An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing

Kan Yang, *Student Member*, *IEEE*, and Xiaohua Jia, *Fellow*, *IEEE*

Abstract—In cloud computing, 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. Some existing remote integrity checking methods can only serve for static archive data and, thus, cannot be applied to the auditing service since the data in the cloud can be dynamically updated. Thus, an efficient and secure dynamic auditing protocol is desired to convince data owners that the data are correctly stored in the cloud. In this paper, we first design an auditing framework for cloud storage systems and propose an efficient and privacy-preserving auditing protocol. Then, we extend our auditing protocol to support the data dynamic operations, which is efficient and provably secure in the random oracle model. We further extend our auditing protocol to support batch auditing for both multiple owners and multiple clouds, without using any trusted organizer. The analysis and simulation results show that our proposed auditing protocols are secure and efficient, especially it reduce the computation cost of the auditor.

Index Terms—Storage auditing, dynamic auditing, privacy-preserving auditing, batch auditing, cloud computing

1 Introduction

Cloud storage is an important service of cloud computing [1], 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 [2]. However, this new paradigm of data hosting service also introduces new security challenges [3]. 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 [4], [5], [6], [7], [8]. Sometimes, cloud service providers might be dishonest. They could discard the data that have 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.

Traditionally, owners can check the data integrity based on two-party storage auditing protocols [9], [10], [11], [12], [13], [14], [15], [16], [17]. 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 that have been proposed in some previous works [18], [19]. 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 [20], [21], [22], [23], [24], [25], [26], [27], [28]. Table 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. From Table 1, we can find that many of them are not privacy preserving or cannot support the data dynamic operations, so that they cannot be applied to cloud storage systems.

In [23], 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 [24], 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 may incur a heavy storage overhead on the server. In [25], 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 [26]. 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

• *The authors are with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong, SAR. E-mail: kanyang3@student.cityu.edu.hk, csjia@cityu.edu.hk.*

Manuscript received 13 Aug. 2011; revised 7 Sept. 2012; accepted 9 Sept. 2012; published online 21 Sept. 2012.
Recommended for acceptance by L.E. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-08-0534.
Digital Object Identifier no. 10.1109/TPDS.2012.278.

TABLE 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 [20]	$O(t)$	$O(t)$	$O(1)$	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 [22]	$O(t\log n)$	$O(t\log n)$	$O(t\log n)$	No	No	No	No	$1-(1-\rho)^t$
Audit [23], [24]	$O(t\log n)$	$O(t\log n)$	$O(t\log n)$	Yes	Yes	Yes	No	$1-(1-\rho)^t$
IPDP [25], [26]	$O(ts)$	$O(t+s)$	$O(t+s)$	Yes	Yes	No	Yes	$1-(1-\rho)^{ts}$
Our Scheme	$O(ts)$	$O(t)$	$O(t)$	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).

that their scheme requires an additional trusted organizer to send a commitment to the auditor during the multicloud 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 paper, we propose an efficient and secure dynamic auditing protocol, which can meet the above-listed requirements. To solve the data privacy problem, our method 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, our method does not require any trusted organizer during the batch auditing for multiple clouds. On the other hand, in our method, we let 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, our method can greatly reduce the computing loads of the auditor by moving it to the cloud server.

Our original contributions can be summarized as follows:

1. We design an auditing framework for cloud storage systems and propose a privacy-preserving and efficient storage auditing protocol. Our auditing protocol ensures the data privacy by using cryptography method and the Bilinearity property of the bilinear pairing, instead of using the mask technique. Our auditing protocol incurs less communication cost between the auditor and the server. It also reduces the computing loads of the auditor by moving it to the server.
2. We extend our auditing protocol to support the data dynamic operations, which is efficient and provably secure in the random oracle model.
3. We further extend our auditing protocol to support batch auditing for not only multiple clouds but also multiple owners. Our multicloud batch auditing does not require any additional trusted organizer. The multiowner batch auditing can greatly improve the auditing performance, especially in large-scale cloud storage systems.

The remaining of this paper is organized as follows: In Section 2, we describe definitions of the system model and security model. In Section 3, we propose an efficient and inherently secure auditing protocol and extend it to support the dynamic auditing in Section 4. We further extend our auditing protocol to support the batch auditing for multiple owners and multiple clouds in Section 5. Section 6 give the performance analysis of our proposed auditing protocols in terms of communication cost and computation cost. The security proof will be shown in the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.278. In Section 7, we give the related work on storage auditing. Finally, the conclusion is given in Section 8.

2 Preliminaries and Definitions

In this section, we first describe the system model and give the definition of storage auditing protocol. Then, we define the threat model and security model for a storage auditing system.

2.1 Definition of a System Model

We consider an auditing system for cloud storage as shown in Fig. 1, which 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.

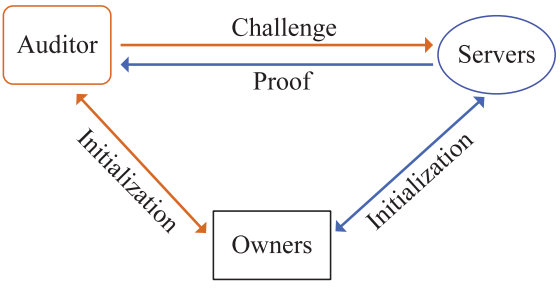


Fig. 1. System model of the data storage auditing.

TABLE 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

Before describing the auditing protocol definition, we first define some notations as listed in Table 2.

Definition 1 (Storage auditing protocol). *A storage auditing protocol consists of the following five algorithms:*

1. $\mathbf{KeyGen}(\lambda) \rightarrow (sk_h, sk_t, pk_t)$. *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).*
2. $\mathbf{TagGen}(M, sk_t, sk_h) \rightarrow T$. *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 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 $T = \{t_i\}_{i\in[1,n]}$.*
3. $\mathbf{Chall}(M_{info}) \rightarrow \mathcal{C}$. *The challenge algorithm takes as input the abstract information of the data M_{info} (e.g., file identity, total number of blocks, version number, time stamp, etc.). It outputs a challenge $\mathcal{C}$.*
4. $\mathbf{Prove}(M, T, \mathcal{C}) \rightarrow \mathcal{P}$. *The prove algorithm takes as inputs the file M, the tags T, and the challenge from the auditor $\mathcal{C}$. It outputs a proof $\mathcal{P}$.*
5. $\mathbf{Verify}(\mathcal{C}, \mathcal{P}, sk_h, pk_t, M_{info}) \rightarrow 0/1$. *The verification algorithm takes as inputs $\mathcal{P}$ 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 Definition of a Security Model

We assume the auditor is 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.

3 Efficient and Privacy-Preserving Auditing Protocol

In this section, we first present some techniques we applied in the design of our efficient and privacy-preserving auditing protocol. Then, we describe the algorithms and the detailed construction of our auditing protocol for cloud storage systems. The correctness proof will be shown in the supplemental file, available online.

3.1 Overview of Our Solution

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, our method 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 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. In our method, we let 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, our method can greatly reduce the computing loads of the auditor by moving it to the cloud server.

To improve the performance of an auditing system, we apply the *data fragment technique* and *homomorphic verifiable tags* in our method. 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.

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, we further split each data block into sectors. The sector size is restricted by the security parameter. We generate one data tag for each data block that consists of s sectors, such that less data tags are generated. 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, we only consider one data component in our construction 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, we first select 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, $s_i < s_{max}$, we 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, we can calculate the number of data blocks as $n = \frac{sizeof(M)}{s \cdot \log p}$. We denote the encrypted data component 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be the bilinear map. Let g_1 and g_2 be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Let $h : \{0,1\}^* \rightarrow \mathbb{G}_1$ be a keyed secure hash function that maps the M_{info} to a point in $\mathbb{G}_1$.

Our storage auditing protocol consists of the following algorithms:

$\mathbf{KeyGen}(\lambda) \rightarrow (pk_t, sk_t, sk_h)$. 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.

$\mathbf{TagGen}(M, sk_t, sk_h) \rightarrow T$. 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

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

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 $T = \{t_i\}_{i \in [1,n]}$.

$\mathbf{Chall}(M_{info}) \rightarrow \mathcal{C}$. 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 $\mathcal{C} = (\{i, v_i\}_{i \in Q}, R)$.

$\mathbf{Prove}(M, T, \mathcal{C}) \rightarrow \mathcal{P}$. The prove algorithm takes as inputs the data M and the received challenge $\mathcal{C} = (\{i, v_i\}_{i \in Q}, R)$.

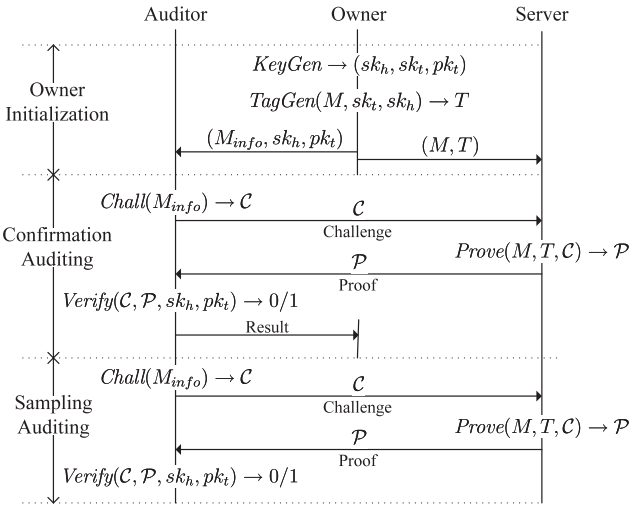


Fig. 2. Framework of our privacy-preserving auditing protocol.

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 $\mathcal{P} = (TP, DP)$.

$\mathbf{Verify}(\mathcal{C}, \mathcal{P}, sk_h, pk_t, M_{info}) \rightarrow 0/1$. 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:

$$DP \cdot e(H_{chal}, pk_t) = e\big(TP, g_2^r\big). \quad (1)$$

If the above verification equation (1) holds, it outputs 1. Otherwise, it outputs 0.

3.3 Construction of Our Privacy-Preserving Auditing Protocol

As illustrated in Fig. 2, our 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.

Phase 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.

Phase 2: Confirmation auditing. In our 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 are 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 are correctly stored on the server, and it may choose to delete the local version of the data.

Phase 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.

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 $Pr(t, s)$.

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.

4.1 Our Solution

To prevent the replay attack, we introduce an *index table* (ITable) 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 time stamp 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 that 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, we can modify the tag generation algorithm TagGen. Specifically, when generating the data tag t_i for the data block m_i, we 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. The detailed proof will be given in the supplemental file, available online.

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 our 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, as illustrated in Fig. 3, we only describe the dynamic auditing phase, which contains three steps: data update, index update, and update confirmation.

Step 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:

$\text{Modify}(m_i^*, sk_t, sk_h) \rightarrow (Msg_{modify}, t_i^*)$. 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 time stamp T_i^*, and calls the TagGen to generate a new data tag t_i^* for data

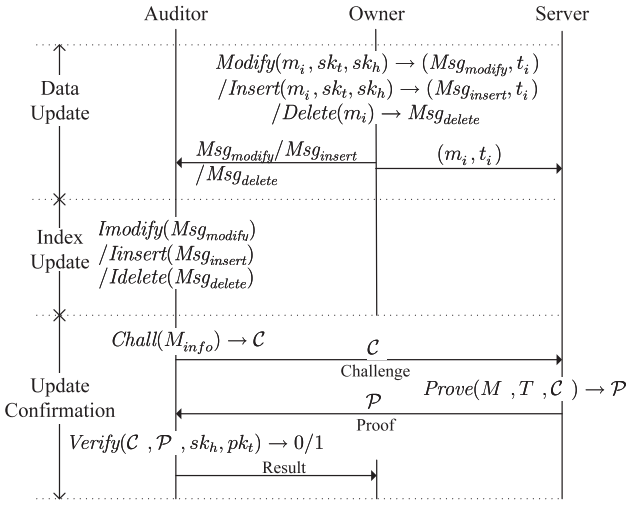


Fig. 3. Framework of auditing for dynamic operations.

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 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 time stamp 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.

Step 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 time stamp 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 ith position in the previous ITable backward in order, with the index number increased by 1.

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 1.

Table 3 shows the change of ITable according to the different type of data update operation. Table 3a describes the initial table of the data $M = \{m_1, m_2, \ldots, m_n\}$, and Table 3b describes the ITable after m_2 is updated. Table 3c shows the ITable after a new data block is inserted before m_2, and Table 3d shows the ITable after m_2 is deleted.

Step 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.

5 Batch Auditing for Multiowner and Multicloud

Data storage auditing is a significant service in cloud computing that 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 [25] 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 that hosts the owner's data and verify all the proofs from them. To reduce the computation cost of the auditor, it is

TABLE 3
ITable of the Abstract Information of Data M

(a) Initial Abstract Information of M.

Index	B_i	V_i	T_i
1	1	1	T_1
2	2	1	T_2
3	3	1	T_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n	1	T_n

(b) After modifying m_2, V_2 and T_2 are updated

Index	B_i	V_i	T_i
1	1	1	T_1
2	2	2	T_2^*
3	3	1	T_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n	1	T_n

(c) After inserting before m_2, all items before m_2 move backward with the index increased by 1.

Index	B_i	V_i	T_i
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

(d) After deleting m_2, all items after m_2 move forward with the index decreased by 1.

Index	B_i	V_i	T_i
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

desirable to combine all these responses together and do the batch verification.

In the previous work [25], the authors proposed a cooperative provable data possession for integrity verification in multicloud 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 multicloud batch auditing. In our method, we apply the encryption method with the bilinearity property of the bilinear pairing to ensure the data privacy, rather than the mask technique. Thus, our multicloud batch auditing protocol does not have any commitment phase, such that our method does not require any additional trusted organizer.

5.1 Algorithms for Batch Auditing for Multiowner and Multicloud

Let O be the set of owners and S be the set of cloud servers. The batch auditing for multiowner and multicloud can be constructed as follows:

Phase 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. We denote each data component 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, we assume that each data block is further split into the same number of sectors. We can use the similar technique proposed in Section 3.2 to deal with the situation that each data block 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

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

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.

Phase 2: Batch auditing for multiowner and multicloud. 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

$$\prod_{l\in S_{chal}} DP_l = \frac{e\left(\prod_{l\in S_{chal}} TP_l, g_2^r\right)}{\prod_{k\in O_{chal}} e(H_{chal,k}, pk_{t,k})}. \quad (2)$$

If (2) is true, it outputs 1. Otherwise, it outputs 0.

TABLE 4
Communication Cost Comparison of Batch Auditing for K Owners and C Clouds

Scheme	Challenge	Proof
Wang's Audit [23], [24]	$O(KCst)$	$O(KCst \log n)$
Zhu's IPDP [25], [26]	$O(KCt)$	$O(KCs)$
Our Scheme	$O(KCt)$	$O(C)$

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.

6 Performance Analysis of Our Auditing Protocols

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 our scheme and two existing works: the Audit protocol proposed by Wang et al. [23], [24] and the IPDP proposed by Zhu et al. [25], [26]. The storage overhead analysis will be shown in the supplemental file, available online.

6.1 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 our scheme. 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 4.

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.

Our scheme 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 our scheme 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 our scheme, the server is only required to send the encrypted proof to the auditor and, thus, incurs less communication cost than Zhu's IPDP.

6.2 Computation Complexity

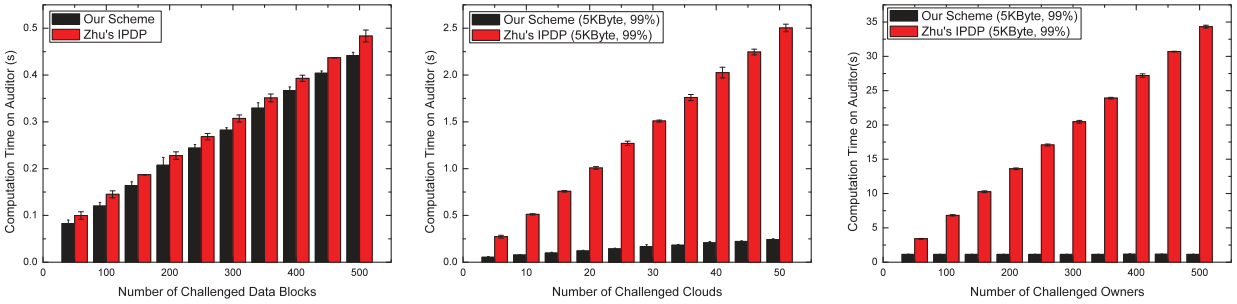
We simulate the computation of the owner, the server, and the auditor on a Linux system with an Intel Core 2 Duo CPU at 3.16 GHz and 4.00-GB RAM. The code uses the pairing-based cryptography library version 0.5.12 to simulate our auditing scheme and Zhu's IPDP scheme (Under the same detection of probability, Wang's scheme requires much more data blocks than our scheme and Zhu's scheme, such that the computation time is almost s times more than our scheme and Zhu's IPDP, and thus, it is not comparable). The elliptic curve we 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.

6.2.1 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. 4.

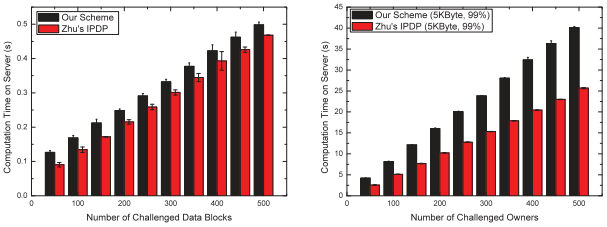
Fig. 4a shows the computation time of the auditor versus the number of challenged data blocks in the single cloud and single owner case. In this figure, the number of data blocks goes to 500 (i.e., the challenged data size equals to 500 KByte), but it can illustrate the linear relationship between the computation cost of the auditor versus the challenged data size. From Fig. 4a, we can see that our scheme incurs less computation cost of the auditor than Zhu's IPDP scheme, when coping with large number of challenged data blocks.

In real cloud storage systems, the data size is very large (e.g., petabytes), our scheme apply 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, we can estimate that it requires 800 seconds to audit for 1-GByte data. However, the computing abilities of the cloud server and the auditor are much more powerful than



(a) Single Owner, Single Cloud (b) Single Owner, 5 blocks/Cloud (c) Single Cloud, 5 blocks/Owner

Fig. 4. Comparison of computation cost of the auditor ($s = 50$).



(a) Single Owner, Single Cloud (b) Single Cloud, 5 blocks/Owner

Fig. 5. Comparison of computation cost on the server ($s = 50$).

our simulation PC, so the computation time can be relatively small. Therefore, our auditing scheme is practical in large-scale cloud storage systems.

Fig. 4b describes the computation cost of the auditor of the multicloud batch auditing scheme versus the number of challenged clouds. It is easy to find that our scheme incurs less computation cost of the auditor than Zhu's IPDP scheme, 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, in our simulation, we repeat the computation for several times that is equal to the number of data owners. Then, as shown in Fig. 4c, we compare the computation cost of the auditor between our multiowner batch auditing and the general auditing protocol that does not support the multiowner batch auditing (e.g., Zhu's IPDP). Fig. 4c also demonstrates that the batch auditing for multiple owners can greatly reduce the computation cost. Although in our simulation the number of data owners goes to 500, it can illustrate the trend of computation cost of the auditor that our scheme is much more efficient than Zhu's scheme in large-scale cloud storage systems that may have millions to billions of data owners.

6.2.2 Computation Cost of the Server

We compare the computation cost of the server versus the number of data blocks in Fig. 5a and the number of data owners in Fig. 5b. Our scheme 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.

7 Related Work

To support the dynamic auditing, Ateniese et al. [29] developed a dynamic provable data possession protocol based on cryptographic hash function and symmetric key encryption. Their idea is to precompute 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. [22] 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 because they relied on the PDP scheme proposed by Ateniese.

In [23], 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 [24], 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 [25], 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 [26]. 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.

8 Conclusion

In this paper, we proposed 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, our multicloud batch auditing protocol does not require any additional organizer. Our batch auditing protocol can also support the batch auditing for multiple owners. Furthermore, our auditing scheme 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] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," technical report, Nat'l Inst. of Standards and Technology, 2009.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Comm. ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[3] T. Velte, A. Velte, and R. Elsenpeter, *Cloud Computing: A Practical Approach*, first ed., ch. 7. McGraw-Hill, 2010.

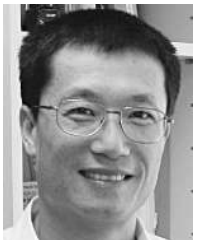
[4] J. Li, M.N. Krohn, D. Mazières, and D. Shasha, "Secure Untrusted Data Repository (SUNDR)," *Proc. Sixth Conf. Symp. Operating Systems Design Implementation*, pp. 121-136, 2004.

[5] G.R. Goodson, J.J. Wylie, G.R. Ganger, and M.K. Reiter, "Efficient Byzantine-Tolerant Erasure-Coded Storage," *Proc. Int'l Conf. Dependable Systems and Networks*, pp. 135-144, 2004.

[6] V. Kher and Y. Kim, "Securing Distributed Storage: Challenges, Techniques, and Systems," *Proc. ACM Workshop Storage Security and Survivability (StorageSS)*, V. Atluri, P. Samarati, W. Yurcik, L. Brumbaugh, and Y. Zhou, eds., pp. 9-25, 2005.

[7] L.N. Bairavasundaram, G.R. Goodson, S. Pasupathy, and J. Schindler, "An Analysis of Latent Sector Errors in Disk Drives," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems,* L. Golubchik, M.H. Ammar, and M. Harchol-Balter, eds., pp. 289-300, 2007.
[8] B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" *Proc. USENIX Conf. File and Storage Technologies,* pp. 1-16, 2007.
[9] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A Cooperative Internet Backup Scheme," *Proc. USENIX Ann. Technical Conf.,* pp. 29-41, 2003.
[10] Y. Deswarte, J. Quisquater, and A. Saidane, "Remote Integrity Checking," *Proc. Sixth Working Conf. Integrity and Internal Control in Information Systems (IICIS),* Nov. 2004.
[11] M. Naor and G.N. Rothblum, "The Complexity of Online Memory Checking," *J. ACM,* vol. 56, no. 1, article 2, 2009.
[12] A. Juels and B.S. Kaliski Jr., "Pors: Proofs of Retrievability for Large Files," *Proc. ACM Conf. Computer and Comm. Security,* P. Ning, S.D.C. di Vimercati, and P.F. Syverson, eds., pp. 584-597, 2007.
[13] T.J.E. Schwarz and E.L. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage," *Proc. 26th IEEE Int'l Conf. Distributed Computing Systems,* p. 12, 2006.
[14] D.L.G. Filho and P.S.L.M. Barreto, "Demonstrating Data Possession and Uncheatable Data Transfer," *IACR Cryptology ePrint Archive,* vol. 2006, p. 150, 2006.
[15] F. Sebé, J. Domingo-Ferrer, A. Martínez-Ballesté, Y. Deswarte, and J.-J. Quisquater, "Efficient Remote Data Possession Checking in Critical Information Infrastructures," *IEEE Trans. Knowledge Data Eng.,* vol. 20, no. 8, pp. 1034-1038, Aug. 2008.
[16] G. Yamamoto, S. Oda, and K. Aoki, "Fast Integrity for Large Data," *Proc. ECRYPT Workshop Software Performance Enhancement for Encryption and Decryption,* pp. 21-32, June 2007.
[17] M.A. Shah, M. Baker, J.C. Mogul, and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," *Proc. 11th USENIX Workshop Hot Topics in Operating Systems (HOTOS),* G.C. Hunt, ed., 2007.
[18] C. Wang, K. Ren, W. Lou, and J. Li, "Toward Publicly Auditable Secure Cloud Data Storage Services," *IEEE Network,* vol. 24, no. 4, pp. 19-24, July/Aug. 2010.
[19] K. Yang and X. Jia, "Data Storage Auditing Service in Cloud Computing: Challenges, Methods and Opportunities," *World Wide Web,* vol. 15, no. 4, pp. 409-428, 2012.
[20] G. Ateniese, R.C. Burns, R. Curtmola, J. Herring, L. Kissner, Z.N.J. Peterson, and D.X. Song, "Provable Data Possession at Untrusted Stores," *Proc. ACM Conf. Computer and Comm. Security,* P. Ning, S.D.C. di Vimercati, and P.F. Syverson, eds., pp. 598-609, 2007.
[21] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology,* J. Pieprzyk, ed., pp. 90-107, 2008.
[22] C.C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," *Proc. ACM Conf. Computer and Comm. Security,* E. Al-Shaer, S. Jha, and A.D. Keromytis, eds., pp. 213-222, 2009.
[23] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing," *IEEE Trans. Parallel Distributed Systems,* vol. 22, no. 5, pp. 847-859, May 2011.
[24] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," *Proc. IEEE INFOCOM,* pp. 525-533, 2010.
[25] Y. Zhu, H. Hu, G. Ahn, and M. Yu, "Cooperative Provable Data Possession for Integrity Verification in Multi-Cloud Storage," *IEEE Trans. Parallel and Distributed Systems,* vol. 23, no. 12, pp. 2231-2244, Dec. 2012.
[26] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S.S. Yau, "Dynamic Audit Services for Integrity Verification of Outsourced Storages in Clouds," *Proc. ACM Symp. Applied Computing,* W.C. Chu, W.E. Wong, M.J. Palakal, and C.-C. Hung, eds., pp. 1550-1557, 2011.
[27] K. Zeng, "Publicly Verifiable Remote Data Integrity," *Proc. 10th Int'l Conf. Information and Comm. Security,* L. Chen, M.D. Ryan, and G. Wang, eds., pp. 419-434, 2008.
[28] G. Ateniese, S. Kamara, and J. Katz, "Proofs of Storage from Homomorphic Identification Protocols," *Proc. Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology,* M. Matsui, ed., pp. 319-333, 2009.
[29] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," *IACR Cryptology ePrint Archive,* vol. 2008, p. 114, 2008.
[30] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-Based Encryption with Non-Monotonic Access Structures," *Proc. ACM Conf. Computer and Comm. Security (CCS '07),* P. Ning, S.D.C. di Vimercati, and P.F. Syverson, eds., Oct. 2007.

Kan Yang received the BEng degree from the University of Science and Technology of China in 2008. He is currently working toward the PhD degree in the Department of Computer Science at the City University of Hong Kong. His research interests include cryptography, information security, cloud computing and distributed systems. He is a student member of the IEEE and the IEEE Computer Society.



Xiaohua Jia received the BSc and MEng degrees from the University of Science and Technology of China, in 1984 and 1987, respectively, and the DSc degree in information science from the University of Tokyo in 1991. He is currently a chair professor in the Department of Computer Science at the City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks and mobile wireless networks. He is an editor of the *IEEE Transactions on Parallel and Distributed Systems* (2006-2009), *Wireless Networks*, *Journal of World Wide Web*, *Journal of Combinatorial Optimization*, and so on. He is the general chair of ACM MobiHoc 2008, TPC cochair of IEEE MASS 2009, area chair of IEEE INFOCOM 2010, TPC cochair of IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symp, and panel cochair of IEEE INFOCOM 2011. He is a fellow of the IEEE and the IEEE Computer Society.