International Journal of Security and Its Applications
Vol.9, No.9 (2015), pp.21-32
http://dx.doi.org/10.14257/ijsia.2015.9.9.03

Public Data Auditing with Constrained Auditing Number for Cloud Storage

Guangyang Yang[1], Hui Xia[1], Wenting Shen[1], Xiuxiu Jiang[1] and Jia Yu[1,2*]

[1]*College of Information Engineering, Qingdao University, Qingdao 266071, China*

[2] *Shandong provincial Key Laboratory of Computer Network, Jinan 250014, China*

ateamguang@126.com, qduxiahui@163.com, shenwenting.0929@163.com, xifeng_smile@163.com, yujia@qdu.edu.cn

Abstract

As the popularity and the proliferation of cloud storage increase, data security is becoming one of the biggest concerns for users of cloud storage. How to preserve the data integrity, as one of the most important security aspects, has been a research hotspot in the field of cloud security. Many data auditing schemes for checking the data integrity have been presented, however, these schemes are based on the assumption that the third party auditor (TPA) is secure and trustworthy. If TPA becomes wicked, these schemes are easy to make cloud server suffer distributed denial-of-service (DDOS) attack. In order to deal with this problem, we propose an authorized auditing scheme with constrained auditing number in this paper. In our scheme, only authorized TPA can make valid challenges to cloud for data integrity checking. Moreover, the total auditing number that an authorized TPA can make is decided by the user. In our construction, a constrained auditing number is integrated into the authorization generated by user to achieve this property. Once the number of a TPA's auditing reaches the constraint, cloud server will not respond to this TPA's challenges, which literally rules out the threat of DDOS attack. Analysis and experimental results show the proposed scheme is secure and efficient.

Keywords: *Cloud storage, Data auditing, Constrained auditing number, DDOS attack*

1. Introduction

In the last decade, cloud computing, as a new terminology of computing model, has been brought up and widely used in many applications. This model becomes one of the most fundamental evolutions in computing paradigm and plays a major role in global economic growth. Cloud computing makes use of idle resources (computing, storage, network, etc.) on the Internet to provide low-cost, convenient, accessible anytime anywhere, scalable and pay-as-you-go services [1-2]. Because the development of computer hardware techniques leads to large capacity and powerful computation with lower prices, computing and storing resources are easier to be purchased and deployed than ever before. And the information technology revolutions, especially electronic commerce, greatly change people's life and assemble enormous resources on the Internet to achieve business requirements at peak time. These result in resources idleness most of the time. And the cloud computing is promoted by the motivation to leverage these resources more efficiently. Therein, cloud storage service allows enterprises, organizations and individuals to store their large-scale data in remote storage. By outsourcing data to cloud, users can get rid of the burden of maintaining data, and the capital expenditure for fundamental infrastructure and essential software [3]. Nowadays, we are in an era of information explosion,

* Corresponding author: yujia@qdu.edu.cn

ISSN: 1738-9976 IJSIA

thousands of data being produced per minute or even second. Both Enterprises and individuals are prone to outsourcing data, after they trade off costs between subscribing services and building storage warehouse. There are many enterprises benefiting from cloud storage service. For instance, the famous file hosting service provider "Dropbox" uses Amazon S3 (Simple Storage Service) to store clients' files. And so does the picture sharing platform "Instagram" that provides service for nearly 50 million users but only has 13 employees owing to outsourcing data [4].

Although there are many benefits that one can get from data outsourcing, users do not completely trust cloud. Considering data are no longer physically possessed by themselves, users might wonder whether their data are properly stored in cloud. Even though the infrastructure and the software of cloud are more powerful and reliable than those of individuals, data stored in cloud may still get corrupted or lost [15-17]. It is because the hardware of cloud platform may encounter unexpected failure, cloud administrator may conduct improper operations and crackers may launch attacks through unrevealed loopholes of system or software, etc. [18-19]. What is worse, to maintain reputation and financial profit, cloud service providers (CSPs) who manage cloud storage servers (CSSs) might hide these incidents from users. In addition, CSPs may deliberately remove data that user does not or rarely access for a long time to save storage space. These concerns are becoming the main obstacles hindering wide application of cloud storage service. Therefore, it is important to explore an efficient way to guarantee data integrity for the development of cloud.

The traditional data integrity checking methods, such as MAC and signature techniques, are not suitable to be directly applied to cloud environment, because they require retrieving the whole data back, which will cause huge communication overhead. To achieve efficient data integrity verification, researchers have proposed many schemes on data auditing [5-9]. For one data auditing procedure, firstly the user issues a challenge message that specifies chosen blocks to be checked. Then cloud server responds with a combined block and an aggregated authenticator as the proof according to the challenge message. As a result, the size of response message is dramatically reduced. Finally, the user validates the proof. However, some terminal users only have limited computing ability and communication bandwidth. Periodical auditing workload is difficult for these users to process. Hence, the concept of public data auditing was then brought up under different system and security models [6-13] that allowed the user to delegate tedious auditing task to a TPA. However, in above mentioned schemes, cloud does not authenticate the TPA's identity. This makes anyone can generate valid challenges, and CSS should respond to all of them. Malicious entity can utilize this feature to launch distributed denial-of-service (DDOS) attack on CSS. In data checking community, DDOS attack is the behavior that malicious entity sends massive challenge requests to CSS in a short time. The resource of CSS may be greatly occupied by responding to these challenges which makes CSS cannot respond to other normal service requests. The quality of cloud service is thus obviously degenerated [13]. Since the real time response is a crucial competiveness for online services, service provider may lose clients and money if instant service availability cannot be guaranteed.

In order to deal with above problem, C. Liu et al. proposed an authorized public auditing scheme in [13]. In their scheme, user computes a signature as authorization evidence and sends it to TPA. When TPA checks the integrity of the cloud data, it needs to show this authorization to CSS. If an auditor cannot provide valid authorization, CSS will not respond to the challenges from this auditor. However, their solution does not solve the problem fundamentally. In reality TPA can be hacked by crackers, performed improper operations by manager or bribed by malicious entity. Once TPA is compromised, the authorization information will be disclosed. Since anyone with authorization can issue valid challenges, CSS will still be degenerated by DDOS attack. In this paper, we propose a public data auditing scheme with constrained auditing number for data storage. The pro-

posed scheme can literally solve the above problem. In our scheme, a constrained auditing number is integrated into the authorization for TPA. When CSS receives an auditing requirement, it validates authorization and checks whether the completed auditing times exceeds the constrained auditing number. When TPA's authorization is exposed, only limited number of valid challenges can be made with this authorization. Even if TPA is malicious and launches DDOS attack, it cannot generate large scale valid challenges in a short time. The DDOS attack can be avoided because nobody can make massive requests in a short period. As analyzed in later chapter, our scheme works well defending malicious entities with negligible system overhead introduced.

The rest of this paper is organized as follows: Research background and related work are depicted in Section 2. Section 3 presents system model, security concerns and design objectives of our scheme. Section 4 describes the details of our scheme. In Section 5 we analyze scheme's security and evaluate its performance through experiments. Finally, Section 6 concludes this paper.

2. Related Work

To publicly validate integrity of cloud data, traditional well-studied cryptographic techniques, such as hash function and signature, were taken into consideration in the first place [20, 21]. Unfortunately, these techniques require possessing local data to verify the integrity which will cause large amounts of communication overhead for outsourced data. Therefore, researchers expected to explore more efficient ways to audit the data stored on cloud.

The notions of proof of retrievability (POR) and provable data possession (PDP) were proposed by Juels *et al.* [5] and Ateniese *et al.* [6] respectively, which made a big progress towards this target. In consideration of efficiency, these approaches adopt probabilistic technique to guarantee data intactness by randomly checking a fraction of data. In the scheme [5] spot checking and error correcting code are utilized to preserve availability and integrity of remote data, but public auditability is not supported in the main scheme. Although they referred to construct a public verifiable version of POR with Merkle tree, their solution only copes with encrypted data. Ateniese *et al.* [6] considered and realized public auditability in the PDP model using RSA-based homomorphic linear authenticator (HLA) technique to authenticate blocks, which enabled aggregation during integrity proof generation process. The aggregation of blocks and authenticators avoids remote data downloading and only consumes O(1) communication resource. However, due to the inherent drawback of RSA scheme, the size of public/private key pair and block authenticator is very large that leads to inefficient computation. Shacham et al. designed a compact POR scheme [7] to achieve the assurance of possession and availability on remote stored data by utilizing BLS signature which essentially decreased the size of system parameters. They presented two schemes and proved the security of these schemes based on the classic formal security model they formulated. One scheme is publicly verifiable and the other is privately verifiable.

Since cloud users often perform update operations on their data, how to support dynamic data operations is considered in data auditing schemes [8-11]. In [8], the authors came up with one scheme called scalable PDP based on symmetric-key cryptography to address this issue. However, their scheme did not support fully dynamic data operations, and imposed prefixed number of updates and challenges that user can perform. Later on, a fully dynamic PDP scheme was proposed by Erway *et al.* [9] that employed an authenticated data structure, rank based skip list, to support data updates, especially insertions. This scheme was proven secure in the standard model. Subsequent schemes were proposed to improve the efficiency of dynamic PDP, based on Merkle hash tree [10] and B+ tree [11], etc. Recently, as a new research direction, Y. Jia *et al.* [12] firstly considered key

exposure in cloud auditing and proposed an efficient solution by using binary tree structure to update user's key.

Nevertheless, all the mentioned schemes assume the auditor, either data owner itself or a delegated TPA, behaves decently, which becomes an exploitable vulnerability for malicious entities. Typically, in public auditing scheme any entity can obtain public parameters and make as many challenges as it wants. As pointed out by C. Liu *et al.* [13], this could be exploited by malicious entities to launch the DDOS attack. In order to resolve this problem, C. Liu *et al.* proposed that the user generates an authorization for delegated TPA. Only when a valid authorization is presented will server respond to challenges of TPA. Although it seems that their solution can tolerant DDOS attack, malicious entities can still get their way launching this attack if TPA gives away its authorization information. To fundamentally address this problem, we propose public data auditing scheme with constrained number for cloud storage in this paper.

3. Problem Statements and Preliminaries

3.1. Problem Analysis

The system model consists of three participating entities: data owner (DO), CSS and TPA. In Figure.1, we present a sketch of cloud storage architecture and interactions among involved entities. The CSS is server hosted in cloud and supervised by CSP to provide online storage services. The DO possesses massive data that are to be stored on CSS. A third party TPA, who has expertise and capability to do auditing task, is delegated by DO to check data integrity on behalf of DO. TPA periodically audits outsourced data on CSS and informs DO results.

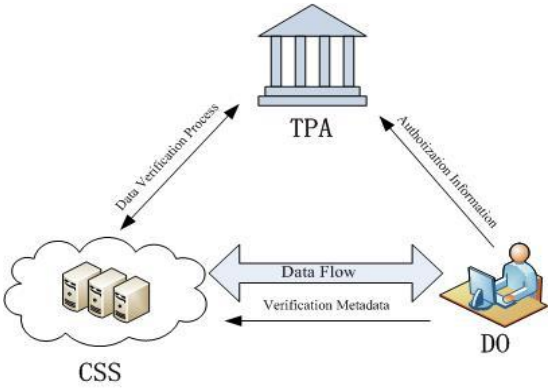


Figure 1. Architecture of Cloud Storage Service and Interactions among Entities

Due to public auditability, any entity can obtain public parameters, like public keys, and challenge CSS for data integrity proof. If a malicious entity controls masses of computers that needn't have much computing capability, then it can produce a challenge request flood to CSS in a short time and cause service degradation of CSS, i.e. DDOS attack. However, preserving high quality of service is critical for online service, since the long response latency or even being out of service is terrible for user which may result in the user financial loss. To protect CSS from DDOS attack, C. Liu et al. [13] proposed that DO delegates TPA for data verification with an authorization. When TPA conducts an auditing, it needs to present the authorization in challenge request to CSS for validation. Only when the authorization is valid will CSS generate proof to reply the request. However, CSS can still be affected by DDOS under this solution. Since TPA may be intruded by crackers, performed improper operations by managers or bribed by malicious entity, the authorization of TPA is thus revealed. Once malicious entities obtain authorization,

they can make valid challenges with no limitation again. In our construction, auditing number is proposed and integrated into authorization. Auditing number is the maximum challenge times that TPA can make for one data file, which is determined by DO and TPA. It is a practical scenario that DO pays TPA for auditing service and decides how many audit times according to the charging metric of TPA. On cloud side, the current audited number that how many challenges have been issued is recorded. When TPA makes a valid challenge, CSS will increase the current audited number. Once the current audited number reaches constrained auditing number, CSS will reject the challenge requests ever after.

3.2. Design Goals

For cloud server, data may get corrupted or lost because of internal and external attacks. CSPs are inclined to hide these incidents from users to maintain good reputation. Hence, CSP will try to cheat users when data auditing operation is performed. At this point, CSP may return a proof generated by uncorrupted blocks or even forged data. If TPA is compromised under attacks, authorization information may be disclosed. And malicious entity will attempt to modify the constrained auditing number in authorization to sufficient large for DDOS attack. Considering these security threats, the proposed scheme should have the following properties:

1) **Correctness**. Ensure that the cloud correctly storing DO's data can pass data auditing scheme.
2) **Security**. Whenever an adversary against the soundness of our verification scheme causes auditor to accept the proof it provides, there exits an extraction algorithm that can retrieve data back, except with negligible probability [7].
3) **The property of constrained auditing number**. CSS only responds to the TPA that is delegated by DO. And the number of data auditing that can be made with one authorization is limited. The constrained auditing number cannot be modified by malicious entities.
4) **Low computation and communication costs**. The computation cost for DO to generate authorization for TPA and for CSS to validate the authorization is constant and negligible. And the additional communication cost of checking parameters for TPA validation is very low, comparing to the total size of challenge message.

3.3. Preliminaries

(1) Bilinear Map: Let G and G_T be two multiplicative groups with prime order *p*. *G* is a gap Diffie-Hellman (GDH) group in which Computational Diffie-Hellman (CDH) problem is difficult while Decisional Diffie-Hellman (DDH) problem is easy. A map $e: G \times G \to G_T$ with following properties is called bilinear map:

1) Bilinearity: for all $u, v \in G_1$ and $a, b \in Z_p^*$, there is $e(u^a, v^b) = e(u, v)^{ab}$.

2) Non-degeneracy: $e(\mathrm{g}, \mathrm{g}) \neq 1$, *g* is a generator of *G*.

3) Computability: there is an efficient algorithm to compute the map *e*.

(2) BLS Signature: The BLS Signature scheme was first proposed by Boneh, Lynn and Shacham [14] on Asiacrypt 2001. This signature is described as follows:

1) Let *G* be a multiplicative group with prime order *p*, *g* be a generator of *G*. A signer randomly chooses $x \in Z_p$ and computes $v = g^x \in G$.
2) Assuming *M* is the message to be signed, the signer computes $\sigma = (H(M))^x$, where *H* is a hash function that maps string to group element: $\{0,1\}^* \to G$.
3) On receiving signature σ and message *M*, the receiver verifies whether σ is the corresponding signature of *M* by checking $e(H(M), v) \stackrel{?}{=} e(\sigma, g)$.

4. Proposed Scheme

4.1. High-level Technique Explanation

In order to deal with the problem that malicious entities can exploit public auditability to launch DDOS attack, restrictions on public auditing are necessary. In our design, the first restriction is to narrow down entities that can challenge data. DO needs to designate TPA for data auditing and CSS only responds to the challenges from these TPAs: DO generates an authorization for designated TPA and TPA sends challenge messages with the authorization to CSS; then CSS sends proof to TPA if the authorization is valid. In this way, only partial TPAs can make valid challenges.

Then auditing number is integrated into authorization as further restriction. That is, after a certain number of challenges are made the authorization becomes invalid. The auditing number is determined by DO and TPA based on financial agreement. This number is put into the authorization. When a challenge message comes, CSS will record how many challenge message that have been made with this authorization. If the recorded number reaches the constrained auditing number, this authorization becomes invalid and CSS will not respond to the challenge with it any more. With limited challenge number, even if TPA gives away its authorization message, malicious entity cannot notably degrade service quality of cloud. Thus this solution literally protects CSS from DDOS attack.

4.2. Scheme Details

Our construction consists of six algorithms: **KeyGen**, **SigGen**, **AuthGen**, **Challenge**, **ProofGen**, and **VerifyProof**. In Setup phase, **KeyGen** algorithm generates system parameters, e.g. public/private key pairs for DO and TPA; DO runs **SigGen** algorithm to preprocess outsourcing data file and generate verification metadata; in **AuthGen** algorithm, DO computes the authorization for the designated TPA. In Verification phase, firstly, TPA runs **Challenge** algorithm to generate a challenge message to specify the checked data blocks; and the CSS responds to challenge with an integrity proof through **ProofGen** algorithm; at last, TPA runs **VerifyProof** algorithm to verify whether the proof is valid. Now we give the detail of our design as follows.

(1) Setup Phase: Let G, G_T be two multiplicative groups with large prime order *p* and $e: G \times G \to G_T$ be a bilinear map. Let *g* be a generator of group *G* and $H: \{0,1\}^* \to G$ be a collision-resistant hash function.

KeyGen(1^κ): This algorithm takes security parameter κ as input and outputs public and private parameters of the system. Firstly, DO chooses signing key pair (*ssk, spk*) corresponding to a provable secure signature scheme, whose singing and verifying algorithms are denoted as *Sign()* and *Verify()* respectively. This signature scheme is selected to compute and validate authorization of delegated TPAs. Then, DO randomly chooses $\alpha \in_R Z_p$, and computes $v \leftarrow g^\alpha$. Let $SK_{DO} = (ssk, \alpha)$ be the secret key and $PK_{DO} = (spk, v)$ be the public key of DO. Similarly, TPA randomly chooses $x \in_R Z_p$ as its secret key SK_{TPA} and computes $X \leftarrow g^x$ as its public key PK_{TPA}.

SigGen(F, SK_{DO}): Given data file *F* and secret key of DO, this algorithm generates metadata for data integrity verification. We denote the outsourcing data file as $F = \{m_1, \cdots, m_n\}$ which is split into *n* blocks in sequence, and each blocks belongs to Z_p. DO chooses a random element from Z_p as file name *name* which should be unique for one storage domain. To generate verification metadata, DO randomly chooses another

generator $u \in_R G$ of group *G*, and computes authenticator for each data block m_i using his private key SK_{DO}:

$\sigma_i \leftarrow (H(name \| i) \cdot u^{m_i})^{\alpha}, 1 \le i \le n.$

"||" denotes concatenation operation. The signature set is denoted as $\Phi = \{\sigma_i\}_{i \in [1,n]}$. Then DO sets $t_0 = name \| n \| u$ and computes the file tag of *F* with t_0 using secret signing key *ssk*: $t = t_0 \| Sign_{ssk}(t_0)$. Finally, DO outputs $\{F, \Phi, t\}$.

AuthGen (N, t, PK_{TPA}, SK_{DO}): This algorithm takes as input constrained auditing number *N*, a file tag *t*, public key of TPA and secret key of DO. *N* is the maximum auditing times TPA can launch on one data file whose file tag is *t*. Moreover, *N* is predetermined by negotiation between TPA and DO based on the cost that DO would like to pay to TPA for data auditing. Then DO computes an authorization for TPA by using the signature scheme chosen in key generation process: $auth = Sign_{ssk}(AUTH \| N \| PK_{TPA})$. AUTH is a string that is used to represent authorization operation and PK_{TPA} is used to identify TPA. Finally, this algorithm outputs *auth*.

In the end, DO sends data file *F* along with $\{\Phi, t, AUTH, N, PK_{TPA}\}$ to CSS, and sends $\{auth, t, AUTH\}$ to TPA.

(2) Verification phase: TPA firstly validates whether the authorization from DO is valid by using *Verify( )* algorithm with input PK_{DO}, and *AUTH*, *N*, and PK_{TPA}; if it is not valid, TPA keeps asking for another authorization until he gets a valid one or rejects the mission either. Then, TPA uses DO's public signing key *spk* to verify whether the file tag *t* is valid; if the signature on *t* is invalid, TPA rejects this auditing request. Otherwise, TPA recovers *name*, *n* and *u* from the file tag *t*.

Challenge (l, auth, SK_{DO}): This algorithm takes as input blocks numbers *l*, authorization *auth* and secret key of DO SK_{DO}. *l* denotes the number of blocks that are going to be checked in this challenge. TPA picks a random *l*-element subset out of [1, *n*] as the chosen block index set *I*, and chooses a random coefficient $v_i \in Z_p$ for each *i* in *I*. TPA computes $sig_{TPA} = (H(AUTH \| N))^x$ using its secret key. The challenge message consists of the authorization from DO, the signature of TPA and *l* index-coefficient pairs, which is denoted as $chal = \{auth, sig_{TPA}, \{(i, v_i)\}_{i \in I}, PK_{TPA}\}$. Then TPA sends the challenge message *chal* to CSS.

ProofGen ($F, \Phi, chal, \mathrm{sig}_{TPA}, auth$): On receiving the challenge message, CSS firstly authenticates the identity of TPA by checking $e(g, sig_{TPA}) = e(PK_{TPA}, H(AUTH \| N))$. If the equation holds, CSS can affirm TPA is legal; otherwise, CSS does not reply this challenge. Then, CSS validates the authorization of TPA by using the *Verify()* algorithm of the chosen signature scheme in key generation process. CSS will not respond to the challenge once the validation failed. At last, CSS checks whether the current audited number reaches the max auditing number restrained in authorization. If reaches, CSS does not reply this auditing request; otherwise CSS increases current audited number. Only when all above checks pass will CSS generate integrity proof. CSS computes a linear combination of blocks and an aggregation of signatures specified by *chal*: $\mu = \sum_{i \in I} v_i m_i, \sigma = \prod_{i \in I} \sigma_i^{v_i}$. Finally, CSS replies the challenge with integrity proof $P = \{\mu, \sigma\}$.

VerifProof(P, PK_{DO}, chal): After receives response from CSS, TPA validates the proof to estimate data intactness by checking if the following equation holds: $e(g, \sigma) = e(v, \prod_{i \in I} H(name \| i)^{v_i} \cdot u^{\mu})$. If the equation holds, TPA can affirm that the challenged data are properly stored.

5. Analysis and Experimental Result

In this section, we analyze the security and efficiency of our proposed scheme, and then demonstrate the performance of our scheme by experimental results.

1) Correctness

From right hand side (RHS) of the verification equation, we can deduce left hand side.

$$\begin{aligned} RHS &= e(v, \prod_{i\in I} H(name \| i) \cdot u^{\mu}) \\ &= e(g^{x}, \prod_{i\in I} H(name \| i)^{v_i} \cdot \prod_{i\in I} u^{v_i m_i}) \\ &= e(g, \prod_{i\in I} (H(name \| i) \cdot u^{m_i})^{xv_i}) \\ &= e(g, \prod_{i\in I} \sigma_i^{v_i}) \\ &= e(g, \sigma) = LHS \end{aligned}$$

Since the proof response is derived from the data, as long as the data stored by remote CSS are indeed intact, we can affirm that this verification equation holds.

2) Security

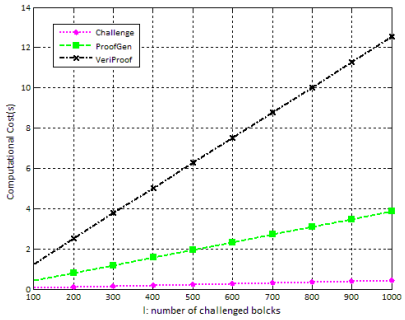
In the verification phase, a malicious CSS would try to cheat auditor about data storage condition by responding with proof derived from uncorrupted or even forged block and authenticator pairs. Our scheme can prohibit this behavior, i.e. any cheating proof cannot pass this verification phase. The construction of our data auditing scheme is similar with the framework in [7]. We add some necessary verification for the legality of TPA. Similarly to the security analysis in [7], whenever an adversary against the soundness of our verification scheme causes auditor to accept the proof it provides, we can construct an extraction algorithm that can retrieve data back, except with negligible probability. We omit the detailed proof here.

3) The property of constrained auditing number

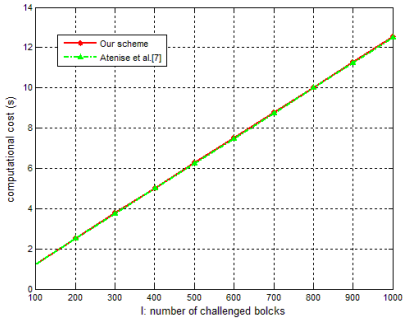
When CSS receives an auditing request from delegated TPA, it compares the current audited number with constrained number implicitly declared in authorization. Once the current audited number exceeds constraint, CSS will not respond to the challenges with this authorization afterwards. In the challenge messages, there is a signature signed by TPA using its secret key. CSS validates this signature to authenticate the TPA's identity. If it is invalid, CSS will reject the auditing request. In our scheme, this signature is constructed from BLS short signature [13]. According to the security of BLS signature scheme, it cannot be forged by malicious entity, except with negligible probability. And the identity of TPA is integrity into authorization to determine whether challenging TPA is delegated by DO. Therefore, no malicious entity in disguise can convince CSS as a delegated TPA. Moreover, the number of data auditing that an authorization can make is limited, because of the constrained auditing number in authorization. Since the authorization is generated using a proven existentially unforgeable signature scheme, no malicious entity can change the constrained auditing number with except with negligible probability. Hence, even if the authorization information of one TPA is exposed, the maximum auditing times that a malicious entity can make stays unchanged. Therefore, our scheme has the property of constrained auditing number. Because the constrained auditing number is determined by DO and TPA based on financial cost, it would not be very large. Nobody can generate large-sale challenges in a short time to significantly degrade service quality of CSS. Therefore, DDOS attack is thus avoided literally.

4) Low computation and communication costs

Similarly with the schemes in [7-10], our construction can also verify data integrity without retrieving back the whole data blocks. To demonstrate the performance of our scheme, we implement our scheme based on Pairing-Based Cryptography (PBC) library [23]. PBC is a convenient C library built on GMP library that performs the mathematical operations underlying pairing-based cryptosystems. We deploy the system on a server running Linux OS with Intel Pentium G630 2.70GHz processor, 4GB memory. In the experiments, super singular curve $Y\hat{}2 = X\hat{}3 + X$ and 160-bit group order of curve group are utilized, achieving 80-bit security level with 512 bits base field size and embedding degree 2. Hence, in our experiments the block size is 160 bits and the outsourcing data size we choose is 20MB. All experimental results are the mean of 20 trials.



(a) Computation cost of each algorithm in verification phase with regard to different choice of challenged blocks from 0 to 1000 increasing by 100.



(b) Comparison of computation cost between our scheme and the scheme in [7] we implemented with regard to different choice of challenged blocks from 0 to 1000 increasing by 1

Figure 2. Overview of Computational Cost and Comparison with Previous Scheme [7]

We explore the time spent by algorithms to measure computational overhead of our scheme in Figure 2. For efficiency consideration, we randomly check partial data blocks to guarantee the integrity in a probabilistic manner instead of challenging all blocks. As illustrated in previous schemes [6, 13], with 1% of data corrupted, to achieve at least 95% detectable probability only l = 300 blocks are needed to be checked, while l = 460 to achieve 99% probability. Hence, we choose to challenge 1,000 blocks at most in our experiments.

Since *SigGen()* algorithm generates metadata for all blocks in setup phase, it is very time-consuming which consumes nearly 140 seconds to generate all authenticators in the experiments. *KeyGen()* and *AuthGen()* algorithms contribute very little to overall computational cost. These three algorithms are executed just once and do not change with the number of challenged blocks. Figure 2 (a) presents computation time of algorithms in verification phase to demonstrate the overall variation trend of computational cost. We can see that the computation time of each algorithm is linear with the number of challenged blocks. As the work of *Challenge()* is to choose parameters for latter algorithms, there is no much computation. So its running time is the shortest and grows the slowest among the three algorithms, ranging from 0.043s with 100 challenged blocks to 0.43s with 1000 challenged. *ProofGen()* runs almost 10 times slower than *Challenge()* to generate the proof of data integrity, ranging from 0.42s to 3.85s. And *VerifyProof()* is the slowest and largely affected by challenged block number, ranging from 1.26s to 12.49s. These three algorithms are noticeably affected by challenged block number, and the growth ratio

is 0.043, 0.38 and 1.25 for *Challenge()*, *ProofGen()* and *VerifyProof()* respectively. We can infer that when challenged block number is large, the computation work can be cumbersome during verification phase. Thus, tradeoff should be made between computational cost and integrity guarantee.

We implemented the PDP scheme proposed by Atenise *et al.* in [6] and compare the computational efficiency of our scheme with that of their scheme in Figure 2 (b). Because the first five processes are similar in the two schemes, we just present the comparison of running time of *VerifyProof()* where additional validity checks are introduced in our scheme. From Figure 2 (b) we can see that the performance of this process is rather close, which means our scheme incurs negligible computational overhead.

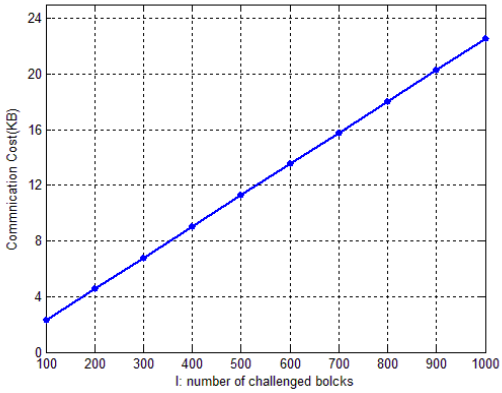


Figure 3. Communication Overhead during Verification Phase

We evaluate all communication overheads during verification phase as a whole, since the setup phase is only conducted once and its overhead is not affected by any external factors, e.g. challenged block number. Therefore, communication overhead includes the challenge message sent by TPA and the proof message generated by CSS. The challenge message consists of l index-coefficient pairs that is $l(logn+160)$ bit, a signature signed by TPA which is 160 bit, the authorization from DO which is 160 bit and the public key of TPA which is 160 bit. Thus the communication overhead of a challenge message is $O(l(logn+160)+480)$ bit and the parameters for checking validity of TPA is 480 bit, where n is the total block number. When the challenged block number l is 300 which achieves 95% probability of detecting data corruption, the checking parameters is negligible compared to the size of challenge message. On the other hand, the proof message consists of a combination of blocks and an aggregation of corresponding signatures. So the length of the proof would not change with varied challenged block number. With above analysis, it is clear that the communication cost mainly depends on the size of challenge message, and the cost for checking TPA's validity is negligible. Fig.3 demonstrates the communication overhead of our scheme. We can see that the communication is greatly affected by the challenged blocks number. And the size of checking parameters is only 0.06 KB, which is negligible compared to the challenge massage. Therefore, the additional communication cost that our proposal introduces is very low.

6. Conclusion

In this paper, we propose an efficient and secure public data auditing scheme with constrained auditing number for cloud storage. Authorization is introduced between DO and TPA to narrow down auditor set. CSS only responds to the TPA with valid authorization. And constrained auditing number is integrated into authorization, which limits the times of auditing from the TPA with a valid authorization. Nobody can generate sufficient challenge requests in a short time to significantly degrade the service quality of CSS. Thus the

threat of DDOS attack is literally eliminated. The analysis and experimental results show that our scheme is secure and efficient.

Acknowledgements

This research is supported by National Natural Science Foundation of China (61272425, 61402245), the Shandong Provincial Natural Science Foundation No. ZR2014FQ010, the Project Funded by China Postdoctoral Science Foundation under Grand No. 2014M551870, Huawei Technology Fund(YB2013120027), Minsheng Project of Huangdao District in Qingdao, and Shandong provincial Key Laboratory of Computer Network (SDKLCN-2013-03).

References

[1] P. Mell and T. Grance, "Draft NIST working definition of cloud computing", Online at http://csrc.nist.gov/group.SNS/ cloud-computing/index.html, **(2009)**.
[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandoc, "Cloud computing and emerging IT platforms: vision, hyper and reality for delivering computing as the 5th utility", Future Generation Computer Systems, vol. 25, no. 6, **(2009),** pp. 599-616.
[3] 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, "Above the clouds: a berkeley view of cloud computing", University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28, **(2009)**.
[4] H. F. Guo, "From Instagram to Dropbox why AWS are losing big clients", Online at http://www. pingwest.com/amazon-aws-problems/. **(2014)**.
[5] A. Juels and B. S. Kaliski Jr., "PORs: Proofs of retrievability for large files", In *Proc. of CCS'07*, **(2007),** pp. 584-579.
[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores. In *Proc. of CCS'07*, **(2007),** pp. 598-609.
[7] H. Shacham and B. Waters, "Compact proofs of retrievability", In *Proc. of ASIACRYPT'08*, vol. 5350, **(2008),** pp. 90-107.
[8] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Scalable and efficient provable data possession", In *SecureComm'08*, **(2008),** pp. 1-10.
[9] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession", In *Proc. of CSS'09*, **(2009),** pp. 213-222.
[10] Q. Wang, C. Wang, K. Ren W. Lou and J. Li, "Enabling public auditability and data dynamic for storage security in cloud computing", In *IEEE Transaction on Parallel and Distributed Systems*, vol. 22, no. 5, **(2011),** pp. 847-859.
[11] Z. Mo, Y. Zhou and S. Chen, "A dynamic proof of retrievalbility (POR) scheme with O(logn) Complexity", In *IEEE Communication and Information Systems security Symposium*, **(2012),** pp. 912-916.
[12] J. Yu, K. Ren, C. Wang and V. Varadharajan, "Enabling Cloud Storage Auditing with Key-Exposure Resistance", In *IEEE Transactions on Information Forensics and Security*, **(2015)** DOI: 10.1109/TIFS.2015.2400425
[13] C. Liu, J. Chen, L. T. Yang, X. Zhang, C. Yang, K. Ramamohanarao, "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates", In *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, **(2013),** pp. 2234-2244.
[14] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing", In *ASIACTYPT 2001*, **(2001),** pp. 514-532.
[15] M. Arrington, "Gmail disaster: Reports of mass email deletions", Online at http://www. techcrunch.com/2006/12/28/gmail-disasterreports-of-mass-email-deletions/, **(2006)**.
[16] J. Kincaid, "MediaMax/TheLinkup Closes Its Doors", Online at http://www.techcrunch. com/2008/07/10/mediamaxthelinkup-closes-its-doors/, **(2008)**.
[17] Amazon.com, Amazon s3 availability event: July 20, 2008, Online at http:// status.aws.amazon.com/s3-20080720.html, **(2008)**.
[18] S. Wilson, "Appengine outage", Online at http://www.cio-weblog.com/50226711/appengine outage.php. **(2008)**.
[19] B. Krebs, "Payment Processor Breach May Be Largest Ever", Online at http://voices. washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html. **(2009)**.
[20] H. C. Hsiao, Y. H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun, and B. Y. Yang. A Study of User Friendly Hash Comparison Schemes. In *Proc. ACSAC'09*, **(2009),** pp. 105-114.
[21] A.R. Yumerefendi and J. S. Chase, "Strong Accountability for Network Storage", In *Proc. Sixth USENIX Conf. File and Storage Technologies (FAST)*, **(2007),** pp. 77-92.
[22] B. Lynn, "The Pairing-Based Cryptographic Library. Online at http://crypto.Stanford. edu/pbc/. **(2014)**.

