Chapter 2
TSAS: Third-Party Storage Auditing Service

Abstract In cloud storage systems, data owners host their data on cloud servers and users (data consumers) can access the data from cloud servers. Due to the data outsourcing, however, this new paradigm of data hosting service also introduces new security challenges, which requires an independent auditing service to check the data integrity in the cloud. In large-scale cloud storage systems, the data may be updated dynamically, so existing remote integrity checking methods served for static archive data are no longer applicable to check the data integrity. Thus, an efficient and secure dynamic auditing protocol is desired to convince data owners that the data is correctly stored in the cloud. In this chapter, we first introduce an auditing framework for cloud storage systems. Then, we describe Third-party Storage Auditing Scheme (TSAS), an efficient and privacy-preserving auditing protocol for cloud storage, which can also support data dynamic operations and batch auditing for both multiple owners and multiple clouds.

2.1 Introduction

Cloud storage is an important service of cloud computing [16], which allows data owners (owners) to move data from their local computing systems to the cloud. More and more owners start to store the data in the cloud [1]. However, this new paradigm of data hosting service also introduces new security challenges [24]. Owners would worry that the 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 cloud service providers would take [5, 11, 13, 14, 18]. Sometimes, cloud service providers might be dishonest. They could discard the data which has not been accessed or rarely accessed to save the storage space and claim that the data are still correctly stored in the cloud. Therefore, owners need to be convinced that the data are correctly stored in the cloud.

K. Yang and X. Jia, *Security for Cloud Storage Systems*, SpringerBriefs in Computer Science, DOI: 10.1007/978-1-4614-7873-7_2,

Traditionally, owners can check the data integrity based on two-party storage auditing protocols [6, 9, 12, 15, 17, 19, 20, 22, 28]. In cloud storage system, however, it is inappropriate to let either side of cloud service providers or owners conduct such auditing, because none of them could be guaranteed to provide unbiased auditing result. In this situation, *third party auditing* is a natural choice for the storage auditing in cloud computing. A third party auditor (auditor) that has expertise and capabilities can do a more efficient work and convince both cloud service providers and owners.

For the third party auditing in cloud storage systems, there are several important requirements which have been proposed in some previous works [25, 29]. The auditing protocol should have the following properties:

1. *Confidentiality* The auditing protocol should keep owner's data confidential against the auditor.
2. *Dynamic Auditing* The auditing protocol should support the dynamic updates of the data in the cloud.
3. *Batch Auditing* The auditing protocol should also be able to support the batch auditing for multiple owners and multiple clouds.

Recently, several remote integrity checking protocols were proposed to allow the auditor to check the data integrity on the remote server [2, 4, 8, 21, 26, 27, 30–32]. Table 2.1 gives the comparisons among some existing remote integrity checking schemes in terms of the performance, the privacy protection, the support of dynamic operations and the batch auditing for multiple owners and multiple clouds. Table 2.1 shows that many of the existing schemes are not privacy-preserving or cannot support the data dynamic operations, so that they cannot be applied to cloud storage systems.

In [27], the authors proposed a dynamic auditing protocol that can support the dynamic operations of the data on the cloud servers, but this method may leak the data content to the auditor because it requires the server to send the linear combinations of data blocks to the auditor. In [26], the authors extended their dynamic auditing scheme to be privacy-preserving and support the batch auditing for multiple owners.

Table 2.1 Comparison of remote integrity checking schemes

Scheme	Computation		Commu-nication	Privacy	Dynamic	Batch operation		Prob. of detection
	Sever	Verifier				Multi-owner	Multi-cloud	
PDP [2]	$O(t)$	$O(t)$	$O(1)$	Yes	No	No	No	$1-(1-\rho)^t$
CPDP [21]	$O(t+s)$	$O(t+s)$	$O(t+s)$	No	No	No	No	$1-(1-\rho)^{ts}$
DPDP [8]	$O(t \log n)$	$O(t \log n)$	$O(t \log n)$	No	No	No	No	$1-(1-\rho)^t$
Audit [27, 26]	$O(t \log n)$	$O(t \log n)$	$O(t \log n)$	Yes	Yes	Yes	No	$1-(1-\rho)^t$
IPDP [31, 32]	$O(ts)$	$O(t+s)$	$O(t+s)$	Yes	Yes	No	Yes	$1-(1-\rho)^{ts}$
TSAS	$O(ts)$	$O(t)$	$O(t)$	Yes	Yes	Yes	Yes	$1-(1-\rho)^{ts}$

n is the total number of data blocks of a file; t is the number of challenged data blocks in an auditing query

s is the number of sectors in each data block; ρ is the probability of block/sector corruption (suppose the probability of corruption is the same for the equal size of data block or sector)

However, due to the large number of data tags, their auditing protocols may incur a heavy storage overhead on the server. In [31], Zhu et al. proposed a cooperative provable data possession scheme that can support the batch auditing for multiple clouds and also extended it to support the dynamic auditing in [32]. However, their scheme cannot support the batch auditing for multiple owners. That is because parameters for generating the data tags used by each owner are different and thus they cannot combine the data tags from multiple owners to conduct the batch auditing. Another drawback is that their scheme requires an additional trusted organizer to send a commitment to the auditor during the multi-cloud batch auditing, because their scheme applies the mask technique to ensure the data privacy. However, such additional organizer is not practical in cloud storage systems. Furthermore, both Wang's schemes and Zhu's schemes incur heavy computation cost of the auditor, which makes the auditor a performance bottleneck.

In this chapter, we introduce Third-party Storage Auditing Service (TSAS) to ensure the data integrity in the cloud, where all the above listed requirements are satisfied. To solve the data privacy problem, the method in TSAS is to generate an encrypted proof with the challenge stamp by using the *Bilinearity* property of the bilinear pairing, such that the auditor cannot decrypt it but can verify the correctness of the proof. Without using the mask technique, it does not require any trusted organizer during the batch auditing for multiple clouds. On the other hand, the auditing protocol lets the server compute the proof as an intermediate value of the verification, such that the auditor can directly use this intermediate value to verify the correctness of the proof. Therefore, it can greatly reduce the computing loads of the auditor by moving it to the cloud server.

2.2 Preliminaries and Definitions

2.2.1 Bilinear Pairing

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

1. Bilinearity: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
2. 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 $\mathbb{G}_T$.
3. Computability: e can be computed in an efficient way.

Such a bilinear map is called a bilinear pairing.

2.2.2 Computational Bilinear Diffie-Hellman Assumption

The definition of the Computational Bilinear Diffie-Hellman (CBDH) assumption is defined as follows.

A challenger chooses a group $\mathbb{G}$ of prime order p according to the security parameter. Let $a, b, c \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}$. When given g, g^a, g^b, g^c, the adversary must compute $e(g, g)^{abc}$.

An algorithm $\mathcal{B}$ that outputs $e(g, g)^{abc}$ has advantage ε in solving CBDH in $\mathbb{G}$ if

$$|Pr[\mathcal{B}(g, g^a, g^b, g^c) = e(g, g)^{abc}]| \geq \varepsilon.$$

Definition 2.1 The (t, ε)-CBDH assumption holds if no t-time algorithm has a non-negligible probability ε in solving the CBDH problem.

2.2.3 Definition of System Model

As shown in Fig. 2.1, an auditing system for cloud storage normally involves data owners (owner), the cloud server (server) and the third party auditor (auditor). The owners create the data and host their data in the cloud. The cloud server stores the owners' data and provides the data access to users (data consumers). The auditor is a trusted third party that has expertise and capabilities to provide data storage auditing service for both the owners and servers. The auditor can be a trusted organization managed by the government, which can provide unbiased auditing result for both data owners and cloud servers.

Before describing the auditing protocol definition, some notations are defined as in Table 2.2.

Definition 2.2 (TSAS). TSAS is a collection of the following five algorithms: KeyGen, TagGen, Chall, Prove and Verify.

- **KeyGen**$(\lambda) \rightarrow (sk_h, sk_t, pk_t)$. This key generation algorithm takes no input other than the implicit security parameter λ. It outputs a secret hash key sk_h and a pair of secret-public tag key (sk_t, pk_t).
- **TagGen**$(M, sk_t, sk_h) \rightarrow T$. The tag generation algorithm takes as inputs an encrypted file M, the secret tag key sk_t and the secret hash key sk_h. For each

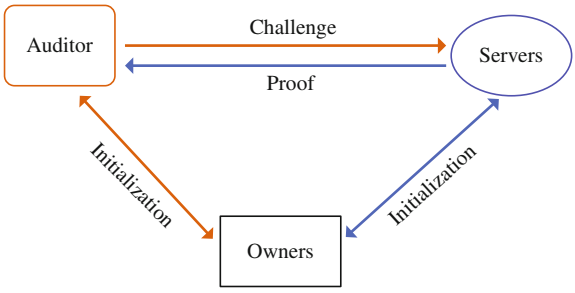

Fig. 2.1 System model of the data storage auditing

Table 2.2 Notations

Symbol	Physical meaning
sk_t	Secret tag key
pk_t	Public tag key
sk_h	Secret hash key
M	Data component
T	Set of data tags
n	Number of blocks in each component
s	Number of sectors in each data block
M_{info}	Abstract information of M
$\mathcal{C}$	Challenge generated by the auditor
$\mathcal{P}$	Proof generated by the server

data block m_i, it computes a data tag t_i based on sk_h and sk_t. It outputs a set of data tags $T = \{t_i\}_{i\in[1,n]}$.

- **Chall**$(M_{info}) \rightarrow \mathcal{C}$. The challenge algorithm takes as input the abstract information of the data M_{info} (e.g., file identity, total number of blocks, version number and timestamp etc.). It outputs a challenge $\mathcal{C}$.
- **Prove**$(M, T, \mathcal{C}) \rightarrow \mathcal{P}$. The prove algorithm takes as inputs the file M, the tags T and the challenge from the auditor $\mathcal{C}$. It outputs a proof $\mathcal{P}$.
- **Verify**$(\mathcal{C}, \mathcal{P}, sk_h, pk_t, M_{info}) \rightarrow 0/1$. The verification algorithm takes as inputs the $\mathcal{P}$ from the server, the secret hash key sk_h, the public tag key pk_t and the abstract information of the data M_{info}. It outputs the auditing result as 0 or 1.

2.2.4 Definition of Security Model

The auditor is assumed to be honest-but-curious. It performs honestly during the whole auditing procedure but it is curious about the received data. But the sever could be dishonest and may launch the following attacks:

1. *Replace Attack*. The server may choose another valid and uncorrupted pair of data block and data tag (m_k, t_k) to replace the challenged pair of data block and data tag (m_i, t_i), when it already discarded m_i or t_i.
2. *Forge Attack*. The server may forge the data tag of data block and deceive the auditor, if the owner's secret tag keys are reused for the different versions of data.
3. *Replay Attack*. The server may generate the proof from the previous proof or other information, without retrieving the actual owner's data.

2.3 An Efficient and Privacy-Preserving Auditing Protocol

In this section, we first present some techniques applied in the design of the auditing protocol. Then, we describe the algorithms and the detailed construction of the auditing protocol for cloud storage systems.

2.3.1 Overview

The main challenge in the design of data storage auditing protocol is the *data privacy problem* (i.e. the auditing protocol should protect the data privacy against the auditor.). This is because: (1) For public data, the auditor may obtain the data information by recovering the data blocks from the data proof. (2) For encrypted data, the auditor may obtain content keys somehow through any special channels and could be able to decrypt the data. To solve the data privacy problem, TSAS generates an encrypted proof with the challenge stamp by using the Bilinearity property of the bilinear pairing, such that the auditor cannot decrypt it. But the auditor can verify the correctness of the proof without decrypting it.

Although the auditor has sufficient expertise and capabilities to conduct the auditing service, the computing ability of an auditor is not as strong as cloud servers. Since the auditor needs to audit for many cloud servers and a large number of data owners, the auditor could be the performance bottleneck. TSAS lets the server compute the proof as an intermediate value of the verification (calculated by the challenge stamp and the linear combinations of data blocks), such that the auditor can use this intermediate value to verify the proof. Therefore, the computing loads of the auditor can be greatly reduced by moving it to the cloud server.

In TSAS, both the *Data Fragment Technique* and *Homomorphic Verifiable Tags* are applied to improve the performance. The data fragment technique can reduce number of data tags, such that it can reduce the storage overhead and improve the system performance. By using the homomorphic verifiable tags, no matter how many data blocks are challenged, the server only responses the sum of data blocks and the product of tags to the auditor, whose size is constant and equal to only one data block. Thus, it reduces the communication cost.

2.3.2 Algorithms for Auditing Protocol

Suppose a file F has m data components as $F = (F_1, \ldots, F_m)$. Each data component has its physical meanings and can be updated dynamically by the data owners. For public data components, the data owner does not need to encrypted it, but for private data component, the data owner needs to encrypt it with its corresponding key.

Each data component F_k is divided into n_k data blocks denoted as

$$F_k = (m_{k1}, m_{k2}, \ldots, m_{kn_k}).$$

Due to the security reason, the data block size should be restricted by the security parameter. For example, suppose the security level is set to be 160-bit (20-Byte), the data block size should be 20-Byte. A 50-KByte data component will be divided into 2,500 data blocks and generate 2,500 data tags, which incurs 50-KByte storage overhead.

By using the data fragment technique, each data block is further split into sectors. The sector size is restricted by the security parameter. One data tag is generated for each data block which consists of s sectors, such that it can reduce the number of data tags. In the same example above, a 50-KByte data component only incurs $50/s$ KByte storage overhead. In real storage systems, the data block size can be various. That is different data blocks could have different number of sectors. For example, if a data block m_i will be frequently read, then s_i could be large, but for those frequently updated data blocks, s_i could be relatively small.

For simplicity, the construction only considers one data component and constant number of sectors for each data block. Suppose there is a data component M, which is divided into n data blocks and each data block is further split into s sectors. For data blocks that have different number of sectors, it first selects the maximum number of sectors s_{max} among all the sector numbers s_i. Then, for each data block m_i with s_i sectors, $s_i < s_{max}$, it can simply consider that the data block m_i has s_{max} sectors by setting $m_{ij} = 0$ for $s_i < j \leq s_{max}$. Because the size of each sector is constant and equal to the security parameter p, the number of data blocks can be calculated as $n = \frac{sizeof(M)}{s \cdot \log p}$. The encrypted data component is denoted as $M = \{m_{ij}\}_{i \in [1,n], j \in [1,s]}$.

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be the multiplicative groups with the same prime order p and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be the bilinear map. Let g_1 and g_2 be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let $h : \{0, 1\}^* \to \mathbb{G}_1$ be a keyed secure hash function that maps the M_{info} to a point in $\mathbb{G}_1$.

The storage auditing protocol consists of the following algorithms:

- **KeyGen**$(\lambda) \to (pk_t, sk_t, sk_h)$. The key generation algorithm takes no input other than the implicit security parameter λ. It chooses two random number $sk_t, sk_h \in \mathbb{Z}_p$ as the secret tag key and the secret hash key. It outputs the public tag key as $pk_t = g_2^{sk_t} \in \mathbb{G}_2$, the secret tag key sk_t and the secret hash key sk_h.
- **TagGen**$(M, sk_t, sk_h) \to T$. The tag generation algorithm takes each data component M, the secret tag key sk_t and the secret hash key sk_h as inputs. It first chooses s random values $x_1, x_2, \ldots, x_s \in \mathbb{Z}_p$ and computes $u_j = g_1^{x_j} \in \mathbb{G}_1$ for all $j \in [1, s]$. For each data block $m_i (i \in [1, n])$, it computes a data tag t_i as

$$t_i = (h(sk_h, W_i) \cdot \prod_{j=1}^{s} u_j^{m_{ij}})^{sk_t},$$

where $W_i = FID||i$ (the "||" denotes the concatenation operation), in which *FID* is the identifier of the data and i represents the block number of m_i. It outputs the set of data tags $T = \{t_i\}_{i\in[1,n]}$.

- **Chall**$(M_{info}) \rightarrow \mathcal{C}$. The challenge algorithm takes the abstract information of the data M_{info} as the input. It selects some data blocks to construct the *Challenge Set* Q and generates a random number $v_i \in \mathbb{Z}_p^*$ for each chosen data block $m_i (i \in Q)$. Then, it computes the challenge stamp $R = (pk_t)^r$ by randomly choosing a number $r \in \mathbb{Z}_p^*$. It outputs the challenge as $\mathcal{C} = (\{i, v_i\}_{i\in Q}, R)$.
- **Prove**$(M, T, \mathcal{C}) \rightarrow \mathcal{P}$. The prove algorithm takes as inputs the data M and the received challenge $\mathcal{C} = (\{i, v_i\}_{i\in Q}, R)$. The proof consists of the *tag proof TP* and the *data proof DP*. The *tag proof* is generated as

$$TP = \prod_{i\in Q} t_i^{v_i}.$$

To generate the *data proof*, it first computes the sector linear combination of all the challenged data blocks MP_j for each $j \in [1, s]$ as

$$MP_j = \sum_{i\in Q} v_i \cdot m_{ij}.$$

Then, it generates the *data proof DP* as

$$DP = \prod_{j=1}^{s} e(u_j, R)^{MP_j}.$$

It outputs the proof $\mathcal{P} = (TP, DP)$.

- **Verify**$(\mathcal{C}, \mathcal{P}, sk_h, pk_t, M_{info}) \rightarrow 0/1$. The verification algorithm takes as inputs the challenge $\mathcal{C}$, the proof $\mathcal{P}$, the secret hash key sk_h, the public tag key pk_t and the abstract information of the data component. It first computes the identifier hash values $h(sk_h, W_i)$ of all the challenged data blocks and computes the challenge hash H_{chal} as

$$H_{chal} = \prod_{i\in Q} h(sk_h, W_i)^{rv_i}.$$

It then verifies the *proof* from the server by the following verification equation:

$$DP \cdot e(H_{chal}, pk_t) = e(TP, g_2^r). \tag{2.1}$$

If the above verification Eq. 2.1 holds, it outputs 1. Otherwise, it outputs 0.

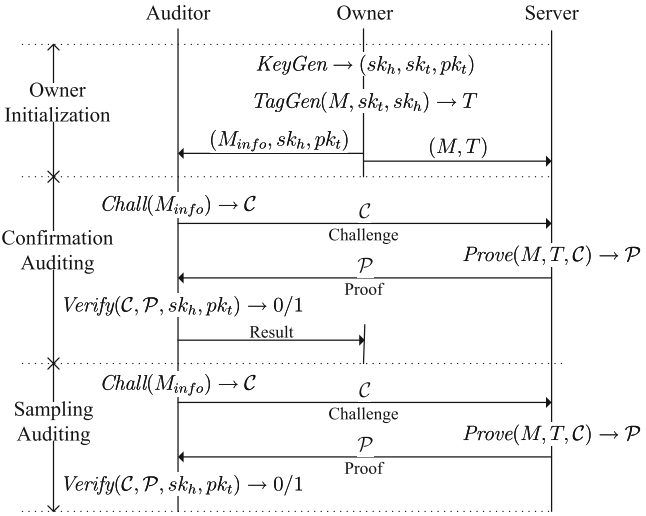

Fig. 2.2 Framework of the privacy-preserving auditing protocol

2.3.3 Construction of the Privacy-Preserving Auditing Protocol

As illustrated in Fig. 2.2, the storage auditing protocol consists of three phases: *Owner Initialization*, *Confirmation Auditing* and *Sampling Auditing*. During the system initialization, the owner generates the keys and the tags for the data. After storing the data on the server, the owner asks the auditor to conduct the confirmation auditing to make sure that their data is correctly stored on the server. Once confirmed, the owner can choose to delete the local copy of the data. Then, the auditor conducts the sampling auditing periodically to check the data integrity.

2.3.3.1 Owner Initialization

The owner runs the key generation algorithm KeyGen to generate the secret hash key sk_h, the pair of secret-public tag key (sk_t, pk_t). Then, it runs the tag generation algorithm TagGen to compute the data tags. After all the data tags are generated, the owner sends each data component $M = \{m_i\}_{i\in[1,n]}$ and its corresponding data tags $T = \{t_i\}_{i\in[1,n]}$ to the server together with the set of parameters $\{u_j\}_{j\in[1,s]}$. The owner then sends the public tag key pk_t, the secret hash key sk_h and the abstract information of the data M_{info} to the auditor, which includes the data identifier *FID*, the total number of data blocks n.

2.3.3.2 Confirmation Auditing

In the auditing construction, the auditing protocol only involves two-way communication: Challenge and Proof. During the confirmation auditing phase, the owner requires the auditor to check whether the owner's data is correctly stored on the server. The auditor conducts the confirmation auditing phase as

1. The auditor runs the challenge algorithm Chall to generate the challenge $\mathcal{C}$ for all the data blocks in the data component and sends the $\mathcal{C} = (\{i, v_i\}_{i \in Q}, R)$ to the server.
2. Upon receiving the challenge $\mathcal{C}$ from the auditor, the server runs the prove algorithm Prove to generate the proof $\mathcal{P} = (TP, DP)$ and sends it back to the auditor.
3. When the auditor receives the proof $\mathcal{P}$ from the server, it runs the verification algorithm Verify to check the correctness of $\mathcal{P}$ and extract the auditing result.

The auditor then sends the auditing result to the owner. If the result is true, the owner is convinced that its data is correctly stored on the server and it may choose to delete the local version of the data.

2.3.3.3 Sampling Auditing

The auditor will carry out the sampling auditing periodically by challenging a sample set of data blocks. The frequency of taking auditing operation depends on the service agreement between the data owner and the auditor (and also depends on how much trust the data owner has over the server). Similar to the confirmation auditing in Phase 2, the sampling auditing procedure also contains two-way communication as illustrated in Fig. 2.2.

Suppose each sector will be corrupted with a probability of ρ on the server. For a sampling auditing involved with t challenged data blocks, the probability of detection can be calculated as

$$Pr(t, s) = 1 - (1 - \rho)^{t \cdot s}.$$

That is this t-block sampling auditing can detect any data corruption with a probability of $Pr(t, s)$.

2.3.4 Correctness Proof

The correctness of the privacy-preserving auditing protocol is concluded as the following theorem:

Theorem 2.1 *In the proposed auditing protocol, the server passes the audit iff all the chosen data blocks and the data tags are correctly stored.*

Proof First, let's prove that if all the chosen data and the corresponding data tags are stored correctly on the server, the server will pass the auditing via the challenge-response protocol. The verification equation can be rewritten in details as the following:

$$
\begin{aligned}
& DP \cdot e(H_{chal}, pk_t) \qquad (2.2)\\
=& \prod_{j=1}^{s} e(u_j, R)^{MP_j} \cdot e(\prod_{i\in Q} h(sk_h, W_i)^{rv_i}, pk_t)\\
=& \prod_{j=1}^{s} e(u_j, pk_t)^{r\prod_{i\in Q} v_i m_{ij}} \cdot e(\prod_{i\in Q} h(sk_h, W_i)^{rv_i}, pk_t)\\
=& \prod_{i\in Q} e(\prod_{j=1}^{s} u_j^{m_{ij}}, pk_t^{rv_i}) e(h(sk_h, W_i), pk_t^{rv_i})\\
=& \prod_{i\in Q} e(\prod_{j=1}^{s} h(sk_h, W_i) u_j^{m_{ij}}, pk_t^{rv_i})\\
=& \prod_{i\in Q} e(\prod_{j=1}^{s} (h(sk_h, W_i) u_j^{m_{ij}})^{sk_t}, g_2^{rv_i})\\
=& \prod_{i\in Q} e(t_i, g_2^{rv_i})\\
=& e(TP, g_2^{r})
\end{aligned}
$$

From Eq. 2.2, we can say that the server can pass the auditing, if the data blocks and the data tags are stored correctly on the server. However, if any of the challenged data block or data tag is corrupted or modified, the verification equation will not hold and the server cannot pass the audit.

2.4 Secure Dynamic Auditing

In cloud storage systems, the data owners will dynamically update their data. As an auditing service, the auditing protocol should be designed to support the dynamic data, as well as the static archive data. However, the dynamic operations may make the auditing protocols insecure. Specifically, the server may conduct two following attacks: (1) *Replay Attack* The server may not update correctly the owner's data on the server and may use the previous version of the data to pass the auditing. (2) *Forge Attack* When the data owner updates the data to the current version, the server may get enough information from the dynamic operations to forge the data tag. If the server could forge the data tag, it can use any data and its forged data tag to pass the auditing.

2.4.1 Solution of Dynamic Auditing

To prevent the replay attack, an *Index Table* (ITable) is introduced to record the abstract information of the data. The ITable consists of four components: *Index*, B_i, V_i and T_i. The *Index* denotes the current block number of data block m_i in the data component M. B_i denotes the original block number of data block m_i and V_i denotes the current version number of data block m_i. T_i is the timestamp used for generating the data tag.

This ITable is created by the owner during the owner initialization and managed by the auditor. When the owner completes the data dynamic operations, it sends an update message to the auditor for updating the ITable which is stored on the auditor. After the confirmation auditing, the auditor sends the result to the owner for the confirmation that the owner's data on the server and the abstraction information on the auditor are both up-to-date. This completes the data dynamic operation.

To deal with the forge attack, it can modify the tag generation algorithm TagGen. Specifically, when generating the data tag t_i for the data block m_i, the owners insert all the abstract information into the data tag by setting $W_i = FID||i||B_i||V_i||T_i$, such that the server cannot get enough information to forge the data tag from dynamic operations.

2.4.2 Algorithms and Constructions for Dynamic Auditing

The dynamic auditing protocol consists of four phases: Owner Initialization, Confirmation Auditing, Sampling Auditing and Dynamic Auditing.

The first three phases are similar to the privacy-preserving auditing protocol as described in the above section. The only differences are the tag generation algorithm TagGen and the ITable generation during the owner initialization phase. Here, Fig. 2.3 only illustrates the dynamic auditing phase, which contains three steps: Data Update, Index Update and Update Confirmation.

2.4.2.1 Data Update

There are three types of data update operations that can be used by the owner: Modification, Insertion and Deletion. For each update operation, there is a corresponding algorithm in the dynamic auditing to process the operation and facilitate the future auditing, defined as follows.

- **Modify**$(m_i^*, sk_t, sk_h) \rightarrow (Msg_{modify}, t_i^*)$. The modification algorithm takes as inputs the new version of data block m_i^*, the secret tag key sk_t and the secret hash key sk_h. It generates a new version number V_i^*, new timestamp T_i^* and calls the TagGen to generate a new data tag t_i^* for data block m_i^*. It outputs the new tag t_i^* and the update message $Msg_{modify} = (i, B_i, V_i^*, T_i^*)$. Then, it sends the new

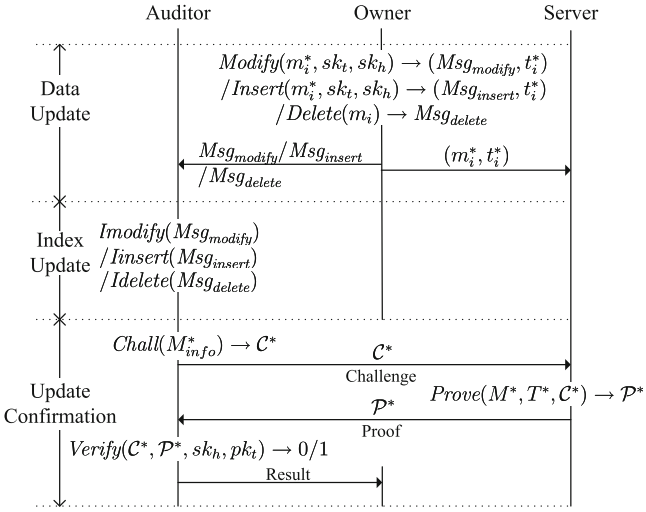

Fig. 2.3 Framework of auditing for dynamic operations

pair of data block and tag (m_i^*, t_i^*) to the server and sends the update message Msg_{modify} to the auditor.

- **Insert**$(m_i^*, sk_t, sk_h) \rightarrow (Msg_{insert}, t_i^*)$. The insertion algorithm takes as inputs the new data block m_i^*, the secret tag key sk_t and the secret hash key sk_h. It inserts a new data block m_i^* before the ith position. It generates an original number B_i^*, a new version number V_i^* and a new timestamp T_i^*. Then, it calls the TagGen to generate a new data tag t_i^* for the new data block m_i^*. It outputs the new tag t_i^* and the update message $Msg_{insert} = (i, B_i^*, V_i^*, T_i^*)$. Then, it inserts the new pair of data block and tag (m_i^*, t_i^*) on the server and sends the update message Msg_{insert} to the auditor.
- **Delete**$(m_i) \rightarrow Msg_{delete}$. The deletion algorithm takes as input the data block m_i. It outputs the update message $Msg_{delete} = (i, B_i, V_i, T_i)$. It then deletes the pair of data block and its tag (m_i, t_i) from the server and sends the update message Msg_{delete} to the auditor.

2.4.2.2 Index Update

Upon receiving the three types of update messages, the auditor calls three corresponding algorithms to update the ITable. Each algorithm is designed as follows.

- **IModify**(Msg_{modify}). The index modification algorithm takes the update message Msg_{modify} as input. It replaces the version number V_i by the new one V_i^* and modifies T_i by the new timestamp T_i^*.

- **IInsert**(Msg_{insert}). The index insertion algorithm takes as input the update message Msg_{insert}. It inserts a new record (i, B_i^*, V_i^*, T_i^*) in ith position in the ITable. It then moves the original ith record and other records after the i-th position in the previous ITable backward in order, with the index number increased by one.
- **IDelete**(Msg_{delete}). The index deletion algorithm takes as input the update message Msg_{delete}. It deletes the ith record (i, B_i, V_i, T_i) in the ITable and all the records after the ith position in the original ITable moved forward in order, with the index number decreased by one.

Table 2.3 shows the change of ITable according to the different type of data update operation. Table 2.3(a) describe the initial table of the data $M = \{m_1, m_2, \ldots, m_n\}$ and Table 2.3(b) describes the ITable after m_2 is updated. Table 2.3(c) is the ITable after a new data block is insert before m_2 and Table 2.3(d) shows the ITable after m_2 is deleted.

Table 2.3 ITable of the abstract information of data M

Index	B_i	V_i	T_i
Initial abstract information of M			
1	1	1	T_1
2	2	1	T_2
3	3	1	T_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n	1	T_n
After modifying m_2, V_2 *and* T_2 *are updated*			
1	1	1	T_1
2	2	2	T_2^*
3	3	1	T_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n	1	T_n
After inserting before m_2, *all items before* m_2 *move backward with the index increased by* 1			
1	1	1	T_1
2	$n+1$	1	T_{n+1}
3	2	1	T_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n+1$	n	1	T_n
After deleting m_2, *all items after* m_2 *move forward with the index decreased by* 1			
1	1	1	T_1
2	3	1	T_3
3	4	1	T_4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	n	1	T_n

2.4.2.3 Update Confirmation

After the auditor updates the ITable, it conducts a confirmation auditing for the updated data and sends the result to the owner. Then, the owner can choose to delete the local version of data according to the update confirmation auditing result.

2.5 Batch Auditing for Multi-Owner and Multi-Cloud

Data storage auditing is a significant service in cloud computing which helps the owners check the data integrity on the cloud servers. Due to the large number of data owners, the auditor may receive many auditing requests from multiple data owners. In this situation, it would greatly improve the system performance, if the auditor could combine these auditing requests together and only conduct the batch auditing for multiple owners simultaneously. The previous work [31] cannot support the batch auditing for multiple owners. That is because parameters for generating the data tags used by each owner are different and thus the auditor cannot combine the data tags from multiple owners to conduct the batch auditing.

On the other hand, some data owners may store their data on more than one cloud servers. To ensure the owner's data integrity in all the clouds, the auditor will send the auditing challenges to each cloud server which hosts the owner's data, and verify all the proofs from them. To reduce the computation cost of the auditor, it is desirable to combine all these responses together and do the batch verification.

In the previous work [31], the authors proposed a cooperative provable data possession for integrity verification in multi-cloud storage. In their method, the authors apply the mask technique to ensure the data privacy, such that it requires an additional trusted organizer to send a commitment to the auditor during the commitment phase in multi-cloud batch auditing. The TSAS applies the encryption method with the Bilinearity property of the bilinear pairing to ensure the data privacy, rather than the mask technique. Thus, the multi-cloud batch auditing protocol does not have any commitment phase, such that it does not require any additional trusted organizer.

2.5.1 Algorithms for Batch Auditing for Multi-Owner and Multi-Cloud

Let O be the set of owners and S be the set of cloud servers. The batch auditing for multi-owner and multi-cloud can be constructed as follows.

2.5.1.1 Owner Initialization

Each owner $O_k (k \in O)$ runs the key generation algorithm KeyGen to generate the pair of secret-public tag key $(sk_{t,k}, pk_{t,k})$ and a set of secret hash key $\{sk_{h,kl}\}_{l \in S}$. That is, for different cloud servers, the owner has different secret hash keys. Each data component is denoted as M_{kl}, which means that this data component is owned by the owner O_k and stored on the cloud server S_l. Suppose the data component M_{kl} is divided into n_{kl} data blocks and each data block is further split into s sectors. (Here each data block is assumed to be further split into the same number of sectors. It can also use the technique proposed in Sect. 2.3.2 to deal with the situation that each data blocks is split into different number of sectors.) The owner O_k runs the tag generation algorithm TagGen to generate the data tags $T_{kl} = \{t_{kl,i}\}_{i \in [1,n_{kl}]}$ as

$$t_{kl,i} = (h(sk_{h,kl}, W_{kl,i}) \cdot \prod_{j=1}^{s} u_{k,j}^{m_{kl,ij}})^{sk_{t,k}}.$$

where $W_{kl,i} = FID_{kl}||i||B_{kl,i}||V_{kl,i}||T_{kl,i}$.

After all the data tags are generated, each owner $O_k (k \in O)$ sends the data component $M_{kl} = \{m_{kl,ij}\}_{i \in [1,n_{kl}], j \in [1,s]}^{k \in O, l \in S}$ and the data tags $T_{kl} = \{t_{kl,i}\}_{i \in [1,n_{kl}]}^{k \in O, l \in S}$ to the corresponding server S_l. Then, it sends the public tag key $pk_{t,k}$, the set of secret hash key $\{sk_{hl,k}\}_{l \in S}$, the abstract information of data $\{M_{info,kl}\}_{k \in O, l \in S}$ to the auditor.

2.5.1.2 Batch Auditing for Multi-Owner and Multi-Cloud

Let O_{chal} and S_{chal} denote the involved set of owners and cloud servers involved in the batch auditing respectively. The batch auditing also consists of three steps: Batch Challenge, Batch Proof and Batch Verification.

- **Step 1: Batch Challenge**
  During this step, the auditor runs the batch challenge algorithm BChall to generate a batch challenge $\mathcal{C}$ for a set of challenged owners O_{chal} and a set of clouds S_{chal}. The batch challenge algorithm is defined as follows.

  – **BChall**$(\{M_{info,kl}\}_{k \in O, l \in S}) \rightarrow \mathcal{C}$. The batch challenge algorithm takes all the abstract information as input. It selects a set of owners O_{chal} and a set of cloud servers S_{chal}. For each data owner $O_k (k \in O_{chal})$, it chooses a set of data blocks as the challenged subset Q_{kl} from each server $S_l (l \in S_{chal})$. It then generates a random number $v_{kl,i}$ for each chosen data block $m_{kl,i} (k \in O_{chal}, l \in S_{chal}, i \in Q_{kl})$. It also chooses a random number $r \in \mathbb{Z}_p^*$ and computes the set of challenge stamp $\{R_k\}_{k \in O_{chal}} = pk_{t,k}^r$. It outputs the challenge as

  $$\mathcal{C} = (\{\mathcal{C}_l\}_{l \in S_{chal}}, \{R_k\}_{k \in O_{chal}}),$$

  where $\mathcal{C}_l = \{(k, l, i, v_{kl,i})\}_{k \in O_{chal}}$.

Then, the auditor sends each $\mathcal{C}_l$ to each cloud server $S_l(l \in S_{chal})$ together with the challenge stamp $\{R_k\}_{k \in O_{chal}}$.

- **Step 2: Batch Proof**
  Upon receiving the challenge, each server $S_l(l \in S_{chal})$ generates a proof $\mathcal{P}_l = (TP_l, DP_l)$ by using the following batch prove algorithm BProve and sends the proof $\mathcal{P}_l$ to the auditor.

  – **BProve**$(\{M_{kl}\}_{k \in O_{chal}}, \{T_{kl}\}_{k \in O_{chal}}, \mathcal{C}_l, \{R_k\}_{k \in O_{chal}}) \rightarrow \mathcal{P}_l$. The batch prove algorithm takes as inputs the data $\{M_{kl}\}_{k \in O_{chal}}$, the data tags $\{T_{kl}\}_{k \in O_{chal}}$, the received challenge $\mathcal{C}_l$ and the challenge stamp $\{R_k\}_{k \in O_{chal}}$. It generates the *tag proof* TP_l as

$$TP_l = \prod_{k \in O_{chal}} \prod_{i \in Q_{kl}} t_{kl,i}^{v_{kl,i}}.$$

  Then, for each $j \in [1, s]$, it computes the sector linear combination $MP_{kl,j}$ of all the chosen data blocks of each owner $O_k(k \in O_{chal})$ as

$$MP_{kl,j} = \sum_{i \in Q_{kl}} v_{kl,i} \cdot m_{kl,ij},$$

  and generates the *data proof* DP_l as

$$DP_l = \prod_{j=1}^{s} \prod_{k \in O_{chal}} e(u_{k,j}, R_k)^{MP_{kl,j}}.$$

  It outputs the proof $\mathcal{P}_l = (TP_l, DP_l)$.

- **Step 3: Batch Verification**
  Upon receiving all the proofs from the challenged servers, the auditor runs the following batch verification algorithm BVerify to check the correctness of the proofs.

  – **BVerify**$(\mathcal{C}, \{\mathcal{P}_l\}, \{sk_{h,lk}\}, \{pk_{t,k}\}, \{M_{info,kl}\}) \rightarrow 0/1$. The batch verification algorithm takes as inputs the challenge $\mathcal{C}$, the proofs $\{\mathcal{P}_l\}_{l \in S_{chal}}$, the set of secret hash keys $\{sk_{h,kl}\}_{k \in O_{chal}, l \in S_{chal}}$, the public tag keys $\{pk_{t,k}\}_{k \in O_{chal}}$ and the abstract information of the challenged data blocks $\{M_{info,kl}\}_{k \in O_{chal}, l \in S_{chal}}$. For each owner $O_k(k \in O_{chal})$, it computes the set of identifier hash values $\{h(sk_{h,kl}, W_{kl,i})\}_{l \in S_{chal}, i \in Q_{kl}}$ for all the chosen data blocks from each challenged server, and use these hash values to compute a *challenge hash* $H_{chal,k}$ as

$$H_{chal,k} = \prod_{l \in S_{chal}} \prod_{i \in Q_{kl}} h(sk_{h,kl}, W_{kl,i})^{rv_{kl,i}}.$$

  When finished the calculation of all the data owners' challenge hash $\{H_{chal,k}\}_{k \in O_{chal}}$, it verifies the proofs by the batch verification equation as

$$\prod_{l \in S_{chal}} DP_l = \frac{e(\prod_{l \in S_{chal}} TP_l, g_2^r)}{\prod_{k \in O_{chal}} e(H_{chal,k}, pk_{t,k})}. \tag{2.3}$$

If Eq. 2.3 is true, it outputs 1. Otherwise, it outputs 0.

2.5.2 Correctness Proof

The correctness of the batch auditing protocol is concluded as the following theorem:

Theorem 2.2 *In the multi-owner multi-cloud batch auditing protocol, all the challenged servers pass the audit iff all the chosen data blocks and the data tags from all the owners are correctly stored.*

Proof If the data blocks and the data tags from all the owners are stored correctly on the challenged servers, the right part of the batch verification equation can be rewritten as

$$\begin{aligned}
&\frac{e(\prod_{l \in S_{chal}} TP_l, g_2^r)}{\prod_{k \in O_{chal}} e(H_{chal,k}, pk_{t,k})} \\
&= \prod_{k \in O_{chal}} \prod_{l \in S_{chal}} \prod_{i \in Q_{kl}} \frac{e(t_{kl,i}^{v_{kl,i}}, g_2^r)}{e(h(sk_{h,kl}, W_{kl,i})^{rv_{kl,i}}, pk_{t,k})} \\
&= \prod_{k \in O_{chal}} \prod_{l \in S_{chal}} \prod_{i \in Q_{kl}} \frac{e((h(sk_{h,kl}, W_{kl,i}) \cdot \prod_{j=1}^{s} u_{k,j}^{m_{kl,ij}})^{sk_{t,k} v_{kl,i}}, g_2^r)}{e(h(sk_{h,kl}, W_{kl,i})^{v_{kl,i} sk_{t,k}}, g_2^r)} \\
&= \prod_{l \in S_{chal}} \prod_{j=1}^{s} \prod_{k \in O_{chal}} e(u_{k,j}^{\sum_{i \in Q_{kl}} v_{kl,i} m_{kl,ij}}, pk_{t,k}^r) \\
&= \prod_{l \in S_{chal}} \prod_{j=1}^{s} \prod_{k \in O_{chal}} e(u_{k,j}, pk_{t,k}^r)^{MP_{kl,j}} \\
&= \prod_{l \in S_{chal}} DP_l
\end{aligned} \tag{2.4}$$

It shows that the batch verification equation can hold, if all the data blocks and tags are correctly stored on the challenged servers. Otherwise, the batch verification Eq. 2.4 does not hold. That is, the server fail to pass the audit, if any of the chosen data block or data tag is corrupted or modified.

2.6 Security Analysis

In this section, we first prove that the auditing protocols are provably secure under the security model. Then, we prove that the auditing protocols can also guarantee the data privacy. Finally, we prove that the auditing system is an interactive proof system.

2.6.1 Provably Secure Under the Security Model

The security proofs of the dynamic auditing protocol and batch auditing protocol are similar. Here, we only demonstrate the security proof for the dynamic auditing protocol, as concluded in the following theorems.

Theorem 2.3 *The dynamic auditing protocol can resist the Replace Attack from the server.*

Proof If any of the challenged data blocks m_l or its data tag t_l is corrupted or not up-to-date on the server, the server cannot pass the auditing because the verification equation cannot hold. The server may conduct the replace attack to try to pass the audit. It uses another pair of data block and data tag (m_k, t_k) to replace the chosen one (m_l, t_l). Then, the *data proof* DP^* becomes

$$DP^* = \prod_{j=1}^{s} e(u_j, R)^{MP_j^*},$$

where each MP_j^* can be expressed as

$$MP_j^* = v_l \cdot m_{kj} + \sum_{i \in Q, i \neq l} v_i \cdot m_{ij}.$$

The *tag proof* TP^* can be calculated as

$$TP^* = t_k^{v_l} \cdot \prod_{i \in Q, i \neq l} t_i^{v_i}.$$

Then, the left hand of the verification equation can be transformed to

$$DP^* \cdot e(H_{chal}, pk_t) = e\left(\left(\frac{h(sk_h, W_l)}{h(sk_h, W_k)}\right)^{v_l sk_t} \cdot TP^*, g_2^r\right). \tag{2.5}$$

Due to the collision resistance of hash function, $h(sk_h, W_l)/h(sk_h, W_k)$ cannot be equal to 1 in the random oracle model and thus the verification equation does not

hold, such that the proof from the server cannot pass the auditing. Therefore, the dynamic auditing protocol can resist the replace attack. □

Theorem 2.4 *The dynamic auditing protocol can resist the Forge Attack.*

Proof The server can forge the tag without knowing the secret tag key and the secret hash key, when the same hash value and the secret tag key are used for two times. For example, suppose the same hash value $h(sk_h, i)$ and the secret tag key sk_t are used for generating the data tags for two different data blocks m_i and m_i'. The two tags can be expressed as $t_i = (h(sk_h, i) \cdot g^{m_i})^{sk_t}$ and $t_i' = (h(sk_h, i) \cdot g^{m_i'})^{sk_t}$. The server can first compute

$$t_i \cdot t_i'^{-1} = g^{(m_i - m_i')sk_t}$$

and get

$$g^{sk_t} = (t_i \cdot t_i'^{-1})^{\frac{1}{m_i - m_i'}}$$

by using the Euclidean algorithm $gcd(m_i - m_i', p)$. Then, for any pair of data block and tag (m_k, t_k), the server can easily compute

$$h(sk_h, k)^{sk_t} = \frac{t_k}{(g^{sk_t})^{m_k}}.$$

Therefore, for any data block m_k^*, the server can forge its tag t_k^* by

$$t_k^* = t_k \cdot (t_i \cdot t_i'^{-1})^{\frac{m_k^* - m_k}{m_i - m_i'}}.$$

The above equation shows that if the same value and the secret tag key is reused for two times, the server can forge the tag and deceive the auditor.

In the dynamic auditing protocol, the server cannot forge the tags and pass the audit successfully. That is because there is no chance to get the same hash value from the abstract information of data blocks in the dynamic auditing protocol. For each data block m_i, the abstract information contains the original block number B_i, the version number V_i and the timestamp T_i. Due to the different value of timestamp T_i for each data block, it is impossible for a hash function to get two same hash values from different abstract information in the random oracle model. □

Theorem 2.5 *The dynamic auditing protocol can resist the Replay Attack.*

Proof On one hand, in the dynamic auditing protocol, there is a challenge stamp R in each challenge-response auditing process. Because different audit processes have different challenge stamps, the server cannot only use the previous proof $\mathcal{P}$ to generate the new proof and pass the auditing without retrieving the challenged data blocks and data tags.

On the other hand, in the dynamic auditing protocol, a timestamp is introduced in the ITable, which is used to generating the tags. For different version of data blocks or

new inserted data blocks, the timestamps used to generate the data tags are different. The update operations will not allow the server to launch the replay attack based on the same hash values of the abstract information. □

2.6.2 Privacy-Preserving Guarantee

The data privacy is an important requirement in the design of auditing protocol in cloud storage systems. The proposed auditing protocols are privacy-preserving as stated in the follow theorem.

Theorem 2.6 *In the proposed auditing protocols, neither the server nor the auditor can obtain any information about the data and the secret tag key during the auditing procedure.*

Proof Because the data are encrypted by owners, it is obvious that the server cannot decrypt the data without the owners' secret key. The secret hash key and the secret tag key are kept secret to the server and the server cannot deduce them based on the received information during the auditing procedure. Therefore, the data and the secret tag key are confidential against the server in the auditing protocols.

On the auditor side, it can only get the product of all the challenged data tags from the *tag proof TP*. The *data proof* in the auditing protocol is in an encrypted way by the exponentiate on the challenge stamps R. It is a discrete logarithm problem to get the linear combinations of the chosen data sectors $\{MP_j\}_{j\in[1,s]}$ from the *data proof DP*, which is similar to obtain the secret tag key sk_t from g^{sk_t}. Hence, the auditor cannot get any information about the data and the secret tag key from the proof generated by the server in the auditing protocol. For the dynamical index update, the index update messages do not contain any information about the secret tag key and the content of the data, and thus the auditor cannot obtain any information about the data content from the dynamic operations. □

2.6.3 Proof of the Interactive Proof System

In this section, we first recall the definition of the interactive proof system and the zero-knowledge in [10] as follows.

Definition 2.3 A system is a zero-knowledge interactive system if the completeness, soundness and zero-knowledge hold.

Then, we prove that the dynamic auditing system is an Interactive Proof system, which provides zero-knowledge proof to ensure both the data integrity and the data confidentiality in the cloud.

Theorem 2.7 *The storage auditing system is a zero-knowledge interactive proof system under the CBDH assumption in random oracle model.*

Proof First, we prove that the TSAS system is an interactive proof system. According to the definition of an interactive proof system in [10], an interactive proof system should satisfy the following two features:

1. *Completeness* The storage auditing scheme is complete if the verification algorithm accepts the response when the server returns a valid response. This can be proved as the correctness proof in Theorem 2.1 and Theorem 2.2.
2. *Soundness* The storage auditing scheme is sound if any cheating server that convinces the auditor that it is storing a file is actually storing that file. In other words, the server cannot conduct the forge attack successfully, which is proved by Theorem 2.4..

Then, we prove that the TSAS is zero-knowledge as follows.

Zero-knowledge Proof The only information can be revealed in each auditing procedure is the data proof DP and the tag proof TP. We construct a simulator S that is not interacted with the protocol $\mathcal{A}$ as follows.

Given the public tag key pk_t, the public parameter g_2 and the challenge hash H_{chal}, the simulator chooses a random $TP \in \mathbb{G}_1$ as the tag proof and a random number $r \in \mathbb{Z}_p$, then the data proof DP can be simulated as

$$DP = \frac{e(TP, g_2^r)}{e(H_{chal}, pk_t)} \in G_T. \tag{2.6}$$

Such randomly generated pair of (TP, DP) are computationally indistinguishable from the pair of proof generated according to the auditing protocol. Thus, the auditing protocol $\mathcal{A}$ is a zero-knowledge protocol. This completes the proof of the theorem. □

2.7 Performance Analysis

Storage auditing is a very resource demanding service in terms of computational resource, communication cost and memory space. In this section, we give the communication cost comparison and computation complexity comparison between the TSAS and two existing works: the Audit protocol proposed by Wang et al. [26, 27] and the IPDP proposed by Zhu et al. [31, 32].

Table 2.4 Storage overhead comparison for $|M|$-bit data

Scheme	Server	Auditor
Wang's audit [26, 27]	$3 \cdot |M|$	$O(1)$
Zhu's IPDP [31, 32]	$|M|/s$	$O(1)$
TSAS	$|M|/s$	$O(1)$

s number of sectors in each data block

2.7.1 Storage Overhead

We compare the storage overhead on both the server and the auditor as described in Table 2.4.

2.7.1.1 Storage Overhead on the Server

The storage overhead on the server mainly comes from the storage of data tags. Suppose the size of data component is $|M|$ and the security parameter is set to 160-bit.

In Wang's auditing scheme, the data is divided into data blocks, and for each data block, there is a data tag. Due to the security reason, the size of each data element (in Wang's scheme, the data element is the data block) should not be larger than the security parameter. In that case, the total size of data tags should be $|M|$-bit, which is the same as the total size of data blocks. Moreover, in Wang's scheme, the server should store a MHT for the dynamic auditing, which incurs $2|M|$-bit storage overhead. Thus, in Wang's auditing scheme, the storage overhead on the server should be $3|M|$-bit, three times of the data size.

Both the TSAS and Zhu's IPDP apply the data fragment technique to further split each data block into s sectors. Since the data element is the sector in the TSAS and Zhu's IPDP, the size of each sector is corresponding to the security parameter. Then, for each data block that consists of s sectors only one data tag is generated, such that a $|M|$-bit data component only incurs $\frac{|M|}{s}$-bit storage overhead, which can greatly reduce the storage overhead.

2.7.1.2 Storage Overhead on the Auditor

The abstract information of the data contributes the main storage overhead on the auditor. In Wang's auditing scheme, the abstract data information only contains the file name, the number of data blocks. Besides the file name and the number of data blocks, in the TSAS and Zhu's IPDP, the abstract data information also includes the index table. However, the value of each item in the index table is only the number from 1 to the total number of data blocks n. The size of each item in the index table is very small compared to the data tags. For example, suppose the security parameter

is 160-bit, and the number of sectors in each data block is set to 50. Then, for 10 MB data component, the number of data block is 1000, which means that TSAS can use 10 bits to describe all the values in the index table. Thus, the size of index table is 500 Bytes, which is 0.005 % of the data size. Therefore, the storage overhead on the auditor is $O(1)$.

2.7.2 Communication Cost

Because the communication cost during the initialization is almost the same in these three auditing protocols, we only compare the communication cost between the auditor and the server, which consists of the challenge and the proof.

Consider a batch auditing with K owners and C cloud servers. Suppose the number of challenged data block from each owner on different cloud servers is the same, denoted as t, and the data block are split into s sectors in Zhu's IPDP and TSAS. We do the comparison under the same probability of detection. That is, in Wang's scheme, the number of data blocks from each owner on each cloud server should be st. The result is described in Table 2.5.

From the table, we can see that the communication cost in Wang's auditing scheme is not only linear to C, K, t, s, but also linear to the total number of data blocks n. As we know, in large scale cloud storage systems, the total number of data blocks could be very large. Therefore, Wang's auditing scheme may incur high communication cost.

TSAS and Zhu's IPDP have the same total communication cost during the challenge phase. During the proof phase, the communication cost of the proof in TSAS is only linear to C, but in Zhu's IPDP, the communication cost of the proof is not only linear to C and K, but also linear to s. That is because Zhu's IPDP uses the mask technique to protect the data privacy, which requires to send both the masked proof and the encrypted mask to the auditor. In TSAS, the server is only required to send the encrypted proof to the auditor and thus incurs less communication cost than Zhu's IPDP.

Table 2.5 Communication cost comparison of batch auditing for K owners and C clouds

Scheme	Challenge	Proof
Wang's audit [26, 27]	$O(KCst)$	$O(KCst \log n)$
Zhu's IPDP [31, 32]	$O(KCt)$	$O(KCs)$
TSAS	$O(KCt)$	$O(C)$

t is the number of challenged data blocks from each owner on each cloud server
s is the number of sectors in each data block
n is the total number of data blocks of a file in Wang's scheme

2.7.3 Computation Complexity

The simulation of the computation on the owner, the server and the auditor is conducted on a Linux system with an Intel Core 2 Duo CPU at 3.16 GHz and 4.00 GB RAM. The code uses the Pairing-Based Cryptography (PBC) library version 0.5.12 to simulate TSAS and Zhu's IPDP (Under the same detection of probability, Wang's scheme requires much more data blocks than TSAS and Zhu's IPDP, such that the computation time is almost s times more than TSAS and Zhu's IPDP and thus it is not comparable). The elliptic curve used is a MNT $d159$-curve, where the base field size is 159-bit and the embedding degree is 6. The $d159$-curve has a 160-bit group order, which means p is a 160-bit length prime. All the simulation results are the mean of 20 trials.

2.7.3.1 Computation Cost of the Auditor

We compare the computation time of the auditor versus the number of data blocks, the number of clouds and the number of owners in Fig. 2.4.

Figure 2.4a shows the computation time of the auditor versus the number of challenged data blocks in the single cloud and single owner case. In this figure, the number of data blocks goes to 500 (i.e. the challenged data size equals to 500 KByte), but it can illustrate the linear relationship between the computation cost of the auditor versus the challenged data size. From the Fig. 2.4a, we can see that TSAS incurs less computation cost of the auditor than Zhu's IPDP, when coping with large number of challenged data blocks.

In real cloud storage systems, the data size is very large (e.g. petabytes), TSAS applies the sampling auditing method to ensure the integrity of such large data. The sample size and the frequency are determined by the service level agreement. From the simulation results, it requires approximate 800 s to audit for 1 GByte data. However, the computing abilities of the cloud server and the auditor are much more powerful than the simulation PC, so the computation time can be relatively small. Therefore, TSAS is practical in large scale cloud storage systems.

Figure 2.4b describes the computation cost of the auditor of the multi-cloud batch auditing scheme versus the number of challenged clouds. It is easy to find that TSAS incurs less computation cost of the auditor than Zhu's IPDP, especially when there are a large number of clouds in the large scale cloud storage systems.

Because Zhu's IPDP does not support the batch auditing for multiple owners, the simulation repeats the computation for several times which is equal to the number of data owners. Figure 2.4c compares the computation cost of the auditor between the multi-owner batch auditing and the general auditing protocol which does not support the multi-owner batch auditing (e.g. Zhu's IPDP). Figure 2.4c also demonstrates that the batch auditing for multiple owners can greatly reduce the computation cost. Although in the simulation the number of data owners goes to 500, it can illustrate the trend of computation cost of the auditor that TSAS is much more efficient than

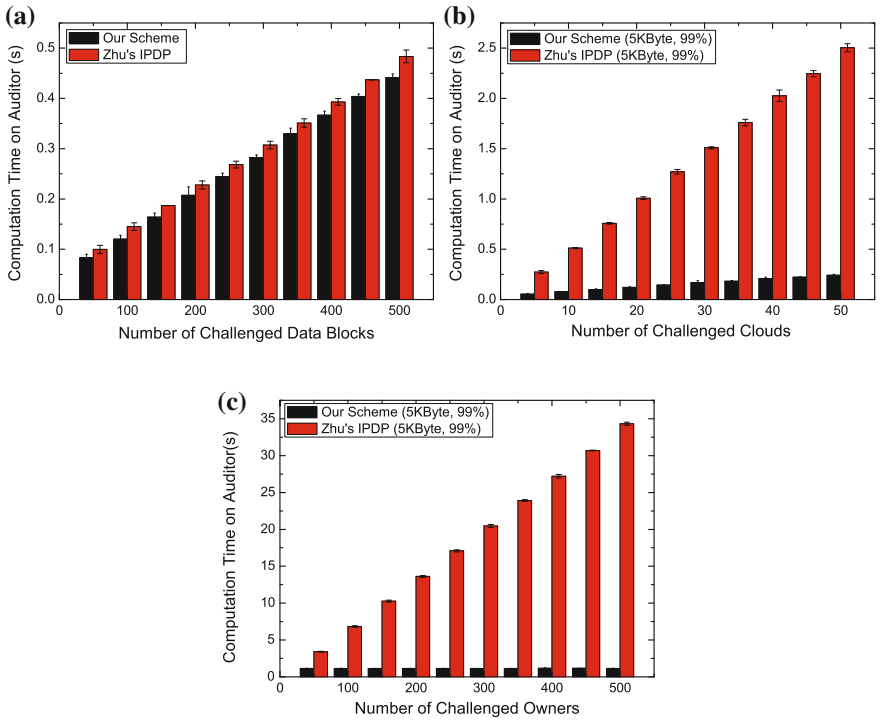

Fig. 2.4 Comparison of computation cost of the auditor (s = 50). **a** Single owner, single cloud. **b** Single owner, 5 blocks/cloud. **c** Single cloud, 5 blocks/owner

Zhu's IPDP in large scale cloud storage systems that may have millions to billions of data owners.

2.7.3.2 Computation Cost of the Server

We compare the computation cost of the server versus the number of data blocks in Fig. 2.5a and the number of data owners in Fig. 2.5b. TSAS moves the computing loads of the auditing from the auditor to the server, such that it can greatly reduce the computation cost of the auditor.

2.7.4 Computation Cost of the Owner

Both TSAS and Zhu's IPDP apply the data fragment technique to reduce the number of data blocks by further splitting data block into sectors. The number of sectors in each data block should be carefully selected. As we mentioned, due to the security

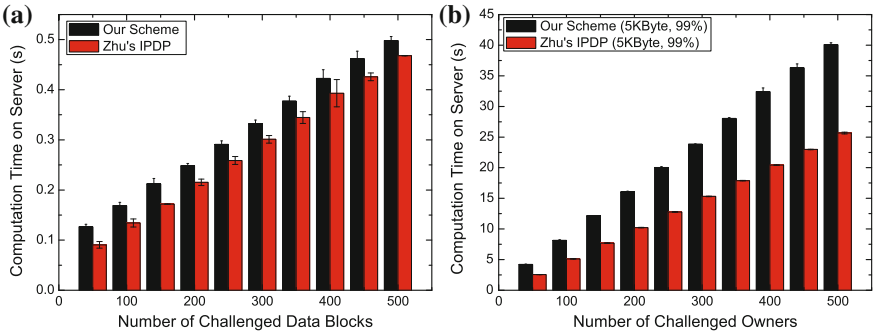

Fig. 2.5 Comparison of computation cost on the server (s = 50). **a** Single owner, single cloud. **b** Single cloud, 5 blocks/owner

reason, the size of the data element should not be larger than the security parameter. In TSAS and Zhu's IPDP, the data element is the data sector, thus the size of each data sector is fixed according to the security parameter. For a constant size data component M, the number of data blocks can be calculated as $n = \frac{sizeof(M)}{s \cdot \log p}$, where s is the number of sectors in the data block and p is the security parameter.

When considering the time of generating a tag for one data block, it is easy to see that the computation time is linear to the number of sectors in the data block. Specifically, let **Exp**. and **Mul**. be an exponentiation computation and a multiplication computation in the group respectively. Let the **H**. be the hash computation. The time of generating a data tag for one data block can be described as

$$Time_{tag}(s) = s \cdot (\mathbf{Exp.} + \mathbf{Mul.}) + \mathbf{Exp.} + \mathbf{H.}$$

The total tag generation time for a constant size of data M can be calculated as

$$TTime_{tag}(s) = \frac{sizeof(M)}{\log p}\left(\mathbf{Exp.} + \mathbf{Mul.} + \frac{1}{s}(\mathbf{Exp.} + \mathbf{H.})\right).$$

It is easy to find that the total tag generation time for a constant size of data is linear to $\frac{1}{s}$. Figure 2.6 shows the total computation time of generating all the data tags for 1 MByte data component versus the number of sectors in each data block. We can see that for the fixed size data, when the number of sectors in a data block is increased, the total time of tag generation goes stable.

2.8 Related Work

Juels et al. proposed a Proofs Of Retrievability (POR) scheme which enables a server (prover) to give a concise proof that a user (verifier) can retrieve a target file [12]. Their POR protocol encrypts the file F and randomly embeds a set of randomly-valued

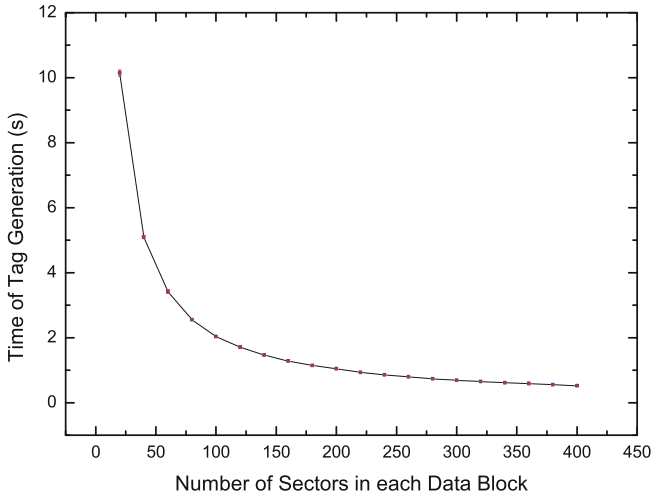

Fig. 2.6 Computation time of tag generation for 1 MByte data

check blocks called *sentinels*. The verifier challenges the prover by specifying the positions of a collection of *sentinels* and asking the prover to return the associated *sentinel* values. If the prover has modified or deleted a substantial portion of F, then it also has suppressed some *sentinels* with high probability and thus it cannot respond correctly to the verifier. The security of this protocol is proved by Dodis et al. in [7] without making any simplifying assumptions on the behavior of the adversary. However, this POR protocol is inappropriate for the proposed problem because it only allows a limited number of auditing times which is related to the number of *sentinels*.

To ensure the data integrity in remote servers, in [22, 23], 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 it. However, the number of times a particular data item can be verified is limited by the number of secret keys that fixed beforehand. Besides, the auditor needs to store several MACs for each file. Therefore, Shah's auditing protocols still cannot be applied to the problem.

Filho et al. [9] proposed a cryptographic protocol based on RSA-based secure hash function, through which a prover can demonstrate possession of a set of data known to the verifier. But in their protocol the prover needs to exponentiate the entire data file which will cause high computation cost. To overcome the drawback of Filho's protocol, Sebe et al. [20] improved the protocol by first dividing data into blocks and fingerprinting each block and then using a RSA-based hash function on each block. Then, a Diffie-Hellman-based approach is used to verify the data integrity. Their protocol can reduce the computation time of verification by trading off the computation time required at the prover against the storage required at the verifier.

Similarly, Yamamoto et al. [28] proposed a fast integrity checking scheme through batch verification of homomorphic hash functions on randomly selected blocks of data. However, in their schemes, the verifier needs to store a copy of the meta-data, such that they cannot be applied to the storage auditing in cloud storage system.

Ateniese et al. proposed a Sampling Provable Data Possession (SPDP) scheme [2], which combines the RSA cryptography with Homomorphic Verifiable Tags (HVT). It divides the data into several data blocks and encrypts each data block. For each auditing query, the auditor only challenge a subset of data blocks. By using such sampling method, the integrity of entire data can be guaranteed, when sufficient number of such sampling auditing queries are conducted. This sampling mechanism is applied in many remote integrity checking scheme, because it could significantly reduce the workloads of the server. Although the SPDP scheme can keep the data privacy, it cannot support the dynamic auditing and the batch auditing for multiple owners.

To support the dynamic auditing, 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. [8] also extended the PDP model to support dynamic updates on the stored data and proposed two dynamic provable data possession scheme by using a new version of authenticated dictionaries based on rank information. However, their schemes may cause heavy computation burden to the server since they relied on the PDP scheme proposed by the Ateniese.

In [27], the authors proposed a dynamic auditing protocol that can support the dynamic operations of the data on the cloud servers, but this method may leak the data content to the auditor because it requires the server to send the linear combinations of data blocks to the auditor. In [26], the authors extended their dynamic auditing scheme to be privacy-preserving and support the batch auditing for multiple owners. However, due to the large number of data tags, their auditing protocols will incur a heavy storage overhead on the server. In [31], Zhu et al. proposed a cooperative provable data possession scheme that can support the batch auditing for multiple clouds and also extend it to support the dynamic auditing in [32]. However, it is impossible for their scheme to support the batch auditing for multiple owners. That is because parameters for generating the data tags used by each owner are different and thus they cannot combine the data tags from multiple owners to conduct the batch auditing. Another drawback is that their scheme requires an additional trusted organizer to send a commitment to the auditor during the batch auditing for multiple clouds, because their scheme applies the mask technique to ensure the data privacy. However, such additional organizer is not practical in cloud storage systems. Furthermore, both Wang's schemes and Zhu's schemes incur heavy computation cost of the auditor, which makes the auditing system inefficient.

2.9 Conclusion

In this chapter, we introduced TSAS, an efficient and inherently secure dynamic auditing protocol. It protects the data privacy against the auditor by combining the cryptography method with the bilinearity property of bilinear paring, rather than using the mask technique. Thus, the multi-cloud batch auditing protocol in TSAS does not require any additional organizer. The batch auditing protocol in TSAS can also support the batch auditing for multiple owners. Furthermore, TSAS incurs less communication cost and less computation cost of the auditor by moving the computing loads of auditing from the auditor to the server, which greatly improves the auditing performance and can be applied to large scale cloud storage systems.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM conference on computer and communications security (CCS'07), pp. 598–609. ACM (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. 1–10. ACM (2008)
4. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: Proceedings of the15th international conference on the theory and application of cryptology and information security: advances in cryptology—ASIACRYPT'09, pp. 319–333. Springer (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 (2007)
6. 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)
7. Dodis, Y., Vadhan, S.P., Wichs, D.: Proofs of retrievability via hardness amplification. In: Proceedings of the 6th theory of cryptography conference (TCC'09), pp. 109–127. Springer (2009)
8. Erway, C.C., Küpçü, 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 (2009)
9. Filho, D.L.G., Barreto, P.S.L.M.: Demonstrating data possession and uncheatable data transfer. IACR Cryptology ePrint Archive **2006**, 150 (2006)
10. Goldreich, O.: Zero-knowledge twenty years after its invention. Electron. Colloquium Comput. Complex. **63** (2002)
11. 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 (DSN'04), pp. 135–144. IEEE Computer Society (2004)
12. Juels, A., Jr., Kaliski, 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 (2007)

13. Kher, V., Kim, Y.: Securing distributed storage: challenges, techniques, and systems. In: Proceedings of the 2005 ACM workshop on storage security and survivability (StorageSS05), pp. 9–25. ACM (2005)
14. Li, J., Krohn, M.N., Mazières, D., Shasha, D.: Secure untrusted data repository (sundr). In: Proceedings of the 6th conference on symposium on operating systems design and implementation, pp. 121–136. Berkeley, CA, USA (2004)
15. Lillibridge, M., Elnikety, S., Birrell, A., Burrows, M., Isard, M.: A cooperative internet backup scheme. In: Proceedings of the general track: 2003 USENIX annual technical conference, pp. 29–41. USENIX (2003)
16. Mell, P., Grance, T.: The NIST definition of cloud computing. Tech. report, National Institute of Standards and Technology (2009)
17. Naor, M., Rothblum, G.N.: The complexity of online memory checking. J. ACM 56(1), 1–46 (2009)
18. 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 (FAST'07), pp. 1–16. USENIX (2007)
19. Schwarz, T.J.E., Miller, E.L.: 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) (2006)
20. Sebé, F., Domingo-Ferrer, J., Martínez-Ballesté, A., Deswarte, Y., Quisquater, J.J.: Efficient remote data possession checking in critical information infrastructures. IEEE Trans. Knowl. Data Eng. **20**(8), 1034–1038 (2008)
21. 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 (2008)
22. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R.: Auditing to keep online storage services honest. In: Proceedings of the 11th workshop on hot topics in operating systems (HotOS'07). USENIX Association (2007)
23. Shah, M.A., Swaminathan, R., Baker, M.: Privacy-preserving audit and extraction of digital contents. IACR Cryptology ePrint Archive **2008**, 186 (2008)
24. Velte, T., Velte, A., Elsenpeter, R.: Cloud computing: a practical approach, 1 edn., chap. 7. McGraw-Hill Inc., New York (2010)
25. Wang, C., Ren, K., Lou, W., Li, J.: Toward publicly auditable secure cloud data storage services. IEEE Netw. **24**(4), 19–24 (2010)
26. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings of the 29th IEEE international conference on computer communications (INFOCOM'10), pp. 525–533. IEEE (2010)
27. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling public auditability and data dynamics for storage security in cloud computing. IEEE Trans. Parallel Distrib. Syst. **22**(5), 847–859 (2011)
28. 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)
29. Yang, K., Jia, X.: Data storage auditing service in cloud computing: challenges, methods and opportunities. World Wide Web **15**(4), 409–428 (2012)
30. 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 (2008)
31. Zhu, Y., Hu, H., Ahn, G., Yu, M.: Cooperative provable data possession for integrity verification in multi-cloud storage. IEEE Trans. Parallel Distrib. Syst. **23**(12) 2231–2244 (2012)
32. Zhu, Y., Wang, H., Hu, Z., Ahn, G.J., Hu, H., Yau, S.S.: Dynamic audit services for integrity verification of outsourced storages in clouds. In: Proceedings of the 2011 ACM symposium on applied computing (SAC'11), pp. 1550–1557. ACM (2011)

http://www.springer.com/978-1-4614-7872-0

Security for Cloud Storage Systems

Yang, K.; Xiaohua, J.

2014, XI, 83 p. 11 illus., Softcover
ISBN: 978-1-4614-7872-0