Enabling Identity-Based Integrity Auditing and Data Sharing With Sensitive Information Hiding for Secure Cloud Storage

Wenting Shen, Jing Qin, Jia Yu, Rong Hao, and Jiankun Hu, *Senior Member, IEEE*

***Abstract*— With cloud storage services, users can remotely store their data to the cloud and realize the data sharing with others. Remote data integrity auditing is proposed to guarantee the integrity of the data stored in the cloud. In some common cloud storage systems such as the electronic health records system, the cloud file might contain some sensitive information. The sensitive information should not be exposed to others when the cloud file is shared. Encrypting the whole shared file can realize the sensitive information hiding, but will make this shared file unable to be used by others. How to realize data sharing with sensitive information hiding in remote data integrity auditing still has not been explored up to now. In order to address this problem, we propose a remote data integrity auditing scheme that realizes data sharing with sensitive information hiding in this paper. In this scheme, a sanitizer is used to sanitize the data blocks corresponding to the sensitive information of the file and transforms these data blocks' signatures into valid ones for the sanitized file. These signatures are used to verify the integrity of the sanitized file in the phase of integrity auditing. As a result, our scheme makes the file stored in the cloud able to be shared and used by others on the condition that the sensitive information is hidden, while the remote data integrity auditing is still able to be efficiently executed. Meanwhile, the proposed scheme is based on identity-based cryptography, which simplifies the complicated certificate management. The security analysis and the performance evaluation show that the proposed scheme is secure and efficient.**

***Index Terms*— Cloud storage, data integrity auditing, data sharing, sensitive information hiding.**

I. Introduction

With the explosive growth of data, it is a heavy burden for users to store the sheer amount of data locally. Therefore, more and more organizations and individuals would like to store their data in the cloud. However, the data stored in the cloud might be corrupted or lost due to the inevitable software bugs, hardware faults and human errors in the cloud [1]. In order to verify whether the data is stored correctly in the cloud, many remote data integrity auditing schemes have been proposed [2]–[8].

In remote data integrity auditing schemes, the data owner firstly needs to generate signatures for data blocks before uploading them to the cloud. These signatures are used to prove the cloud truly possesses these data blocks in the phase of integrity auditing. And then the data owner uploads these data blocks along with their corresponding signatures to the cloud. The data stored in the cloud is often shared across multiple users in many cloud storage applications, such as Google Drive, Dropbox and iCloud. Data sharing as one of the most common features in cloud storage, allows a number of users to share their data with others. However, these shared data stored in the cloud might contain some sensitive information. For instance, the Electronic Health Records (EHRs) [9] stored and shared in the cloud usually contain patients' sensitive information (patient's name, telephone number and ID number, etc.) and the hospital's sensitive information (hospital's name, etc.). If these EHRs are directly uploaded to the cloud to be shared for research purposes, the sensitive information of patient and hospital will be inevitably exposed to the cloud and the researchers. Besides, the integrity of the EHRs needs to be guaranteed due to the existence of human errors and software/hardware failures in the cloud. Therefore, it is important to accomplish remote data integrity auditing on the condition that the sensitive information of shared data is protected.

A potential method of solving this problem is to encrypt the whole shared file before sending it to the cloud, and then generate the signatures used to verify the integrity of

Manuscript received February 13, 2018; revised May 10, 2018; accepted June 11, 2018. Date of publication June 25, 2018; date of current version August 2, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61772311, Grant 61572267, and Grant 61272091, in part by the National Cryptography Development Fund of China under Grant MMJJ20170118, in part by the Open Project of Co-Innovation Center for Information Supply & Assurance Technology, Anhui University, and in part by the Open Project of the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences under Grant 2017-MS-21 and Grant 2017-MS-05. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ivan Visconti. *(Corresponding author: Jing Qin.)*

W. Shen is with the School of Mathematics, Shandong University, Shandong 250100, China (e-mail: shenwentingmath@163.com).

J. Qin is with the School of Mathematics, Shandong University, Shandong 250100, China, and also with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: qinjing@sdu.edu.cn).

J. Yu is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China, and also with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: qduyujia@gmail.com).

R. Hao is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China (e-mail: hr@qdu.edu.cn).

J. Hu is with the Cyber Security Laboratory, School of Engineering and IT, University of New South Wales, Australian Defence Force Academy, Canberra, ACT 2612, Australia (e-mail: j.hu@adfa.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2018.2850312

this encrypted file, finally upload this encrypted file and its corresponding signatures to the cloud. This method can realize the sensitive information hiding since only the data owner can decrypt this file. However, it will make the whole shared file unable to be used by others. For example, encrypting the EHRs of infectious disease patients can protect the privacy of patient and hospital, but these encrypted EHRs cannot be effectively utilized by researchers any more. Distributing the decryption key to the researchers seems to be a possible solution to the above problem. However, it is infeasible to adopt this method in real scenarios due to the following reasons. Firstly, distributing decryption key needs secure channels, which is hard to be satisfied in some instances. Furthermore, it seems very difficult for a user to know which researchers will use his/her EHRs in the near future when he/she uploads the EHRs to the cloud. As a result, it is impractical to hide sensitive information by encrypting the whole shared file. Thus, how to realize data sharing with sensitive information hiding in remote data integrity auditing is very important and valuable. Unfortunately, this problem has remained unexplored in previous researches.

Contribution

The contribution of this paper can be summarized as follows:

(1) We investigate how to achieve data sharing with sensitive information hiding in remote data integrity auditing, and propose a new concept called identity-based shared data integrity auditing with sensitive information hiding for secure cloud storage. In such a scheme, the sensitive information can be protected and the other information can be published. It makes the file stored in the cloud able to be shared and used by others on the condition that the sensitive information is protected, while the remote data integrity auditing is still able to be efficiently executed.

(2) We design a practical identity-based shared data integrity auditing scheme with sensitive information hiding for secure cloud storage. A sanitizer is used to sanitize the data blocks corresponding to the sensitive information of the file. In our detailed scheme, firstly, the user blinds the data blocks corresponding to the personal sensitive information of the original file and generates the corresponding signatures, and then sends them to a sanitizer. The sanitizer sanitizes these blinded data blocks into a uniform format and also sanitizes the data blocks corresponding to the organization's sensitive information. It also transforms the corresponding signatures into valid ones for the sanitized file. This method not only realizes the remote data integrity auditing, but also supports the data sharing on the condition that sensitive information is protected in cloud storage. To the best of our knowledge, this is the first scheme with the above functions. Besides, our scheme is based on identity-based cryptography, which simplifies the complex certificate management.

(3) We give the security analysis of the proposed scheme, and also justify the performance by concrete implementations. The result shows that the proposed scheme achieves desirable security and efficiency.

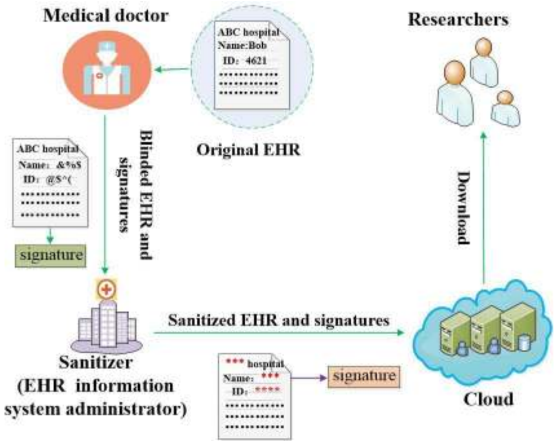


Fig. 1. Example of EHRs.

A. An Illustrative Example for EHRs

Here, we give an illustrative example for EHRs in Fig. 1. In this example, the sensitive information of EHRs contains two parts. One is the personal sensitive information (patient's sensitive information), such as patient's name and patient's ID number. The other is the organization's sensitive information (hospital's sensitive information), such as the hospital's name.* Generally speaking, the above sensitive information should be replaced with wildcards when the EHRs are uploaded to cloud for research purpose. The sanitizer can be viewed as the administrator of the EHR information system in a hospital. The personal sensitive information should not be exposed to the sanitizer. And all of the sensitive information should not be exposed to the cloud and the shared users. A medical doctor needs to generate and send the EHRs of patients to the sanitizer for storing them in the EHR information system. However, these EHRs usually contain the sensitive information of patient and hospital, such as patient's name, patient's ID number and hospital's name. To preserve the privacy of patient from the sanitizer, the medical doctor will blind the patient's sensitive information of each EHR before sending this EHR to the sanitizer. The medical doctor then generates signatures for this blinded EHR and sends them to the sanitizer. The sanitizer stores these messages into EHR information system. When the medical doctor needs the EHR, he sends a request to the sanitizer. And then the sanitizer downloads the blinded EHR from the EHR information system and sends it to the medical doctor. Finally, the medical doctor recovers the original EHR from this blinded EHR. When this EHR needs to be uploaded and shared in the cloud for research purpose, in order to unify the format, the sanitizer needs to sanitize the data blocks corresponding to the patient's sensitive information of the EHR. In addition, to protect the privacy of hospital, the sanitizer needs to sanitize the data blocks corresponding to the hospital's sensitive information.

*In some situations, the hospital's name in the EHR can be viewed as the sensitive information. If the name of hospital in the EHR is known by medical devices suppliers or drug providers, these people could analyze the number of patients with a certain disease in each hospital. As a result, they can easily select the appropriate hospital to do door to door sales which will threaten the privacy of the hospital.

Generally, these data blocks are replaced with wildcards. Furthermore, the sanitizer can transform these data blocks' signatures into valid ones for the sanitized EHR. It makes the remote data integrity auditing still able to be effectively performed. During the process of sanitization, the sanitizer does not need to interact with medical doctors. Finally, the sanitizer uploads these sanitized EHRs and their corresponding signatures to the cloud. In this way, the EHRs can be shared and used by researchers, while the sensitive information of EHRs can be hidden. Meanwhile, the integrity of these EHRs stored in the cloud can be ensured.

The sanitizer is necessary because of the following reasons. Firstly, after the data blocks corresponding to the patient's sensitive information are blinded, the contents of these data blocks might become messy code. The sanitizer can unify the format by using wildcards to replace the contents of these data blocks. In addition, the sanitizer also can sanitize the data blocks corresponding to the hospital's sensitive information such as hospital's name by using wildcards, which protects the privacy of the hospital. Secondly, the sanitizer can facilitate the information management. It can sanitize the EHRs in bulk, and uploads these sanitized EHRs to the cloud at a fixed time. Thirdly, when the medical doctor needs the EHR, the sanitizer as the administrator of EHR information system can download the blinded EHR from the EHR information system and sends it to the medical doctor. The medical doctor can recover the original EHR from the blinded one.

B. Related Work

In order to verify the integrity of the data stored in the cloud, many remote data integrity auditing schemes have been proposed. To reduce the computation burden on the user side, a Third Party Auditor (TPA) is introduced to periodically verify the integrity of the cloud data on behalf of user. Ateniese *et al.* [2] firstly proposed a notion of Provable Data Possession (PDP) to ensure the data possession on the untrusted cloud. In their proposed scheme, homomorphic authenticators and random sampling strategies are used to achieve blockless verification and reduce I/O costs. Juels and Kaliski [3] defined a model named as Proof of Retrievability (PoR) and proposed a practical scheme. In this scheme, the data stored in the cloud can be retrieved and the integrity of these data can be ensured. Based on pseudorandom function and BLS signature, Shacham and Waters [4] proposed a private remote data integrity auditing scheme and a public remote data integrity auditing scheme.

In order to protect the data privacy, Wang *et al.* [5] proposed a privacy-preserving remote data integrity auditing scheme with the employment of a random masking technique. Worku *et al.* [6] utilized a different random masking technique to further construct a remote data integrity auditing scheme supporting data privacy protection. This scheme achieves better efficiency compared with the scheme in [5]. To reduce the computation burden of signature generation on the user side, Guan *et al.* [7] designed a remote data integrity auditing scheme based on the indistinguishability obfuscation technique. Shen *et al.* [8] introduced a Third Party Medium (TPM) to design a light-weight remote data integrity auditing scheme. In this scheme, the TPM helps user generate signatures on the condition that data privacy can be protected. In order to support data dynamics, Ateniese *et al.* [10] firstly proposed a partially dynamic PDP scheme. Erway *et al.* [11] used a skip list to construct a fully data dynamic auditing scheme. Wang *et al.* [12] proposed another remote data integrity auditing scheme supporting full data dynamics by utilizing Merkle Hash Tree. To reduce the damage of users' key exposure, Yu *et al.* [13] and [14], and Yu and Wang [15] proposed key-exposure resilient remote data integrity auditing schemes based on key update technique [16].

The data sharing is an important application in cloud storage scenarios. To protect the identity privacy of user, Wang *et al.* [17] designed a privacy-preserving shared data integrity auditing scheme by modifying the ring signature for secure cloud storage. Yang *et al.* [18] constructed an efficient shared data integrity auditing scheme, which not only supports the identity privacy but only achieves the identity traceability of users. Fu *et al.* [19] designed a privacy-aware shared data integrity auditing scheme by exploiting a homomorphic verifiable group signature. In order to support efficient user revocation, Wang *et al.* [20] proposed a shared data integrity auditing scheme with user revocation by using the proxy re-signature. With the employment of the Shamir secret sharing technique, Luo *et al.* [21] constructed a shared data integrity auditing scheme supporting user revocation.

The aforementioned schemes all rely on Public Key Infrastructure (PKI), which incurs the considerable overheads from the complicated certificate management. To simplify certificate management, Wang [22] proposed an identity-based remote data integrity auditing scheme in multicloud storage. This scheme used the user's identity information such as user's name or e-mail address to replace the public key. Wang *et al.* [23] designed a novel identity-based proxy-oriented remote data integrity auditing scheme by introducing a proxy to process data for users. Yu *et al.* [24] constructed a remote data integrity auditing scheme with perfect data privacy preserving in identity-based cryptosystems. Wang *et al.* [25] proposed an identity-based data integrity auditing scheme satisfying unconditional anonymity and incentive. Zhang *et al.* [26] proposed an identity-based remote data integrity auditing scheme for shared data supporting real efficient user revocation.

Other aspects, such as privacy-preserving authenticators [27] and data deduplication [28], [29] in remote data integrity auditing have also been explored. However, all of existing remote data integrity auditing schemes cannot support data sharing with sensitive information hiding. In this paper, we explore how to achieve data sharing with sensitive information hiding in identity-based integrity auditing for secure cloud storage.

C. Organization

The rest of this paper is organized as follows: In Section II, we present notations and preliminaries. In Section III, the system model and security model are presented. We intro-

TABLE I
NOTATIONS

Notation	Meaning
p	One large prime
G_1, G_2	Multiplicative cyclic groups with order p
g	A generator of group G_1
e	A bilinear pairing map $e : G_1 \times G_1 \to G_2$
Z_p^*	A prime field with nonzero elements
H	A cryptographic hash function: $H : \{0,1\}^* \to G_1$
x	An element in Z_p^*
$u', \mu_1, \mu_2, ..., \mu_l, u, g_2$	The elements in G_1
g_1	A public value
n	The number of data blocks of file F
$F = \{m_1, m_2, ..., m_n\}$	The original file F
$F^* = \{m_1^*, m_2^*, ..., m_n^*\}$	The blinded file F^* sent to the sanitizer
$F' = \{m_1', m_2', ..., m_n'\}$	The sanitized file F' stored in the cloud
ID	The user's identity
K_1	The set of the indexes of the data blocks corresponding to the personal sensitive information
K_2	The set of the indexes of the data blocks corresponding to the organization's sensitive information
msk	The master secret key
sk_{ID}	The private key of the user *ID*
$\Phi = \{\sigma_i\}_{1 \le i \le n}$	The signature set of the blinded file F^*
$\Phi' = \{\sigma_i'\}_{1 \le i \le n}$	The signature set of the sanitized file F'

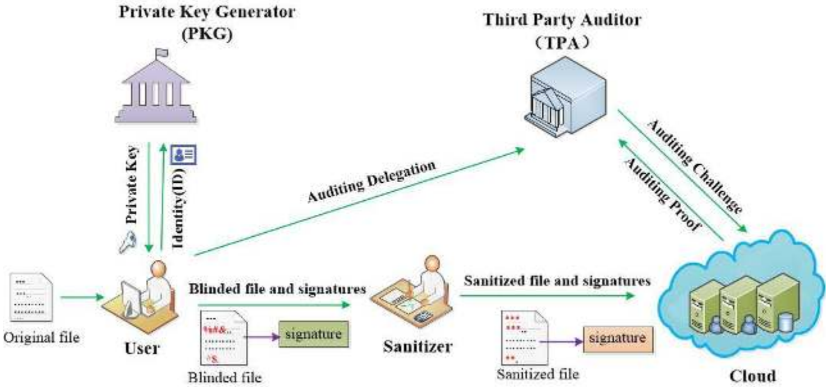


Fig. 2. The system model.

duce the proposed scheme in Section IV. In Section V and Section VI, the security analysis and the performance evaluation are given respectively. Finally, we conclude our paper in Section VII.

II. NOTIONS AND PRELIMINARIES

A. Notions

We show some notations used in the description of our scheme in Table I.

B. Preliminaries

In this section, we review some preliminary cryptography knowledge, including bilinear map, Computational Diffie-Hellman (CDH) problem and Discrete Logarithm (DL) problem.

1) **Bilinear Map**
Let G_1, G_2 be two multiplicative cyclic groups of large prime order p, and g be a generator of G_1. Bilinear map is a map $e : G_1 \times G_1 \to G_2$ with the following properties:
   a) *Bilinearity*: for all $u, v \in G_1$ and $a, b \in Z_p^*$, $e\left(u^a, v^b\right) = e(u, v)^{ab}$.
   b) *Computability*: there exists an efficiently computable algorithm for computing map e.
   c) *Non-degeneracy*: $e\,(g, g) \neq 1$.
2) **Computational Diffie-Hellman (CDH) Problem**
For unknown x,$y \in Z_p^*$, given g, g^x and g^y as input, output $g^{xy} \in G_1$. The CDH assumption in G_1 holds if it is computationally infeasible to solve the CDH problem in G_1.
3) **Discrete Logarithm (DL) Problem**
For unknown $x \in Z_p^*$, given g and g^x as input, outputs x. The DL assumption in G_1 holds if it is computationally infeasible to solve the DL problem in G_1.

III. SYSTEM MODEL AND SECURITY MODEL

A. System Model

The system model involves five kinds of different entities: the cloud, the user, the sanitizer, the Private Key Generator (PKG) and the Third Party Auditor (TPA), as shown in Fig.2.

(1) Cloud: The cloud provides enormous data storage space to the user. Through the cloud storage service, users can upload their data to the cloud and share their data with others.

(2) User: The user is a member of an organization, which has a large number of files to be stored in the cloud.

(3) Sanitizer: The sanitizer is in charge of sanitizing the data blocks corresponding to the sensitive information (personal sensitive information and the organization's sensitive information) in the file, transforming these data blocks' signatures into valid ones for the sanitized file, and uploading the sanitized file and its corresponding signatures to the cloud.

(4) PKG: The PKG is trusted by other entities. It is responsible for generating system public parameters and the private key for the user according to his identity *ID*.

(5) TPA: The TPA is a public verifier. It is in charge of verifying the integrity of the data stored in the cloud on behalf of users.

The user firstly blinds the data blocks corresponding to the personal sensitive information of the file, and generates the corresponding signatures. These signatures are used to guarantee the authenticity of the file and verify the integrity of the file. Then the user sends this blinded file and its corresponding signatures to the sanitizer. After receiving the message from the user, the sanitizer sanitizes these blinded data blocks and the data blocks corresponding to the organization's sensitive information, and then transforms the signatures of sanitized data blocks into valid ones for the sanitized file. Finally, the sanitizer sends this sanitized file and its corresponding signatures to the cloud. These signatures are used to verify the integrity of the sanitized file in the phase of integrity auditing.

When the TPA wants to verify the integrity of the sanitized file stored in the cloud, he sends an auditing challenge to the cloud. And then, the cloud responds to the TPA with an auditing proof of data possession. Finally, the TPA verifies the integrity of the sanitized file by checking whether this auditing proof is correct or not.

B. Design Goals

To efficiently support data sharing with sensitive information hiding in identity-based integrity auditing for secure cloud storage, our scheme is designed to achieve the following goals:

1) The correctness:
   a) Private key correctness: to ensure that when the PKG sends a correct private key to the user, this private key can pass the verification of the user.
   b) The correctness of the blinded file and its corresponding signatures: to guarantee that when the user sends a blinded file and its corresponding valid signatures to the sanitizer, the blinded file and its corresponding signatures he generates can pass the verification of the sanitizer.
   c) Auditing correctness: to ensure that when the cloud properly stores the user's sanitized data, the proof it generates can pass the verification of the TPA.
2) Sensitive information hiding: to ensure that the personal sensitive information of the file is not exposed to the sanitizer, and all of the sensitive information of the file is not exposed to the cloud and the shared users.
3) Auditing soundness: to assure that if the cloud does not truly store user's intact sanitized data, it cannot pass the TPA's verification.

C. Definition

Definition 1: An identity-based shared data integrity auditing scheme with sensitive information hiding for secure cloud storage consists of the following six algorithms: *Setup*, *Extract*, *SigGen*, *Sanitization*, *ProofGen* and *ProofVerif* y. Specifically, these algorithms are described as follows:

1) $\mathbf{Setup(1^k)}$ is a setup algorithm run by the PKG. It takes as input a security parameter k. It outputs the master secret key *msk* and the system public parameters *pp*.
2) $\mathbf{Extract(pp, msk, ID)}$ is an extraction algorithm run by the PKG. It takes as input the system public parameters *pp*, the master secret key *msk*, and a user's identity *ID*. It outputs the user's private key sk_{ID}. The user can verify the correctness of sk_{ID} and accept it as his private key only if it passes the verification.
3) $\mathbf{SigGen(F, sk_{ID}, ssk, name)}$ is a signature generation algorithm run by the user *ID*. It takes as input the original file F, the user's private key sk_{ID}, the user's signing private key *ssk* and the file identifier name *name*. It outputs a blinded file F^*, its corresponding signature set Φ and a file tag τ.
4) $\mathbf{Sanitization(F^*, \Phi)}$ is a sensitive information sanitization algorithm run by the sanitizer. It takes as input the blinded file F^* and its signature set Φ. It outputs the sanitized file F' and its corresponding signature set Φ'.
5) $\mathbf{ProofGen(F', \Phi', chal)}$ is a proof generation algorithm run by the cloud. It takes as input the sanitized file F', the corresponding signature set Φ' and the auditing challenge *chal*. It outputs an auditing proof P that is used to demonstrate the cloud truly possesses this sanitized file F'.
6) $\mathbf{ProofVerify(chal, pp, P)}$ is a proof verification algorithm run by the TPA. It takes as input the auditing challenge *chal*, the system public parameters *pp* and the auditing proof P. The TPA can verify the correctness of proof P.

D. Security Model

To formalize the security model, we indicate a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ to show how the adversary $\mathcal{A}$ is against the security of an identity-based shared data integrity auditing scheme with sensitive information hiding. The data owner is viewed as a challenger $\mathcal{C}$ and the untrusted cloud server is viewed as an adversary $\mathcal{A}$ in our security model. This game includes the following phases:

1) **Setup phase**. The challenger $\mathcal{C}$ runs the *Setup* algorithm to obtain the master secret key *msk* and the system public parameters *pp*, and then sends the public parameters *pp* to the adversary $\mathcal{A}$.
2) **Query phase**. In this phase, the adversary $\mathcal{A}$ makes the following two queries to the challenger $\mathcal{C}$.
   a) *Extract Queries*: The adversary $\mathcal{A}$ queries the private key for the identity *ID*. The challenger $\mathcal{C}$ runs

the *Extract* algorithm to generate the private key sk_{ID}, and sends it to the adversary $\mathcal{A}$.

b) *SigGen Queries*: The adversary $\mathcal{A}$ queries the signatures of the file F. By running the *Extract* algorithm, the challenger $\mathcal{C}$ gets the private key. And then the challenger $\mathcal{C}$ runs the *SigGen* algorithm to calculate the signatures of the file F. Finally, the challenger $\mathcal{C}$ sends these signatures to the adversary $\mathcal{A}$.

3) **Challenge phase**. In this phase, the adversary $\mathcal{A}$ acts as a prover and the challenger $\mathcal{C}$ plays the role of a verifier. The challenger $\mathcal{C}$ sends the challenge $chal = \{i, v_i\}_{i \in I}$ to the adversary $\mathcal{A}$, where $I \in \{\gamma_1, \gamma_2, \ldots, \gamma_c\}$ ($\gamma_j \in [1, n]$, $j \in [1, c]$ and $c \in [1, n]$). Meanwhile, it requests the adversary $\mathcal{A}$ to provide a data possession proof P for the data blocks $\{m_{\gamma_1}, m_{\gamma_2}, \ldots, m_{\gamma_c}\}$ under the *chal*.

4) **Forgery phase**. After receiving the challenge from the challenger $\mathcal{C}$, the adversary $\mathcal{A}$ generates a data possession proof P for the data blocks indicated by *chal* to reply the challenger $\mathcal{C}$. If this proof P can pass the verification of the challenger $\mathcal{C}$ with non-negligible probability, we say that the adversary $\mathcal{A}$ succeeds in the above game.

In the above security model, we need to prove that if an adversary $\mathcal{A}$, who does not keep all the data blocks challenged by the challenger $\mathcal{C}$, cannot guess all the corrupted data blocks, then it cannot generate a valid proof P to pass the verification of the challenger $\mathcal{C}$. The goal of the adversary $\mathcal{A}$ is to pass the verification of the challenger $\mathcal{C}$ by generating a valid proof P for the challenged data blocks. The definition 2 presents that there exists a knowledge extractor that can capture the challenged data blocks whenever the adversary can output a valid data possession proof P. The definition 3 is to describe the detectability for the data integrity auditing scheme, which can ensure that the cloud truly keeps the data blocks that are not challenged with high probability.

Definition 2: We say a remote data integrity auditing scheme is secure if the following condition holds: whenever an adversary $\mathcal{A}$ in the aforementioned game can generate a valid proof P to pass the verification of the challenger $\mathcal{C}$ with non-negligible probability, there exists a knowledge extractor that can capture the challenged data blocks except possibly with negligible probability.

Definition 3: A remote data integrity auditing scheme is (ρ, δ) $(0 < \rho, \delta < 1)$ detectable if the cloud corrupted ρ fraction of the whole file, these corrupted data blocks can be detected with the probability at least δ.

We consider the sanitizer is not fully trustworthy. The sanitizer might be curious about the personal sensitive information of the file. In addition, the cloud and the shared users might be curious about all of the sensitive information of the file. Thus, the personal sensitive information of the file should not be exposed to the sanitizer, and all of the sensitive information of the file should not be exposed to the cloud and the shared users in our scheme. That is to say, even if the sanitizer is untrustworthy, the personal sensitive information of the file will not be exposed to it. Furthermore, even if the cloud and the shared users are untrustworthy, all of the sensitive information of the file will not be exposed to them. Therefore, we give the following security definition.

Definition 4: We say a remote data integrity auditing scheme achieves sensitive information security if the sanitizer cannot derive the personal sensitive information of the file, besides the cloud and the shared users cannot derive any sensitive information of the file.

IV. The Proposed Scheme

A. An Overview

In order to achieve data sharing with sensitive information hiding, we consider making use of the idea in the sanitizable signature [30] to sanitize the sensitive information of the file by introducing an authorized sanitizer. Nonetheless, it is infeasible if this sanitizable signature is directly used in remote data integrity auditing. Firstly, this signature in [30] is constructed based on chameleon hashes [31]. However, a lot of chameleon hashes exhibit the key exposure problem. To avoid this security problem, the signature used in [30] requires strongly unforgeable chameleon hashes, which will inevitable incur huge computation overhead [31]. Secondly, the signature used in [30] does not support blockless verifiability. It means that the verifier has to download the entire data from the cloud to verify the integrity of data, which will incur huge communication overhead and excessive verification time in big data storage scenario. Thirdly, the signature used in [30] is based on the PKI, which suffers from the complicated certificate management.

In order to address above problems, we design a new efficient signature algorithm in the phase of signature generation. The designed signature scheme supports blockless verifiability, which allows the verifier to check the integrity of data without downloading the entire data from the cloud. In addition, it is based on identity-based cryptography, which simplifies the complicated certificate management.

In our proposed scheme, the PKG generates the private key for user according to his identity *ID*. The user can check the correctness of the received private key. When there is a desire for the user to upload data to the cloud, in order to preserve the personal sensitive information of the original file from the sanitizer, this user needs to use a blinding factor to blind the data blocks corresponding to the personal sensitive information of the original file. When necessary, the user can recover the original file from the blinded one by using this blinding factor. And then this user employs the designed signature algorithm to generate signatures for the blinded file. These signatures will be used to verify the integrity of this blinded file. In addition, the user generates a file tag, which is used to ensure the correctness of the file identifier name and some verification values. The user also computes a transformation value that is used to transform signatures for sanitizer. Finally, the user sends the blinded file, its corresponding signatures, and the file tag along with the transformation value to the sanitizer. When the above messages from user are valid, the sanitizer firstly sanitizes the blinded data blocks into a uniform format and also sanitizes the data blocks corresponding to the organization's sensitive information to protect the privacy of organization, and then transforms their corresponding signatures into valid ones for sanitized file using transformation value. Finally, the sanitizer uploads the sanitized file and the corresponding signatures to the cloud. When the data integrity auditing task is performed, the cloud generates an auditing proof according to the challenge from the TPA. The TPA can verify the integrity of the sanitized file stored the cloud by checking whether this auditing proof is correct or not. The details will be described in the following subsection.

B. Description of the Proposed Scheme

In our scheme, an original file F is divided into n blocks $(m_1, m_2, \ldots, m_n)$, where $m_i \in Z_p^*$ denotes the i-th block of file F. Assume the user's identity *ID* is l-bit, which is described as $ID = (ID_1, ID_2, \ldots, ID_l) \in \{0, 1\}^l$. In previous remote data integrity auditing schemes [5], [12], a signature *SSig* is used to guarantee the integrity of the file identifier name. In our scheme, we also employ a similar identity-based signature *SSig* to guarantee the integrity of the file identifier name and the correctness of verification values. Assume *ssk* is the signing private key used to generate file tag in signature *SSig* and is held by user. Under such an assumption, our scheme is more clear and simple. Let K_1 be the set of indexes of the data blocks corresponding to the personal sensitive information of the file F. Let K_2 be the set of indexes of the data blocks corresponding to the organization's sensitive information of the file F. In order to preserve the personal sensitive information of the file from the sanitizer, the data blocks whose indexes are in the set K_1 should be blinded before the file is sent to the sanitizer. Assume the blinded file is $F^* = \left(m_1^*, m_2^*, \ldots, m_n^*\right)$ which is different from the original file $F = (m_1, m_2, \ldots, m_n)$ in index set K_1. That is to say, $m_i = m_i^*$ only if $i \in [1, n]$ and $i \notin K_1$; otherwise, $m_i \neq m_i^*$. To unify the format, the sanitizer needs to sanitize the blinded data blocks with wildcards. Furthermore, to protect the privacy of organization, the sanitizer also needs to sanitize the data blocks corresponding to the organization's sensitive information. The sanitized file is $F' = \left(m_1', m_2', \ldots, m_n'\right)$ which is different from the blinded file $F^* = \left(m_1^*, m_2^*, \ldots, m_n^*\right)$ in index set $K_1 \bigcup K_2$. That is to say, $m_i^* = m_i'$ only if $i \in [1, n]$ and $i \notin K_1 \bigcup K_2$; otherwise, $m_i^* \neq m_i'$. In general, there is not too much sensitive information in a file, which makes the sanitizer only need to sanitize a few fields. For example, the sensitive information of the EHRs only contain the fields such as patient's name, patient's ID number and hospital's name. Thus, in EHRs, only these fields containing the sensitive information need to be sanitized, and other fields do not need to be sanitized.

The details of the proposed scheme are as follows.

1) **Algorithm Setup($\mathbf{1^k}$)**
   a) The PKG chooses two multiplicative cyclic groups G_1 and G_2 of prime order p, a generator g of G_1, a bilinear map $e : G_1 \times G_1 \to G_2$ and a pseudo-random function $f : Z_p^* \times Z_p^* \to Z_p^*$.
   b) The PKG randomly chooses an element $x \in Z_p^*$, elements $\mu', \mu_1, \mu_2, \ldots, \mu_l, u, g_2 \in G_1$ and a cryptographic hash function $H : \{0, 1\}^* \to G_1$.

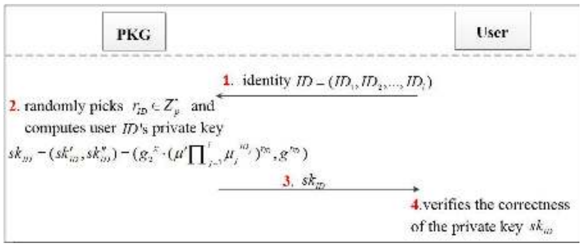

PKG
User
1. identity $ID = (ID_1, ID_2, \ldots, ID_l)$
2. randomly picks $r_{ID} \in Z_p^*$ and
computes user ID's private key
$sk_{ID} = (sk'_{ID}, sk''_{ID}) = (g_2^x \cdot (\mu' \prod_{j=1}^{l} \mu_j^{ID_j})^{r_{ID}}, g^{r_{ID}})$
3. sk_{ID}
4.verifies the correctness
of the private key sk_{ID}

Fig. 3. The process of private key generation.

   c) The PKG computes the public value $g_1 = g^x$ and the master secret key $msk = g_2{}^x$.
   d) The PKG publishes system parameters $pp = (G_1, G_2, p, e, g, \mu', \mu_1, \mu_2, \ldots, \mu_l, u, g_1, g_2, H, f)$ and holds the master secret key *msk*.

2) **Algorithm Extract(pp, msk, ID)**
   This process is illustrated in Fig. 3.
   a) After receiving the user's identity $ID = (ID_1, ID_2, \ldots, ID_l) \in \{0, 1\}^l$, the PKG randomly picks a value $r_{ID} \in Z_p^*$ and computes $sk_{ID} = (sk'_{ID}, sk''_{ID}) = (g_2{}^x \cdot (\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j})^{r_{ID}}, g^{r_{ID}})$ as the private key of the user *ID*. The PKG sends it to the user *ID*.
   b) The user *ID* verifies the correctness of the received private key sk_{ID} by checking whether the following equation holds or not.

$$e(sk'_{ID}, g) = (g_1, g_2) \cdot e(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j}, sk''_{ID}). \tag{1}$$

   If above equation does not hold, the user *ID* refuses the private key sk_{ID}; otherwise, accepts it.

3) **Algorithm SigGen(F, $\mathbf{sk_{ID}}$, ssk, name)**
   The process is illustrated in Fig. 4.
   a) The user *ID* randomly chooses a value $r \in Z_p^*$, and calculates a verification value g^r. Then the user *ID* randomly chooses a seed $k_1 \in Z_p^*$ as the input secret key of pseudo-random function f. The user *ID* employs the secret seed k_1 to calculate the blinding factor $\alpha_i = f_{k_1}(i, name)(i \in K_1)$ which is used to blind the data blocks corresponding to the personal sensitive information, where $name \in Z_p^*$ is a random value chosen as the file identifier.
   b) In order to preserve the personal sensitive information from the sanitizer, the user *ID* should blind the data blocks corresponding to the personal sensitive information of the original file F before sending it to the sanitizer. The indexes of these data blocks are in set K_1. The user *ID* computes the blinded data block $m_i^* = m_i + \alpha_i$ for each block $m_i \in Z_p^*$ $(i \in K_1)$ of the original file F. The blinded file is $F^* = \left(m_1^*, m_2^*, \ldots, m_n^*\right)$, where $m_i^* = m_i$ only if $i \in [1, n]$ and $i \notin K_1$; otherwise, $m_i^* \neq m_i$.
   c) For each block $m_i^* \in Z_p^*$ $(i \in [1, n])$ of the blinded file F^*, the user *ID* calculates

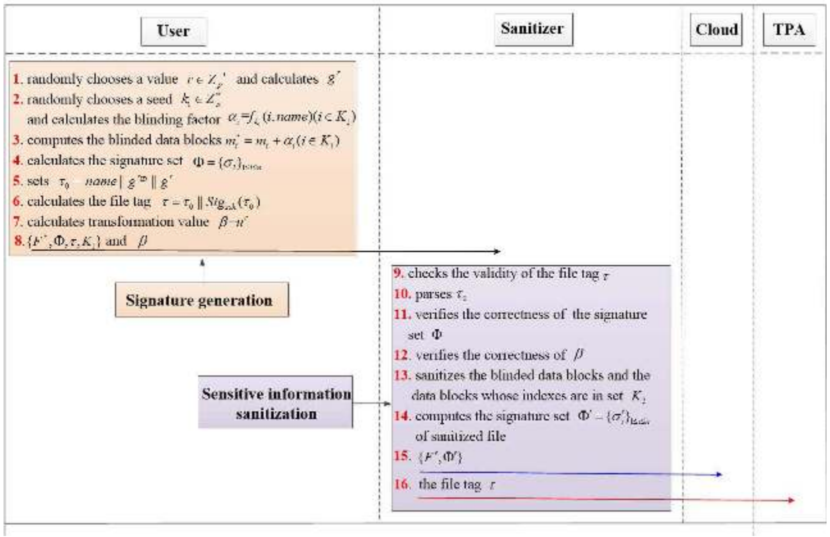

User
Sanitizer
Cloud
TPA
1. randomly chooses a value $r \in Z_p^*$ and calculates g^r
2. randomly chooses a seed $k_1 \in Z_p^*$
and calculates the blinding factor $\alpha_i = f_{k_1}(i, name)(i \in K_1)$
3. computes the blinded data blocks $m_i^* = m_i + \alpha_i (i \in K_1)$
4. calculates the signature set $\Phi = \{\sigma_i\}_{1 \le i \le n}$
5. sets $\tau_0 = name \| g^{r_{ID}} \| g^r$
6. calculates the file tag $\tau = \tau_0 \| Sig_{ssk}(\tau_0)$
7. calculates transformation value $\beta = u^r$
8. $\{F^*, \Phi, \tau, K_1\}$ and β
Signature generation
9. checks the validity of the file tag τ
10. parses τ_0
11. verifies the correctness of the signature set Φ
12. verifies the correctness of β
13. sanitizes the blinded data blocks and the data blocks whose indexes are in set K_2
14. computes the signature set $\Phi' = \{\sigma_i'\}_{1 \le i \le n}$ of sanitized file
15. $\{F', \Phi'\}$
16. the file tag τ
Sensitive information sanitization

Fig. 4. The processes of signature generation and sensitive information sanitization.

the signature σ_i on block m_i^* as follows: $\sigma_i = g_2{}^x(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j})^{r_{ID}}(H(name||i) \cdot u^{m_i^*})^r$. Let $\Phi = \{\sigma_i\}_{1 \le i \le n}$ be the set of signatures for the blinded file F^*.

d) The user *ID* sets $\tau_0 = name||g^{r_{ID}}||g^r$ and calculates the file tag by computing $\tau = \tau_0||SSig_{ssk}(\tau_0)$, where $SSig_{ssk}(\tau_0)$ is the signature on τ_0 under the signing private key *ssk*.

e) The user *ID* calculates a transformation value $\beta = u^r$ which is used to transform the signature in *Sanitization* algorithm. He sends $\{F^*, \Phi, \tau, K_1\}$ along with β to the sanitizer, and then deletes these messages from local storage. In addition, when the user *ID* wants to retrieve his file *F*, he can send a request to the sanitizer. And then the sanitizer downloads and sends the blinded file F^* to the user. The user *ID* can recover the original file *F* using the blinding factor.

4) **Algorithm Sanitization**$(\mathbf{F}^*, \Phi)$

The process is illustrated in Fig. 4.

a) The sanitizer checks the validity of the file tag τ by verifying whether $SSig_{ssk}(\tau_0)$ is a valid signature. If it is a valid signature, the sanitizer parses τ_0 to obtain file identifier name *name* and verification values $g^{r_{ID}}$ and g^r, and then does the following steps.

b) The sanitizer respectively verifies the correctness of signature σ_i $(i \in [1, n])$ as follows:

$$e(\sigma_i, g) = e(g_1, g_2) \cdot e(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j}, g^{r_{ID}}) \cdot e(H(name||i) \cdot u^{m_i^*}, g^r). \quad (2)$$

If the equation (2) does not hold, the sanitizer thinks the signatures invalid; otherwise, goes to the step c.

c) The sanitizer verifies the correctness of the transformation value β by checking whether $e(u, g^r) = e(\beta, g)$ holds or not. If the above equation holds, the sanitizer will sanitize the blinded data blocks and the data blocks corresponding to the organization's sensitive information. The indexes of these data blocks are in sets K_1 and K_2. In SigGen algorithm, the data blocks whose indexes are in set K_1 have been blinded by the user *ID*, which will make the contents of these data blocks become messy code. In order to unify the format, the sanitizer can use wildcards to replace the contents of these data blocks. For example, in an EHR, Bob is a user's name. After blinded by the medical doctor, the contents of this sector will become messy code. To unify the format, the sanitizer replaces these messy code with ***, as shown in Fig. 1. In addition, to protect the privacy of organization, the sanitizer also sanitizes the data blocks whose indexes are in set K_2. For example, in an EHR, the sanitizer replaces the information such as hospital's name with ***. And then the sanitizer transforms the signatures of data blocks in sets K_1 and K_2 into valid ones for sanitized file F' as follows:

$$\sigma_i' = \left\{ \begin{array}{ll} \sigma_i(\beta)^{m_i' - m_i^*} & i \in K_1 \cup K_2 \\ \sigma_i & i \in [1, n] \text{ and } i \notin K_1 \cup K_2 \end{array} \right\}$$
$$= g_2{}^x(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j})^{r_{ID}}(H(name||i) \cdot u^{m_i'})^r$$

Let $\Phi' = \{\sigma_i'\}_{1 \le i \le n}$ be the set of sanitized file's signatures.

d) The sanitizer sends $\{F', \Phi'\}$ to the cloud, and then sends the file tag τ to the TPA. Finally, he deletes these messages from local storage.

We give an example of sensitive information sanitization in Fig. 5. Assume an original file F is divided into n blocks $(m_1, m_2, \ldots, m_n)$, the index set K_1 of the data blocks corresponding to the personal sensitive

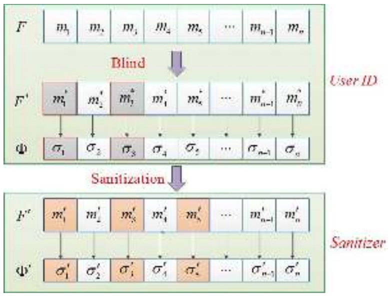
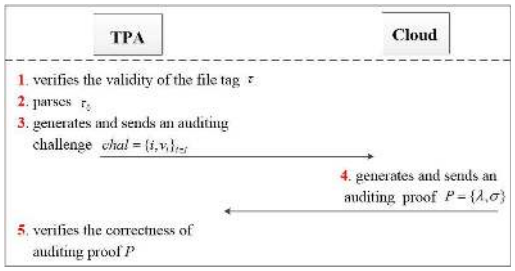

F
m_1 m_2 m_3 m_4 m_5 ⋯ m_{n-1} m_n
Blind
F'
m_1^* m_2^* m_3^* m_4^* m_5^* ⋯ m_{n-1}^* m_n^*
Φ
σ_1 σ_2 σ_3 σ_4 σ_5 ⋯ σ_{n-1} σ_n
User ID
Sanitization
F'
m_1' m_2' m_3' m_4' m_5' ⋯ m_{n-1}' m_n'
Φ'
σ_1' σ_2' σ_3' σ_4' σ_5' ⋯ σ_{n-1}' σ_n'
Sanitizer

Fig. 5. An example of sensitive information sanitization.



1. verifies the validity of the file tag τ
2. parses τ_0
3. generates and sends an auditing challenge $chal=\{i,v_i\}_{i\in I}$
4. generates and sends an auditing proof $P=\{\lambda,\sigma\}$
5. verifies the correctness of auditing proof P

Fig. 6. The processes of integrity auditing.

information is $\{1, 3\}$ and the index set K_2 of the data blocks corresponding to the organization's sensitive information is $\{5\}$. That is to say, in the original file F, m_1 and m_3 are the data blocks corresponding to the personal sensitive information, and m_5 is the data block corresponding to the organization's sensitive information. To preserve the personal sensitive information from the sanitizer, the user *ID* needs to blind the data blocks $\{m_1, m_3\}$. The blinded file is $F^*=\left(m_1^*, m_2^*, \ldots, m_n^*\right)$, where $m_i^* = m_i$ only if $i \in [1, n]$ and $i \notin \{1, 3\}$; otherwise, $m_i^* \neq m_i$. To unify the format, the sanitizer sanitizes the data blocks m_1^* and m_3^* after receiving the blinded file F^* from the user *ID*. In addition, to protect the privacy of organization, the sanitizer sanitizes the data block m_5^*. Then the sanitizer transforms the signatures of data blocks m_1^*, m_3^* and m_5^* into valid ones for sanitized file F'. As shown in Fig. 5, we can see that data blocks $m_i^* \neq m_i'$ and their corresponding signatures $\sigma_i \neq \sigma_i'$ only if $i \in \{1, 3, 5\}$; otherwise, $m_i^* = m_i'$ and $\sigma_i=\sigma_i'$.

5) **Algorithm ProofGen($\mathbf{F'}$, Φ', chal)**
The process is illustrated in Fig. 6.
   a) The TPA verifies the validity of the file tag τ. The TPA will not execute auditing task if the file tag τ is invalid; otherwise, the TPA parses τ_0 to obtain file identifier name *name* and verification values $g^{r_{ID}}$ and g^r, and then generates an auditing challenge *chal* as follows:
      i) Randomly picks a set I with c elements, where $I \subseteq [1, n]$.
      ii) Generates a random value $v_i \in Z_p^*$ for each $i \in I$.
      iii) Sends the auditing challenge $chal = \{i, v_i\}_{i\in I}$ to the cloud.
   b) After receiving an auditing challenge from the TPA, the cloud generates a proof of data possession as follows:
      i) Computes a linear combination of data blocks $\lambda = \sum_{i\in I} m_i' v_i$.
      ii) Calculates an aggregated signature $\sigma = \prod_{i\in I} \sigma_i'^{v_i}$.
      iii) Outputs an auditing proof $P = \{\lambda, \sigma\}$ to the TPA.

6) **Algorithm ProofVerify(chal, pp, P)**
The TPA verifies the correctness of auditing proof as follows:

$$e(\sigma, g)=e(g_1, g_2)^{\sum_{i\in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j}, g^{r_{ID}})^{\sum_{i\in I} v_i} \cdot e(\prod_{i\in I} H(name||i)^{v_i} \cdot u^{\lambda}, g^r). \quad (3)$$

If the equation (3) holds, the sanitized file stored in the cloud is intact; otherwise, it is not.

V. Security Analysis

In this section, we prove that the proposed scheme is secure in terms of correctness, sensitive information hiding and audit soundness.

Theorem 1 (The Correctness of Private Key): Our proposed scheme satisfies the following properties:

1) (Private key correctness) When the PKG sends a correct private key to the user, this private key can pass the verification of the user.
2) (Correctness of the blinded file and its corresponding signatures) When the user sends the blinded file and its corresponding valid signatures to the sanitizer, this file and its corresponding signatures can pass the verification of sanitizer.
3) (Auditing correctness) When the cloud properly stores the user's sanitized file, the proof it generates can pass the verification of the TPA.

Proof:

1) Given a correct private key $sk_{ID} = (sk'_{ID}, sk''_{ID})$ generated by the PKG, the verification equation (1) in *Extract* algorithm will hold. Based on the properties of bilinear maps, the equation (1) can be proved correct by deducing the left-hand side from the right-hand side:

$$\begin{aligned} e(sk'_{ID}, g) &= e(g_2{}^x(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j})^{r_{ID}}, g) \\ &= e(g_2{}^x, g) \cdot e(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j}, g^{r_{ID}}) \\ &= e(g_2, g^x) \cdot e(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j}, sk''_{ID}) \\ &= e(g_1, g_2) \cdot e(\mu' \prod_{j=1}^{l} \mu_j{}^{ID_j}, sk''_{ID}) \end{aligned}$$

2) Given a blinded file F and its corresponding valid signatures $\{\sigma_i\}_{1\leq i\leq n}$ from the user *ID*, the verification equation (2) in *SigGen* algorithm will hold. According to the properties of bilinear maps, the correctness of

equation (2) is presented as follows:

$$\begin{aligned} e(\sigma_i, g) &= e({g_2}^x \cdot (\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID}} \\ &\quad \cdot (H(name||i) \cdot u^{m_i^*})^r, g) \\ &= e({g_2}^x, g) \cdot e(\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}}) \\ &\quad \cdot e(H(name||i) \cdot u^{m_i^*}, g^r) \\ &= e(g_2, g^x) \cdot e(\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}}) \\ &\quad \cdot e(H(name||i) \cdot u^{m_i^*}, g^r) \\ &= e(g_1, g_2) \cdot e(\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}}) \\ &\quad \cdot e(H(name||i) \cdot u^{m_i^*}, g^r) \end{aligned}$$

3) Given a valid proof $P = \{\lambda, \sigma\}$ from the cloud, the verification equation (3) in *ProofVerify* algorithm will hold. Based on the properties of bilinear maps, the verification equation (3) can be proved correct by deducing the left-hand side from the right-hand side:

$$\begin{aligned} e(\sigma, g) &= e(\prod\nolimits_{i \in I} {\sigma_i'}^{v_i}, g) \\ &= e(\prod\nolimits_{i \in I} ({g_2}^x (\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID}} \\ &\quad \cdot (H(name||i) \cdot u^{m_i'})^r)^{v_i}, g) \\ &= e(\prod\nolimits_{i \in I} ({g_2}^x)^{v_i}, g) \\ &\quad \cdot e(\prod\nolimits_{i \in I} (\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID} \cdot v_i}, g) \\ &\quad \cdot e(\prod\nolimits_{i \in I} (H(name||i) \cdot u^{m_i'})^{r v_i}, g) \\ &= e(g_1, g_2)^{\sum_{i \in I} v_i} \cdot e(\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{i \in I} v_i} \\ &\quad \cdot e(\prod\nolimits_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda}, g^r) \end{aligned}$$

Theorem 2 (Auditing Soundness): Suppose the CDH problem is hard in bilinear groups and the signature scheme employed for generating file tags is existentially unforgeable. In the proposed scheme, for an adversary or an untrusted cloud, it is computationally infeasible to forge a proof that can pass the TPA's verification if the sanitized data stored in the cloud have been corrupted.

Proof: We will prove this theorem by using the method of knowledge proof. If the cloud can pass the TPA's verification without keeping the intact data, then, we can extract the intact challenged data blocks by repeatedly interacting between the proposed scheme and the constructed knowledge extractor. We will accomplish our proof by a series of games. Note that in our scheme, the data stored in the cloud is the sanitized data $\{m_i'\}_{1 \le i \le n}$, and their corresponding signatures are $\{\sigma_i'\}_{1 \le i \le n}$.

Game 0: In Game 0, both the challenger and the adversary behave in the way defined in Section III. That is, the challenger runs the *Setup* algorithm and *Extract* algorithm to obtain the master secret key *msk*, the system public parameters *pp* and the private key sk_{ID}, and then sends the parameters *pp* to the adversary. The adversary queries the signatures of a series of data blocks. The challenger runs the *SigGen* algorithm to compute the corresponding signatures of these data blocks, and sends these signatures to the adversary. And then the challenger sends a challenge to the adversary. Finally, the adversary returns a data possession proof $P = \{\lambda, \sigma\}$ to the challenger. If this proof can pass the verification of the challenger with non-negligible probability, then the adversary succeeds in this game.

Game 1: Game 1 is the same as Game 0, with one difference. The challenger holds a list of records about the queries of adversary. The challenger observes each instance of the challenge and response process with the adversary. If the adversary can generate a proof that passes the verification of the challenger, while the aggregate signature σ generated by the adversary is not equal to $\prod_{i \in I} {\sigma_i'}^{v_i}$ generated by the challenger based on maintained file, then the challenger declares failure and aborts.

Analysis: Assume that the adversary wins the Game 1 with non-negligible probability. Then, we can construct a simulator to solve the CDH problem. The simulator is given $g, g^\alpha, h \in G_1$, its goal is to generate h^α. The simulator acts like the challenger in Game 0, but with the following differences:

1) It randomly chooses an element $x \in Z_p^*$, and sets $g_1 = g^x$, $g_2 = h$ and the master secret key $msk = {g_2}^x$. And then, it selects two random values $a, b \in Z_p^*$, and sets $u = {g_2}^a g^b$.
2) It programs the random oracle H. It stores a list of queries and responses them in a consistent manner. When responding the queries of the form $H(name||i)$, the simulator answers them in the following way.
3) When processing a file F, it firstly extracts a private key $sk_{ID} = (sk_{ID}', sk_{ID}'') = ({g_2}^x \cdot (\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID}}, g^{r_{ID}})$ for *ID* by running *Extract* algorithm. And then, it chooses a random unique identifier name for file F and a random value $\bar{x} \in Z_p^*$, and calculates a verification value $g^r = (g^\alpha)^{\bar{x}}$, which implies $r = \alpha \bar{x}$. For each $i (1 \le i \le n)$ in the challenge, the simulator chooses a random value $r_i \in Z_p^*$, and programs the random oracle at i as

$$H(name||i) = g^{r_i} / ({g_2}^{a m_i'} \cdot g^{b m_i'}) \tag{4}$$

The simulator can compute σ_i' for data block m_i', since we have

$$\begin{aligned} (H(name||i) \cdot u^{m_i'})^r &= (g^{r_i} / ({g_2}^{a m_i'} \cdot g^{b m_i'}) \cdot u^{m_i'})^r \\ &= (g^{r_i} / ({g_2}^{a m_i'} \cdot g^{b m_i'}) \\ &\quad \cdot ({g_2}^a g^b)^{m_i'})^r \\ &= (g^{r_i} / ({g_2}^{a m_i'} \cdot g^{b m_i'}) \\ &\quad \cdot ({g_2}^{a m_i'} \cdot g^{b {m_i}'}))^r \\ &= (g^r)^{r_i} \end{aligned}$$

Therefore, the simulator calculates σ_i' as follows:

$$\begin{aligned} \sigma_i' &= {g_2}^x (\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID}} (H(name||i) \cdot u^{m_i'})^r \\ &= {g_2}^x (\mu' \prod\nolimits_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID}} (g^r)^{r_i}. \end{aligned}$$

4) The simulator continues interacting with the adversary to perform the remote data integrity scheme. As defined in Game 1, if the adversary succeeds, but the aggregate

signature σ' it generated is not equal to the expected aggregate signature σ, then this game aborts.

Assume that an honest prover provides a correct proof $\{\lambda, \sigma\}$. From the correctness of our scheme, we can know that this proof $\{\lambda, \sigma\}$ can pass the verification of the following equation, i.e., that

$$e(\sigma, g) = e(g_1, g_2)^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda}, g^r) \quad (5)$$

Assume that the adversary provides a proof $\{\lambda', \sigma'\}$ which is different from the honest prover provided. Because the game aborted, we can know that the forgery of adversary is successful. That is to say, the aggregate signature $\sigma' \neq \sigma$, but this aggregate signature σ' still can pass the verification of the following equation:

$$e(\sigma', g) = e(g_1, g_2)^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda'}, g^r) \quad (6)$$

Obviously, $\lambda \neq \lambda'$, otherwise $\sigma = \sigma'$, which contradicts our above assumption. We define $\Delta\lambda = \lambda' - \lambda$, and construct a simulator that could solve the CDH problem if the adversary makes the challenger abort with a non-negligible probability.

Now, dividing equation (6) by equation (5) and assuming $g^r = (g^{\alpha})^{\bar{x}}$, we obtain

$$e(\sigma'/\sigma, g) = e(u^{\Delta\lambda}, g^r) = e(({g_2}^a g^b)^{\Delta\lambda}, (g^{\alpha})^{\bar{x}}).$$

Thus, we can know that

$$e(\sigma' \cdot \sigma^{-1} \cdot (g^{\alpha})^{-\bar{x}b\Delta\lambda}, g) = e(h, g^{\alpha})^{\bar{x}a\Delta\lambda}. \quad (7)$$

From the equation (7), we can know that $h^{\alpha} = (\sigma'\sigma^{-1}(g^{\alpha})^{-\bar{x}b\Delta\lambda})^{1/(\bar{x}a\Delta\lambda)}$. Note that the probability of game failure is the same as the probability of $\bar{x} \cdot a \cdot \Delta\lambda = 0 \bmod p$. The probability of $\bar{x} \cdot a \cdot \Delta\lambda = 0 \bmod p$ is $1/p$ which is negligible since p is a large prime. Therefore, we can solve the CDH problem with a probability of $1-1/p$, which contradicts the assumption that the CDH problem in G_1 is computationally infeasible.

It means that if the difference between the adversary's probabilities of success in Game 1 and Game 2 is non-negligible, the constructed simulator can solve the CDH problem.

Game 3: Game 3 is the same as Game 2, with one difference. The challenger still keeps and observes each instance of the proposed remote data integrity auditing scheme. For one of these instances, if the aggregate message λ' is not equal to the expected λ generated by the challenger, then the challenger declares failure and aborts.

Analysis: Assume that the adversary wins the game 2 with non-negligible probability. We will construct a simulator that uses the adversary to solve the DL problem. The simulator is given $g, h \in G_1$, its goal is to calculate a value α satisfying $h = g^{\alpha}$. The simulator acts like the challenger in Game 2, but with the following differences:

1) When processing a file F, it firstly extracts a private key $sk_{ID} = (sk'_{ID}, sk''_{ID}) = ({g_2}^x \cdot (\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j})^{r_{ID}}, g^{r_{ID}})$ for ID by running *Extract* algorithm. And then, it chooses two random values $a, b \in Z_p^*$, and sets $= {g_2}^a g^b$, where $g_2 = h$.
2) The simulator continues interacting with the adversary to perform the remote data integrity scheme. As defined in Game 2, if the adversary succeeds, but the aggregation of data blocks λ' it generated is not equal to the expected aggregation λ of data blocks, then this game aborts.

Assume that an honest prover provides a correct proof $\{\lambda, \sigma\}$. From the correctness of the scheme, we can know that the following verification equation $e(\sigma, g) = e(g_1, g_2)^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda}, g^r)$ holds. Assume that the adversary provides a proof $\{\lambda', \sigma'\}$ which is different from what the honest prover provided. Because the game aborted, we can know that the forgery of the adversary is successful. Thus, this forged proof can pass the verification of the equation $e(\sigma', g) = e(g_1, g_2)^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda'}, g^r)$. According to Game 2, we know that $\sigma' = \sigma$. Define $\Delta\lambda = \lambda' - \lambda$. Based on the above two verification equations, we have

$$\begin{aligned} & e(g_1, g_2)^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{\mathrm{i} \in I} v_i} \\ & \quad \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda}, g^r) \\ & = e(\sigma, g) = e(\sigma', g) \\ & = e(g_1, g_2)^{\sum_{\mathrm{i} \in I} v_i} \cdot e(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}})^{\sum_{\mathrm{i} \in I} v_i} \\ & \quad \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda'}, g^r). \end{aligned}$$

Therefore, we have

$$u^{\lambda} = u^{\lambda'},$$

and can further imply that

$$1 = u^{\Delta\lambda} = ({g_2}^a g^b)^{\Delta\lambda} = h^{a\Delta\lambda} \cdot g^{b\Delta\lambda}.$$

In addition, we have $\Delta\lambda \neq 0 \bmod p$. Otherwise, we have $\lambda' = \lambda \bmod p$, which contradicts the aforementioned assumption.

Therefore, we can find the solution to the DL problem as follows,

$$h = g^{-\frac{b\Delta\lambda}{a\Delta\lambda}} = g^{-\frac{b}{a}}.$$

However, a is zero only with the probability $1/p$, which is negligible because p is a large prime. Then, we can get a solution to the DL problem with a probability of $1 - 1/p$, which contradicts the assumption that the DL problem in G_1 is computationally infeasible.

It means that if the difference between the adversary's probabilities of success in Game 2 and Game 3 is non-negligible, the constructed simulator can solve the DL problem.

Note that the hardness of the CDH problem implies the hardness of the DL problem. Thus, the differences between these games defined can be ignored on the condition that the CDH problem in G_1 is hard.

Finally, we construct a knowledge extractor to extract all of challenged data blocks $m'_i (i \in I, |i| = c)$ by selecting c different coefficients $v_i (i \in I, |I| = c)$ and executing c times different challenges on the same data blocks

TABLE II
FUNCTIONALITY COMPARISON WITH EXISTING RELATED SCHEMES

Schemes	Public verifiability	Certificate management simplification	Data sharing	Sensitive information hiding
Shacham et al. [4]	Yes	No	No	No
Wang et al.[20]	Yes	No	Yes	No
Li et al.[32]	Yes	Yes	No	No
Wang et al.[33]	No	No	No	No
Shen et al.[34]	Yes	No	No	No
Ours	Yes	Yes	Yes	Yes

$m_i'(i \in I, |i| = c)$. The knowledge extractor can get independently linear equations in the variables $m_i'(i \in I, |i| = c)$. By solving these equations, this knowledge extractor can extract $m_i'(i \in I, |i| = c)$. It means that if the cloud can pass the TPA's verification, it must correctly maintain the user's sanitized data.

Theorem 3 (The Detectability): Assume the sanitized file F' stored in the cloud is divided into n blocks, k data blocks in this file F' are modified or deleted by the cloud and c data blocks are challenged by the TPA. Our remote data integrity auditing scheme is $\left(\frac{k}{n}, 1 - \left(\frac{n-k}{n}\right)^c\right)$ detectable.

Proof: Let Y be the set of block-signature pairs corrupted by the cloud. Let S be the set of block-signature pairs challenged by the TPA. Let X be the random variable set, which is denoted as $X = |Y \cap S|$. Let P_X denote the probability of detecting the corrupted data blocks. In other words, if the cloud modifies or deletes k data blocks of the sanitized file F', the TPA can detect the cloud's misbehavior with probability P_X by challenging c data blocks. Therefore, we have

$$\begin{aligned} P_X &= P\{X \geq 1\} \\ &= 1 - P\{X = 0\} \\ &= 1 - \frac{n \quad - k}{n} \times \frac{n-1-k}{n-1} \times \ldots \times \frac{n-c+1-k}{n-c+1} \end{aligned}$$

We can know that $1 - (\frac{n-k}{n})^c \leq P_X \leq 1 - (\frac{n-c+1-k}{n-c+1})^c$, since $\frac{n-i-1-k}{n-i-1} \leq \frac{n-i-k}{n-i}$. Thus, we can conclude that the proposed scheme can detect the misbehavior of the cloud with probability at least $1 - \left(\frac{n-k}{n}\right)^c$.

Theorem 4: (Sensitive information hiding) The sanitizer cannot derive the personal sensitive information of the file, besides the cloud and the shared users cannot derive any sensitive information of the file in our scheme.

Proof: (1) In our scheme, the user needs to blind the data blocks corresponding to the personal sensitive information of the original file F before sending this file to the sanitizer. Because the blinding factors $\alpha_i = f_{k_1}(i, name)(i \in K_1)$ are randomly generated by the user, the blinded data blocks $m_i^* = m_i + \alpha_i$ $(i \in K_1)$ of F^* received by the sanitizer are unpredictable. Therefore, the sanitizer cannot know the real data blocks m_i $(i \in K_1)$ according to the blinded data blocks $m_i^* = m_i + \alpha_i$ $(i \in K_1)$ from the user. That is to say, the sanitizer cannot derive the personal sensitive information from F^* it received.

(2) After receiving the blinded file F^*, the sanitizer sanitizes the data blocks corresponding to the personal sensitive information and the organization's sensitive information, and uploads the sanitized file F' to the cloud. In the sanitized file F', the data blocks corresponding to all of the sensitive information of the original file F have been replaced with wildcards. It means that the cloud and the shared users cannot derive the sensitive information from the sanitized file F'. Therefore, the cloud and the shared users cannot know any sensitive information of the original file F. The sensitive information of the original file can be hidden.

VI. PERFORMANCE EVALUATION

In this section, we first give the functionality comparison among our scheme and several related schemes, and the computation overhead comparison between our scheme and Shacham and Waters scheme [4]. And then discuss the communication overhead and the computation complexity of our scheme. At last, we evaluate the performance of our scheme in experiments.

A. Functionality Comparison

We give the functionality comparison of our scheme with several related schemes [4], [20], [32]–[34]. As shown in Table II, our scheme is the only scheme that can satisfy all of the following properties: public verifiability, certificate management simplification, data sharing and sensitive information hiding. Note that schemes [4], [20], [32]–[34] all cannot support the sensitive information hiding.

B. Performance Analysis and Comparison

We define the following notations to denote the operations in our scheme. Let $Hash_{G_1}$, Exp_{G_1} and Mul_{G_1} respectively denote one hashing operation, one exponentiation operation and one multiplication operation in G_1. Similarly, $Sub_{Z_p^*}$, $Mul_{Z_p^*}$ and $Add_{Z_p^*}$ denote one subtraction operation, one multiplication operation and one addition operation in Z_p^*, respectively. $Pair$ denotes one pairing operation. Mul_{G_2} and Exp_{G_2} respectively denote one multiplication operation and one exponentiation operation in G_2. n is the total number of data blocks. c is the number of challenged data blocks. d_1 is the number of data blocks corresponding to the personal sensitive information. d_2 is the number of data blocks corresponding to the organization's sensitive information. l is the length of user identify. $|n|$ is the size of an element of set $[1, n]$, $|p|$ is the size of an element in Z_p^*, and $|q|$ is the size of an element in G_1.

TABLE III
THE COMPUTATION OVERHEAD OF OUR SCHEME AND SHACHAM ET AL. SCHEME [4] IN DIFFERENT PHASES

Phase	Computation overhead (Our scheme)	Computation overhead (Shacham et al.[4])
Data blinding	$d_1 Add_{Z_p^*}$	-
Signature generation	$n(Hash_{G_1}+2Mul_{G_1}+2Exp_{G_1})$	$n(Hash_{G_1}+Mul_{G_1}+2Exp_{G_1})$
Sanitization	$(d_1+d_2)(Exp_{G_1}+Mul_{G_1}+Sub_{Z_p^*})$	-
Proof generation	$(c-1)Mul_{G_1}+cExp_{G_1}+$ $cMul_{Z_p^*}+(c-1)Add_{Z_p^*}$	$(c-1)Mul_{G_1}+cExp_{G_1}+$ $cMul_{Z_p^*}+(c-1)Add_{Z_p^*}$
Proof verification	$4Pair+2Mul_{G_2}+2(c-1)Add_{Z_p^*}+2Exp_{G_2}$ $+(l+c+1)Exp_{G_1}+(c+l)Mul_{G_1}+cHash_{G_1}$	$2Pair+(c+1)Exp_{G_1}+cMul_{G_1}+$ $cHash_{G_1}$

TABLE IV
THE COMMUNICATION OVERHEAD OF THE PROPOSED SCHEME

Entity	Phase	Communication overhead
TPA	Auditing challenge	$c \cdot (\lvert n\rvert + \lvert p\rvert)$
Cloud	Auditing proof	$\lvert p\rvert + \lvert q\rvert$

TABLE V
THE COMPUTATION COMPLEXITY OF DIFFERENT ENTITIES IN DIFFERENT PHASES

	User	Sanitizer	TPA	Cloud
Data blinding	$O(d_1)$	-	-	-
Signature generation	$O(n)$	-	-	-
Sanitization	-	$O(d_1+d_2)$	-	-
Challenge generation	-	-	$O(c)$	-
Proof generation	-	-	-	$O(c)$
Proof verification	-	-	$O(c)$	-

1) **Computation overhead comparison**. In Table III, we give the computation overhead comparison between our scheme and Shacham and Waters scheme [4] in different phases. From Section IV, we can know that, before sending the original file to the sanitizer, the user needs to blind the data blocks corresponding to the personal sensitive information of the file. The computation overhead of data blinding is $d_1 Add_{Z_p^*}$. And then the user generates the signatures for the blinded file. The computation overhead of computing signatures is $n(Hash_{G_1}+2Mul_{G_1}+2Exp_{G_1})$. The sanitizer sanitizes the blinded data blocks and the data blocks corresponding to the organization's sensitive information, and transforms the signatures of these data blocks into valid ones for the sanitized file. In the phase of sanitization, the computation overhead on the sanitizer side is $(d_1+d_2)(Exp_{G_1}+Mul_{G_1}+Sub_{Z_p^*})$. In the phase of integrity auditing, the TPA firstly needs to generate and send an auditing challenge to the cloud, which only costs negligible computation overhead. And then, the cloud outputs an auditing proof $P=\{\lambda,\sigma\}$ to reply the TPA. The computation overhead of the cloud is $(c-1)Mul_{G_1}+cExp_{G_1}+cMul_{Z_p^*}+(c-1)Add_{Z_p^*}$. When the TPA verifies this auditing proof, the computation overhead of the TPA is $4\ Pair+2Mul_{G_2}+2(c-1)Add_{Z_p^*}+2Exp_{G_2}+(l+c+1)Exp_{G_1}+(c+l)Mul_{G_1}+cHash_{G_1}$.
The construction of the scheme [4] is generally regarded as one of the most efficient one among all existing remote data integrity auditing schemes. In this scheme [4], one data block is divided into multiple sectors, which can reduce storage overhead. Actually, our scheme also can support multiple sectors. But for simplification, we only consider the situation that the file F is divided into n data blocks, and do not consider the situation that each data block is divided into s sectors. For fairness comparison, we set $s=1$ in scheme [4]. As shown in Table III, we can see that our scheme and the scheme [4] have almost the same computation overhead in the phase of signature generation. It means that our scheme and the scheme [4] have the same efficiency when processing the same file. In the phase of proof generation, the cloud's computation overhead in our scheme and the scheme [4] are the same. For the TPA in the phase of proof verification, our scheme costs more computation overhead than the scheme [4]. On the other hand, as Table II shows, our scheme can satisfy the following properties: data sharing, sensitive information hiding and certificate management simplification. However, the scheme [4] cannot satisfy the above properties. Therefore, the above comparison shows that our scheme has the same efficiency level with the scheme [4], but meets more properties.

2) **Communication overhead**. According to the description of Section IV, we can know that the communication overhead of the proposed scheme is mainly from the integrity auditing phase, as shown in Table IV. Thus, we only consider the communication overhead incurred in the remote data integrity auditing. In the phase of integrity auditing, the TPA sends an auditing challenge $chal=\{i,v_i\}_{i\in I}$ to the cloud. The size of an auditing challenge is $c\cdot\ (|n|+|p|)$ bits. After receiving the auditing challenge from the TPA, the cloud generates an auditing proof $P=\{\lambda,\sigma\}$ to reply the TPA. The size of an auditing proof $P=\{\lambda,\sigma\}$ is $|p|+|q|$ bits. Therefore, for one auditing task, the whole communication overhead is $c\cdot|n|+(c+1)\cdot|p|+|q|$ bits.

3) **Computation complexity**. We analyze the computation complexity of the different entities in different phases in Table V. The computation complexity of different

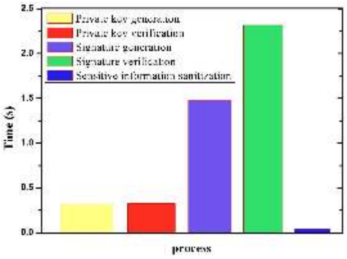


Fig. 7. Performance of different processes.

entities in our scheme respectively depends on the number c of challenged blocks, the total number n of data blocks, the number d_1 of data blocks corresponding to the personal sensitive information and the number d_2 of data blocks corresponding to the organization's sensitive information. From Table V, we see that the computation complexities of data blinding and signature generation for the user are $O(d_1)$ and $O(n)$ respectively. On the sanitizer side, the computation complexity of data sanitization is $O(d_1 + d_2)$. The computation overheads of challenge generation and proof verification are both $O(c)$ on the TPA side. The computation complexity of proof generation for the cloud is $O(c)$.

C. Experimental Results

In this subsection, we evaluate the performance of the proposed scheme by several experiments. We run these experiments on a Linux machine with an Intel Pentium 2.30GHz processor and 8GB memory. All these experiments use C programming language with the free Pairing-Based Cryptography (PBC) Library [35] and the GNU Multiple Precision Arithmetic (GMP) [36]. 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 data file to be 20MB composed by 1,000,000 blocks, and the length of user identify to be 160 bits.

1) Performance of Different Processes: To effectively evaluate the performance in different processes, we set the number of data blocks to be 100 and the number of sanitized data blocks to be 5 in our experiment. As shown in Fig. 7, private key generation and private key verification spend nearly the same time, which are nearly 0.31s. The time consumed by the signature generation is 1.476s. The time of signature verification and that of sensitive information sanitization respectively are 2.318s and 0.041s. So we can conclude that in these processes, the signature verification spends the longest time and the sensitive information sanitization spends the shortest time.

To evaluate the performance of signature generation and signature verification, we generate the signatures for different number of blocks from 0 to 1000 increased by an interval of 100 in our experiment. As shown in Fig. 8, the time cost of the signature generation and the signature verification both linearly increases with the number of the data blocks. The time of signature generation ranges from 0.121s to 12.132s. The time of signature verification ranges from 0.128s to 12.513s.

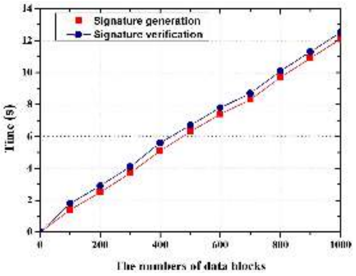


Fig. 8. The computation overhead in the process of signature generation and signature verification.

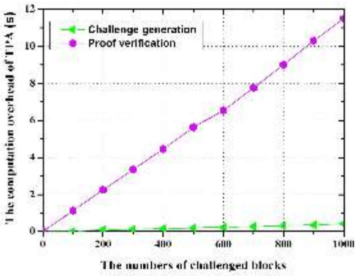


Fig. 9. The computation overhead of the TPA in the phase of integrity auditing.

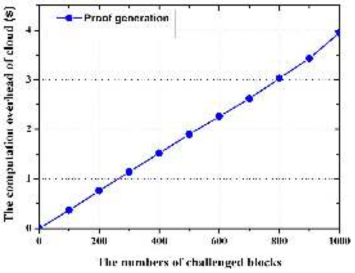


Fig. 10. The computation overhead of the cloud in the phase of integrity auditing.

2) Performance of Auditing: With the different number of challenged data blocks, we respectively show the computation overhead of the TPA and that of the cloud in integrity auditing phase in Fig. 9 and Fig. 10. In our experiment, the number of challenged data blocks varies from 0 to 1,000. As shown in Fig. 9, we see the that the computation overheads of challenge generation and proof verification on the TPA side linearly increase with the number of challenged data blocks. The computation overhead of proof verification varies from 0.317s to 11.505s. Compared with the time of proof verification, the time of challenge generation increases slowly, just varying from 0.013s to 0461s. From Fig. 10, we have the observation that the computation overhead of proof generation on the cloud side varies from 0.021s to 3.981s. So we can conclude that, with the more challenged data blocks, both the TPA and the cloud will spend the more computation overheads.

VII. Conclusion

In this paper, we proposed an identity-based data integrity auditing scheme for secure cloud storage, which supports data sharing with sensitive information hiding. In our scheme, the file stored in the cloud can be shared and used by others on the condition that the sensitive information of the file is protected. Besides, the remote data integrity auditing is still able to be efficiently executed. The security proof and the experimental analysis demonstrate that the proposed scheme achieves desirable security and efficiency.

References

[1] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan. 2012.

[2] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.

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

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

[5] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013.

[6] S. G. Worku, C. Xu, J. Zhao, and X. He, "Secure and efficient privacy-preserving public auditing scheme for cloud storage," *Comput. Electr. Eng.*, vol. 40, no. 5, pp. 1703–1713, 2014.

[7] C. Guan, K. Ren, F. Zhang, F. Kerschbaum, and J. Yu, "Symmetric-key based proofs of retrievability supporting public verification," in *Computer Security—ESORICS*. Cham, Switzerland: Springer, 2015, pp. 203–223.

[8] W. Shen, J. Yu, H. Xia, H. Zhang, X. Lu, and R. Hao, "Light-weight and privacy-preserving secure cloud auditing scheme for group users via the third party medium," *J. Netw. Comput. Appl.*, vol. 82, pp. 56–64, Mar. 2017.

[9] J. Sun and Y. Fang, "Cross-domain data sharing in distributed electronic health record systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 6, pp. 754–764, Jun. 2010.

[10] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw.*, 2008, Art. no. 9.

[11] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 213–222.

[12] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, May 2011.

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

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

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

[16] J. Yu, R. Hao, H. Xia, H. Zhang, X. Cheng, and F. Kong, "Intrusion-resilient identity-based signatures: Concrete scheme in the standard model and generic construction," *Inf. Sci.*, vols. 442–443, pp. 158–172, May 2018.

[17] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2012, pp. 295–302.

[18] G. Yang, J. Yu, W. Shen, Q. Su, Z. Fu, and R. Hao, "Enabling public auditing for shared data in cloud storage supporting identity privacy and traceability," *J. Syst. Softw.*, vol. 113, pp. 130–139, Mar. 2016.

[19] A. Fu, S. Yu, Y. Zhang, H. Wang, and C. Huang, "NPP: A new privacy-aware public auditing scheme for cloud data sharing with group users," *IEEE Trans. Big Data*, to be published, doi: 10.1109/TBDATA.2017.2701347.

[20] B. Wang, B. Li, and H. Li, "Panda: Public auditing for shared data with efficient user revocation in the cloud," *IEEE Trans. Serv. Comput.*, vol. 8, no. 1, pp. 92–106, Jan./Feb. 2015.

[21] Y. Luo, M. Xu, S. Fu, D. Wang, and J. Deng, "Efficient integrity auditing for shared data in the cloud with secure user revocation," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, pp. 434–442.

[22] H. Wang, "Identity-based distributed provable data possession in multi-cloud storage," *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 328–340, Mar./Apr. 2015.

[23] H. Wang, D. He, and S. Tang, "Identity-based proxy-oriented data uploading and remote data integrity checking in public cloud," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1165–1176, Jun. 2016.

[24] Y. Yu *et al.*, "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 767–778, Apr. 2017.

[25] H. Wang, D. He, J. Yu, and Z. Wang, "Incentive and unconditionally anonymous identity-based public provable data possession," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2016.2633260.

[26] Y. Zhang, J. Yu, R. Hao, C. Wang, and K. Ren, "Enabling efficient user revocation in identity-based cloud storage auditing for shared big data," *IEEE Trans. Depend. Sec. Comput.*, to be published, doi: 10.1109/TDSC.2018.2829880.

[27] W. Shen, G. Yang, J. Yu, H. Zhang, F. Kong, and R. Hao, "Remote data possession checking with privacy-preserving authenticators for cloud storage," *Future Gener. Comput. Syst.*, vol. 76, pp. 136–145, Nov. 2017.

[28] 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.

[29] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 3113–3125, Nov. 2016.

[30] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik, "Sanitizable signatures," in *Proc. 10th Eur. Symp. Res. Comput. Secur.* Berlin, Germany: Springer-Verlag, 2005, pp. 159–177.

[31] G. Ateniese and B. de Medeiros, "On the key exposure problem in chameleon hashes," in *Security in Communication Networks*. Berlin, Germany: Springer, 2005, pp. 165–179.

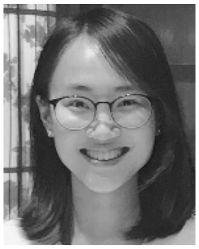
[32] Y. Li, Y. Yu, G. Min, W. Susilo, J. Ni, and K.-K. R. Choo, "Fuzzy identity-based data integrity auditing for reliable cloud storage systems," *IEEE Trans. Depend. Sec. Comput.*, to be published, doi: 10.1109/TDSC.2017.2662216.

[33] H. Wang, "Proxy provable data possession in public clouds," *IEEE Trans. Serv. Comput.*, vol. 6, no. 4, pp. 551–559, Oct./Dec. 2013.

[34] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2402–2415, Oct. 2017.

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

[36] *The GNU Multiple Precision Arithmetic Library (GMP)*. Accessed: Nov. 2017. [Online]. Available: http://gmplib.org



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.

Jing Qin received the B.S. degree from Information Engineering University, Zhengzhou, China, in 1982, and the Ph.D. degree from the School of Mathematics, Shandong University, China, in 2004. She is currently a Professor with the School of Mathematics, Shandong University, China. Her research interests include computational number theory, information security, and design and analysis of security about cryptologic protocols. She has co-authored two books and has published about 30 professional research papers. She is a Senior Member of Chinese Association for Cryptologic Research and China Computer Federation.

Jia Yu received the B.S. and M.S. degrees from the School of Computer Science and Technology, Shandong University, in 2000 and 2003, respectively, and the Ph.D. degree from the Institute of Network Security, Shandong University, in 2006. He was a Visiting Professor with the Department of Computer Science and Engineering, the State University of New York at Buffalo, from 2013 to 2014. He is currently a Professor with the College of Computer Science and Technology, Qingdao University. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.

Rong Hao is currently with the College of Computer Science and Technology, Qingdao University. Her research interest is cloud computing security and cryptography.

Jiankun Hu receive the Ph.D. degree in control engineering from the Harbin Institute of Technology, China, in 1993, and the master's degree in computer science and software engineering from Monash University, Australia, in 2000. He was a Research Fellow with Delft University, The Netherlands, from 1997 to 1998, and The University of Melbourne, Australia, from 1998 to 1999. His main research interest is in the field of cyber security, including biometrics security, where he has published many papers in high-quality conferences and journals including the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. He has served on the editorial boards of up to seven international journals and served as a Security Symposium Chair of the IEEE Flagship Conferences of IEEE ICC and IEEE GLOBECOM. He has obtained seven Australian Research Council (ARC) Grants. He is currently serving on the prestigious Panel of Mathematics, Information and Computing Sciences, ARC ERA (The Excellence in Research for Australia) Evaluation Committee.