JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 653-673 (2017)
DOI: 10.6688/JISE.2017.33.3.4

Large Branching Tree Based Dynamic Provable Data Possession Scheme*

YONG LI[1,2,3], GE YAO[1], LI-NAN LEI[1], HUA-QUN WANG[4] AND CHANG-LU LIN[2]
[1]*School of Electronic and Information Engineering*
Beijing Jiaotong University
Beijing, 100044 P.R. China
E-mail: liyong@bjtu.edu.cn
[2]*Fujian Provincial Key Laboratory of Network Security and Cryptology*
Fujian Normal University
Fuzhou, 350007 P.R. China
[3]*Guangxi Key Laboratory of Cryptography and Information Security*
Guilin University of Electronic Technology
Guilin, 541004 P.R. China
[4]*School of Computer Science and Technology*
Nanjing University of Posts and Telecommunications
Nanjing, 210023 P.R. China

As the era of big data is coming, more and more users choose to store data in the cloud. Cloud storage provides users with flexible, dynamic and cost effective data storage service. However, the new paradigm of service introduces new security challenges such as users loss control of the remote data and they cannot ensure data integrity in the cloud. Moreover, supporting dynamic data updates is also a practical requirement of cloud storage. It is imperative to provide an efficient and secure dynamic auditing protocol to check the data integrity in the cloud. In this paper, we first analyze the dynamic performance of some prior works. In order to solve the inefficiency problem caused by the Merkle Hash Tree (MHT) in dynamic update, the Large Branching Tree (LBT) data structure was introduced in our *Dynamic Provable Data Possession* (DPDP) scheme. LBT structure simplifies the process of updates and supports updating several blocks synchronously, and reduces the auxiliary information during the challenge-respond process as well. Our scheme is able to efficiently support *fully dynamic* data updates and *batch* updates. Based on the LBT structure, we use bilinear algebraic maps to optimize the authenticate process. A signature scheme is used to authenticate both the value and the position of data blocks, which reduces computation overhead during the dynamic update phase. The security and performance analysis show that our DPDP scheme is provably secure and efficient.

Keywords: cloud storage, provable data possession, large branching tree, dynamic update, public auditability

1. INTRODUCTION

Cloud computing paradigm has gained widespread popularity in the industry and academia and it has been envisioned as the next-generation architecture of IT enterprise [1]. It enables users to access to the infrastructure and application services on a subscription basis. This computing service can be categorized into three service models: Infra-

Received June 30, 2016; revised August 17, 2016; accepted September 16, 2016.
Communicated by Balamurugan Balusamy.
* A preliminary version [2] of this paper was presented at the 11th International Conference on Green, Pervasive and Cloud Computing (GPC 2016).

structure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [3]. Due to the advantage characteristics including large scale computation and data storage, virtualization, high scalability and elasticity, cloud computing technologies have been developing fast, of which one important application is cloud storage system. Cloud storage service is a new paradigm for delivering storage service on demand, over network and billed for just what is used. Many international IT corporations now offer cloud storage service on a scale from individual to enterprise, such as Amazon Simple Storage Service (S3) and EMC Atoms Cloud Storage, and a growing number of users choose to store their data in the cloud.

Although cloud storage is growing in popularity, data security is still one of the major concerns in the adoption of this new paradigm of data hosting. For concerning about data confidentiality and privacy, a general approach is to encrypt the data before outsourcing it to the cloud service providers [4]. However, search techniques for encrypted data [5], and more generally, techniques for computing over encrypted data are needed to balance data confidentiality and data utilization, which are not focus of this paper. Let's consider from another way. For example, the cloud service providers may discard the data which has not been accessed or rarely accessed to save the storage space or keep fewer replicas than promised [6]. And the storage service provider, which experiences Byzantine failures occasionally, may decide to hide the data errors from the client for the benefit of their own [1]. Furthermore, disputes occasionally suffer from the lack of trust on cloud service provider (CSP) because the data change may not be timely known by the cloud client, even if these disputes may result from the user's own improper operations [7]. Therefore, clients would like to check the *integrity* and *availability* of their stored data. However, the large size of the outsourced data and the limited resource capability present an additional restriction: the client should perform the integrity check *without downloading all stored data*.

To date, extensive researches are carried out to address this problem [8-17, 21-27]. Early work concentrated on enabling data owners to check the integrity of remote data, which can be denoted as *private verifiability.* Although schemes with private verifiability can achieve higher scheme efficiency, *public verifiability* (or *public auditability*) allows *anyone* not just the client (data owner), to challenge the cloud server for correctness of data storage while keeping no private information [1]. In cloud computing, instead of performing frequent integrity checks by themselves, data owners are able to delegate the verification of data integrity to a trusted third party – auditor (TPA), who has expertise and capabilities to audit the outsourced data on demand.

Recently, public auditability has become one of the basic requirements of data storage auditing scheme. However, there are still some major concerns need to be solved before put the auditing schemes into practical use. Many big data applications keep clients' data on the cloud and offer frequently update operations. A most typical example is Twitter. Data stored in cloud may not only be accessed but also updated by the clients through either modify an existing data block, or insert a new block, or delete any block. To support the most general forms of update operation is important to broaden the scope of practical application of cloud storage. Therefore, it is imperative to extend the auditing scheme to support provable updates to outsourced data. Unfortunately, traditional data integrity verification schemes are mainly designed for *static* data storage. The direct extension of these schemes may lead to functional defect or security vulnerability. In this

paper, we will focus on better support for *dynamic* and *efficient* data integrity verification for cloud storage applications. We employ a secure signature scheme from bilinear maps [18] and the Large Branching Tree (LBT) to achieve that aim. Our contribution can be summarized as follows:

(1) We formally define the framework of dynamic provable data possession scheme and provide an efficient construction, which supports *fully dynamic* updates including modification, insertion and deletion, and supports *batch dynamic* updates as well.
(2) We analyze the existing schemes and point out the disadvantages of the Merkle Hash Tree (MHT) used as the data structure for dynamic updates. For better efficiency, we replace MHT with LBT. This multiple branching data structure enables reduction in size of auxiliary information, thereby causes less communication cost compared to MHT-based schemes.
(3) We combine a secure signature algorithm with LBT data structure. The characteristics of bilinear pairings in the signature algorithm only cause $O(1)$ computation cost on CSP for each dynamic update. Besides, the client no longer needs to construct LBT structure to support dynamic operation. Consequently, this algorithm greatly reduces computation cost both on CSP and client as well as simplifies the update process.
(4) We prove the security of our proposed construction and show the performance of our scheme through comparisons with existing data integrity verification schemes [1, 8-10, 14, 15].

The rest of this paper is organized as follows. Section 2 discusses related works. In Section 3 we introduce main techniques, system model and security model. Then, Section 4 presents the specific description of our proposed scheme. Section 5 provides security analysis. We further analyze the experimental results in Section 6. Section 7 concludes the paper.

2. RELATED WORKS

Recently, the integrity verification for data outsourced in cloud has attracted extensive attention. The existing provable data integrity schemes can be classified into two categories: *proof of retrievability* (POR) and *provable data possession* (PDP). POR scheme was first proposed by Juels *et al.* in 2007 [8]. In their scheme, the client can not only check their remote data integrity, but also recover outsourced data in its entirety by employing erasure-correcting code. The following researches of POR focused on providing security analysis [10] and improving the construction. However, most of existing POR schemes can only be used to the static archive storage system, *e.g.*, libraries and scientific data sets [8, 10-12]. The reason is that the erasure-correcting codes using in POR system bring a problem: the *whole* outsourced data is required to perform a small update. This is the main issue towards making POR dynamic.

In cloud computing, the dynamic update is a significant issue for various applications which means that the outsourced data can be dynamically updated by the clients such as modification, deletion and insertion. Therefore, an efficient dynamic auditing protocol is essential in practical cloud storage systems [13].

In 2007, Ateniese *et al.* [9] first proposed PDP framework, which allows data owner to verify the integrity of its data stored at an untrusted server without retrieving the entire file. Compared to POR scheme, PDP did not use erasure-correcting codes, and hence

was more efficient. Although PDP did not provide the retrievability guarantee, the dynamic techniques of PDP are developed well in follow-up studies. Ateniese *et al.* [14] gave a dynamic PDP scheme based on their prior work [9], in which the client precomputes and stores at the server a limited number of random challenges with the corresponding answers. This scheme cannot perform insertion since that would affect all remaining answers.

The first fully dynamic PDP protocol was proposed by Erway *et al.* [15] in 2009. They considered using dynamic data structure to support data updates, so they constructed the rank-based authenticated dictionaries based on the skip list. However, the skip list requires a long authentication path and large amount of auxiliary information during the verification process. Wang *et al.* [1] employed homomorphic signature and MHT data structure to achieve supporting fully dynamic updates. Zhu *et al.* [7] proposed a dynamic auditing system based on fragment, random sampling and Index-Hash Tree (IHT) that supports provable updates and timely anomaly detection. Later on, researches are focus on supplying additional properties [19], distribute and replicate [16] or enhance efficiency and using other data structure [20]. For instance, Wang *et al.* [21] firstly proposed a proxy provable data possession (PPDP) system. Their protocol supports the general access structure so that only authorized clients are able to store data to public cloud servers. Lin *et al.* [22] proposed a novel provable data possession scheme, in which data of different values are integrated into a data hierarchy, and clients are classified and authorized different access permissions. Their scheme also allows the data owner to efficiently enroll and revoke clients which make it more practical in cloud environment.

Recently, Gritti *et al.* proposed an efficient and practical PDP system by adopting asymmetric pairings [23]. Their scheme outperforms other existing schemes because there are no exponentiation and only three pairings are required. However, this scheme is vulnerable to three attacks as they later pointed out [24]. Several solutions are proposed by Gritti *et al.* corresponding to all the vulnerabilities of scheme [23]. They used IHT and MHT techniques to resist the replace attack and replay attack. They also employed a weaker security model to achieve data privacy. Although system security can be guaranteed, the performance of the system still needs improvement.

Zhang *et al.* proposed a new solution to provable data possession with support for dynamic operations, access to shared data by multiple users, and revision control [25]. They used a new data structure called *balanced update tree* and removed the need for interaction during update operations in addition to communicating the updates themselves. For dealing with outsourced data be replicated on multiple servers across multiple data centers, Barsoum *et al.* proposed a map-based provable *multi-copy* dynamic data possession scheme [26], which is considered as extension of dynamic *single-copy* PDP scheme. In [27], Ren *et al.* proposed an efficient mutual verifiable provable data possession scheme, which utilizes Diffie-Hellman shared key to construct the homomorphic authenticator.

3. PRELIMINARIES

3.1 Large Branching Tree

Compared to Merkle Hash Tree (MHT), Large Branching Tree (LBT) allows to in-

crease the number of a node's children and decrease the depth of the tree. Each node of the tree except leaves has more than 2 children nodes. That is, each node has only one parent but has a number of sibling nodes. For the same number of leaf nodes, the depth of the constructed LBT is much smaller than that of the MHT. This is one difference between LBT and MHT. For concreteness, as shown in Fig. 1, we take the outdegree of the tree to be n, the height of the tree is l, then the number of leaf nodes is n^l and all nodes except the leaf nodes have n children nodes. Each leaf node in the tree corresponds to a hash value of a data block, and the non-leaf node is the link form of the hash value of all its children nodes. The depth of the tree is l=2 in Fig. 1. The number of leaf nodes is n^2, corresponding to the n^2 hash value of the data block. The value of node A_1 is $H(H(m_1)||H(m_2)||\ldots||H(m_n))$. The value of node A_i is $H(H(m_i)||H(m_{i+1})||\ldots||H(m_{i+n-1}))$. The value of node A_n and root node R is $H(H(m_n{}^2{}_{-n+1})||H(m_n{}^2{}_{-n+2})||\ldots||H(m_n{}^2))$, and $H(H(A_1)||H(A_2)||\ldots||H(A_n))$ respectively.

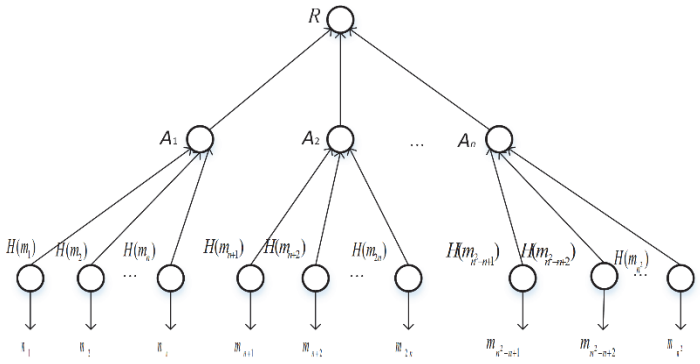


Fig. 1. LBT structure.

An authentication LBT scheme produces signatures that represent paths connecting data blocks to the root of the tree. The authentication mechanism works inductively: the root authenticates its children nodes, these nodes authenticate their children nodes, and the authentication proceeds recursively down to the data blocks authenticated by its parent [18]. Another difference between LBT and MHT is the way the sibling nodes are authenticated. The authenticity of a node can be verified given its parent and its authentication value, whose size is independent of the branching factor. In a LBT, each node is labelled a number to denote its position among silibings with a unique authentication value that can be verified independently. In our scheme, the way the sibling nodes are authenticated is different. Since every node has multiple brother nodes, we label them with a number to denote its position among siblings. And a unique authentication value that can be verified independently has been generated for the verification.

3.2 Dynamic PDP System

The dynamic PDP system for outsourced data in cloud consists of three entities: *Client*, who has limited storage resource and computational ability but large amount of data to be stored in the cloud; *Cloud Storage Server* (CSS), an entity which has huge storage space and is able to provide data maintenance and computation; *Third Party Au-*

ditor (TPA), who specializes in verifying the integrity of outsourced data in cloud when received a request from the client. The system model is shown in Fig. 2.

We assume the communication between any two of these three entities is reliable. The whole auditing scheme is based on *challenge-response* protocol, which contains three phases: first, the client completes initializing work and then hosts his/her data files in cloud; second, the client makes an update operation by communication with CSS; third, TPA and CSS work together to provide data auditing service through exchanging the challenge and proof messages. TPA would report the audit results to the client.

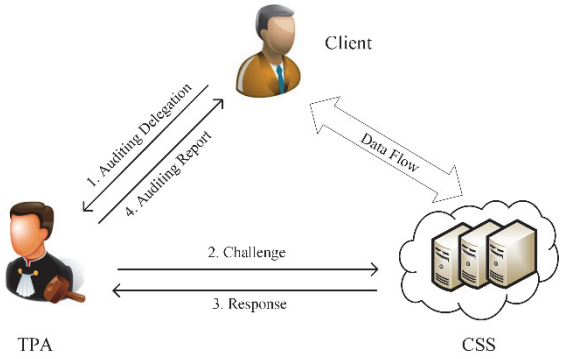


Fig. 2. System model.

Definition 1: In a *DPDP* system, the client, *CSS* and *TPA* cooperate with each other to accomplish the challenge-response procedure. A DPDP scheme consists of the following algorithms:

- $KeyGen(1^k) \rightarrow \{sk, pk\}$. This probabilistic algorithm is run by the client. It takes as input security parameter 1^k, and returns private key *sk* and public key *pk*.
- $TagGen(F, sk) \rightarrow \{T\}$. This algorithm is run by the client to generate the metadata. It takes as input the data file *F* and private key *sk* and outputs the tag sets *T*, which is a collection of signatures $\{\tau_i\}$ on $\{m_i\}$.
- $Update(F, Info, \Omega, pk) \rightarrow \{F', P_{update}\}$. This algorithm is run by *CSS* in response to an update request from *TPA*. As input, it takes the data file *F*, update information Info, the previous auxiliary information Ω and the public key *pk*. The output is the new version of the data file *F'* along with its proof P_{update}. *CSS* sends the proof to *TPA*.
- $VerifyUpdate(P_{update}, sk, pk) \rightarrow \{accept, regect\}$. This algorithm is run by *TPA* to verify *CSS* updated the data correctly. The input contains the proof P_{update} from *CSS*, the new file *F'* with its corresponding metadata *T'*, and the private and public keys. The output is accept if the proof is valid or reject otherwise.
- $Challenge(\bullet) \rightarrow \{chal\}$. TPA runs this algorithm to start a challenge and send the challenge information chal to *CSS*.
- $GenProof(F, T, chal, pk) \rightarrow \{P\}$. This algorithm is run by *CSS*. It takes data file *F*, metadata *T*, the challenge information *chal* and the public key as inputs, and outputs the proof for the verification.
- $VerifyProof(P, pk) \rightarrow \{accept, reject\}$. *TPA* run this algorithm to verify the response *P* from *CSS*. It outputs "*accept*" if the proof is correct, or "*reject*" otherwise.

3.3 Security of Dynamic PDP

Following the security model defined in [15, 23], we define the security model for our proposed DPDP scheme by a data possession game between a challenger C and an adversary A. The challenger plays the role of verifier and the adversary acts as a malicious CSS.

***KeyGen*:** The challenger runs $(pk, sk) \leftarrow KeyGen(1^k)$, then sends pk to the adversary.

***ACF Queries*:** The adversary can make adaptively chosen file (ACF) queries as follows. First, the adversary interact with the tag generation oracle $\mathcal{O}_{TG}$. For each query, A chooses a data block m_i and sends it to $\mathcal{O}_{TG}$. Then the oracle responds each query with a corresponding verification metadata $\tau_i \leftarrow (H(m_i) \cdot \omega^{m_i})^x$. The adversary keeps making n times queries. Then, it enables to create an ordered collection of metadata $T = \{\tau_i\}_{1 \le i \le n}$ for all the selected data blocks $F = \{m_1, m_2, ..., m_n\}$. Second, the adversary is given access to a data update oracle $\mathcal{O}_{UP}$. A chooses a data block m_i ($i = 1, 2, ..., n$) and generates corresponding update information $Info_i$ indicating what operation the adversary wants to perform. Then the adversary runs *Update* algorithm and outputs a new version of data file F' and an update proof P_{update}. After receiving these information submitted by the adversary, the oracle $\mathcal{O}_{UP}$ verifies the proof P_{update} by running algorithm *VerifyUpdate*. The output is *accept* or *reject*. The adversary can repeat the above interaction in polynomial times.

***Setup*:** The adversary decides on data block m_i^* and corresponding update information $info_i^*$ for all $i \in I \in [0, n + 1]$. The *ACF Queries* are performed again by the adversary, with the first $info_i^*$ specifying a full re-write (this corresponds to the first time the client sends a file to CSS). The challenger verifies the update information and update his local metadata.

***Challenge*:** The final version of data file F is created according to the data update requested by A, and verified then accepted by the challenger. Now the challenger generates a challenge *chal* and sends it to the adversary.

***Forge*:** The adversary computes a data possession proof P based on *chal*. Then the challenger runs algorithm *VerifyProof* and outputs the result belonging to *accept*/*reject*. If the output is *accept*, then the adversary wins.

Definition 2: We say that a *DPDP* scheme is secure if for any probabilistic polynomial time (*PPT*) adversary A (*i.e.*, malicious *CSS*), the probability that A wins the data possession game is negligible.

4. CONSTRUCTION

The main building blocks of our scheme include LBT, a secure signature scheme proposed by Boneh *et al.* [18] and Homomorphic Verifiable Tags (HTVs) [9]. LBT data structure is an expansion of MHT, which is intended to prove that a set of elements are

undamaged and unaltered [1]. Naturally, we consider employing the hash algorithm used in MHT structure to authenticate the values of nodes in LBT, but this algorithm brings undesirable effects on the performance. During the update process, that the client modify, insert, or delete the data if only for one block will affect the whole data structure, causing $O(n)$ computation overhead for both the client and CSS. Therefore, it is imperative to find a better method to authenticate LBT data structure. Instead of using hash functions, we employ the signature scheme [18] to improve the efficiency of verifying the elements in LBT. The computation complexity decreases to $O(1)$ in the update process. As for the public auditability, we resort to the homomorphic verifiable tags. The reason is that HVTs make it possible to verify the integrity of the data blocklessly.

The procedure of our scheme is summarized in three phase: Setup, Dynamic Operation and Periodical Auditing. The details are as follows:

4.1 Setup

In this phase, we assume the data file F is segmented into $\{m_1, m_2, ..., m_n\}$, where $n=q^l$ and q, l are arbitrary positive integers. Bilinear map $e:G\times G\rightarrow G_T$ is secure. Group G has a prime order p. Let g be the generator of G. $H:\{0, 1\}^*\rightarrow G$ is a family of collision-resistant hash functions. Note that all exponentiations in following algorithms are performed modulo p on G and for simplicity we omit writing "(mod p)" explicitly.

***KeyGen*(1^k)** The client runs this algorithm to generate a pair of private and public keys. Choose a random $x\leftarrow Z_p$ and compute $y=g^x$. Pick $\alpha_1, \alpha_2, \ldots, \alpha_q \leftarrow Z_p$ and $\lambda \leftarrow G$. Compute $\lambda_1 \leftarrow \lambda^{1/\alpha_1}$, $\lambda_2 \leftarrow \lambda^{1/\alpha_2}$, …, $\lambda_q \leftarrow \lambda^{1/\alpha_q} \in G$. Pick $\mu \leftarrow G$, $\beta_0 \leftarrow Z_p$, then compute $\nu = e(\mu, \lambda)$ and $e(\mu, \lambda)^{\beta_0}$ where η_0 denotes the root of LBT (the root of MHT is the hashes of all the nodes). And for every node in LBT tree, the client chooses $\{\beta_j\}_{1\le j\le n}$. The client also generates a random signing key pair (spk, ssk). The public key is $pk = \{y, \lambda, \nu, \mu, \{\alpha_i\}_{1\le i\le q}, \{\beta_i\}_{1\le j\le n}, spk\}$ and the private key is $sk = \{x, \beta_0, ssk\}$.

***TagGen*(F, sk)** For each data block m_i (i=1, 2, ..., n), the client chooses a random element $\omega \leftarrow G$, and computes a signature tag $\tau_i \leftarrow (H(m_i) \cdot \omega^{m_i})^x$. The set of all the tags is denoted by $T=\{\tau_i\}_{1\le i\le n}$. Then the client computes $\gamma = Sig_x(\eta_0)$ and sends $Ini = \{F, T, t, \gamma\}$ to CSS. Let $t = name||n||\omega||Sig_{ssk}(name||n||\omega)$ be the tag for file F. The client will then compute $sig = Sig_{ssk}(t)$ and sends sig along with the auditing delegation request to TPA for it to compose a challenge later on.

Upon receiving the initialize information Ini, CSS first stores all the data blocks, and then construct a LBT as follows: for the ith data block m_i (i = 1, 2, ..., n), CSS generates the ith leaf of LBT together with a path from the leaf to the root. We denote the leaf by $\eta_l \in G$, where l is the layer of the leaf and the nodes on its path to the root are (η_l, i_l, η_{l-1}, i_{l-1}, …, η_1, i_1), where η_j is the i_jth child of η_{j-1}, $1 \le j \le l$. The authentication values for these nodes are computed as follow steps:

Step 1: For every node on the path from leaf η_l to the root, CSS generates $\eta_j \leftarrow e(\mu, \lambda_{i_j})^{\beta_j}$.
Step 2: The authentication value of node η_j, the i_jth child of η_{j-1}, is $f_i \leftarrow \mu^{\alpha_{ij}(\beta_{i-1}+(\eta_j))}$.
Step 3: The authentication value of $H(m_i)$, the child of the leaf node η_l, is $f \leftarrow \mu^{\beta_l+H(m_i)}$.

Therefore, the signature on data block m_i is $\Omega_i = (f, f_l, i_l, \ldots, f_1, i_1)$ which is also the auxiliary information for authentication in the dynamic update process. The construction of LBT is illustrated in Fig. 3.

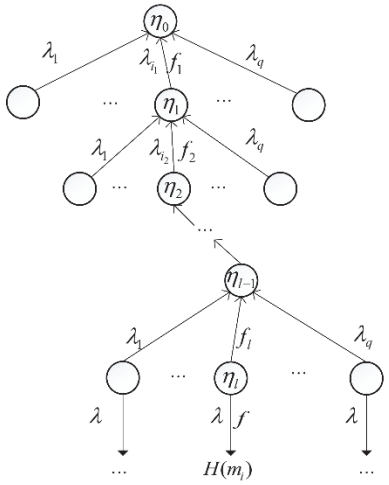


Fig. 3. Construction of LBT.

4.2 Dynamic Operation

(1) **Modification:** The client composes an update request $Info = (m, i, m_i', \tau_i')$, it denotes that the client wants to modify m_i to m_i', and $\tau_i' \leftarrow (H(m_i') \cdot \omega^{m_i'})^x$ is the signature of m_i'. Then he/she sends the update information Info to CSS.

***Update*(F, *Info*, Ω, *pk*)** Upon receiving the update request, CSS first modifies the data block m_i to m_i', and replaces the $H(m_i)$ with $H(m_i')$ in LBT. As shown in the Fig. 4, CSS generates the new authentication value $f' \leftarrow \mu^{\beta_l + H(m_i')}$ and updates the signature Ω into Ω′. Note that, CSS only consumes $O(1)$ computation overhead. Finally, CSS responds $P_{update} = (H(m_i'), \Omega', \gamma)$ to TPA.

***VerifyUpdate*(P_{update}, *sk*, *pk*)** TPA generates root η_0' based on $H(m_i')$, Ω′ as follows:
Step 1: Compute $\eta_l' \leftarrow e(f', \lambda) \cdot v^{H(m_i')}$.
Step 2: Computes $\eta_{j-1}' \leftarrow e(f_j', \lambda_{ij}) \cdot v^{H(\eta_j')}$ for $j = l, \ldots, 1$.
Step 3: The proof is accepted if $e(\gamma, g) = e(\eta_0', y)$ or otherwise rejected.

(2) **Insertion:** As the insert operation would change the structure of LBT, this process is different from data modification. We assume the client wants to insert block m^* after the ith block m_i. First, the client generates a tag $\tau^* \leftarrow (H(m^*) \cdot \omega^{m^*})^x$. Then the client chooses two parameters $\beta_{l+1}, \beta_{l+1}^*$ and sends an update request $Infor = (i, m^*, \tau^*, \beta_{l+1}, \beta_{l+1}^*)$ to CSS.

***Update*(F, *Info*, Ω, *pk*)** Upon receiving the update information, CSS updates data files and turns the leaf node η_l into a father node whose first child node is η_{l+1} and the second

one is η^*_{l+1}. Data blocks m_i and m^* are authenticated with respect to the leaves η_{l+1} and η^*_{l+1}. As shown in the Fig. 4, CSS computes the authentication values f_{l+1} and f^*_{l+1} by η_{l+1}. and η^*_{l+1} respectively. The authentication values of the two blocks are computed as $f \leftarrow \mu^{\beta_{l+1}+H(m_i)}$ and $f^* \leftarrow \mu^{\beta^*_{l+1}+H(m^*)}$. Finally, CSS responses TPA with a proof $P_{update} = \{(\Omega'_i, H(m_i)), (\Omega^*, H(m^*)), \gamma\}$. The process is shown in Fig. 5.

***VerifyUpdate*(P_{update}, *sk, pk*)**. This process is similar to the update verification process in modification operation except that the data blocks and the auxiliary information are different.

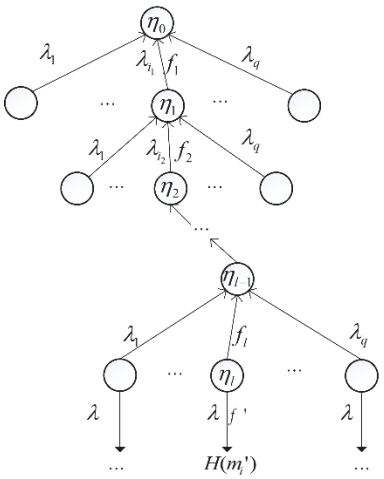


Fig. 4. LBT update under modification.

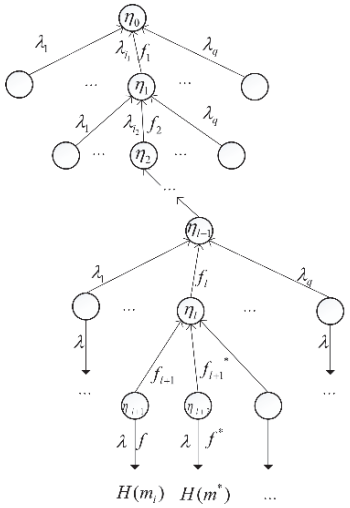


Fig. 5. LBT update under insertion.

(3) **Deletion:** Suppose the client wants to delete the block m_i. The update process is very simple. The only thing CSS needs to do is deleting m_i from its storage space and taking out the $H(m_i)$ from LBT structure. The subsequent update verification process is same as **Modification** operation. Since other nodes are not affected by deleting leaf node, CSS needn't recalculate node values on authentication path. Therefore, LBT don't need to be reconstructed and computation overhead of CSS was reduced significantly.

4.3 Batch Dynamic Operation

In MHT-based scheme, dynamic update applies to only *single* data block. For each insertion, the new data block should be no more than 160 bits, otherwise the block needs to be divided into equal small blocks and be inserted separately. Moreover, each update operation leads to reconstruction of the MHT structure, which costs huge computation overhead. On the contrast, our scheme support *batch dynamic* data updates, which was not discussed in [2]. The LBT structure allows multiple child nodes attached to one father node. Therefore, when the client request to update a large block, CSS only needs to update several corresponding leaves while the whole structure remain unchanged. Specifically, our scheme allows insertion of data block with no more than $160(q-1)$*bits*. The *insertion* process is as follows (shown in Fig. 6):

Suppose client wants to insert data f (size is $160(q-1)bits$) after data block m_i. Data that are smaller than $160(q-1)bits$ can be padded to the length by appending null data blocks. First, client divides this data into data blocks $f = (m'_1, m'_2, \ldots, m'_{q-1})$, and generates authentication tags $\tau'_i \leftarrow (H(m'_i) \cdot \omega^{m_i})^x$ for each block $1 \le i \le (q-1)$. Then client chooses parameters β_{l+1}, $\beta_{(l+1,1)}$, …, $\beta_{(l+1,q-1)}$ and sends an update request $Info = (i, f, \tau'_i, \beta_{l+1}, \beta_{(l+1,1)}, \ldots, \beta_{(l+1,q-1)})$ to CSS.

***Update*(F, Info, Ω, pk)** Upon receiving the update information, CSS updates data files and turns the leaf node η_l into a father node whose first child node is η_{l+1} and following child nodes are $\eta_{(l+1,1)}$, …, $\eta_{(l+1,q-1)}$. Data blocks m_i and $m'_1, m'_2, \ldots, m'_{q-1}$ are authenticated with respect to the leaves η_{l+1} and $\eta_{(l+1,1)}$, …, $\eta_{(l+1,q-1)}$. As shown in the Fig. 6, CSS computes the authentication values f_{l+1} and $f_{(l+1,1)}$, …, $f_{(l+1,q-1)}$ by η_{l+1} and $\eta_{(l+1,1)}$, …, $\eta_{(l+1,q-1)}$ respectively. The authentication values of these blocks are computed as $f \leftarrow \mu^{\beta_{l+1}+H(m_i)}$ and $f'_1 \leftarrow \mu^{\beta_{(l+1,1)}+H(m'_1)}$, …, $f'_{q-1} \leftarrow \mu^{\beta_{(l+1,q-1)}+H(m'_{q-1})}$. Finally, CSS responses TPA with a proof

$$P_{update} = \{(\Omega'_i, H(m_i)), (\Omega'_1, H(m'_1)), \ldots, (\Omega'_{q-1}, H(m'_{q-1})), \gamma\}.$$

***VerifyUpdate*(P_{update}, sk, pk)** This process is similar to the update verification process in modification operation except that the data blocks and the auxiliary information are different.

Batch modification is similar to the modification of single block in Section 4.2, except that multiple leaf nodes update at the same time. Besides, batch deletion is the inverse of batch insertion. Since the operation process is simple, the concrete update process is omitted here.

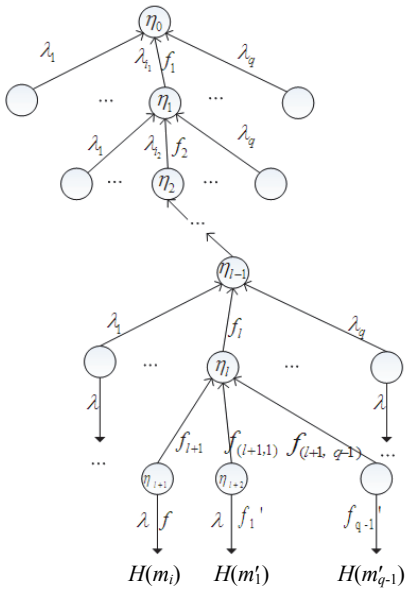


Fig. 6. Batch insertion.

4.4 Auditing

After the Setup process, no matter whether the update operation is executed or not, the integrity verification is available for TPA to perform his/her duty as an auditor. The integrity verification process is a challenge-response protocol, TPA generates a challenge information *chal* and sends it to CSS. CSS responds with a proof *P*. Then TPA verifies the correctness of the proof and outputs *accept*/*reject*.

***Challenge*(•)** Before challenging, TPA first use *ssk* to verify the signature on *t* to recover ω. Suppose TPA wants to challenge *c* blocks. The indexes of these blocks are randomly selected from [1, *n*]. Namely, let $I = \{i_1, i_2, \ldots, i_c\}$ be the indexes of the challenged blocks. For each $i \in I$, TPA chooses a random element $\pi_i \leftarrow Z_p$. TPA then sends $chal = \{(i, \pi_i)_{i \in I}\}$ to CSS.

***GenProof*(*F*, *T*, *chal*, *pk*)** Upon receiving the challenge, CSS takes the data *F*, tags *T* and challenge information *chal* as inputs, and outputs: $\varphi = \sum_{i \in I} \pi_i m_i$ and $\tau = \sum_{i \in I} \tau_i^{\pi_i}$.

Moreover, CSS also provides TPA with the auxiliary information $\{\Omega_i\}_{i \in I}$, which denotes the authentication path from the challenged data blocks to the root. CSS sends proof $P = \{\varphi, \tau, \{H(m_i)\}, \{\Omega_i\}_{i \in I}, \gamma\}$ to TPA.

***VerifyProof*(*P*, *pk*)** For each challenged block m_i, $i \in I$, TPA first use the auxiliary information to reconstruct the nodes $\eta_l, \eta_{l-1}, \ldots, \eta_0$ in a bottom-up order by the following steps:

Step 1: Compute $\eta_l \leftarrow e(f, \lambda) \cdot v^{H(m_i)}$.
Step 2: For $j = l, l-1, \ldots, l$, compute $\eta_{j-1} \leftarrow e(f_j, \lambda_{ij}) \cdot v^{H(\eta_j)}$.
Step 3: Verify $e(\gamma, g) = e(\eta_0, y)$.

If the equality in step 3 holds, TPA continues to verify $e(\tau, g) = e(\sum_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{\varphi}, y) \cdot$ If so, the proof is accepted, otherwise rejected.

5. CORRECTNESS AND SECURITY

Correctness The correctness of our scheme is that both the proof generated for dynamic auditing and integrity checking passes the verification algorithm. The correctness of the proof for dynamic auditing is easy to prove. Indeed, Step 1 of the verification algorithm results in

$$e(f, \lambda) \cdot v^{H(m_i)} = e(\mu^{\beta_l + H(m_i)}, \lambda) \cdot e(\mu, \lambda)^{-H(m_i)} = e(\mu^{\beta_l}, \lambda) = \eta_l.$$

For any $j \in \{l, l-1, \ldots, 1\}$, the result of computation in step 2 of the verification algorithm is

$$e(f_j, \lambda_{ij}) \cdot v^{H(\eta_j)} = e(\mu^{\alpha_{ij}(\beta_{j-1} + H(\eta_j))}, \lambda^{1/\alpha_{ij}}) \cdot e(\mu, \lambda)^{-H(\eta_j)}$$

$= e(\mu^{\beta_{j-1}+H(\eta_j)}, \lambda) \cdot e(\mu, \lambda)^{-H(\eta_j)}, \lambda) = e(\mu^{\beta_{j-1}}, \lambda) = \eta_{j-1}$

The proof for integrity checking is also based on the properties of bilinear maps.

$$e(\tau, g) = e(\sum_{i \in I} (H(m_i) \cdot \omega^{m_i})^{x\pi_i}, g) = e(\sum_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{m_i \pi_i}, g^x) = e(\sum_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{\varphi}, y)$$

Now we show that our proposed scheme is secure in the random oracle model. The security of our scheme is depending on responding correctly generated proof. We divide the security analysis of our scheme into two parts:

(1) Prove that if the challenger accepts the proof $P = \{\varphi, \tau, \{H(m_i), \Omega_i\}_{i \in I}, \gamma\}$, where τ denotes the tag proof which aggregates some forged tags for all the challenged blocks, the Computational Diffie-Hellman (CDH) problem is tractable within non-negligible probability.
(2) Prove that if the challenger accepts the proof $P = \{\varphi, \tau, \{H(m_i), \Omega_i\}_{i \in I}, \gamma\}$, where φ denotes the data proof generated by the adversary with all the challenged blocks $\{m_i\}_{i \in I}$, the Discrete Logarithm (DL) problem is tractable within non-negligible probability.

Security During the analysis of existing schemes, we found that different schemes have different security levels. We classify some typical schemes' security level by their key techniques. Most of MAC-based schemes are semantically secure. RSA-based schemes and BLS-based schemes are both provably secure since they rely on public keys. Like most homomorphic tag-based schemes, our scheme is provably secure in the random oracle model.

Theorem 1: If the tag generation scheme we use is existentially unforgeable, *CDH* problem and *DL* problem is intractable in bilinear groups in the random oracle model, there exists no adversary against our provable data possession scheme could cause the verifier to accept a corrupted proof in the challenge-verify process, within non-negligible probability, except by responding the correctly computed proof.

***Proof*:** We firstly prove that the tag generation scheme is existentially unforgeable with the assumption that BLS short signature scheme is secure. We prove this by reduction. Assume BLS signature scheme is secure and its public key is $pk = g^x$. If there exists an adversary who can win the challenge game with non-negligible probability, then the adversary must be able to forge a signature in BLS scheme. Pick $x \leftarrow Z_p$, and compute $u=g^x$. When the adversary queries about a data block m_i, he/she sends the block to BLS signature oracle, and the oracle responds with the signature $s_i=H(m_i)^x$. The adversary queries the oracle about the same block in our scheme, and be replied with the tag $\tau_i = (H(m_i) \cdot \omega^{m_i})^x$. Let $\omega = g^\alpha$, then $\tau_i = s_i \cdot \mu^{\alpha m_i}$. Suppose that the adversary can forge a new tag $\tau_j = (H(m_j) \cdot \omega^{m_j})^x$ for the block m_j that has never been queried. Therefore, the adversary can compute BLS signature on m_j as $s_j = \tau_j / \mu^{\alpha m_j}$. This completes the proof of the security of the tag generation scheme.

Now we prove the Theorem 1 by using a sequence of games.

Game 1: The first game is the data possession game as we defined in section 3.3.
Game 2: Game 2 is the same as Game 1, with one difference. When the challenger responds the *ACF Queries* made by the adversary, he/she keeps a list of all his/her responses. Then the challenger observes each instance of the challenge-response process with the adversary. If in any of these instances the adversary responds a valid proof which can make the challenger accept, but the adversary's tag proof is not equal to the $\tau = \prod_{i \in I} \tau_i^{\pi_i}$, which is the expected response that would have been obtained from an honest prover, the challenger declares *reject* and aborts.

Analysis Before we analyzing the difference in probabilities between Game 1 and Game 2, we firstly describe the notion and draw a few conclusions. Suppose the data file that causes the abort is divided into n blocks, and the tags of data blocks are $\tau_i = (H(m_i) \cdot \omega^{m_i})^x$ for $i \in [1, n]$. Assume $chal = \{i, \pi_i\}_{i \in I}$ is the query that causes the challenger to abort, and the adversary's response to that query was $P' = \{\varphi', \tau', \{H(m_i), \Omega_i\}_{i \in I}, \gamma\}$. Let the expected response be $P = \{\varphi, \tau, \{H(m_i), \Omega_i\}_{i \in I}, \gamma\}$. The correctness of $H(m_i)$ can be verified through $\{H(m_i), \Omega_i\}_{i \in I}$ and γ. Because of the correctness of the scheme, the expected response can pass the verification equation, that is

$$e(\tau, g) = e(\prod_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{\varphi}, y).$$

Because the challenger aborted, we know that $\tau \neq \tau'$ and that τ' passes the verification equation $e(\tau', g) = e(\prod_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{\varphi'}, y)$. Observe that if $\varphi' = \varphi$, it follows from the verification equation that $\tau = \tau'$, which contradicts our assumption above. Therefore, it must be the case that $\Delta\varphi$ is nonzero, here we define $\Delta\varphi = \varphi' - \varphi$.

With this in mind, we show that if the adversary win Game 2 and causes the challenger to abort, we can construct a simulator to solve CDH problem.

Given the values g, g^x, $h \in G$ as inputs, the goal of the simulator is to output h^x. The simulator behaves like the challenger in Game 2 and interacts with the adversary as follows:

(1) To generate a tag key, the simulator sets the public key y to g^x, and then forwards y to the adversary.
(2) The simulator programs the random oracle H and keeps a list of queries to respond consistently. Upon receiving the adversary's queries, the simulator chooses a random $r \leftarrow Z_p$ and responds with $g^r \in G$. It also responds queries of the form $H(m_i)$ in a special way, as we will see below.
(3) When requested to store the data file which is divided into n blocks $\{m_i\}_{1 \leq i \leq n}$, the simulator responds as follows. It firstly chooses a random block m_i. For each $1 \leq i \leq n$, the simulator chooses a random value $r_i \leftarrow Z_p$ and sets $\omega = g^a h^b$ for $a, b \leftarrow Z_p$, then it outputs $H(m_i) = g^{r_i} h^{-m_i}$. Therefore, the simulator can compute the tag $\tau_i = (H(m_i) \cdot \omega^{m_i})^x = (g^{r_i} h^{-m_i} \cdot (g^a h^b)^{m_i})^x$.
(4) The simulator continues interacting with the adversary until the adversary succeeds in responding with a tag τ' that is not equal to the expected tag τ. After receiving the valid proof P' from the adversary, the simulator is able to compute $e(\tau'/\tau, g) = e(\omega^{\Delta\varphi}, g) = e((g^a h^b)^{\Delta\varphi}, g)$.

Rearranging terms yields $e(\tau' \tau^{-1} y^{-a\Delta\varphi}, g) = e(h, y)^{b\Delta\varphi}$.

Since $y=g^x$, we obtain $h^x = (\tau' \tau^{-1} y^{-a\Delta\varphi})^{\frac{1}{b\Delta\varphi}}$. To analyze the probability that the challenger aborts in the game, we only need to compute the probability that $b\Delta\varphi = 0 (\text{mod } p)$. Because b is chosen by the challenger and hidden from the adversary, the probability that $b\Delta\varphi = 0 (\text{mod } p)$ will be only $1/p$, which is negligible.

Therefore, if there is a non-negligible difference between the adversary's probabilities of success in Games 1 and 2, we can construct a simulator that solves CDH problem by interacting with the adversary.

Game 3: Game 3 is the same as Game 2, with one difference. When the challenger responds the *ACF Queries* made by the adversary, he keeps a list of all his responses. Then the challenger observes each instance of the challenge-response process with the adversary. If in any of these instances the adversary responds a valid proof which can make the challenger accept, but the adversary's data proof is not equal to the $\varphi = \prod_{i \in I}$, which is the expected response that would have been obtained from an honest prover, the challenger declares *reject* and aborts.

Analysis Again, let us describe some notation. Suppose the data file that causes the abort is divided into n blocks. Assume $chal = \{i, \pi_i\}_{i \in I}$ is the query that causes the challenger to abort, and the adversary's response to that query was $P' = \{\varphi', \tau', \{H(m_i), \Omega_i\}_{i \in I}, \gamma\}$.

Let the expected response be $P = \{\varphi, \tau, \{H(m_i), \Omega_i\}_{i \in I}, \gamma\}$, among which the data proof should be $\varphi = \prod_{i \in I} \pi_i m_i$. Game 2 already guarantees that we have $\tau' = \tau'$. It is only the values of φ' and φ that can differ. Define $\Delta\varphi = \varphi' - \varphi$, again, it must be the case that $\Delta\varphi$ is nonzero.

We now show that if the adversary causes the challenger in Game 3 to abort with non-negligible probability, we can construct a simulator to solve DL problem.

Given the values g, $h \in G$ as inputs, the goal of the simulator is to output α such that $h = g^{\alpha}$. The simulator behaves like the challenger in Game 2 and interacts with the adversary as follows:

(1) When requested to store the data file which is divided into n blocks $\{m_i\}_{1 \le i \le n}$, the simulator first sets $\omega = g^a h^b$ for $a, b \leftarrow Z_p$. Then, it responds to the adversary according to the *TagGen* algorithm.

(2) The simulator continues interacting with the adversary until the adversary succeeds in responding with a data proof φ' that is not equal to the expected φ. After receiving the valid proof P' from the adversary, the simulator is able to compute

$$e(\prod_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{\varphi'}, y) = e(\tau', g) = e(\tau, g) = e(\prod_{i \in I} H(m_i)^{\pi_i} \cdot \omega^{\varphi}, y).$$

From this equation, we have $1 = \omega^{\Delta\varphi} = (g^a h^b)^{\Delta\varphi}$.

Thus, the solution to DL problem has been found, that is $h = g^{-\frac{a\Delta\varphi}{b\Delta\varphi}}$, unless the denominator is zero. However, $\Delta\varphi$ is not equal to zero, and the value of b is chosen by the challenger and hidden from the adversary, the probability that $b\Delta\varphi = 0 (\text{mod } p)$ will be only $1/p$, which is negligible.

Therefore, if there is a non-negligible difference between the adversary's probabilities of success in Games 2 and 3, we can construct a simulator that solves DL problem by interacting with the adversary.

Wrapping Up As we analyzed above, there is only negligible difference probability of the adversary between game sequences Game i (i=1, 2, 3), if the tag generation scheme is existentially unforgeable, CDH problem and DL problem are hard in bilinear groups. This completes the proof of Theorem 1.

6. PERFORMANCE

In this section, we analyze the performance of our scheme in the terms of storage overhead, computation cost and communication complexity.

6.1 Storage Overhead

Through analysis of the state-of-the-art, we find that what affects the storage overhead most is the metadata. For example, in [8], the verifier (the client) has to store the sentinels for verification. In [17], the verifier (the client) needs to store MACs.

In our scheme, the metadata is stored in CSS instead of the verifier (TPA). The client sends the metadata together with data to CSS during the setup phase. For each challenge, CSS responds both the data proof and the tag proof to TPA.

Table 1 shows the comparison of the storage overhead of different schemes. In the table, k denotes the total number of the sentinels, n denotes the total number of the data blocks, λ is the security parameter, p denotes the order of the group G and N is RSA modulus.

Table 1. Comparison of the storage overhead.

Schme	CSS	Verifier
[6]	$k \cdot \|sentinel\|$	$k \cdot \|sentinel\|$
[7]	$O(\lambda)$	$n \cdot \|N\|$
[13]	$O(\lambda)$	$n \cdot \|N\|$
[8](BLS)	$O(\lambda)$	$n \cdot \|p\|$
[1]	$O(\lambda)$	$n \cdot \|p\|$
Our scheme	$O(\lambda)$	$n \cdot \|p\|$

6.2 Computation Complexity

There are three entities in our scheme: the client, CSS and TPA. We discuss their computation cost respectively in different phase. In the setup phase, the client needs to compute 2 pairings, $2n$+2 exponentiations and n multiplications on G.

For better comparison, we implemented both our scheme and MHT-based scheme [1] in Linux. All experiments are conducted on a system with an Intel Core i5 processor running at 2.6GHz, 750MB RAM. Algorithms such as paring and SHA1 are employed by installing the Paring-Based Cryptography (PBC) library and the crypto library of OpenSSL. All experimental results represent the mean of 10 trials. Fig. 7 shows the pre-processing time as a function of block numbers for client. The MHT-based scheme

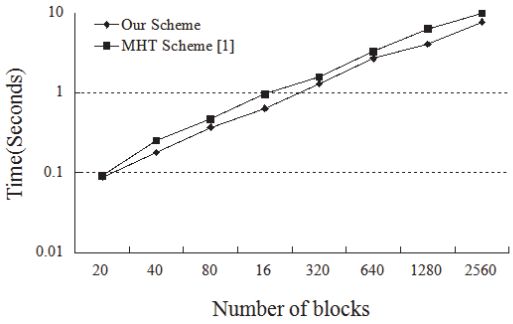


Fig. 7. Comparison of pre-processing time.

[1] exhibits slower pre-processing performance. Our scheme only performs an exponentiation on every data block in order to create the metadata. However, in scheme [1], client needs to perform the exponentiation as well as constructing a MHT to generate the root.

Besides, in the dynamic update phase, CSS only needs to compute 1 exponentiation in modification, 2 exponentiations in insertion and causes no computation in deletion. Note that the computation complexity of CSS in scheme [1] is $O(n)$ in all three update operations, where n is the number of data blocks. Therefore, the secure signature scheme based on bilinear maps [18] introduced in our scheme has greatly reduced the computation overhead during the dynamic update phase. In the auditing phase, TPA needs to do $2c$ summations and $2c$ multiplications, where c is the number of challenged data blocks. The computation complexity of TPA is $O(n)$.

6.3 Communication Cost

The main communication cost we concern is the communication cost between CSS and TPA during each challenge-response query. Since the metadata is stored in CSS, the proof sended from CSS to TPA is increased. There is a trade-off between the storage overhead and the communication cost. The major component of the communication cost is the proof sent to TPA by CSS. We compare our scheme with MHT scheme [1]. Fig. 8 shows the proof size as a function of the number of challenged blocks. Apparently, our

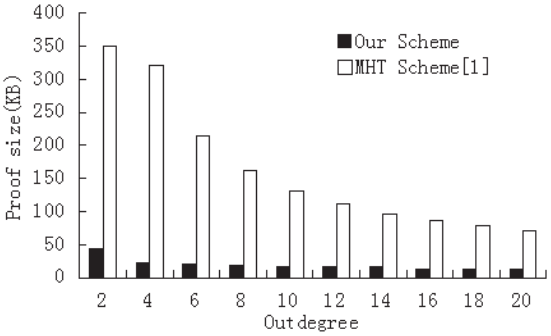


Fig. 8. Comparison of communication cost.

scheme causes less communication cost between CSS and TPA. The auxiliary information accounts for that gap. In our scheme, the size of auxiliary information grows linearly as the number of challenged blocks increase, while it grows exponentially as the number of challenged blocks increase in the MHT scheme [1].

7. CONCLUSION

In this paper, we propose an efficient dynamic provable data possession scheme based on a secure signature scheme [18] and LBT data structure. LBT structure enables reduction in size of auxiliary information, thereby causes less communication cost compared to MHT-based schemes. Moreover, the characteristics of bilinear pairings in the signature algorithm only cause computation cost on CSS for each dynamic update. And the client no longer needs to construct LBT structure to support dynamic operation. Therefore, our scheme reduces computation cost both on CSS and client as well as simplify the update process. Through security analysis and performance analysis, our scheme is provably secure and efficient.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for useful comments. This research is supported in part by National Natural Science Foundation of China under Grant Nos. 61472032, 61272522, 61572132, the Fundamental Research Funds for the Central Universities (No. 2016YJS003), and Fujian Provincial Key Laboratory of Network Security and Cryptology Research Fund (Fujian Normal University) (No. 15007), and Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201609).

REFERENCES

1. Q. Wang, C. Wang, J. Li, K. Ren, and W. J. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proceedings of ESORICS*, LNCS, Vol. 5789, 2009, pp. 355-370.
2. G. Yao, Y. Li, L. N. Lei, H. Q. Wang, and C. L. Lin, "An efficient dynamic provable data possession scheme in cloud storage," in *Proceedings of the 11th International Conference on Green*, *Pervasive and Cloud Computing*, LNCS, Vol. 9663, 2016, pp. 63-81.
3. P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST SP 800-145*, 2011. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.
4. Z. H. Xia, X. H. Wang, X. M. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, 2015, pp. 340-352.
5. Z. J. Fu, K. Ren, J. G. Shu, X. M. Sun, and F. X. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, 2016, pp. 2546-2559.

6. K. Yang and X. H. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, 2013, pp. 1717-1726.
7. Y. Zhu, G. J. Ahn, H. X. Hu, S. S. Yau, H. G. An, and S. Chen, "Dynamic audit services for outsourced storages in clouds," *IEEE Transactions on Services Computing*, Vol. 6, 2013, pp. 227-238.
8. A. Juels and B. S. Kaliski Jr., "PORs: proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 584-597.
9. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 598-609.
10. H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of ASIACRYPT*, Vol. 5350, 2008, pp. 90-107.
11. Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proceedings of the 6th Theory of Cryptography Conference*, LNCS, Vol. 5444, 2009, pp. 36-53.
12. G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Proceedings of ASIACRYPT*, LNCS, Vol. 5912, 2009, pp. 319-333.
13. K. Yang and X. H. Jia, "Data storage auditing service in cloud computing: challenges, methods and opportunities," in *Proceedings of WWW*, LNCS Vol. 15, 2012, pp. 409-428.
14. G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of ACM SecureComm*, 2008, pp. 1-10.
15. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of ACM International Conference on Computer and Communications Security*, 2009, pp. 13-222.
16. H. Wang, "Identity-based distributed provable data possession in multicloud storage," *IEEE Transactions on Services Computing*, Vol. 8, 2015, pp. 328-340.
17. M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," *Cryptology ePrint Archive*, 2008/186, http://eprint.iacr.org/2008/186.
18. D. Boneh, I. Mironov, and V. Shoup, "A secure signature scheme from bilinear maps," in *Proceedings of CT-RSA*, LNCS, Vol. 2612, 2003, pp. 98-110.
19. A. Barsoum and A. Hasan, "Enabling dynamic data and indirect mutual trust for cloud computing storage systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, 2013, 2375-2385.
20. C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Transactions on Services Computing*, Vol. 5, 2012, pp. 220-232.
21. H. Wang and D. He, "Proxy provable data possession with general access structure in public clouds," in *Proceedings of International Conference on Information Security and Cryptology*, LNCS, Vol. 9589, 2016, pp. 283-300.
22. C. Lin, F. Luo, H. Wang, and Y. Zhu, "A provable data possession scheme with data hierarchy in cloud," in *Proceedings of International Conference on Information Se-*

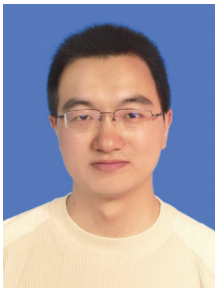
curity and Cryptology, LNCS, Vol. 9589, 2016, pp. 301-321.
23. C. Gritti, W. Susilo, and T. Plantard, "Efficient dynamic provable data possession with public verifiability and data privacy," in *Proceedings of Australasian Conference on Information Security and Privacy*, LNCS, Vol. 9144, 2015, pp. 395-412.
24. C. Gritti, W. Susilo, T. Plantard, and R. Chen, "Improvements on efficient dynamic provable data possession protocols with public verifiability and data privacy," *Cryptology ePrint Archive*, 2015/645, 2015, http://eprint.iacr.org/2015/645.
25. Y. Zhang and M. Blanton, "Efficient dynamic provable possession of remote data via update trees," *ACM Transactions on Storage*, Vol. 12, 2016, Article 9.
26. A. F. Barsoum and M. A. Hasan, "Provable multicopy dynamic data possession in cloud computing systems," *IEEE Transactions on Information Forensics and Security*, Vol. 10, 2015, pp. 485-497.
27. Y. Ren, J. Shen, J. Wang, J. Han, and S. Lee, "Mutual verifiable provable data auditing in public cloud storage," *Journal of Internet Technology*, Vol. 16, 2015, pp. 317-324.

Yong Li (李勇) received his M.S. degree in Computer Science from Wuhan University in 2003, and the Ph.D. degree from State Key Laboratory of Information Security, Graduate University of Chinese Academy of Sciences in 2007. Currently, he is an Associate Professor at the School of Electronic and Information Engineering, Beijing Jiaotong University. He has over 30 publications and filed 6 patents. His research interests include cryptographic protocols and cloud computing security.

Ge Yao (姚戈) received Master degree in Information Security from Beijing Jiaotong University in 2016. She will work toward the Ph.D. programme in University of Melbourne this year. Her research interest includes cryptography and cloud computing.

Li-Nan Lei (雷丽楠) received B.S. degree in Computer Science from Huaqiao University in 2014. Currently, she is an M.S. student at the School of Electronic and Information Engineering, Beijing Jiaotong University. Her research interests include cryptographic protocols and cloud computing security.

Hua-Qun Wang (王化群) received the BS degree in Mathematics Education from the Shandong Normal University and the MS degree in Applied Mathematics from the East China Normal University, both in China, in 1997 and 2000, respectively. He received the Ph.D. degree in Information Security from Nanjing University of Posts and Telecommunications in 2006. He is currently a Professor of Nanjing University of Posts and Telecommunications, China. His research interests include applied cryptography, network security, and cloud computing security.



Chang-Lu Lin (林昌露) received the Ph.D. degree in Information Security from the state key laboratory of information security, Graduate University of Chinese Academy of Sciences, P.R. China, in 2010. He was a Visiting Scholar in the Information Security Group at Royal Holloway, University of London from July 2011 to January 2012. He was a Visiting Scholar in the Division of Mathematical Science, School of Physical and Mathematical Sciences, Singapore Nanyang Technological University from February 2015 to February 2016. He is interested in cryptography and network security, and has conducted research in diverse areas, including secret sharing, secure multi-party computation, public key cryptography and their applications.