. With the file index, the cloud can verify whether the file uploaded by the user is duplicated or not. With the file label, the user can generate some keys for encryption and authenticator generation.

When an initial user wants to upload a file F to the cloud, he initially needs to interact with the AS to generate a file index for the uploaded file. Specifically, in order to protect the privacy of the file, the initial user firstly computes a blinded hash value F' for the file F. Then the initial user sends F' to the AS. Upon receiving F', the AS computes T with his private key, then returns it to the initial user. The initial user can generate the file index with T. The initial user sends the file index to the cloud as the file upload request. If the cloud did not maintain this file index, then the initial user encrypts his file and generates authenticators for the encrypted file. Finally, the initial user uploads the ciphertext and its corresponding authenticators to the cloud.

Similarly, when a subsequent user wants to upload a file F to the cloud, he firstly needs to interact with the AS to generate a file index for the uploaded file. Then the subsequent user sends the file index to the cloud as the file upload request. If the cloud has stored this file index, it will perform PoW (Proof of Ownership) protocol with the subsequent user to verify whether the subsequent user keeps the file indeed.

When the user (initial user or the subsequent user) wants to verify whether the cloud stores his intact ciphertext, he will send an auditing challenge to the cloud. Upon receiving the auditing challenge, the cloud generates and sends an auditing proof to the user. Finally, the user checks the integrity of the ciphertext by verifying whether this auditing proof is correct or not.

C. THREAT MODEL

The security goal in the proposed scheme is to fully resist the brute-force dictionary attacks and provide strong privacy protection for the users' files. We consider two types of adversaries: internal adversary and external adversary. The malicious cloud or the AS plays the role of the internal adversary, which is honest-but-curious. That is to say, the cloud performs the deduplication protocol honestly but tries to cheat the user about the data corrupt event or derive the file's content from the encrypted file. The AS faithfully performs the assigned operations, but intents to guess the contents of users' files. We do not consider the collusion of the cloud and the AS. The user, who somehow knows the file's content or the file's hash value, acts as the external adversary. He intends to obtain a link of a file but does not keep the corresponding file.

III. DESIGN GOALS AND DEFINITION

A. DESIGN GOALS

To achieve lightweight cloud storage auditing with deduplication supporting strong privacy protection, our scheme should satisfy the following goals:

1) **Correctness**: to ensure that the cloud can pass the user's validation only if the auditing proof it generates is valid, and the ciphertext retrieved from the cloud can pass the user's validation only if the ciphertext is intact.
2) **Auditing soundness**: to assure that if the cloud corrupts the user's data, it cannot pass the user's validation.
3) **Strong privacy protection**: to guarantee that both the cloud and the AS cannot extract the useful information of the file by launching the brute-force dictionary attacks.
4) **Lightweight computation on the user side**: to ensure that the user only needs to execute lightweight computation operations for authenticator generation, data integrity auditing and file retrieval.

5) **Efficient storage**: to guarantee that the cloud keeps only a single copy of the duplicated file and its corresponding authenticators.

B. DEFINITION

Definition 1: A lightweight cloud storage auditing with deduplication supporting strong privacy protection is composed by the following seven algorithms:

1) **Setup algorithm**. It takes as input a security parameter λ, and outputs the AS's private-public key pair (x, y) and the system public parameters pp.
2) **Initial Processing algorithm**. It takes as input the AS's private-public key pair (x, y) and the file F, and outputs the file index β_1 and the file label β_2. The initial user computes the symmetric encryption key sk_{enc} with the file label β_2 and the file F, then generates the ciphertext C_F with the key sk_{enc}.
3) **Authenticator Generation algorithm**. It takes as input the file index β_1, the file label β_2, the file identifier *name*, the file F and the PRF key k_{prf}. It generates the initial user's private key sk and a file tag ε. The initial user generates an authenticator set Φ for the ciphertext C_F with the PRF key k_{prf}.
4) **Subsequent Processing algorithm**. It takes as input the AS's private-public key pair (x, y) and the file F, and generates the file index β_1 and the file label β_2. The subsequent user generates the symmetric encryption key sk_{enc} with the file label β_2 and the file F, and outputs the ciphertext C_F with the key sk_{enc}. The subsequent user computes his private key sk according to the file label β_2, the file F and the file identifier *name*. Furthermore, the subsequent user executes the PoW protocol with the cloud to prove that he indeed owns the file.
5) **Proof Generation algorithm**. It takes as input the ciphertext C_F, the corresponding authenticator set Φ and the auditing challenge *Auditing.Chall*, and generates an auditing proof *Auditing.proof* that is used to prove the cloud stores the entire ciphertext C_F.
6) **Proof Verification algorithm**. It takes as input the auditing challenge *Auditing.Chall* and the auditing proof *Auditing.proof*, and returns "true" if the proof is valid; or "false", otherwise.
7) **Data Retrieval algorithm**. It takes as input the ciphertext C_F, its corresponding authenticators Φ and the file tag ε. The user verifies the validity of file tag ε, then checks the integrity of the ciphertext C_F. If the ciphertext C_F is intact, the user decrypts the ciphertext C_F, then recovers the file F.

Definition 2: We say a lightweight cloud storage auditing scheme with deduplication supporting strong privacy protection is secure if the following condition holds: whenever an adversary is able to pass the validation of challenger by generating a valid auditing proof *Auditing.proof* with non-negligible probability, there is a knowledge extractor that can extract the challenged data blocks except possibly with negligible probability.

IV. THE PROPOSED SCHEME

The details of the proposed scheme are as follows.

1) **Setup algorithm:** Let G_1 be a multiplicative cyclic group of the order p. Let g and u be two random generators of the group G_1. Pick four cryptographic hash functions $H_1 : \{0,1\}^* \to G_1, H_2 : G_1 \to \{0,1\}^\lambda$, $h_1 : \{0,1\}^* \to \{0,1\}^\lambda$ and $h_2 : \{0,1\}^* \to Z_p^*$, where λ is a security parameter. Select a pseudo-random function $f : \{0,1\}^* \times K_{prf} \to Z_p^*$. AS selects a random value $x \in Z_p^*$ as his private key, and calculates $y = g^x$ as his public key. The system public parameters are $pp = (G_1, p, g, u, H_1, H_2, h_1, h_2, y, f)$.
2) **Initial Processing algorithm:** The initial user interacts with AS to generate the file index β_1 and the file label β_2. The file index β_1 is used to verify duplicate file by the cloud. The file label β_2 is used to output some keys for encryption and authenticator generation. Then the initial user sends the file index β_1 to the cloud as the file upload request. If the cloud did not store this file index β_1, then the initial user encrypts his file F with the symmetric encryption key sk_{enc}. This process is illustrated in Fig.2.
   a) The initial user chooses $t \in Z_p^*$ at random, then computes the blinded hash value $F' = H_1(F)g^t$ and sends it to AS.
   b) Upon receiving F' from the initial user, AS computes $T = F'^x$ with his private key x, and sends it to the initial user.
   c) The initial user calculates $\alpha = Ty^{-t}$ with AS's public key y, then computes $\beta_1 = H_2(\alpha\|1)$ and $\beta_2 = H_2(\alpha\|2)$. Set β_1 as the file index, which is used to check duplicate file by the cloud. Set β_2 as the file label, which is used to generate the symmetric encryption keys sk_{enc} and k_{enc}, and the MAC key k_{mac}.
   d) The initial user sends the file index β_1 to the cloud as the file upload request. If the cloud did not keep β_1, the initial user computes the symmetric encryption key $sk_{enc} = h_1(\beta_2\|F)$, then encrypts the file F as follows: $C_F = Enc(F, sk_{enc})$. The ciphertext C_F is divided into n blocks $(c_1, c_2, ..., c_n)$, where $c_i \in Z_p^*$ denotes the i-th block of the ciphertext C_F.
3) **Authenticator Generation algorithm:** The initial user generates his private key $sk = (k_{enc}, k_{mac})$, the file tag ε, and the authenticator set Φ. The initial user uploads the ciphertext C_F, the set of authenticators Φ and the file tag ε to the cloud. The cloud makes a link to the file F for the initial user. The initial user holds the symmetric encryption key sk_{enc} and his private key $sk = (k_{enc}, k_{mac})$. The symmetric encryption key sk_{enc} is used to decrypt the ciphertext in Data

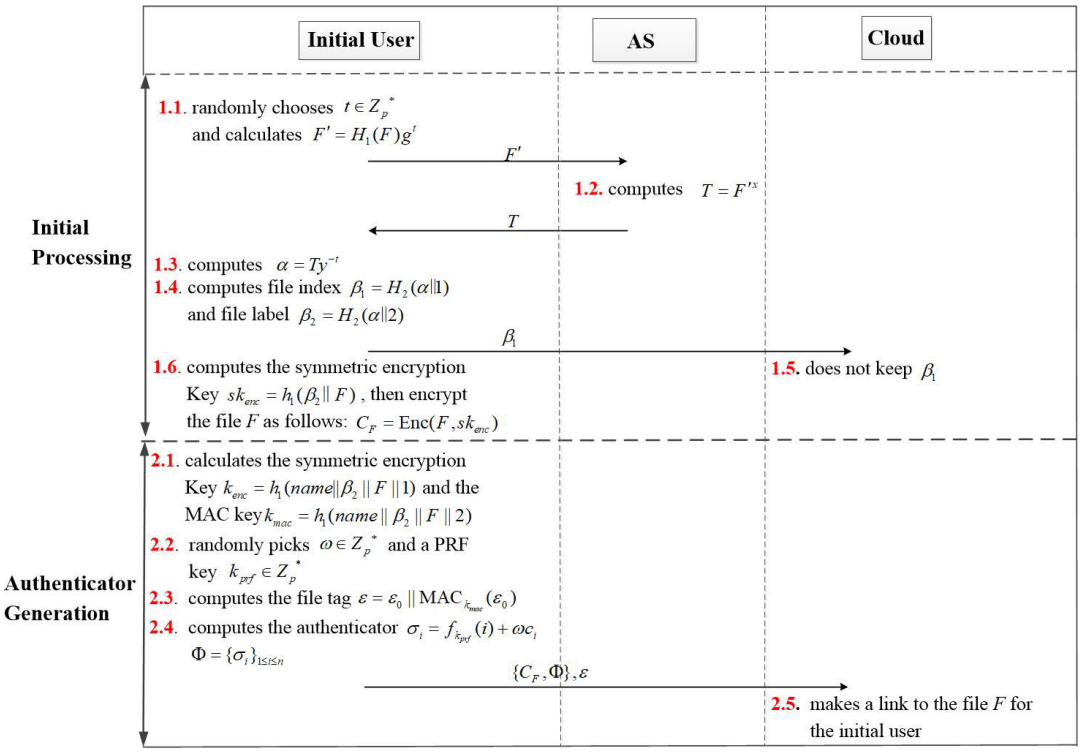


FIGURE 2. The processes of initial processing and authenticator generation.

Retrieval algorithm. The private key sk is used to generate the file tag ε and encrypt the PRF key k_{prf} and the random value ω in the *Authenticator Generation* algorithm, and verify the validity of file tag ε, decrypt the encrypted portion and recover the PRF key k_{prf} and the random value ω in the Proof Verification algorithm. This process is illustrated in Fig.2.

a) The initial user calculates the symmetric encryption key $k_{enc} = h_1(name||\beta_2||F||1)$ and the MAC key $k_{mac} = h_1(name||\beta_2||F||2)$, where $name$ is the identifier of the file F. The initial user's private key is $sk = (k_{enc}, k_{mac})$. The initial user randomly picks $\omega \in Z_p^*$ and a PRF key $k_{prf} \in Z_p^*$ for the pseudo-random function f.
b) The initial user computes the file tag $\varepsilon = \varepsilon_0 || MAC_{k_{mac}}(\varepsilon_0)$, where $\varepsilon_0 = n||Enc(k_{prf}||\omega, k_{enc})$. For each block $c_i \in Z_p^*(i \in [1, n])$ of the ciphertext C_F, the initial user computes the authenticator σ_i for block c_i as follows: $\sigma_i = f_{k_{prf}}(i) + \omega c_i$. Denote the set of authenticators as $\Phi = \{\sigma_i\}_{1 \le i \le n}$. The initial user uploads $\{C_F, \Phi\}$ along with the file tag ε to the cloud, then deletes these messages from local storage and holds the symmetric encryption key sk_{enc} and his private key $sk = (k_{enc}, k_{mac})$. The cloud makes a link to the file F for the initial user.

4) **Subsequent Processing algorithm:** The subsequent user interacts with AS to generate the file index β_1 and the file label β_2. Then the subsequent user sends the file index β_1 to the cloud as the file upload request. If the cloud has stored this file index β_1, then performs the PoW protocol with the subsequent user. With PoW, the subsequent user is able to prove to the cloud that he indeed keeps the file F without sending the entire file. The subsequent user generates his private key, and holds the symmetric encryption key and his private key. This process is illustrated in Fig.3.

a) The subsequent user chooses $t' \in Z_p^*$ at random, then computes the blinded hash value $F'' = H_1(F)g^{t'}$ and sends it to AS.
b) After receiving F'' from the subsequent user, AS computes $T' = F''^x$ with his private key x, and sends it to the subsequent user.
c) The subsequent user calculates $\alpha = T'y^{-t'}$ with AS's public key y, then computes the file index $\beta_1 = H_2(\alpha||1)$ and the file label $\beta_2 = H_2(\alpha||2)$. The subsequent user sends the file index β_1 to the cloud as the file upload request. If the cloud has stored β_1, then he will perform the following PoW protocol with the subsequent user.
   i) The cloud randomly picks a set of I with c elements, where $I \subseteq [1, n]$. For each $i \in I$, the cloud outputs a random value $v_i \in Z_p^*$, and sends a PoW challenge $PoW.Chall = \{v_i\}_{i \in I}$ to the subsequent user.
   ii) The subsequent user computes the symmetric encryption key $sk_{enc} = h_1(\beta_2||F)$,

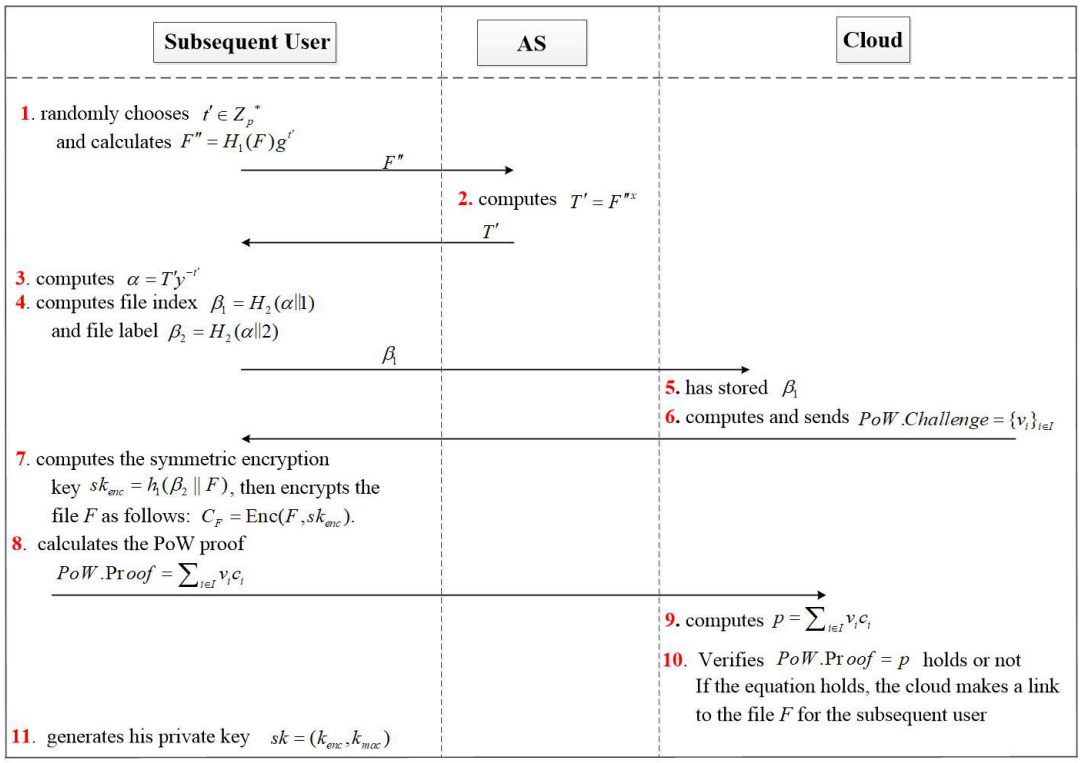


FIGURE 3. The process of subsequent processing.

and encrypts the file F as follows: $C_F = Enc(F, sk_{enc})$. Then the subsequent user calculates the PoW proof $PoW.\mathrm{Pr}oof = \sum_{i\in I} v_i c_i$, and sends it to the cloud.

iii) Upon receiving the PoW proof, the cloud computes $p = \sum_{i\in I} v_i c_i$, then verifies whether the following equation holds or not: $PoW.\mathrm{Pr}oof = p$. If the equation holds, the cloud believes that the subsequent user indeed owns the file F, and provides the subsequent user with a link to the file F. It means that the subsequent user does not need to upload the ciphertext C_F and its corresponding authenticators to the cloud any more.

d) The subsequent user generates his private key $sk = (k_{enc}, k_{mac})$ with the file label β_2, the file F and the file identifier *name*, where $k_{enc} = h_1(name||\beta_2||F||1)$ and $k_{mac} = h_1(name||\beta_2||F||2)$. The subsequent user holds the symmetric encryption key sk_{enc} and his private key $sk = (k_{enc}, k_{mac})$.

5) **Proof Generation algorithm:** The user outputs and sends an auditing challenge to the cloud. Then the cloud generates an auditing proof to respond to the user.

a) The user (the initial user or the subsequent user) randomly selects a c-elements subset $I (I \subseteq [1, n])$. For each $i \in I$, outputs a random value $v_i \in Z_p^*$. Then, the user generates and submits an auditing challenge $Auding.Chall = \{i, v_i\}_{i\in I}$ to the cloud.

b) Upon receiving the auditing challenge from the user, the cloud computes a linear combination of encrypted data blocks $\mu = \sum_{i\in I} c_i v_i$ and an aggregated authenticator $\sigma = \sum_{i\in I} \sigma_i v_i$. The cloud sends the auditing proof $Auditing.\mathrm{Pr}oof = \{\mu, \sigma\}$ along with the file tag ε to the user.

6) **Proof Verification algorithm:** The user firstly utilizes his private key k_{mac} to verify the validity of MAC on file tag ε. If the MAC is valid, the user parses ε, then decrypts the encrypted portion $Enc(k_{prf}||\omega, k_{enc})$ under his private key k_{enc} and recovers the PRF key k_{prf} and the random value ω. Finally, the user checks whether the following verification equation holds or not.

$$\sigma = \sum_{i\in I} v_i f_{k_{prf}}(i) + \omega\mu. \tag{1}$$

If the verification equation holds, it means that the cloud indeed keeps the user's intact data.

7) **Data Retrieval algorithm:** The user, who wants to access his file, will send a request to the cloud. The cloud returns the ciphertext C_F, its corresponding authenticators Φ and the file tag ε to the user. After receiving these messages, the user firstly verifies the integrity of the ciphertext C_F, then recovers his file F with the symmetric encryption key sk_{enc}.

a) The user, who wants to use his file, submits a request for the ciphertext C_F to the cloud.

b) Upon receiving the request from the user, the cloud firstly verifies whether the user is the data owner of the ciphertext C_F. If he is, the cloud sends the ciphertext C_F, its corresponding authenticators Φ and the file tag ε to the user. Otherwise, the cloud rejects the user's request.

c) After receiving the messages from the cloud, the user firstly verifies the validity of MAC on file tag ε with his private key k_{mac}. If the MAC is valid, the user parses ε, then decrypts the encrypted portion $Enc(k_{prf}||\omega, k_{enc})$ by using his private key k_{enc} and recovers the PRF key k_{prf} and the random value ω. The user checks whether the following verification equation holds or not.

$$\sum_{i\in[1,n]} \sigma_i = \sum_{i\in[1,n]} f_{k_{prf}}(i) + \omega \sum_{i\in[1,n]} c_i. \quad (2)$$

If the equation holds, the user believes the ciphertext C_F stored in the cloud is intact, then utilizes the symmetric encryption key sk_{enc} to decrypt the ciphertext C_F, and recovers the file F: $F = Dec(C_F, sk_{enc})$.

V. SECURITY ANALYSIS

Theorem 1 (Correctness): Our proposed scheme satisfies the following properties:

1.(Auditing correctness) If the auditing proof generated by the cloud is valid, the cloud is able to pass the verification of the user.

2.(Ciphertext correctness) If the ciphertext retrieved from the cloud is intact, this ciphertext is able to pass the verification of the user.

Proof: 1. Given a valid auditing proof $Auditing.Proof = \{\mu, \sigma\}$ from the cloud, the verification equation (1) can be proved correct as follows:

$$\begin{aligned}\sigma &= \sum_{i\in I} (f_{k_{prf}}(i) + \omega c_i) v_i \\ &= \sum_{i\in I} v_i f_{k_{prf}}(i) + \sum_{i\in I} v_i \omega c_i \\ &= \sum_{i\in I} v_i f_{k_{prf}}(i) + \omega \sum_{i\in I} v_i c_i \\ &= \sum_{i\in I} v_i f_{k_{prf}}(i) + \omega\mu\end{aligned}$$

2. Given a correct ciphertext C_F retrieved from the cloud and the corresponding authenticators Φ, the verification equation (2) can be proved correct as follows:

$$\begin{aligned}\sum_{i\in[1,n]} \sigma_i &= \sum_{i\in[1,n]} (f_{k_{prf}}(i) + \omega c_i) \\ &= \sum_{i\in[1,n]} f_{k_{prf}}(i) + \omega \sum_{i\in[1,n]} c_i\end{aligned}$$

□

Theorem 2 (Strong Privacy Protection): The data privacy of the proposed scheme satisfies the following properties:

1. The AS cannot derive the content of the real file F from the blinded hash value F' generated by the initial user or the blinded hash value F'' generated by the subsequent user by launching the brute-force dictionary attacks.

2. The cloud cannot extract the content of the real file F from the file index β_1 and the ciphertext C_F sent by the user by launching the brute-force dictionary attacks.

Proof: 1. As described in Section IV, the initial user computes a blinded hash value $F' = H_1(F)g^t$ with a random value t, and sends it to the AS. In this way, the hash of file F can be hidden. Thus, the AS is not able to extract the file F or the hash of the file F without the random value t of the initial user, even if the AS knows that the file F is from a dictionary $\{F_1, F_2, ..., F_s\}$. For the same reason, the AS cannot also derive the file F or the hash of file F without the random value chosen by the subsequent user, even if the AS knows that the file F is from a dictionary $\{F_1, F_2, ..., F_s\}$.

2. In our scheme, the file index β_1 for duplicate check is generated with the help of the AS instead of being produced by the file's hash value. Specifically, the user computes a blinded hash value $F' = H_1(F)g^t$, and sends it to the AS. The AS calculates $T = F'^x$ with his private key x, then returns it to the user. The user generates the file index β_1 by computing $\alpha = Ty^{-t} = H(F)^x$ and $\beta_1 = H_2(\alpha||1)$. Thus, the cloud is not able to generate the file index β_1 for each file $\{F_1, F_2, ..., F_s\}$ in the dictionary by itself without the AS's private key x. It means that the cloud is not able to guess or extract the file F without the AS's private key x, even if the cloud knows that the file F is from a dictionary $\{F_1, F_2, ..., F_s\}$. In addition, the ciphertext C_F is generated by the user with the symmetric encryption key $sk_{enc} = h_1(\beta_2||F)$, where β_2 is the file label. The generation method of file label β_2 is the same as that of file index β_1. Therefore, we can know that the cloud is not able to generate the file label β_2 for each file $\{F_1, F_2, ..., F_s\}$ in the dictionary by itself without the AS's private key x. And then, the cloud is not able to guess or extract the file F without the AS's private key x, even if the cloud knows that the file F is from a dictionary $\{F_1, F_2, ..., F_s\}$. □

Theorem 3 (Auditing Soundness): Suppose the symmetric encryption scheme is semantically secure, the MAC scheme is unforgeable and the pseudo-random function (PRF) is secure. In the proposed scheme, for adversary or untrusted cloud, it is computationally infeasible to forge an auditing proof that is able to pass the validation of the user if the cloud does not store the intact data.

Proof: We complete this proof by constructing a knowledge extractor and employing the method of knowledge proof [23]. If the cloud can pass the validation of the user without storing the intact data, the extractor is able to interact with the proposed scheme to extract the intact challenged data blocks. We will prove this theorem by a sequence of games.

Game 0. The challenger generates the key for generating authenticators and the system public parameters, and sends these public parameters to the adversary. The adversary chooses a series of encrypted data blocks $c_1, c_2, ..., c_n$, and queries the corresponding authenticators of these data blocks. The challenger calculates the corresponding

MAC-authenticated tags and data authenticators, and sends these messages to the adversary. The challenger submits an auditing challenge $Auding.Chall = \{i, v_i\}_{i\in I}$ to the adversary. The adversary responds an auditing proof $Auditing.Proof = \{\mu, \sigma\}$ based on challenge. If this proof is correct, the adversary succeeds in this game.

Game 1. Game 1 is the same as Game 0, except that the challenger maintains a list which records all the MAC-authenticated tags issued as part of a store queries. If the adversary issues one tag which is valid but not on the challenger's record list, the challenger declares failure and aborts.

Analysis. If the adversary is able to make the challenger abort in Game 1 with non-negligible probability, it means the adversary is a valid forger against the MAC scheme. It contradicts that the MAC scheme is unforgeable. Thus, the MAC-authenticated tags are all generated by the challenger.

Game 2. In Game 2, the challenger employs a random bit-string of the same length to replace the encryption of $k_{prf}||\omega$ in the tags. When the adversary gives a tag, in which the MAC on this tag is verified valid, the challenger utilizes the values which would have been encrypted in the tag, instead of intending to decrypt the ciphertext.

Analysis. In Game 2, the challenger does not decrypt the ciphertext which is not generated by himself since he can only see the tags with valid MACs generated by himself. Therefore, the challenger stores a table of plaintexts values $k_{prf}||\omega$ and the corresponding bit string emitted by himself as their tags.

We can break the semantic security of the symmetric encryption scheme by using the adversary if the adversary succeeds between Game 1 and Game 2 with non-negligible probability. In order to bridge the gap between Game 1 and Game 2, we must employ a hybrid argument between ''no valid encryptions'' and ''all valid encryptions''. It will cause the reduction suffer a $1/q_s$ security loss, where q_s is the number of queries made by the adversary.

Specifically, the challenger interacts with the adversary based on Game 0 and records the files stored by the adversary. Then, if the adversary wins in any cloud storage auditing but the auditing proof $\{\mu, \sigma\}$ he generates is different from what would be generated by the honest prover, then the challenger will abort and output ''true''; otherwise, output ''false''. Assume that the challenger outputs ''true'' with some non-negligible probability ε_0 if its behavior is as specified in Game 0, outputs ''true'' with some non-negligible probability ε_1 if its behavior is as specified in Game 1, and outputs ''true'' with some non-negligible probability ε_2 if its behavior is as specified in Game 2, we will show that the difference between ε_0 and ε_1 is negligible as long as the MAC scheme is unforgeable and the difference between ε_1 and ε_2 is negligible as long as the symmetric encryption scheme is secure.

In Game 1, the challenger employs the ciphertext of $k_{prf}||\omega$ to generate each tag. In Game 2, the challenger encrypts a random string with the same length instead of what in each tag it generates. Assume that $|\varepsilon_2 - \varepsilon_1|$ is non-negligible. Consider the hybrid in which the challenger generates the first i tags by encrypting a random string and the remaining $q_s - i$ tags by encrypting random values. Therefore, there must be a value of i such that the difference between the outputs of the challenger in hybrid i and hybrid $i+1$ is at least $|\varepsilon_2 - \varepsilon_1|/q_s$, which is non-negligible. Based on this, we will design an algorithm B which is used to break the security of the symmetric encryption scheme.

The encryption oracle for the encryption key k_{enc} is accessible to algorithm B, as well as a left-or-right oracle, which given strings m_0 and m_1 with the same length, generates the encryption of m_a, where a is a bit selected at random. Algorithm B interacts with the adversary, playing the role of the challenger. When algorithm B answers first i store queries of the adversary, it gets the encryption of $k_{prf}||\omega$ by employing its encryption oracle. The encryption of $k_{prf}||\omega$ is included in the tag. When algorithm B answers $(i + 1)$ st store query of the adversary, it calculates the correct plaintext $m_0 = k_{prf}||\omega$ and a random plaintext m_1 with the same length, and sends them to its left-or-right oracle. When algorithm B answers the remaining store queries of the adversary, it calculates the correct plaintext and a random plaintext with the same length, and employs its encryption oracle to encrypt this random plaintext, and contains the result in the tags. Algorithm B records the files kept by the adversary. If adversary wins in any cloud storage auditing but the auditing proof $\{\mu, \sigma\}$ he outputs is different from what would be generated by the Proof Generation algorithm, then algorithm B will abort and output ''true''; otherwise, will output ''false''.

Algorithm B interacts with adversary based on hybrid i if the left-or-right oracle encrypts its left input. Algorithm B interacts with adversary based on hybrid $i + 1$ if the left-or-right oracle encrypts its right input. The difference in the behavior of adversary and that of algorithm B is non-negligible, which breaks the security of the symmetric encryption scheme. Note that, the values $k_{prf}||\omega$ are selected randomly for each file and independent of each file, thus the values which are given by algorithm B to its left-or-right oracle are consistent with a query it makes to its encryption oracle with negligible probability.

Game 3. In Game 3, the challenger utilizes the random values in Z_p rather than the outputs of PRF. The challenger holds these values which are used to check the correctness of the adversary's responses in cloud storage auditing schemes. Concretely, the challenger evaluates $f_{k_{prf}}(i)$ by outputting a random value r in Z_p and inserting an entry (k_{prf}, i, r) in a table instead of applying the PRF algorithm. The challenger queries this table when evaluating the PRF to guarantee consistency.

Analysis. We can break the semantic security of PRF by using the adversary if the adversary succeeds between Game 2 and Game 3 with non-negligible probability. Note that, the tags given to the adversary in Game 2 do not include k_{prf}, thus the simulator does not need to know

this value. The adversary only can know the outputs of PRF. It means that if the adversary can distinguish the outputs of PRF from random values, we can break the security of the PRF by using this adversary.

As the analysis of Game 2, the difference in behavior we employ to break the PRF's security is the event that the adversary wins in a cloud storage auditing interaction but generates an auditing proof $\{\mu, \sigma\}$ which is different from what would be generated by the honest prover.

As before, it is necessary to use a hybrid argument to prove Game 3 with a security loss $1/(Mqs)$ in the reduction, where M is a bound on the number of blocks in any file that the adversary requests to have been stored.

Game 4. In Game 4, the challenger executes cloud storage auditing scheme initiated by the adversary, which is different from Game 3. In each such cloud storage auditing, the challenger issues an auditing challenge as before. Nevertheless, the challenger checks the correctness of auditing proof generated by the adversary, which is different from the auditing proof generated by the honest prover.

The challenger holds a table of the Authenticator Generation queries issued by the adversary and its corresponding responses; The challenger knows that the honest prover would generate an auditing proof $\{\mu, \sigma\}$ to respond the query it issued according to the table stored by himself. If the auditing proof generated by the adversary is equal to the above auditing proof generated by the honest prover, the challenger accepts the response of the adversary and returns ''true''. If the auditing proof generated by the adversary is not equal to the above proof generated by the honest prover, the challenger rejects the response of the adversary, and returns ''false''.

Analysis. The adversary in Game 4 is the same as in Game 3, with one difference. That is the auditing proof generated by the adversary (1) can pass the verification of verifier but (2) is not what would have been calculated by the challenger who plays the role of an honest prover in one of the cloud storage auditing interactions. We show that the probability that this happens is negligible.

Assume that $Auding.Chall = \{i, v_i\}_{i\in I}$ is the challenge generated by the challenger. An honest prover generates a valid auditing proof $\{\mu, \sigma\}$, where $\mu = \sum_{i\in I} c_i v_i$ and $\sigma = \sum_{i\in I} \sigma_i v_i$. From the correctness of our scheme, we obtain

$$\sigma = \sum_{i\in I} v_i r_{k_{prf},i} + \omega\mu. \tag{3}$$

Suppose the adversary generates an auditing proof $\{\mu', \sigma'\}$ which is different from the honest prover generated. Because the challenger aborted, we can know that the forgery of adversary is successful. In other words, $\sigma' \neq \sigma$ but the auditing proof $\{\mu', \sigma'\}$ still can pass the verification of the following equation:

$$\sigma' = \sum_{i\in I} v_i r_{k_{prf},i} + \omega\mu', \tag{4}$$

where $r_{k_{prf},i}$ is a random value which is used to replace to $f_{k_{prf}}(i)$ in Game 2.

Obviously, $\mu' \neq \mu$, otherwise $\sigma'=\sigma$, which contradicts our assumption above. Thus, define $\Delta\mu = \mu' - \mu$, and subtract the verification equation for σ from that for σ', we obtain $\Delta\sigma = \omega\Delta\mu$.

The bad event occurs exactly when $\Delta\mu$ is not zero. It means that the auditing proof generated by adversary is different from what generated by the honest server.

However, the value ω for every file is selected at random, thus it is independent of the adversary's view. The value ω is no longer encrypted in the tag, and its only other appearance is in calculating $\sigma_i = r_{k_{prf},i} + \omega c_i$. The output $f_{k_{prf}}(i)$ of PRF is replaced by a random value $r_{k_{prf},i}$. As a result, σ_i is independent of ω. Thus, the probability that the bad event happens if the challenger first chooses the random value ω for each stored file and then executes the cloud storage auditing interactions is the same as the probability that the bad event happens if the challenger first executes the cloud storage auditing interactions and then selects the value ω for each file.

Fix the sequence the values $\Delta\mu$ and $\Delta\sigma$ in auditing proof generated by the adversary and the choice of ω. The probability of challenger aborts is the same as the probability of $\Delta\sigma = \omega\Delta\mu(modp)$ holds for a specific entry in an interaction, which is $1/p$. Therefore, the probability that the equation $\Delta\sigma = \omega\Delta\mu(\mathrm{mod}p)$ holds for a nonzero number of entries is at most q_p/p, where q_p is the number of the cloud storage auditing interactions initiated by the adversary. Therefore, the adversary never outputs an auditing proof which is different from an honest server generated except with negligible probability q_p/p. Thus, the view of the adversary in Game 4 is the same as the view in Game 3 with negligible probability.

Finally, we construct a knowledge extractor to extract all of challenged data blocks $c_i(i \in I, |i| = c)$. By using c different coefficients $v_i(i \in I, |I| = c)$ and generating c times different challenges on the same data blocks $c_i(i \in I, |i| = c)$, the knowledge extractor is able to get c independently linear equations in the variables $c_i(i \in I, |i| = c)$. The knowledge extractor is able to obtain $c_i(i \in I, |i| = c)$ by solving these equations. □

VI. PERFORMANCE EVALUATIONS

In this section, we first compare the functionalities of our scheme and several related schemes, and evaluate the computation overhead and the communication overhead of our scheme by theory analysis and simulated experiments.

A. FUNCTIONALITIES COMPARISON

We compare our scheme with several related schemes [5], [11], [14], [18] on the functionality in Table 2. The scheme [18] cannot support strong privacy protection, in which data privacy will be leaked to the key server. The scheme [5] cannot achieve data deduplication and authenticator deduplication, which incurs heavy storage overhead on the cloud side. In the scheme [14], every user needs to perform time-consuming operations to generate data authenticators based on the BLS signatures. In [11], the data stored in the cloud might be corrupted or lost.

TABLE 2. Functionality comparison between our scheme and related schemes.

Schemes	Data integrity autiting	Strong privacy protection	Light-weight computation on the user side	Data deduplication	Authenticator deduplication
Li et al. [18]	Yes	No	No	Yes	Yes
Ding et al.[5]	Yes	Yes	Yes	No	No
Hou et al.[14]	Yes	No	No	Yes	No
Halevi et al.[11]	No	No	No	Yes	No
Ours	Yes	Yes	Yes	Yes	Yes

In conclusion, our scheme is the only scheme with all of the following properties: data integrity auditing, strong privacy protection, light-weight computation on the user side, data deduplication and authenticator deduplication.

B. THEORY ANALYSIS

We define the following notations to denote the operations in our scheme. Mul_{G_1}, Exp_{G_1} and $Hash_{G_1}$ represent one multiplication operation, one exponentiation operation and one hashing operation in G_1 respectively. PRF_f represents one PRF function operation. $Add_{Z_p^*}$ and $Mul_{Z_p^*}$ respectively represent one addition operation and one multiplication operation in Z_p^*. $E_{sk_{enc}}$ represents one symmetric encryption operation. n is the total number of data blocks. c is the number of challenged data blocks. $|n|$ is the size of an element of set $[1, n]$. $|p|$ is the size of an element in Z_p^*.

1) COMPUTATION OVERHEAD

We mainly evaluate the computation overhead of the user and the cloud. Both the initial user and the subsequent user need to cost $2(Mul_{G_1} + 2Exp_{G_1} + 2Hash_{G_1})$ to generate the file index and the file label, and cost $c(PRF_f + Add_{Z_p^*}) + (c+1)Mul_{Z_p^*}$ to verify the integrity of cloud data. The initial user needs to cost $n(PRF_f + Mul_{Z_p^*} + Add_{Z_p^*})$ to generate data authenticators. The subsequent user needs to cost $E_{sk_{enc}} + cMul_{Z_p^*} + (c-1)Add_{Z_p^*}$ to make the cloud convince that he exactly owns the file. For generating an auditing proof $Auditing.\Pr oof = \{\mu, \sigma\}$ ($\mu = \sum_{i\in I} c_i v_i$, $\sigma = \sum_{i\in I} \sigma_i v_i$), the cloud consumes $2cMul_{Z_p^*} + (2c-2)Add_{Z_p^*}$.

2) COMMUNICATION OVERHEAD

The communication overhead of our scheme mainly comes from the auditing phase, which includes two parts: auditing challenge and auditing proof. For an auditing challenge $Auditing.Chall = \{i, v_i\}_{i\in I}$, its size is $c \cdot (|n| + |p|)$ bits. The size of an auditing proof $Auditing.Proof = \{\mu, \eta\}$ is $2|p|$ bits. Thus, in the phase of auditing phase, the total communication overhead is $c \cdot |n| + (c+2) \cdot |p|$ bits.

C. EXPERIMENTAL RESULTS

In order to evaluate the performance of the proposed scheme, we conducted several experiments on the proposed scheme by utilizing free Pairing-Based Cryptography (PBC) Library [21] and the GNU Multiple Precision Arithmetic (GMP) [1]. All algorithms were coded using C programming language and conducted on a 64-bit Linux system with an Intel Core i5-6200 with 2.3GHz processor and 8Gb memory. In our experiments, we set the base field size to be 512 bits, the size of an element in Z_p^* to be $|p| =$ 160 bits, the size of a data file to be 20MB composed by 1,000,000 blocks.

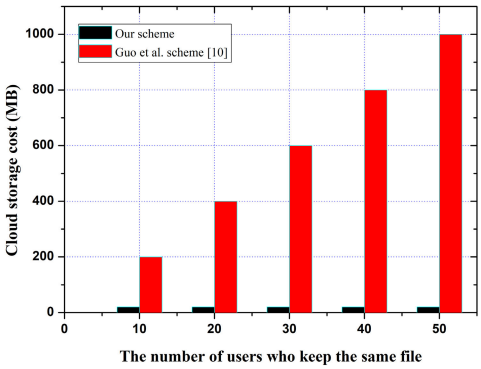


FIGURE 4. Storage overhead comparison between our scheme and the scheme [10] without deduplication.

1) STORAGE OVERHEAD

In all cloud storage auditing schemes without deduplication, the cloud needs to store all of files uploaded by users even if these files are identical. However, in our scheme, the cloud only stores a single copy of these identical files even from different users. Thus, to investigate the importance of deduplication in cloud storage and to evaluate the storage efficiency of our scheme, we select the scheme [10] as a benchmark because it is a new and classic cloud storage auditing scheme that does not support deduplication. As shown in Fig.4, with the increasing of the number of users who keep the identical file, the storage overhead of cloud in the scheme [10] linearly increases, however that in our scheme keeps unchanged. As a result, our scheme is more efficient compared with the scheme [10].

2) THE PHASES OF AUTHENTICATOR GENERATION AND AUDITING

In order to evaluate the computation efficiency of our scheme, we select the scheme [14] as a benchmark. As same as most of cloud storage schemes, the scheme [14] uses BLS signature to generate data authenticators and utilities publicly verifiable method to check data integrity. Through the following comparison, we can conclude that our scheme is more efficient

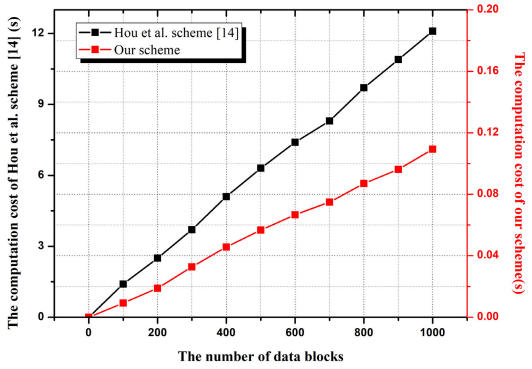


FIGURE 5. Computation overhead for authenticator generation.

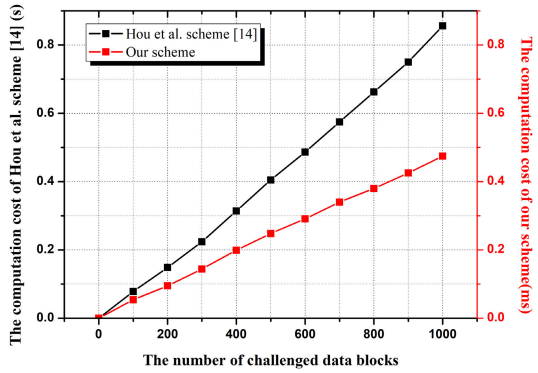


FIGURE 7. Computation overhead for proof generation.

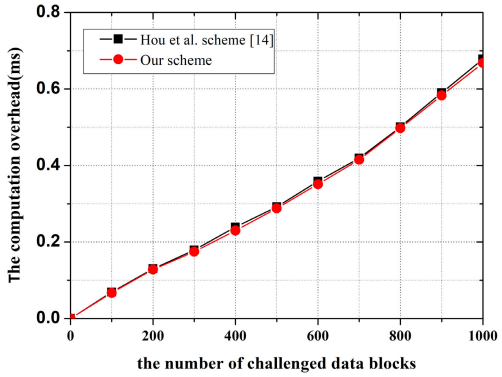


FIGURE 6. Computation overhead for challenge generation.

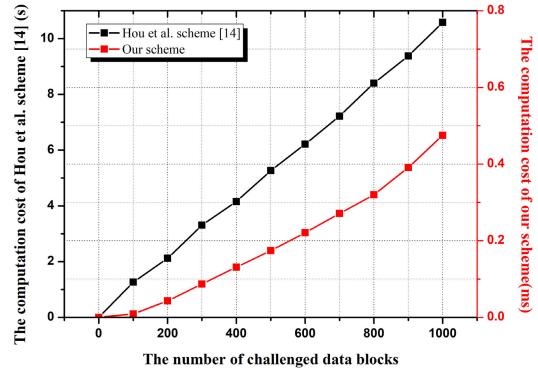


FIGURE 8. Computation overhead for proof verification.

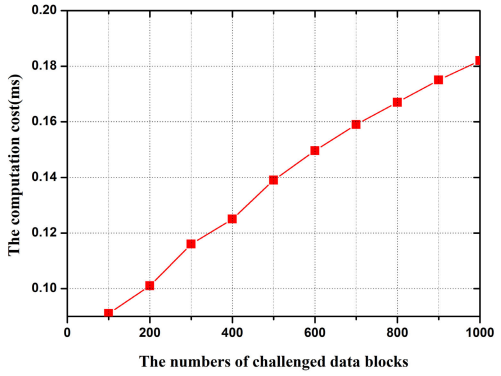


FIGURE 9. Computation overhead for ciphertext verification.

than the scheme [14] in the phases of authenticator generation and auditing. In our experiments, the number of data blocks and challenged data blocks both vary from 100 to 1,000 by an interval of 100.

From Fig.5, we can know that the computation overheads of authenticator generation in our scheme and the scheme [14] both linearly increase with the number of data blocks. The running time in the scheme [14] ranges from 1.432371s to 12.108713s. However, the running time in our scheme varies from 0.009225s to 0.109354s. Thus, in our scheme, the user only needs to perform lightweight computation to generate authenticators for data blocks.

Fig.6 shows that the computation efficiency for generating auditing challenge in our scheme and that in the scheme [14] are very comparable. As shown in Fig.7 and Fig.8, the computation overhead of auditing proof generation and verification in our scheme and the scheme [14] both grow linearly with the number of the challenged data blocks. When generating auditing proof, the running time in scheme [14] varies from 0.078359s to 0.856151s, whereas in our scheme, the running time ranges from 0.054214ms to 0.475325ms. In the scheme [14], proof verification takes 1.262326s when one hundred blocks are challenged, while that in our scheme only takes 0.009021ms. When one thousand blocks are challenged, the proof verification in the scheme [14] takes 10.578678s, however that in our scheme only takes 0.475361s. Therefore, compared with the scheme [14], our scheme is more efficient in the phase of auditing.

3) THE PHASES OF CIPHERTEXT VERIFICATION

As analyzed in Section IV, the user needs to verify the correctness of ciphertext when downloading the ciphertext from the cloud. Fig.9 depicts the computation overhead for ciphertext verification when different numbers of data blocks are challenged. When 100 blocks are challenged, the running time of ciphertext verification takes 0.091031ms. The running time increases to 0.182273ms when 1000 blocks are challenged.

VII. CONCLUSION

In this paper, we study on how to solve the problem of user's privacy leakage in cloud storage auditing with deduplication when brute-force dictionary attacks are launched. We design a lightweight cloud storage auditing scheme with deduplication supporting strong privacy protection. In the proposed scheme, the privacy of user can be well preserved against the cloud and other parties. The user relieves the heavy computation burden for generating data authenticators and verifying data integrity. The security proof shows that the proposed scheme is secure. We also provide detailed comparisons among our proposed scheme and other existing schemes by experiments. Experimental results show the proposed scheme achieves higher storage efficiency and is more efficient in authenticator generation phase and auditing phase.

REFERENCES

[1] *The Gnu Multiple Precision Arithmetic Library (GMP)*. Accessed: Oct. 2019. [Online]. Available: http://gmplib.org/

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 598–609.

[3] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin Germany: Springer, 2013, pp. 296–312.

[4] H. Cui, R. H. Deng, Y. Li, and G. Wu, "Attribute-based storage supporting secure deduplication of encrypted data in cloud," *IEEE Trans. Big Data*, vol. 5, no. 3, pp. 330–342, Sep. 2019.

[5] R. Ding, H. Zhong, J. Ma, X. Liu, and J. Ning, "Lightweight privacy-preserving identity-based verifiable IoT-based health storage system," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8393–8405, Oct. 2019.

[6] 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 *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, Jul. 2002, pp. 617–624.

[7] Y. Fan, X. Lin, G. Tan, Y. Zhang, W. Dong, and J. Lei, "One secure data integrity verification scheme for cloud storage," *Future Gener. Comput. Syst.*, vol. 96, pp. 376–385, Jul. 2019.

[8] J. Gantz and D. Reinsel. (2012). *The Digital Universe Decade–Are You Ready (2010)*. [Online]. Available: http://www.emc.com/collateral/analyst-reports/idcdigital-universe-are-you-ready.pdf

[9] X. Ge, J. Yu, H. Zhang, C. Hu, Z. Li, Z. Qin, and R. Hao, "Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification," *IEEE Trans. Dependable Secure Comput.*, to be published.

[10] W. Guo, H. Zhang, S. Qin, F. Gao, Z. Jin, W. Li, and Q. Wen, "Outsourced dynamic provable data possession with batch update for secure cloud storage," *Future Gener. Comput. Syst.*, vol. 95, pp. 309–322, Jun. 2019.

[11] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2011, pp. 491–500.

[12] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Secur. Privacy Mag.*, vol. 8, no. 6, pp. 40–47, Nov. 2010.

[13] K. He, J. Chen, Q. Yuan, S. Ji, D. He, and R. Du, "Dynamic group-oriented provable data possession in the cloud," *IEEE Trans. Dependable Secure Comput.*, to be published.

[14] H. Hou, J. Yu, and R. Hao, "Cloud storage auditing with deduplication supporting different security levels according to data popularity," *J. Netw. Comput. Appl.*, vol. 134, pp. 26–39, May 2019.

[15] A. Juels and B. S. Kaliski, "Pors: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 584–597.

[16] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proc. 22nd Secur. Symp.*, Washington, DC, USA, 2013, pp. 179–194.

[17] J. Li, X. Chen, X. Huang, S. Tang, Y. Xiang, M. M. Hassan, and A. Alelaiwi, "Secure distributed deduplication systems with improved reliability," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3569–3579, Dec. 2015.

[18] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2386–2396, Aug. 2016.

[19] Y. Li, Y. Yu, G. Min, W. Susilo, J. Ni, and K.-K.-R. Choo, "Fuzzy identity-based data integrity auditing for reliable cloud storage systems," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 72–83, Jan. 2019.

[20] X. Liu, W. Sun, W. Lou, Q. Pei, and Y. Zhang, "One-tag checker: Message-locked integrity auditing on encrypted cloud deduplication storage," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[21] B. Lynn. (2015). *The Pairing-Based Cryptographic Library*. [Online]. Available: https://crypto.stanford.edu/pbc/

[22] S. Peng, F. Zhou, J. Li, Q. Wang, and Z. Xu, "Efficient, dynamic and identity-based remote data integrity checking for multiple replicas," *J. Netw. Comput. Appl.*, vol. 134, pp. 72–88, May 2019.

[23] H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptol.*, vol. 26, no. 3, pp. 442–483, Jul. 2013.

[24] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 2, pp. 331–346, Feb. 2019.

[25] W. Shen, J. Qin, J. Yu, R. Hao, J. Hu, and J. Ma, "Data integrity auditing without private key storage for secure cloud storage," *IEEE Trans. Cloud Comput.*, to be published.

[26] P. Singh, N. Agarwal, and B. Raman, "Secure data deduplication using secret sharing schemes over cloud," *Future Gener. Comput. Syst.*, vol. 88, pp. 156–167, Nov. 2018.

[27] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud data protection for the masses," *Computer*, vol. 45, no. 1, pp. 39–45, Jan. 2012.

[28] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A secure data deduplication scheme for cloud storage," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Berlin, Germany: Springer, 2014, pp. 99–118.

[29] S. Bhagyashri and P. Y. B. Gurav, "Privacy-preserving public auditing for secure cloud storage," *IOSR J. Comput. Eng.*, vol. 16, no. 4, pp. 33–38, 2014.

[30] H. Wang, D. He, A. Fu, Q. Li, and Q. Wang, "Provable data possession with outsourced data transfer," *IEEE Trans. Services Comput.*, to be published.

[31] J. Xiong, Y. Zhang, S. Tang, X. Liu, and Z. Yao, "Secure encrypted data with authorized deduplication in cloud," *IEEE Access*, vol. 7, pp. 75090–75104, 2019.

[32] Y. Xu, S. Sun, J. Cui, and H. Zhong, "Intrusion-resilient public cloud auditing scheme with authenticator update," *Inf. Sci.*, vol. 512, pp. 616–628, Feb. 2020.

[33] Z. Yan, L. Zhang, W. Ding, and Q. Zheng, "Heterogeneous data storage management with deduplication in cloud computing," *IEEE Trans. Big Data*, vol. 5, no. 3, pp. 393–407, Sep. 2019.

[34] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Trans. Cloud Comput.*, to be published.

[35] J. Yu, K. Ren, and C. Wang, "Enabling cloud storage auditing with verifiable outsourcing of key updates," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 6, pp. 1362–1375, Jun. 2016.

[36] J. Yu, K. Ren, C. Wang, and V. Varadharajan, "Enabling cloud storage auditing with key-exposure resistance," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 6, pp. 1167–1179, Jun. 2015.

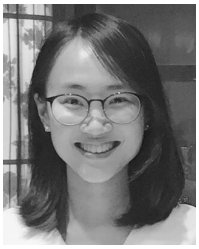
[37] J. Yu and H. Wang, "Strong key-exposure resilient auditing for secure cloud storage," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 8, pp. 1931–1940, Aug. 2017.

[38] Y. Yu, Y. Li, B. Yang, W. Susilo, G. Yang, and J. Bai, "Attribute-based cloud data integrity auditing for secure outsourced storage," *IEEE Trans. Emerg. Topics Comput.*, to be published.

[39] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2013, pp. 145–153.

[40] P. Zhao, J. Yu, H. Zhang, Z. Qin, and C. Wang, "How to securely outsource finding the min-cut of undirected edge-weighted graphs," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 315–328, 2020.

[41] Y. Zheng, H. Duan, and C. Wang, "Towards secure and efficient outsourcing of machine learning classification," in *Proc. Eur. Symp. Res. Comput. Secur.* Berlin, Germany: Springer, 2019, pp. 22–40.
[42] Y. Zheng, X. Yuan, X. Wang, J. Jiang, C. Wang, and X. Gui, "Toward encrypted cloud media center with secure deduplication," *IEEE Trans. Multimedia*, vol. 19, no. 2, pp. 251–265, Feb. 2017.



WENTING SHEN received the B.S. and M.S. degrees from the College of Computer Science and Technology, Qingdao University, China, in 2014 and 2017, respectively. She is currently pursuing the Ph.D. degree with the School of Mathematics, Shandong University, China. Her research interests include cloud security and big data security.

YE SU received the bachelor's degree from the School of Mathematics Sciences, University of Jinan, Shandong, China. She is currently pursuing the Ph.D. degree with the School of Mathematics, Shandong University, China. Her research interests include cloud security and privacy protection.

RONG HAO received the master's degree from the Institute of Network Security, Shandong University. She is currently working at the College of Information Engineering, Qingdao University. Her research interest includes information security.