World Wide Web (2012) 15:409–428
DOI 10.1007/s11280-011-0138-0

Data storage auditing service in cloud computing: challenges, methods and opportunities

Kan Yang · Xiaohua Jia

Received: 9 March 2011 / Revised: 16 June 2011 /
Accepted: 28 June 2011 / Published online: 16 July 2011

Abstract Cloud computing is a promising computing model that enables convenient and on-demand network access to a shared pool of configurable computing resources. The first offered cloud service is moving data into the cloud: data owners let cloud service providers host their data on cloud servers and data consumers can access the data from the cloud servers. This new paradigm of data storage service also introduces new security challenges, because data owners and data servers have different identities and different business interests. Therefore, an independent auditing service is required to make sure that the data is correctly hosted in the Cloud. In this paper, we investigate this kind of problem and give an extensive survey of storage auditing methods in the literature. First, we give a set of requirements of the auditing protocol for data storage in cloud computing. Then, we introduce some existing auditing schemes and analyze them in terms of security and performance. Finally, some challenging issues are introduced in the design of efficient auditing protocol for data storage in cloud computing.

Keywords data storage auditing · data owner auditing · third party auditing · cloud computing

1 Introduction

Cloud computing is a promising computing model that enables convenient and on-demand network access to a shared pool of computing resources [32]. Cloud

K. Yang (✉) · X. Jia
Department of Computer Science,
City University of Hong Kong, Kowloon, Hong Kong
e-mail: lvkiky@gmail.com

X. Jia
e-mail: csjia@cityu.edu.hk

computing offers a group of services, including Software as a Service (SAAS), Platform as a Service (PAAS) and Infrastructure as a Service (IAAS) [1]. Cloud storage is an important service of cloud computing, which allows data owners to move data from their local computing systems to the Cloud. More and more data owners start choosing to host their data in the Cloud. The main reason is because of cost effectiveness, which is particularly true for small and medium-sized businesses. By hosting their data in the Cloud, data owners can avoid the initial investment of expensive infrastructure setup, large equipments, and daily maintenance cost. The data owners only need to pay the space they actually use, e.g., cost-per-gigabyte-stored model [48]. Another reason is that data owners can rely on the Cloud to provide more reliable services, so that they can access data from anywhere and at any time. Individuals or small-sized companies usually do not have the resource to keep their servers as reliable as the Cloud does.

By hosting data in the Cloud, it introduces new security challenges. Firstly, data owners would worry their data could be mis-used or accessed by unauthorized users. Extensive researches have been done on this security issue of data hosting [6, 11–13, 16, 19, 20, 24, 27, 41, 53]. Secondly, the data owners would worry their data could be lost in the Cloud. This is because data loss could happen in any infrastructure, no matter what high degree of reliable measures the cloud service providers would take [5, 40]. Some recent data loss incidents are the Sidekick Cloud Disaster in 2009 [9] and the breakdown of Amazon's Elastic Compute Cloud (EC2) in 2010 [34]. Sometimes, the cloud service providers may be dishonest and they may discard the data which has not been accessed or rarely accessed to save the storage space or keep fewer replicas than promised. Moreover, the cloud service providers may choose to hide data loss and claim that the data are still correctly stored in the Cloud. As a result, data owners need to be convinced that their data are correctly stored in the Cloud.

Checking on retrieval is a common method for checking the data integrity, which means data owners check the data integrity when accessing their data. This method has been used in peer-to-peer storage systems [25, 35], network file systems [22, 26], long-term archives [31], web-service object stores [54] and database systems [30]. However, *checking on retrieval* is not sufficient to check the integrity for all the data stored in the Cloud. There is usually a large amount of data stored in the Cloud, but only a small percentage is frequently accessed. There is no guarantee for the data that are rarely accessed. An improved method was proposed by generating some virtual retrievals to check the integrity of rarely accessed data. But this causes heavy I/O overhead on the cloud servers and high communication cost due to the data retrieval operations.

Therefore, it is desirable to have storage auditing service to assure data owners that their data are correctly stored in the Cloud. But data owners are not willing to perform such auditing service due to the heavy overhead and cost. In fact, it is not fair to let any side of the cloud service providers or the data owners conduct the auditing, because neither of them could be guaranteed to provide unbiased and honest auditing result [49]. Third party auditing is a natural choice for the storage auditing. A third party auditor who has expertise and capabilities can do a more efficient work and convince both the cloud service provider and the data owner. Data storage auditing is a very resource demanding operation in terms of computational resource, memory space, and communication cost. There are three performance criteria in the design of storage auditing protocols:

- *Low storage overhead*. The additional storage used for auditing should be as small as possible on both the Auditor and the cloud server.
- *Low communication cost*. The communication cost required by the auditing protocol should be as low as possible.
- *Low computational complexity*. The computational complexity for storage auditing should be low, especially on the Auditor.

In this paper, we give an extensive survey of storage auditing methods and evaluate these methods against the above performance criteria. The remaining of this paper is organized as follows: In Section 2, we discuss the system model for data storage auditing in cloud computing. Section 3 presents some literature methods for data storage auditing, followed by the security and performance analysis in Section 4. In Section 5, we propose some challenging issues in data storage auditing. Then, the conclusion is given in Section 6.

2 Data storage auditing model

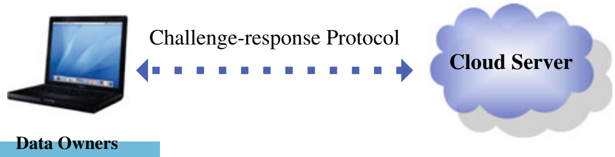
In this section, we describe the system model and threat model of data storage auditing protocol in cloud computing.

2.1 System model

In 1991, Blum et al. [6] first described the auditing problem that enables data owners to check the integrity of remote data without explicit knowledge of the entire data. In recent years, with the development of distributed storage systems and online storage systems [23], the data storage auditing problem becomes even more significant and many protocols have been proposed: e.g., Remote Integrity Checking (RIC) protocols [14, 28], Proof of Retrievability (POR) protocols [21] and Provable Data Possession (PDP) protocols [18]. However, most of the existing protocols only allowed data owners to check the integrity of their remote stored data. We denote this type of auditing protocols as the *Data Owner Auditing*. The system model of the *Data Owner Auditing* only contains the remote cloud server and data owners as shown in Figure 1.

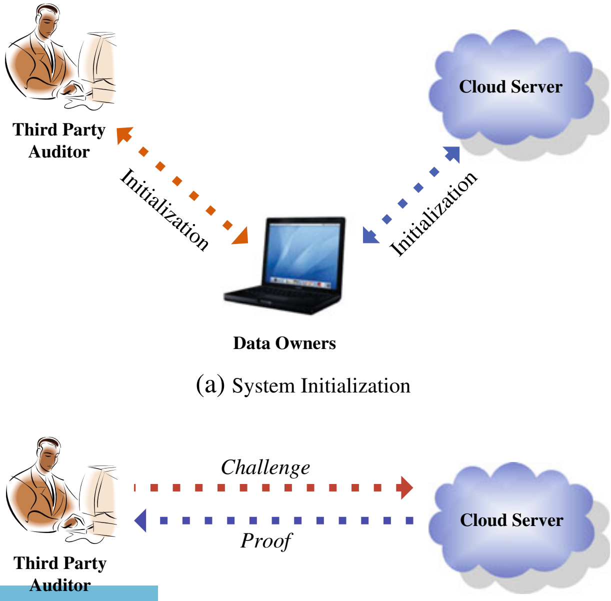
However, in cloud computing, the data storage auditing service is preferred to be provided by a third party auditor, denoted as the *Third Party Auditing*, rather than

Figure 1 System model of the data owner auditing.



by data owners. For the *Third Party Auditing*, the system model contains three types of entities: data owners, the cloud server and the third party auditor, as shown in Figure 2. During the system initialization, as demonstrated in Figure 2a, data owners compute the metadata of their data and negotiate the cryptographic keys with the third party auditor and the cloud server. After that, data owners can choose to be off-line and do not require to engage any auditing query. Figure 2b shows the system model for each auditing query which only involves the third party auditor and the cloud server. Each auditing query is conducted via a challenge-response auditing protocol, which contains three phases: *Challenge*, *Proof* and *Verification*. When the third party auditor wants to check the correctness of data owners' data stored on the cloud server, it generates and sends a *challenge* to the cloud server. On receiving the *challenge* from the third party auditor, the cloud server generates a *proof* of data storage and sends it back to the third party auditor. Then, the third party auditor runs the *verification* to check the correctness of the *proof* from the cloud server and extracts the result on this audit query.

As we can see in Figure 2b, the system model of each auditing query in the *Third Party Auditing* is quite similar to the one of the *Data Owner Auditing*. But the methods for the *Data Owner Auditing* may not be directly used in the *Third Party Auditing* due to the following two reasons: (1) Sending data to the third party auditor can leak out the data owners' data, and (2) The third party auditor has no responsibility to store all the metadata. Thus, both the *Data Owner Auditing* protocols and the *Third Party Auditing* protocols will be introduced and discussed in the next section.



(a) System Initialization

(b) Auditing Query via the Challenge-response Protocol

Figure 2 System model of the third party auditing.

2.2 Threat model

We consider the third party auditor is honest-but-curious. It performs honestly during the whole auditing procedure but it is curious about the received data. Thus, for the storage of secured data, there is also a privacy requirement for the third party auditing protocol. That is, no data will be leaked out to the third party auditor during the auditing procedure.

But the Sever is dishonest and may conduct the following attacks:

Replace Attack Suppose the Server discarded a challenged data block m_i or its metadata t_i, in order to pass the auditing, it may choose another valid and uncorrupted pair of data block and metadata (m_k, t_k) to replace the original challenged pair of data block and metadata (m_i, t_i).

Replay Attack The Server generates the proof from the previous proof or other information, without querying the actual Owner's data.

Forge Attack The Server may forge the metadata of data block and deceive the Auditor.

3 Methods of data storage auditing

The existing data storage auditing methods can be classified into three categories: Message Authentication Code (MAC)-based methods, RSA-based Homomorphic methods and Boneh–Lynn–Shacham signature [7] (BLS)-based Homomorphic methods. For simplification, we use the Owner, the Server and the Auditor to denote the data owner, the cloud service provider (or cloud server) and the third party auditor respectively.

3.1 MAC-based methods

The message authentication code (MAC) is a kind of hash function which has been used for checking the data integrity for a long time. In the cooperative Internet backup scheme proposed by Lillibridge et al. [28], the Reed-Solomon erasure-correcting codes [39] are applied to generate redundancy blocks. After that, the MAC is calculated for each encrypted data block. Then, the system peers perform spot-checks of data blocks using MACs.

Naor and Rothblum [36] extended the memory checking schemes of Blum et al. [6] and proposed a protocol in which an error-correcting code is applied to a file and blocks are then MACed. Whenever a verifier (the Owner or the Auditor) needs to check data integrity, he retrieves a random set of blocks from the Server and re-computes the MAC of each block for comparison.

Based on the precomputed MACs stored on the verifier, the protocols proposed by Lillibridge et al. and Naor et al. can detect any data loss or corruption with high probability. However, in these two protocols, the Server was required to send the original data to the Auditor, which would leak data to the Auditor. Furthermore, they also required the Auditor to store all the MACs. Therefore, both Lillibridge's and Naor's protocols are not suitable for the *Third Party Auditing*.

Juels and Kaliski [21] gave a formal definition of *Proof of Retrievability* (POR) and proposed a POR scheme in which the file F is divided into blocks and each block is encoded by using an error-correcting code. After encrypting the encoded file, a set of randomly-valued check blocks called *sentinels* are embedded into the encrypted file. For auditing the data storage, the verifier (the Owner or the Auditor) challenges the Server by specifying the positions of a subset of *sentinels* and asking the Server to return the associated *sentinels* values. The security of this protocol is proved by Dodis et al. in [15] without making any simplifying assumptions on the behavior of the adversary. However, this POR protocol is still not suitable for the *Third Party Auditing* because it only allows a limited number of auditing times which is related to the number of *sentinels*.

In Shah's auditing protocols [44, 45], the Owner pre-computes some MACs of the data with different secret keys and sends all the MACs and keys to the Auditor. When verifying data integrity, the Auditor selects and sends a key k to the Server. Then, the Server computes the MAC with k and returns it to the Auditor for comparison with the one stored on the Auditor. However, the number of times a particular data item can be verified is limited by the number of secret keys that fixed beforehand. Also, the Auditor needs to store several MACs for each file. Therefore, Shah's auditing protocols still cannot be applied for the *Third Party Auditing*.

Shacham et al. [43] proposed an MAC-based batch verification for multiple data blocks, which was also proposed in [10]. During the system initialization of this protocol, the Owner divides the erasure encoded data into n blocks $m_1, \cdots, m_n \in \mathbb{Z}_p$ for some large prime p. Then, the Owner randomly chooses two numbers α and s from $\mathbb{Z}_p^*$, which are kept as secrets. Let $s_i = G(s, i)(i = 1, 2, \cdots, m)$, where G is a secure pseudo random number generator. The Owner calculates an MAC for each data block i as $\sigma_i = \alpha m_i + s_i \ mod \ p$ and sends both the data blocks $\{m_i\}$ and their corresponding MACs $\{\sigma_i\}$ to the Server. Their proof of retrievability protocol can be described as follows. The Auditor chooses a random challenge set Q and generates some random coefficients $v_i (i \in Q)$ in $\mathbb{Z}_p$ and sends the *challenge* $\{(i, v_i)\}_{i \in Q}$ to the Server. The Server then calculates and sends the *proof* (σ, μ) back to the Auditor, where $\sigma = \sum_{i \in Q} v_i \sigma_i$ and $\mu = \sum_{i \in Q} v_i m_i$. The Auditor accepts the *proof* if $\sigma = \alpha\mu + \sum_{i \in Q} v_i s_i$. In this method, the Server needs to send the linear combination of all the challenged data blocks $\mu = \sum_{i \in Q} v_i m_i$ to the Auditor, which may also leak the data to the Auditor. Since the Auditor has the full knowledge of all the coefficients $\{(i, v_i)\}$, it is possible for the Auditor to recover the data blocks when received enough linear combinations of data blocks.

3.2 RSA-based homomorphic methods

A homomorphism is a mapping $f : \mathbb{P} \rightarrow \mathbb{Q}$ between two groups, which has the property of $f(g_1 \oplus g_2) = f(g_1) \otimes f(g_2)$ for all $g_1, g_2 \in \mathbb{P}$, where $\oplus$ and $\otimes$ denote the operations in $\mathbb{P}$ and $\mathbb{Q}$ respectively. The homomorphism has been used to define the homomorphic hash value or homomorphic tag which have two main types: One is based on RSA, such as the homomorphic hash value in [14, 18] and homomorphic tag in [2, 10, 52]. The other type is based on BLS, such as the homomorphic tags used in [43] and [55].

We first introduce the RSA-based homomorphic methods. Let $N = pq$ be a RSA modulus, where p and q are prime numbers, such that $p' = (p-1)/2$ and $q' = (q-1)/2$ are also prime numbers.

3.2.1 RSA-based homomorphic hash value

Filho et al. [18] proposed a cryptographic protocol via an RSA-based homomorphic hash function, through which the Server can demonstrate possession of a set of data to the Owner. In this protocol, during the system initialization, the Owner calculates a hash value $h(m) = m \; mod \; \phi(N)$ (where $\phi(N) = (p-1)(q-1)$ is an Euler Function of N) and sends the data m to the Server. Then, the Owner may delete the data m on his own system and only keeps the hash value $h(m)$. For the data storage auditing, the Owner randomly chooses an integer $b \, (1 < b < N-1)$ and sends it to the Server. The Server computes $M_p = b^m \; mod \; N$ and sends it back the Owner. Then, the Owner computes $M'_p = b^{h(m)} \; mod \; N$ and checks whether $M_p = M'_p$. If true, the Owner is convinced that the Server stores the data m correctly.

Sebe et al. [42] improved Filho's protocol by first dividing data into blocks and generating an RSA-based homomorphic hash function on each data block. Their data possession checking protocol is based on the Diffie–Hellman key exchange method which can be described as follows:

(a) System initialization: the owner divides the data m into l-bit blocks $m_1, \cdots, m_n$ ($n = \lceil |m|/l \rceil$) (the last block is padded with 0s in the most significant bit positions if its length is less than l) and stores a homomorphic hash value $M_i = m_i \; mod \; \phi(N)$ for each data block m_i.

(b) Challenge-response protocol: the owner generates a random seed S and a random element $a \in \mathbb{Z}_N$, then sends the *challenge* (a, S) to the Server. Upon receiving the *challenge*, the Server generates n pesudorandom values $v_i \in [1, 2^t]$ $(i = 1, \cdots, n)$ using *PRNG* seeded by S, where t is the security parameter and *PRNG* is a pseudorandom number generator which can generate t-bit integer values. Then the Server sends a *proof* $R = a^r \; mod \; N$ to the Owner, where $r = \sum_{i=1}^{n} v_i m_i$. The Owner also generates n pesudorandom values $v_i \in [1, 2^t]$ $(i = 1, \cdots, n)$ using *PRNG* seeded by S and computes $R' = a^{r'} \; mod \; N$, where $r' = \sum_{i=1}^{n} v_i T_i \; mod \; \phi(N)$. The *proof* of the Server is correct if $R = R'$.

Both Filho's protocol and Sebe's protocol are designed for the *Data Owner Auditing* and require the Owner to store the homomorphic hash value. Thus, these two protocols cannot be applied for the *Third Party Auditing*.

3.2.2 RSA-based homomorphic tag

Deswarte et al. [14] defined a homomorphic tag for a file m, which can help the Owner check the data integrity based on the Diffie–Hellman key exchange method. In this protocol, during the system initialization, the Owner randomly chooses an element $a (1 < a < N-1)$, and computes the homomorphic tag of the file m: $t = a^m \; mod \; N$. Then, the Owner sends the file m to the Server and stores the homomorphic tag t. When checking the integrity of m, the Owner chooses a random value $r (1 < r < N-1)$ and sends the *challenge* $R = a^r \; mod \; N$ to the Server. Upon receiving the *challenge*, the Server calculates and responses a *proof* $P = R^m \ mod \ N$ to the Owner. If $t^r \ mod \ N = P$, the Owner considers that m is correctly stored on the Server.

Similarly, Yamamoto et al. [52] presented a fast integrity checking scheme for verifying the integrity of message. Their idea is to use homomorphic tags for all message blocks and make batch verification based on the homomorphism of the tags. Suppose there are n message blocks $(m_1, m_2, \cdots, m_n)$. The sender generates the tags $(t_1, t_2, \cdots, t_n)$ for all blocks as $t_i = g^{m_i} \ mod \ N$, where g is a generator of $\mathbb{Z}_N$. The receiver checks the data integrity by randomly choosing $v \in [0, s-1]$ and computes the verification vector $v = (v_1, v_2, \cdots, v_n)$ where $v_i = v^{i-1} \ mod \ s (i = 1, \cdots, n)$ and s is the system parameter. Then, the receiver computes $M_p = \sum_{i=1}^{n} v_i m_i$ in $\mathbb{Z}$ and $T_p = \prod_{i=1}^{n} t_i^{v_i}$ in $\mathbb{Z}_N$. If $M_p = T_p \ mod \ N$, all the received message blocks are correct.

Although the batch verification based on the homomorphic tags can improve the efficiency of auditing, both Deswarte's protocol and Yamamoto's protocol are still not suitable for the *Third Party Auditing* due to the storage of tags on the Owner. For the *Third Party Auditing*, Ateniese et al. [2] proposed a Sampling Provable Data Possession (S-PDP) scheme which combines the RSA cryptography with homomorphic tags. In this scheme, the Auditor does not need to keep the homomorphic tags and the sampling mechanism greatly reduces the workload of the Server. The S-PDP scheme can be described as follows.

Let f be a pseudorandom function, let π be a pseudorandom permutation and let H be a cryptographic hash function. Let $h : \{0,1\}^* \rightarrow \mathbb{QR}_N$ be a secure deterministic hash value and encode function that maps strings uniformly to $\mathbb{QR}_N$, where $\mathbb{QR}_N$ is a unique cyclic subgroup of $\mathbb{Z}_N^*$.

(a) System initialization: suppose the data M is encrypted and divided into n data blocks $(m_1, m_2, \cdots, m_n)$. The Owner generates the security keys $pk = (N, g)$ and $sk = (e, d, v)$, such that $ed = 1 \ mod \ p'q'$. e is a large secret prime such that $e > \lambda$ and $d > \lambda$, where λ is a system parameter. g is a generator of $\mathbb{QR}_N$ and $v \overset{R}{\leftarrow} \{0,1\}^\kappa$. Then, the Owner computes the data tag for all data blocks m_i: $T_i = (h(W_i) \cdot g^m)^d \ mod \ N$, where $W_i = v||i$ ($||$ denotes the combination operation) and sends pk, $M = (m_1, m_2, \cdots, m_n)$ and $T = (T_1, T_2, \cdots, T_n)$ to the Server for storage. Then, the Owner sends pk, e and the data information (e.g., the data name and the number of blocks n etc.) to the Auditor. After that, the Owner may delete the data M and its data tag T from its local storage system.

(b) Challenge-response protocol:

(1) *Challenge* The Auditor generates a *challenge* (c, k_1, k_2, g_s), where $k_1 \overset{R}{\leftarrow} \{0,1\}^\kappa$, $k_2 \overset{R}{\leftarrow} \{0,1\}^\kappa$, $g_s = g^s \ mod \ N$ and $s \overset{R}{\leftarrow} \mathbb{Z}_N^*$ and sends the *challenge* to the Server.

(2) *Proof* Upon reception, the Server computes the indices of the blocks $i_j = \pi_{k_1}(j)$ and the coefficients $a_j = f_{k_2}(j)$ for each $j (1 \le j \le c)$; Then computes and sends both the *tag proof* $T = \prod_{j=1}^{c} T_{i_j}^{a_j}$ and the *data proof* $\rho = H(g_s^{\sum_{j=1}^{c} a_j m_j}) \ mod \ N$ to the Auditor. □

(3) *Verification* The Auditor computes $i_j = \pi_{k_1}(j)$, $a_j = f_{k_2}(j)$, $W_{i_j} = v||i_j$ for $1 \leq j \leq c$ and compute

$$\tau = \frac{T^e}{\prod_{j=1}^{c} h(W_{i_j}^{a_j})} \ mod \ N.$$

If $H(\tau^s \ mod \ N) = \rho$, the Auditor considers that all the challenged data blocks are correctly stored on the Server.

In Ateniese's protocol, the homomorphic tag is encrypted by a private key d, so that it allows the public verification with the public key e. This protocol also does not require the Auditor to store the data tags, so that it can be applied to the *Third Party Auditing*. Further to this protocol, Ateniese et al. [4] also provided a framework for building public-key homomorphic tags from any identification protocol satisfying certain homomorphic properties. They also illustrated the transformations by applying them to an identification protocol by Shoup [46] and thus obtained a proof of storage protocol based on the hardness of factoring.

As we can see, in the protocols using the RSA-based homomorphic tag, the Server is always required to exponentiate the entire data [14] or data blocks [2, 52]. That is because the Server has no knowledge about the factorization of the RSA modulus N, so that it cannot use the Theorem of Euler or the Chinese Remainder Theorem to simplify the computation complexity. In order to reduce the computation complexity of the exponentiation on the Server, Shacham et al. [43] proposed a method which also appeared in other protocols using the BLS-based homomorphic tags [55]. Similar to the methods used in Sebe's protocol [42] by splitting the data into data blocks, their method is to further divide the data blocks to several sectors, so that it can reduce the computation complexity by just exponentiating the linear combination of sectors.

In Shacham's protocol [43], given the data M, the Owner first applies the erasure code to obtain M', then splits M' into n blocks and s sectors $\{m_{ij}\}$ $(1 \leq i \leq n, 1 \leq j \leq s)$ for each block. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ be a full-domain hash function. Let e and d be the public key and the private key respectively. For each block m_i, the Owner computes the tag $\sigma_i = (H(name||i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}})^d \ mod \ N$, where u_j are randomly selected elements in $\mathbb{Z}_N^*$. The data $\{m_{ij}\}$ are sent to the Server together with the tags $\{\sigma_i\}$. The Owner sends the public key e and the data information to the Auditor. For data storage auditing, the Auditor randomly chooses a subset of data blocks Q and generates the coefficients v_i for each selected data block m_i. Then, it sends a *challenge* $\{(i, v_i)\}_{i \in Q}$ to the Server. When received the *challenge*, the Server computes and sends back both the *tag proof* $\sigma = \prod_{i \in Q} \sigma_i^{v_i} \ mod \ N$ and the *data proof* $\mu_j = \sum_{i \in Q} v_i \cdot m_{ij}$, where the sum is computed in $\mathbb{Z}$ without the modular reduction. The verification equation is

$$\sigma^e \overset{?}{=} \prod_{i \in Q} H(name||i)^{v_i} \cdot \prod_{j=1}^{s} u_j^{\mu_j} \ mod \ N.$$

Although the computation complexity can be reduced by further splitting the data blocks into several sectors, the exponentiation computation on the linear combination of sectors is still costly. This problem can be solved by using the homomorphic tags based on the BLS signatures [7].

3.3 BLS-based homomorphic methods

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative groups with the same prime order p. A bilinear mapping is a mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- **Bilinearity**: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
- **Non-degeneracy**: There exist $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ such that $e(u, v) \neq I$, where I is the identity element of $\mathbb{G}_T$.
- **Computability**: e can be efficiently computed.

Such a bilinear mapping is called a bilinear pairing. Especially, if g_1 and g_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, $e(g_1, g_2)$ is the generator of $\mathbb{G}_T$. The BLS-based homomorphic tag is generated in the scenario of bilinear pairing.

Shacham and Waters [43] proposed a proofs of retrievability scheme by using the homomorphic tags based on BLS signatures. Suppose the data is divided into n blocks $m = \{m_1, \cdots, m_n\}$. Let $\mathbb{G}$ and $\mathbb{G}_T$ be the multiplicative groups with the same prime order p and $e : G \times G \to G_T$ be a computable bilinear map. Let $x \in \mathbb{Z}_p$ and $v = g^x \in G$ be the private key and public key of the Owner. The homomorphic tag on block m_i is $\sigma_i = (H(i)u^{m_i})^x$, where u is another generator of G. Then, the Owner sends the data $\{m_i\}$ and the tags $\{\sigma_i\}$ to the Server. For auditing query, the Auditor chooses a random subset of data blocks Q and generates the coefficients v_i for each chosen data block m_i and sends a *challenge* $\{(i, v_i)\}$ to the Server. The Server computes and sends back $\sigma = \prod_{i \in Q} \sigma_i^{v_i}$ and $\mu = \sum_{i \in Q} v_i \cdot m_i$. The *proof* is correct if the verification equation $e(\sigma, g) = e(\prod_{i \in Q} H(i)^{v_i} \cdot u^{\mu}, v)$ is true. This scheme supports the public verification: the private key x is used for generating the homomorphic tags $\{\sigma_i\}$ and the public key v is sufficient for the verifier to do auditing.

Shacham and Waters also proposed an extension of this protocol by further splitting the data blocks into several sectors, similar to the protocol proposed independently by Zeng in [55]. However, in Shacham's protocols, the Server needs to send the linear combinations of data blocks back to the Auditor. As we analyzed, it will leak the data to the Auditor. To enhance Shacham's protocols, Wang et al. proposed a privacy-preserving public auditing protocol with a random mask technique [50]. In their protocol, the Server does not directly send back the linear combination of data blocks to the Auditor. Instead, the Server will choose a random number and combine this random number with these chosen data blocks. Then the Server encrypts this random number and sends it back to the Auditor together with the linear combination of both data blocks and the random number. Similar mask techniques were also applied in other protocols to protect the data privacy of Owners, such as the IPDP protocol proposed in [56].

Zhu et al. [56] proposed an IPDP scheme to check the data integrity and ensure the confidentiality of secret data in private Cloud. In IPDP scheme, let $e : G \times G \to G_T$ be a computable bilinear map with randomly selected generators $g, h \in_R \mathbb{G}$, where $\mathbb{G}$ and $\mathbb{G}_\mathbb{T}$ are two multiplicative groups with the same prime order p. Let $H_k()$ be a keyed hash function and $sk = (\alpha, \beta)$ be the secret key, where α and β are randomly chosen in $\mathbb{Z}_p$. Let $pk = (g, h, h_1, h_2)$ be the public key, where $h_1 = h^{\alpha}$ and $h_2 = h^{\beta}$.

The file F is divided into n blocks and s sectors per block $F = \{m_{ij}\}(1 \leq i \leq n, 1 \leq j \leq s)$. The Owner chooses s random numbers $k_1, k_2, \cdots, k_s \in \mathbb{Z}_p$ as the secret of this file and computes $u_i = g^{k_i} \in \mathbb{G}$ and $H_1 = H_k(F_n)$, where $k = \sum_{i=1}^{s} k_i$ for all

$k_i (i = 1, \cdots, n)$ and F_n is the file name. For each data block, the Owner builds an index $\chi_i = (B_i = i, V_i = 1, R_i \in_R \{0,1\}^*)$ and calculates a tag $t_i = (H_2)^{\alpha} g^{\sum_{j=1}^{s} k_j m_{ij} \beta} \in \mathbb{G}$, where $H_2 = H_{H_1}(\chi_i)$ $1 \leq i \leq n$. Then, the Owner sends (K, U, χ) to the Auditor, where $K = \{k_1, \cdots, k_s\}$, $U = \{u_1, \cdots, u_s\}$ and $\chi = \{\chi_1, \cdots, \chi_n\}$ and sends (F, T) to the Server, where $F = \{m_{ij}\}$ and $T = \{t_i\} (1 \leq i \leq n, 1 \leq j \leq s)$.

The IPDP protocol between the Server and the Auditor can be shown as follows:

(1) Commitment The Server chooses a random number $r \in \mathbb{Z}_p$, and sends its commitment $C = (h_1^r)$ to the Auditor.

(2) Challenge The Auditor chooses a random subset of data blocks Q and generates a random coefficient v_i for each chosen data block. Then, it sends the *challenge* $\{(i, v_i)\}_{i \in Q}$ to the Server.

(3) Proof The Server choose s integers λ_j $(1 \leq j \leq s)$ at random and calculates the *data proof* $M = \{M_j\}$ where $M_j = \lambda_j + r \cdot \sum_{i \in Q} v_i m_{ij}$ and the *tag proof* $T_p = \prod_{i \in Q} t_i^{r v_i}$. Then it sends the *proof* (M, T_p, π) to the Auditor, where $\pi = \{\pi_j\}$ and $\pi_j = u_j^{\lambda_j} (1 \leq j \leq s)$.

(4) Verification The Auditor verifies the *proof* by checking the following equation

$$e(T_p, h) \overset{?}{=} e\left(\prod_{i \in Q} H_2^{v_i}, h_1^r\right) \cdot e\left(\prod_{j=1}^{s} \frac{u_j^{M_j}}{\pi_j}, h_2\right).$$

If the above equation is true, the Auditor believes that the challenged data and their tags are correctly stored on the Server. Moreover, Zhu et al. also proposed a cooperative PDP protocol for hybrid Cloud with an organizer. □

4 Analysis of auditing methods

In this section, we first analyze the security of auditing methods we introduced and then give the performance of them in terms of storage overhead, communication cost and computation complexity.

4.1 Security analysis

During the description of the existing auditing protocols, we find that different protocols may have different security levels. For example, some protocols can only allow the data owner to conduct the auditing and some protocols may be provably secure in random oracle model or standard model.

We summarize the security analysis in Table 1. From the table, it is not difficult to find that most of the MAC-based methods are semantically secure since they do not rely on public keys. Some RSA-based protocols are provably secure in the standard model, while others which rely on homomorphic tags are provably secure only in the random oracle model.

We also analyze and compare the privacy-preservation of those protocols. As we described in the threat model in Section 2.2, it is significant to guarantee the data privacy during the procedure of *Third Party Auditing*. Hence, those schemes which are not privacy-preserving cannot be applied in the *Third Party Auditing*.

Table 1 Security analysis.

Protocol	Security	Assumption	Privacy-Preserving
Lillibridge's [28]	SS	N/A	No
Naor's [36]	SS	N/A	No
Juels's [21]	SS	N/A	Yes
Shah's [45]	PS/RO	DL	Yes
Shacham's (MAC) [43]	SS	N/A	No
Filho's [18]	PS/ST	IF	Yes
Sebe's [42]	PS/ST	IF, DH	Yes
Deswarte's [14]	PS/ST	DH	Yes
Yamamoto's [52]	PS/ST	DH	No
Ateniese's [2]	PS/RO	IF, DH	Yes
Shacham's (RSA) [43]	PS/RO	IF, DH	No
Shacham's (BLS) [43]	PS/RO	CDH	No
Wang's [50]	PS/RO	CDH	Yes
Zhu's [56]	PS/RO	CDH	Yes

SS: Semantically Secure, *PS*: Provably Secure, *ST*: Standard Model, *RO*: Random Oracle Model, *DL*: Discrete Logarithm, *IF*: Integer Factorization, *DH*: Diffie–Hellman, *CDH*: Computational DH

4.2 Storage overhead

As we can see in the existing data storage auditing protocols, the metadata (e.g., MAC, signature and tag etc.) plays an important part during the auditing procedure, while it also contributes the major part of the storage overhead. For example, in [28] and [36], the verifier (the Owner) has to store the MACs of the data or data blocks. In homomorphic methods [42, 52], the verifier (the Owner or the Auditor) needs to store the homomorphic hash value or homomorphic tag.

But in some MAC-based methods [10, 43], the metadata is as long as each data blocks and the storage overhead of metadata is 100% of the data. Also, in RSA-based homomorphic methods, the size of metadata is equal to the size of RSA modulus. For example, in the SPDP scheme [3], given a 2048-bit RSA modulus, if the RSA public key e is chosen to be a 6168 bits prime, the SPDP scheme requires that each block should be less than $e/2$, otherwise the SPDP scheme is not provable secure. In this case, a 64-MB data would be divided into at least $87,056$ blocks, each of which has a 2048-bit long metadata. Thus, the total size of all the metadata is about 21.2MB [55].

In order to reduce the storage overhead caused by metadata, short metadata are preferred in data storage auditing protocol. For example, in some BLS-based homomorphic auditing protocols [43, 55, 56], the homomorphic tags based on BLS signatures are much shorter than the RSA-based homomorphic tags. Besides, some other methods are proposed to reduce the storage overhead in previous works. For instance, Oprea et al. [37] proposed a space-efficient block storage integrity checking scheme which exhibits a tradeoff between the level of security and the additional client's storage overhead. And Bowers et al. [8] also improved the protocols of Juels [21] to achieve lower storage overhead.

However, in the *Third Party Auditing*, there is no need for the Auditor to store the metadata for all Owners. First, the Auditor is a third party organizer who provides the data storage auditing service for both Servers and Owners. Another reason is that the storage ability of the Auditor is limited and not as powerful as the Server. Although the metadata may be very small compared to the data, the total storage of

Table 2 Storage overhead analysis.

Protocol	Storage overhead	
	On the auditor	On the server
Lillibridge's [28]	$\|h(m)\|$	N/A
Naor's [36]	$n \cdot \|h(m_i)\|$	N/A
Juels's [21]	$k \cdot \|sentinel_i\|$[a]	$k \cdot \|sentinel_i\|$[a]
Shah's [45]	$O(l \cdot \lambda) + l \cdot \|h(k_i, m)\|$[b]	N/A
Shacham's (MAC) [43]	N/A	$n \cdot \|p\|$
Filho's [18]	$\|\phi(N)\|$	N/A
Sebe's [42]	$n \cdot \|\phi(N)\|$	N/A
Deswarte's [14]	$\|N\|$	N/A
Yamamoto's [52]	$n \cdot \|N\|$	N/A
Ateniese's [2]	$O(\lambda)$	$n \cdot \|N\|$
Shacham's (RSA) [43]	$O(\lambda)$	$n \cdot \|N\|$
Shacham's (BLS) [43]	$O(\lambda)$	$n \cdot \|p\|$
Wang's [50]	$O(\lambda)$	$n \cdot \|p\|$
Zhu's [56]	$O(\lambda)$	$n \cdot \|p\|$

[a] k denotes the total number of *sentinels* in the file.

[b] l denotes the total number of precomputed keys.

metadata will still require a large space. That is because: (1) The number of Owners is very large and still is increasing; (2) For each Owner, the number of data is also an increasing large number. Thus, the Auditor is not able to serve the storage of the metadata whose size is linear with the data stored in Servers.

In this situation, an obvious method is storing the metadata on the Server instead of storing on the Auditor, which has already been used in some protocols [2, 43, 55]. During the system initialization, after computing the metadata, the Owner stores the metadata together with the data on the Server. For each auditing query, the Server will generate and send back both the *data proof* and the *tag proof* to the Auditor instead of only the *data proof*. In fact, the less storage on the Auditor the more communication cost will be. There is a trade-off between the storage overhead on the Auditor and the communication cost between the Servers and the Auditor during the auditing procedure.

Table 2 shows the storage overhead comparison of some protocols we introduced. In Table 2, the data m is divided into n blocks $m = \{m_1, \cdots, m_n\}$ and each data block $m_i (i = 1, \cdots, n)$ is divided into s sectors $m_i = \{m_{i1}, \cdots, m_{is}\}$. Let λ be the security parameter which is usually the size of key. Let $|x|$ be the length of x and $h()$ be the hash function. Let p denote the order of the groups and $\phi(N)$ denotes the Euler Function on the RSA modulus N.

4.3 Communication cost

Since Owners are not involved in each auditing query in the *Third Party Auditing* protocols, the main communication cost we concerned is the communication cost between the Server and the Auditor during each challenge-response auditing query. Suppose Q is the random selected challenge query set with c items, we compare the communication cost of some protocols as shown in Table 3. It is easy to find that the communication cost can be reduced by using short homomorphic tags, such as

Table 3 Communication cost for each challenge-response auditing query.

Protocol	Communication cost	
	Challenge	*Proof*
Lillibridge's [28]	N/A	$\|m\|$
Naor's [36]	N/A	$c \cdot \|m_i\|$
Juels's [21]	N/A	$\sum_{i\in Q} \|sentinel_i\|$[a]
Shah's [45]	k	$\|h(k_i, m)\|$
Shacham's (MAC) [43]	$c \cdot \|v_i\|$	$\|p\| + \|m_i\|$
Filho's [18]	N/A	$\|N\|$
Sebe's [42]	$\|N\|$	$\|N\|$
Deswarte's [14]	$\|N\|$	$\|N\|$
Yamamoto's [52]	$c \cdot \|v_i\|$	$\|m_i\|$
Ateniese's [2]	$\|N\|$	$\|N\| + \|N\|$
Shacham's (RSA) [43]	$c \cdot \|v_i\|$	$\|N\| + \|m_{ij}\|$
Shacham's (BLS) [43]	$c \cdot \|v_i\|$	$\|p\| + \|p\|$
Wang's [50]	$c \cdot \|v_i\|$	$\|p\| + \|p\| + \|p\|$
Zhu's [56]	$c \cdot \|v_i\|$	$\|p\| + \|p\| + s \cdot \|p\|$

[a] All the *sentinels* within the challenged block set Q.

the BLS-based homomorphic tags. To further reduce the communication cost, two main mechanisms are used in the existing protocols: *Sampling* and *Batch Auditing*.

Sampling In the protocol proposed by Lillibridge et al. [28], the Server has to send the entire data back to the Auditor which will cause heavy communication cost. Further to [28], Naor and Rothblum [36] proposed a protocol to reduce the communication cost by dividing the data into several blocks and just returning some sampled data blocks during each query. This sampling method is also applied in those homomorphic methods, such as Ateniese's PDP protocol [2] and Shacham's compact POR protocol [43] etc.

Batch auditing The batch processing for multiple message blocks was first applied in Yamamoto's protocol [52]. Via a RSA-based homomorphic hash function h, to check that all of $t_1 = h(m_1)$, $t_2 = h(m_2)$, $\cdots$, $t_n = h(m_n)$ hold, it suffices to check $T_p = h(M_p)$, where M_p and T_p are linear combinations of m_i and t_i respectively with a weight randomly chosen by the verifier. With the batch auditing, the Server just needs to send the linear combination of all the sampled data blocks whose size is equal to one data block, so that it can save the communication bandwidth.

In the *Third Party Auditing* protocols [2, 42, 43, 55], the metadata are stored on the Server together with the data. As a result, during each auditing query, the Server needs to send both *data proof* and *tag proof* to the Auditor which contributes higher communication cost than those protocols which require the Auditor to maintain a copy of metadata. Thus, as we discussed, there is a tradeoff between the storage overhead on the Auditor and the communication cost between the Auditor and the Server.

4.4 Computation complexity

Under the *Third Party Auditing* model, the computation cost of the auditing system comes from three parts: the computation cost on Owners during the system initialization, the computation cost on both the Server and the Auditor for each challenge-response auditing query.

Computation cost on owners In the *Third Party Auditing* model, Owners are only involved in the system initialization. So the computation cost on Owners comes from two main parts: error-correcting code computation and metadata computation. During the system initialization, the Owner first needs to encode the data with an error-correcting code. In [21], Juels et al. pointed that the error-correcting encoding on the entire data used in [36] is inefficient, and they suggested to apply the error-correcting code on each data block to reduce the computation cost of error-correcting code. For the metadata computation, fortunately, the computation can be conducted efficiently: With the full knowledge of the factorization of the RSA modulus, Owners can use the Theorem of Euler to simplify the tag computation by replacing the exponent m by the short value ($m \, mod \, \phi(N)$) of length around 1024 bits or 2048 bits which is independent of the length of the data blocks and the Chinese Remainder Theorem can also be applied to reduce the computation cost.

Computation cost on the auditor Since the Auditor is an organization who provides the auditing service for both Owners and Servers, its computation ability is not as strong as the Server. In many previous works, the computation cost on the Auditor during each auditing query can be reduced by shifting the computation load from the Auditor to the data storage server. However, in some protocols, the computation cost on the Auditor is still heavy.

Computation cost on the server In RSA-based homomorphic methods, the computation of the exponentiation of the data contributes the main part of all the computation cost on the Server. As we mentioned, in some protocols [14, 18], the Server needs to exponentiate the entire data which will cause heavy computation cost, since the Server has no knowledge about the factorization of the RSA modulus. To reduce the computation cost of the exponentiation, in [52] and [42], the Owners first divide the data into several blocks and do the batch processing for some sampled data blocks, so that the Server only needs to exponentiate the linear combination of all the sampling data blocks whose size is the same as one data block. To further reduce the exponentiation computation cost, Shacham et al. [43] and Zeng [55] divided each data block into several sectors and the Server just needs to exponentiate the linear combinations of the sampled data sectors whose size is the same to each data sector. However, in BLS-based homomorphic methods, the computation cost on the exponentiation of the data can be reduced by the modular reduction in $\mathbb{Z}_p$.

5 Challenging issues of data storage auditing

5.1 Dynamic auditing

As we can see, most of the previous auditing protocols (also denoted as Proof of Retrievability (POR) or Provable Data Possession (PDP)) are designed for the static archive storage system, e.g., libraries and scientific datasets. However, in cloud computing, the dynamic scalability is a significant issue for various applications which means that the data stored on the cloud server can be dynamically updated by data owners such as: block modification, deletion and insertion. Therefore, an efficient dynamic auditing protocol is essential in practical cloud storage systems.

Ateniese et al. developed a dynamic provable data possession protocol [3] based on cryptographic hash function and symmetric key encryption. Their idea is to pre-compute a certain number of metadata during the setup period, so that the number of updates and challenges is limited and fixed beforehand. In their protocol, each update operation requires recreating all the remaining metadata, which is problematic for large files. Moreover, their protocol cannot perform block insertions anywhere (only append-type insertions are allowed).

Erway et al. [17] proposed a definitional framework for dynamic provable data possession, which extends the PDP model to support probable updates on the stored data. They also proposed two dynamic provable data possession scheme by using a new version of authenticated dictionaries based on rank information. They constructed the rank-based authenticated dictionaries in two ways: one is based on the Skip List and the other is using the Merkle Hash Tree [33, 38]. As illustrated in [17], there are some challenging issues for dynamic data storage auditing:

(1) *Computation complexity for updates*. The data update operations (e.g., data modification, deletion and insertion) will cause additional computation cost on both data owners and cloud servers. Thus, low computation complexity is essential for an efficient dynamic data storage auditing protocol. For example, in [3], the authors used the symmetric key cryptographic to reduce the computation cost, while the recreation of all the remaining metadata still cause a large number of computation cost on the Server.
(2) *Communication cost for updates*. For data modification and insertion, it is required to send the new data block and its metadata to the Server. The communication cost should be as low as possible.
(3) *Storage overhead for updates*. Dynamic operations can also contribute additional storage for both the cloud server and the third party auditor, e.g., the rank-based authenticated dictionary in [17], the Merkle Hash Tree in [51] and the index table in [56]. Thus, the storage overhead for updates is required to be as small as possible in the dynamic auditing protocol in cloud computing.
(4) *Security requirement for updates*. The update operations may introduce additional threats to the auditing system. For instance, in Ateniese's protocol [3], the Server can deceive the verifier by using the previous metadata or responses due to the lack of the randomness in the *challenge*. Also, in Wang's auditing protocol [51], the data block updating makes the auditing system insecure due to the replay attack on the same hash values. To avoid the replay attack, Zhu et al. [56] proposed remote data integrity checking protocols to support the dynamic change of the data by using an index table.

These challenging issues should be seriously considered when designing an efficient and secure dynamical auditing protocol for data storage.

5.2 Collaborative auditing

In [29], Lin pointed out that the group collaboration is an important application in cloud computing. There are some reasons: First, the client-server essence of cloud computing makes it suitable to support group collaboration. Second, the benefit of group collaboration may be the main reason for an organization or a corporation deciding to subscribe a cloud service. Lin also proposed a data storage auditing protocol for the scenario of group collaboration based on the protocols proposed by Smart et al. [47] and Wang et al. [51].

For the group collaboration, Zhu et al. [56] proposed a Cooperative Provable Data Possession (CPDP) protocol to deal with the PDP problem in a hybrid Cloud which combines the public Clouds with the private Clouds. In their protocol, the verification is performed by a 5-way interactive *proof* among an organizer, an auditor and some provers. (1) The organizer initiates the protocol and sends the commitment to the auditor; (2) The auditor returns a challenge set of random index-coefficient pairs Q to the organizer; (3) The organizer relays them into each prover P_i in the group $\mathbb{P}$ according to the exact position of each data block; (4) Each prover P_i returns its response of *challenge* to the organizer; (5) The organizer synthesizes a final response from these responses and sends it to the auditor. However, Zhu's protocol [56] may not be feasible in real cloud computing systems due to the requirement of a honest organizer.

There are some challenging issues for the collaborative auditing for data storage:

(1) *Job assignment* As we discussed above, the collaborative auditing protocol are preferred to involve only the third party auditor and several cloud servers. Without a honest third party organizer, the job assignment will be a significant challenging problem for designing the collaborative auditing protocol for data storage.

(2) *Communication cost* Since the collaborative auditing is carried on several cloud servers, the data transfer among these cloud servers will play an important part in the collaborative auditing. Therefore, the communication cost for collaborative auditing should be as slow as possible.

(3) *Security guarantee* The collaborative auditing will involve different cloud service providers, which may have different security levels, such as the public Clouds and the private Clouds. During the collaborative auditing, the cloud service providers may need to send their auditing data to other intermediate cloud service providers, which may leak the data out. Therefore, it is necessary for the collaborative auditing protocol to provide the security guarantee for all the involved data.

In addition, it is required to consider the *Storage Overhead* and the *Computation Complexity* on both the third party auditor and cloud servers.

5.3 Batch auditing

The idea of batch auditing was first proposed in Yamamoto's protocol [52]. Via a RSA-based homomorphic hash function h, to check that all of $t_1 = h(m_1), t_2 = h(m_2), \cdots, t_n = h(m_n)$ hold, it suffices to check $T_p = h(M_p)$, where M_p and T_p are linear combinations of m_i and t_i respectively with a weight randomly chosen by the verifier. Compared with auditing individually, the batch auditing has several advantages: (1) It can save the communication bandwidth, because the Server just needs to send the linear combination of all the sampled data blocks whose size is equal to one data block; (2) It can reduce the computation complexity for auditing on both the third party auditor and the cloud server.

The batch auditing for multiple data blocks are applied in many auditing protocols, such as Ateniese's PDP protocol [2], Shacham's POR protocols [43] and Zhu's CPDP protocol [56] etc. However, in these protocols, the batch auditing only support multiple data blocks of the same data owner. Wang et al. proposed a batch auditing protocol for multiple data blocks from multiple data owners [50]. For example, given k auditing delegations on k distinct data files from k different data owners, it is more efficient for the third party auditor to batch these k tasks together and audit it once.

Although the batch auditing can greatly improve the efficiency of auditing, when designing the batch auditing protocol, it is necessary to consider the *Computation Complexity* and *Communication Cost* for batch operations.

6 Conclusion

In this paper, we investigated the auditing problem for data storage in cloud computing and proposed a set of requirements of designing the *Third Party Auditing* protocols. We also described and analyzed the existing auditing methods in the literature. Finally, we discussed some challenging issues in the design of efficient auditing protocols for data storage in cloud computing.

References

1. Armbrust, M., et al.: A view of cloud computing. Commun. ACM **53**, 50–58 (2010)
2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS '07, pp. 598–609. ACM, New York, NY, USA (2007)
3. Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, SecureComm '08, pp. 9:1–9:10. ACM, New York, NY, USA (2008)
4. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT '09, pp. 319–333. Springer, Berlin, Heidelberg (2009)
5. Bairavasundaram, L.N., Goodson, G.R., Pasupathy, S., Schindler, J.: An analysis of latent sector errors in disk drives. In: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '07, pp. 289–300. ACM, New York, NY, USA (2007)
6. Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, SFCS '91, pp. 90–99. IEEE Computer Society, Washington, DC, USA (1991)
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptol. **17**, 297–319 (2004)
8. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW '09, pp. 43–54. ACM, New York, NY, USA (2009)
9. Cellan-Jones, R.: The Sidekick Cloud Disaster. BBC News, vol. 1 (2009)
10. Chang, E.C., Xu, J.: Remote integrity check with dishonest storage server. In: Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security, ESORICS '08, pp. 223–237. Springer, Berlin, Heidelberg (2008)
11. Clarke, D., Devadas, S., van Dijk, M., Gassend, B., Suh, G.E.: Incremental multiset hash functions and their application to memory integrity checking. In: Proceedings of the 9th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT'03, pp. 188–207. Springer (2003)

12. Curtmola, R., Khan, O., Burns, R.: Robust remote data checking. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability, StorageSS '08, pp. 63–68. ACM, New York, NY, USA (2008)
13. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: MR-PDP: multiple-replica provable data possession. In: Proceedings of the 2008 the 28th International Conference on Distributed Computing Systems, ICDCS '08, pp. 411–420. IEEE Computer Society, Washington, DC, USA (2008)
14. Deswarte, Y., Quisquater, J., Saidane, A.: Remote integrity checking. In: The Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS). Springer Netherlands (2004)
15. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, TCC '09, pp. 109–127. Springer (2009)
16. Dwork, C., Naor, M., Rothblum, G.N., Vaikuntanathan, V.: How efficient can memory checking be? In: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, TCC '09, pp. 503–520. Springer (2009)
17. Erway, C., Kupccu, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, pp. 213–222. ACM, New York, NY, USA (2009)
18. Gazzoni Filho, D., Barreto, P.: Demonstrating data possession and uncheatable data transfer. Tech. Rep., Citeseer (2006)
19. Goodson, G.R., Wylie, J.J., Ganger, G.R., Reiter, M.K.: Efficient byzantine-tolerant erasure-coded storage. In: Proceedings of the 2004 International Conference on Dependable Systems and Networks, pp. 135–. IEEE Computer Society, Washington, DC, USA (2004)
20. Hu, L., Ying, S., Jia, X., Zhao, K.: Towards an approach of semantic access control for cloud computing. In: Cloud Computing, pp. 145–156 (2009)
21. Juels, A., Kaliski, Jr., B.S.: Pors: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS '07, pp. 584–597. ACM, New York, NY, USA (2007)
22. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: scalable secure file sharing on untrusted storage. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp. 29–42. USENIX Association, Berkeley, CA, USA (2003)
23. Kher, V., Kim, Y.: Securing distributed storage: challenges, techniques, and systems. In: Proceedings of the 2005 ACM workshop on Storage Security and Survivability, StorageSS '05, pp. 9–25. ACM, New York, NY, USA (2005)
24. Krohn, M., Freedman, M., Mazieres, D.: On-the-fly verification of rateless erasure codes for efficient content distribution. In: Proceedings of IEEE Symposium on Security and Privacy, pp. 226–240 (2004)
25. Kubiatowicz, J., et al.: Oceanstore: an architecture for global-scale persistent storage. SIGPLAN Not. **35**, 190–201 (2000)
26. Li, J., Krohn, M., Mazieres, D., Shasha, D.: Secure untrusted data repository (sundr). In: Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, vol. 6, pp. 9–9. USENIX Association, Berkeley, CA, USA (2004)
27. Li, M., Yu, S., Ren, K., Lou, W.: Securing personal health records in cloud computing: patient-centric and fine-grained data access control in multi-owner settings. In: Security and Privacy in Communication Networks, pp. 89–106 (2010)
28. Lillibridge, M., Elnikety, S., Birrell, A., Burrows, M., Isard, M.: A cooperative internet backup scheme. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, pp. 3–3. USENIX Association, Berkeley, CA, USA (2003)
29. Lin, J.: Cloud Data Storage for Group Collaborations. Lecture Notes in Engineering and Computer Science, vol. 2183 (2010)
30. Maheshwari, U., Vingralek, R., Shapiro, W.: How to build a trusted database system on untrusted storage. In: Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation, OSDI'00, vol. 4, pp. 10–10. USENIX Association, Berkeley, CA, USA (2000)
31. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H., Baker, M.: The lockss peer-to-peer digital preservation system. ACM Trans. Comput. Syst. **23**, 2–50 (2005)
32. Mell, P., Grance, T.: The NIST definition of cloud computing. Tech. Rep., National Institute of Standards and Technology (2009)
33. Merkle, R.C.: Protocols for public key cryptosystems. IEEE Symposium on Security and Privacy, p. 122 (1980)
34. Miller, R.: Amazon addresses EC2 power outages. Data Center Knowledge **1** (2010)

35. Muthitacharoen, A., Morris, R., Gil, T.M., Chen, B.: Ivy: a read/write peer-to-peer file system. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, OSDI '02, pp. 31–44. ACM, New York, NY, USA (2002)
36. Naor, M., Rothblum, G.N.: The complexity of online memory checking. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS '05, pp. 573–584. IEEE Computer Society, Washington, DC, USA (2005)
37. Oprea, A., Reiter, M., Yang, K.: Space-efficient block storage integrity. In: Proceedings of the NDSS Symposium, Citeseer (2005)
38. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables. In: Proceedings of the 15th ACM conference on Computer and Communications Security, CCS '08, pp. 437–448. ACM, New York, NY, USA (2008)
39. Plank, J.S.: A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. Softw. Pract. Exp. **27**, 995–1012 (1997)
40. Schroeder, B., Gibson, G.A.: Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In: Proceedings of the 5th USENIX conference on File and Storage Technologies. USENIX Association, Berkeley, CA, USA (2007)
41. Schwarz, T., Miller, E.: Store, forget, and check: Using algebraic signatures to check remotely administered storage. In: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), p. 12 (2006). doi:10.1109/ICDCS.2006.80
42. Sebe, F., Domingo-Ferrer, J., Martinez-Balleste, A., Deswarte, Y., Quisquater, J.J.: Efficient remote data possession checking in critical information infrastructures. IEEE Trans. Knowl. Data Eng. **20**, 1034–1038 (2008)
43. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT '08, pp. 90–107. Springer, Berlin, Heidelberg (2008)
44. Shah, M., Swaminathan, R., Baker, M.: Privacy-preserving audit and extraction of digital contents. Tech. rep., Cryptology ePrint Archive, Report 2008/186, 2008. http://eprint.iacr.org (2008)
45. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R.: Auditing to keep online storage services honest. In: Proceedings of the 11th USENIX workshop on Hot Topics in Operating Systems, pp. 11:1–11:6. USENIX Association, Berkeley, CA, USA (2007)
46. Shoup, V.: On the security of a practical identification scheme. In: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques, EUROCRYPT'96, pp. 344–353. Springer, Berlin, Heidelberg (1996)
47. Smart, N.P., Warinschi, B.: Identity based group signatures from hierarchical identity-based encryption. In: Proceedings of the 3rd International Conference Palo Alto on Pairing-Based Cryptography, Pairing '09, pp. 150–170. Springer, Berlin, Heidelberg (2009)
48. Velte, T., Velte, A., Elsenpeter, R.: Cloud Computing: a Practical Approach, 1 edn., chap. 7. McGraw-Hill, New York, NY, USA (2010)
49. Wang, C., Ren, K., Lou, W., Li, J.: Toward publicly auditable secure cloud data storage services. IEEE Netw. **24**(4), 19–24 (2010). doi:10.1109/MNET.2010.5510914
50. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings of the 29th Conference on Information Communications, INFOCOM'10, pp. 525–533. IEEE Press, Piscataway, NJ, USA (2010)
51. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: Proceedings of the 14th European conference on Research in Computer Security, ESORICS'09, pp. 355–370. Springer, Berlin, Heidelberg (2009)
52. Yamamoto, G., Oda, S., Aoki, K.: Fast integrity for large data. In: Proceedings of the ECRYPT Workshop on Software Performance Enhancement for Encryption and Decryption, pp. 21–32. ECRYPT, Amsterdam, The Netherlands (2007)
53. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proceedings of the 29th Conference on Information Communications, pp. 534–542. IEEE Press (2010)
54. Yumerefendi, A.R., Chase, J.S.: Strong accountability for network storage. Trans. Storage **3** (2007)
55. Zeng, K.: Publicly verifiable remote data integrity. In: Proceedings of the 10th International Conference on Information and Communications Security, ICICS '08, pp. 419–434. Springer, Berlin, Heidelberg (2008)
56. Zhu, Y., Wang, H., Hu, Z., Ahn, G., Hu, H., Yau, S.: Cooperative provable data possession **0** (2010)