Int. J. Inf. Secur.
DOI 10.1007/s10207-014-0263-8

REGULAR CONTRIBUTION

Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage

Yong Yu · Man Ho Au · Yi Mu · Shaohua Tang · Jian Ren · Willy Susilo · Liju Dong

Abstract Remote data integrity checking (RDIC) enables a server to prove to an auditor the integrity of a stored file. It is a useful technology for remote storage such as cloud storage. The auditor could be a party other than the data owner; hence, an RDIC proof is based usually on publicly available information. To capture the need of data privacy against an untrusted auditor, Hao et al. formally defined "privacy against third party verifiers" as one of the security requirements and proposed a protocol satisfying this definition. However, we observe that all existing protocols with public verifiability supporting data update, including Hao et al.'s proposal, require the data owner to publish some meta-data related to the stored data. We show that the auditor can tell whether or not a client has stored a specific file and link various parts of those files based solely on the published meta-data in Hao et al.'s protocol. In other words, the notion "privacy against third party verifiers" is not sufficient in protecting data privacy, and hence, we introduce "zero-knowledge privacy" to ensure the third party verifier learns nothing about the client's data from all available information. We enhance the privacy of Hao et al.'s protocol, develop a prototype to evaluate the performance and perform experiment to demonstrate the practicality of our proposal.

Keywords Cloud computing · Data integrity · Privacy · Remote data integrity checking

Y. Yu
School of Computer Science and Engineering,
University of Electronic Science and Technology of China,
Chengdu, China
e-mail: yyucd2012@gmail.com

Y. Yu · M. H. Au · Y. Mu · W. Susilo · L. Dong
School of Computer Science and Software Engineering, University of Wollongong, Wollongong, Australia

Y. Mu
e-mail: ymu@uow.edu.au

W. Susilo
e-mail: wsusilo@uow.edu.au

L. Dong
e-mail: liju@uow.edu.au

M. H. Au (✉)
Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
e-mail: csallen@comp.polyu.edu.hk

S. Tang
School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China
e-mail: csshtang@scut.edu.cn

J. Ren
Department of Electrical and Computer Engineering,
Michigan State University, East Lansing, USA
e-mail: renjian@egr.msu.edu

L. Dong
School of Information Science and Engineering,
Shenyang University, Shenyang, China

1 Introduction

Cloud computing is emerging as a dominant technology category in the business community. While the benefits of cloud computing are clear, it also introduces new security challenges. See [1] for a comprehensive survey. Cloud storage services, which allow data owners to migrate their data from local storage systems to the cloud, relieve the burden of storage management and maintenance. They offer scalable, pay-on-demand, location-independent storage service for users [2]. However, this new kind of data hosting service does trigger many new security challenges [3]. Indeed,

Published online: 29 August 2014

the Cloud Security Alliance (CSA) [4] regards Data Loss and Leakage as the second among the top seven security threats to cloud computing. For example, Business Insiders reported[1] that some data were destroyed in an EC2 cloud services crash in 2011. In addition, it is not mandatory for the service providers to report these incidents. In cloud storage setting, due to the loss of physical possession of data, a major concern of cloud users is whether their data are stored in the cloud safely. If the cloud servers are not fully trusted, the integrity of stored data could not be ensured. Consequently, there is a need for the development of protocols allowing the data owners to verify that their data are correctly stored in the cloud.

Traditional cryptographic technologies for data integrity checking such as message authentication codes and digital signatures are not ideal to remote data integrity checking (RDIC) because the original file is required in the verification procedure. It is an expensive exercise to download the entire file from the cloud for verification. Blum presented a scheme enabling data owners to verify the integrity of remote data without explicit knowledge of the entire data [5]. Provable data possession (PDP) [6,7], introduced by Ateniese et al., is a technique for validating data integrity over remote servers. In a typical PDP system, the data owner generates some metadata for a file, which will be used later for integrity checking via a challenge-response protocol with the remote server. Data owner then sends his file to a remote server, which may be untrusted, and deletes the file from its local storage. To generate a proof that the server hosts the file in its original form, the server computes a response to a challenge from the verifier. The verifier validates that the file is not being tampered with via checking the correctness of the response. Ateniese et al. also proposed two PDP schemes by utilizing the RSA-based homomorphic linear authenticators. At the same time, Juels et al. proposed the notion of proof of retrievability (POR) [8], in which error-correcting codes and spot-checking are employed to achieve the properties of possession and retrievability of files. PDP and POR have become a research hotspot of secure cloud storage and a number of schemes have been proposed [9–18]. Besides integrity checking, three advanced features, namely data dynamics, public verifiability and privacy against verifiers are also considered for practical purposes.

- **Data dynamics** This property allows the data owners to dynamically update their stored data after they store their data at the remote server. The main dynamics operation involves data insertion, data modification, data deletion and data appending. Ateniese et al. [19] described a dynamic PDP scheme based on cryptographic hash functions and symmetric key encryptions that is highly efficient. However, there is a priori bound on the number of queries, and block insertion is not explicitly supported. Wang et al. [20] proposed dynamic data storage in a distributed application but support for dynamic data operation is still partial. Erway et al. [21] extended the PDP model due to Ateniese et al. to data update by leveraging rank-based authenticated skip lists. They constructed a fully dynamic PDP by cleverly moving the index part from tag computation and authenticating the tag of challenged or updated blocks using authenticated skip list before the integrity-checking process. Wang et al. [13] improved the previous PDP models by manipulating the classic Merkle Hash Tree (MHT) for block tag authentication. They used MHT to authenticate both the data values and the positions of data blocks by treating the leaf nodes as the left-to-right sequence such that any leaf node can be uniquely determined by following left-to-right sequence and the approach to computing the root in MHT.
- **Public verifiability** This property allows an external auditor or anyone, not just the data owner, to have the capability to verify the integrity of the stored data on demand. Publicly verifiable data integrity-checking schemes are gaining favor due to their practicality in many applications in which data owners are not able to afford the overhead of periodical auditing. Ateniese et al. [6] considered this issue for the first time in their PDP model and described a variant with public verifiability of their basic PDP scheme. Shacham and Waters [9] proposed compact proofs of retrievability by making use of publicly verifiable homomorphic authenticators built from the BLS signature [22]. Their scheme relies on the homomorphic properties to aggregate a proof into a small authenticator value, and the public retrievability are also achieved. Due to the short signature length of BLS signature, the Shacham and Waters scheme is space efficient. Subsequent works based on their constructions include [13,14,16]. These schemes provide additional properties in addition to public verifiability.
- **Data privacy** In the remote data integrity-checking schemes with public verifiability, the data privacy issue should be considered since external auditors (or anyone) can check the integrity of the stored data. Data privacy against third party verifiers is highly essential for data owners in the sense that they may store confidential or sensitive files like business contracts and medical records to cloud. However, the importance of data privacy in the publicly verifiable checking has not received adequate attention [11,14], and this issue has not been fully investigated. Although data privacy is discussed in [14], a formal analysis is absent. Informally speaking, "data privacy" requires that the verifier learns no information about the outsourced data. Note that encrypting files before storing them on the

[1] Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data, http://www.businessinsider.com.au/amazon-lost-data-2011-4.

cloud could be a solution to the data privacy problem. However, this solution reduces the issue to the complex key management domain. Moreover, encrypting the files before outsourcing is unnecessary in many applications such as in public cloud data, say outsourced libraries or scientific datasets.

Contributions Sebé et al. [23] proposed a data possession-checking protocol which achieves many desirable properties such as unlimited number of verifications, low communication cost, blockless verification and security against a malicious server. However, this scheme does not support public verifiability nor guarantees data privacy. In a subsequent work, Hao et al. [24] adopted Sebé et al.'s protocol [23] to support data dynamics and data privacy. They formalized a model for data privacy and claimed that their protocol does not leak any private information to third party verifiers. In this paper, we study Hao et al.'s protocol and make the following three contributions.

– We observe that Hao et al.'s model does not capture the information leak regarding meta-data and demonstrate how a third party verifier can make use of the meta-data to learn information about the stored data.
– We develop an enhanced model to address this weakness, propose an improved scheme and prove that the new construction is secure under the enhanced model.
– We demonstrate that the new scheme retains other desirable properties of the original scheme [24] and also develop a prototype to show our proposal is practical.

2 Preliminaries

In this section, we review some preliminary knowledge that will be used in this paper.

2.1 Remote data integrity checking for secure cloud storage

A publicly verifiable remote data integrity-checking architecture [7,10,13] for security cloud storage is illustrated in Fig. 1. Three different entities, namely the cloud user, the cloud server and the third party auditor (TPA) are involved in the system. The cloud user has large amount of data to be stored on the cloud server without keeping a local copy, and the cloud server has significant storage space and computation resources and provides data storage services for cloud users. TPA possesses expertise and capabilities that cloud users do not have and is trusted to check the integrity of the cloud data on behalf of the cloud user upon request. They have their own obligations and benefits, respectively. The cloud server can be self-interested, and for his own benefits, such as to maintain reputation, the cloud server might hide

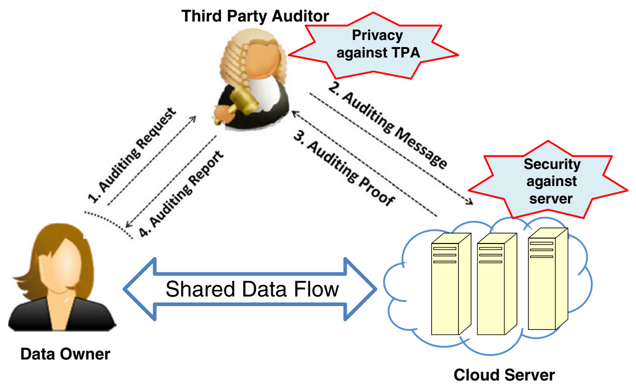


Fig. 1 The system model of publicly verifiable remote data checking

data corruption incidents to users. However, we assume that the cloud server has no incentives to reveal the hosted data to TPA because of regulations and financial incentives. The TPA's job is to perform the auditing on behalf of the cloud user in case that the user has no time, resources or feasibility to monitor his data. However, the TPA is also curious and may try to deduce some information of the data during the auditing process.

2.2 System components and its security

Following the definition in [9], a public remote data-checking scheme or auditing scheme is a tuple of five algorithms, namely Setup, TagGen, Challenge, ProofGen and ProofCheck, which can be described as follows.

– Setup. On input a security parameter (k), this algorithm generates the public key (pk) and secret key (sk) for the data owner. pk is public to everyone but sk is kept secret by the data owner.
– TagGen. On input the key pair (pk, sk) and a data block (m_i), this algorithm outputs a tag (D_{m_i}) for the block, which will be used for public verification of data integrity.
– Challenge. TPA generates a challenge *chal* to request for the integrity proof of the file by sending *chal* to the server.
– GenProof. The server computes response R using *chal*, the file and the tags, and returns R to TPA.
– CheckProof. TPA validates response R using *chal*, the tags and public key pk. Secret key sk is not required in a publicly verifiable data integrity-checking scheme.

Three security requirements, namely completeness, security against a malicious server (soundness) and privacy against the TPA (privacy), should be met for a public data integrity-checking scheme. Following the security model due to Shacham and Waters [9], a data auditing scheme is secure against a server if there exists no polynomial-time algorithm that can cheat the TPA with non-negligible probability. Formally, it is required that there exists a polynomial-time extractor capable of recovering the file by performing the challenge-response protocols multiple times. Completeness and soundness were defined as well by Shacham and Waters in [9]. Completeness says that when interacting with a valid server, the algorithm of CheckProof will accept the response. Soundness indicates that a cheating prover who can convince a TPA it is storing the file is actually storing that file. We now review the security model against a malicious server with public verifiability, in which two entities are involved: an adversary and a challenger who plays the role of the untrusted server and a data owner, respectively.

Security against the server This security game captures the requirement that an adversary cannot successfully generate a valid proof without storing all the file blocks. The game consists of the following four phases, namely *Setup, Query, Challenge* and *Forge*.

- Setup. The challenger runs Setup algorithm to generate the public-secret key pair (pk, sk), forwards pk to the adversary and keeps sk secret.
- Query. The adversary adaptively selects some data blocks $m_i (i = 1, \ldots, n)$ and makes tag queries. The challenger computes the corresponding tag $D_i (i = 1, \ldots, n)$ for these blocks and sends them back to the adversary.
- Challenge. The challenger generates a challenge *chal* and requests the adversary to respond a proof of possession for the challenged blocks.
- Forge. The adversary computes a response R for the data blocks indicated by *chal* and returns it to the challenger.

We say the adversary wins the game if $CheckProof(pk, chal, D_i (i = 1, \ldots, n), R)$ succeeds. We say that a public remote data integrity-checking scheme is secure against the server if for all polynomial-time adversary that wins the above game, there exists another polynomial-time algorithm Σ, denoted as knowledge extractor, that is capable of extracting the file blocks. The rationale of the model is that for any adversary that can win the game, it can employ Σ to compute the data blocks in polynomial time. In other words, any algorithm that can answer the challenge must be in possession of the underlying file blocks stored in one form or another.

Privacy against the TPA The term "zero-knowledge public auditing" was first used by Wang et al. [14] to describe the auditing schemes offering data privacy against the TPA. However, no formal model is presented. Following their terminology, we formally define the notion "zero-knowledge privacy" as an enhanced privacy model of Hao et al. [24] with the goal of capturing data privacy against the TPA in practice. The spirit of our definition is that the TPA learns nothing about the data blocks based on the publicly available information and the interaction with the cloud server. To further strengthen the definition, we allow the TPA to choose two equal length messages, $m_0 = (m_{0,1}, \ldots, m_{0,n})$ and $m_1 = (m_{1,1}, \ldots, m_{1,n})$ and requires that given a set of meta-data as well as interaction with the cloud server, the TPA cannot tell whether the cloud server is storing m_0 or m_1. The major difference between our model and Hao et al.'s model [24] is that our model considers information leak from the meta-data in addition to the execution of the challenge-response protocol.

Formally, we consider the following game.

- Setup. The challenger runs Setup algorithm to generate the public-secret key pair (pk, sk), forwards pk to the adversary and keeps sk secret.
- Query. The adversary submits two equal-length files, $m_0 = (m_{0,1}, \ldots, m_{0,n})$ and $m_1 = (m_{1,1}, \ldots, m_{1,n})$ to the challenger. The challenger chooses a random bit $b \in_R \{0, 1\}$ and returns

$$D_i \leftarrow \text{tagGen}(m_{b,i}, pk, sk) \quad \text{for } i = 1 \text{ to } n.$$

- Challenge. The adversary sends *chal* to the challenger.
- GenProof. The challenger computes response R using m_b, *chal* and the tags $\{D_i\}$.
- Guess. The adversary outputs a guess bit b'. It wins the game if $b' = b$.

We say that the advantage of an adversary is the probability that it wins the above game minus 0.5.[2] A public remote data integrity-checking scheme is data private if the advantage of any polynomial-time adversary is negligible.

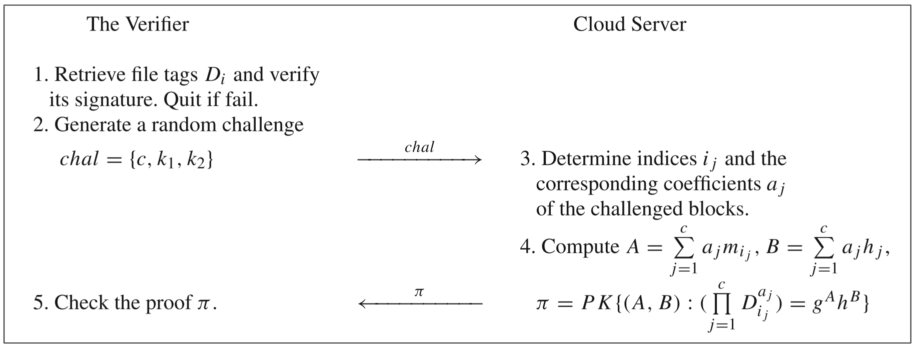
3 Analysis of Hao et al.'s RDIC protocol

Hao et al. [24] proposed a privacy-preserving remote data integrity-checking protocol with data dynamics and public verifiability. Their construction is based on Sebé et al.'s protocol [23] and the homomorphic verifiable tag technique due to Ateniese et al. [6]. While it can be shown that the challenge-response protocol "does not leak any information of the data to TPA," it does not prevent the verifier from learning information about the data from the meta-data. In other words, the scheme itself cannot be said to be data private.

3.1 Review of the RDIC protocol

The remote data integrity-checking protocol in [24] consists of the following algorithms.

[2] This is to offset the winning probability based on random guessing.



The Verifier | Cloud Server

1. Retrieve file tags D_i and verify its signature. Quit if fail.
2. Generate a random challenge $chal = \{c, k_1, k_2\}$ $\xrightarrow{chal}$
3. Determine indices i_j and the corresponding coefficients a_j of the challenged blocks.
4. Compute $A = \sum_{j=1}^{c} a_j m_{i_j}$, $B = \sum_{j=1}^{c} a_j h_j$,
$\pi = PK\{(A, B) : (\prod_{j=1}^{c} D_{i_j}^{a_j}) = g^A h^B\}$
5. Check the proof π. $\xleftarrow{\pi}$

Fig. 2 Improved RDIC protocol supporting zero-knowledge privacy

– Setup. Let $N = pq$ be a publicly known RSA modulus [25], in which $p = 2p' + 1, q = 2q' + 1$ are large primes. p', q' are two large primes as well. QR_N denotes the multiplicative cyclic group of the quadratic residues modulo N. g, whose order is $p'q'$, is a generator of QR_N. The public key of the data owner is $pk = (N, g)$ while the secret key is $sk = (p, q)$.
– TagGen. For each file block m_i, $i \in [1, n]$, the data owner computes the block tag as

$$D_i = g^{m_i} \pmod N.$$

– Challenge. To check the integrity of the file m, the TPA generates a random key $r \in [1, 2^k - 1]$ and a random group element $s \in Z_n \setminus \{0\}$, computes $g_s = g^s \pmod N$ and sends $chal = (r, g_s)$ to the cloud server.
– GenProof. Upon receiving the challenge $chal = (r, g_s)$, the server generates a sequence of block indexes $a_1, \ldots, a_n$ by calling $f_r(i)$ for $i \in [1, n]$ iteratively, computes

$$R = g_s^{\sum_{i=1}^{n} a_i m_i} \pmod N,$$

and sends R to the verifier.
– CheckProof. Upon receiving R from the server, the TPA generates $\{a_i\}_{i=1,\ldots,n}$ firstly and then computes P and R' as follows:

$$P = \prod_{i=1}^{n} (D_i^{a_i}) \pmod N,$$
$$R' = P^s \pmod N.$$

If $R = R'$, this algorithm outputs "success" to indicate the data are kept virgin; Otherwise, outputs "failure".

3.2 On the privacy of the protocol

The verifier can determine whether the client is storing a file block m^* from the public key (N, g) and meta-data $\{D_i\}_{i=1}^n$ by evaluating the following equations,

$$D_i \overset{?}{=} g^{m^*} \quad \text{for } i = 1 \text{ to } n.$$

If any of the equality holds, the client is storing the file block m^*.

We remark again that this "attack" is outside the original security model which only assures no "additional" information can be obtained from the challenge-response protocol. The issue being overlooked is the fact that meta-data could contain information about the underlying data. Indeed, any public auditing scheme could be made "data private" under this model if the meta-data contains the original data. For if this is the case, the challenge-response protocol will leak no "additional" information. On the other hand, the above attack is captured by our enhanced model. Indeed, Hao et al.'s construction is not data private under our definition.

4 An improved scheme and its security

In this section, we provide an improved remote data integrity-checking protocol supporting zero-knowledge privacy. Our construction is inspired by the sprit of zero-knowledge proof, and the details of the new protocol are as follows Fig. 2.

– Setup. Let k, l be two security parameters. $N = pq$ is a public RSA modulus, in which $p = 2p' + 1$, $q = 2q' + 1$ are large primes. p', q' are two large primes as well. QR_N denotes the multiplicative cyclic group of the quadratic residues modulo N. g, h are two generators of QR_N. The public key of the data owner is $pk = (N, g)$ while the secret key is $sk = (p, q)$. A file m is divided into n blocks $m_1, \ldots, m_n$. $H_1, H_2 : \{0, 1\}^* \rightarrow Z_N$ are secure cryptographic hash functions. $f : \{0, 1\}^k \times \{0, 1\}^{\log_2(n)} \rightarrow \{0, 1\}^l$ denotes a pseudo-random function and $\pi : \{0, 1\}^k \times \{0, 1\}^{\log_2(n)} \rightarrow \{0, 1\}^{\log_2(n)}$ represents a pseudo-random permutation.
– TagGen. We assume that a file $m = m_1, \ldots, m_n$ has been encoded using the RS code [26] and the blocks are distinct. The data owner picks a random $t \in Z_N$ called a tag salt and appends t at the end of the file. For each block m_i, $i \in [1, n]$, the data owner computes the block tag as

$$D_i = g^{m_i} h^{H_1(m_i,t)} \pmod N,$$

and stores $(m\|t, D_i(1 \le i \le n))$ into the server.

- Challenge. To check the integrity of file m, the verifier picks a random positive integer c and two keys k_1, k_2 for f and π respectively and sends the challenge $chal = (c, k_1, k_2)$ to the server.
- GenProof. Upon receiving $chal = (c, k_1, k_2)$, the server computes a response as follows.
  1. For $1 \le j \le c$, compute $i_j = \pi_{k_2}(j)$ as the indices of the challenged blocks and computes $a_j = f_{k_1}(j)$ as coefficients.
  2. Compute $A = \sum_{j=1}^{c} a_j m_{i_j}$, and $B = \sum_{j=1}^{c} a_j h_j$ where $h_j = H_1(m_{i_j}, t)$.
  3. Randomly pick $\rho_1, \rho_2 \in Z_N$, compute $T = g^{\rho_1} h^{\rho_2} \pmod N$, $\xi = H_2(T, chal)$ and $z_1 = \rho_1 - \xi A$, $z_2 = \rho_2 - \xi B$.
  4. Send $R = (\xi, z_1, z_2)$ as the response to the verifier.
- CheckProof. Upon receiving R from the server, the verifier firstly generates the indices i_j of the challenged blocks and the corresponding coefficients a_j and checks if

$$\xi \overset{?}{=} H_2\left(g^{z_1} h^{z_2} (\prod_{j=1}^{c} D_{i_j}^{a_j}\right)^{\xi} \pmod N), chal).$$

If the verification holds, this algorithm outputs "success" to indicate the data are kept virgin; Otherwise, outputs "failure".

5 Security analysis of the new protocol

In this section, we show that the newly proposed scheme achieves the properties of completeness, soundness and zero-knowledge privacy.

5.1 Completeness

If both the data owner and the server are honest, for each valid tag D_i and a random challenge $chal = (c, k_1, k_2)$, the completeness of the protocol can be elaborated as follows.

$$\begin{aligned}
&g^{z_1} h^{z_2} \left(\prod_{j=1}^{c} D_{i_j}^{a_j}\right)^{\xi} \\
&= g^{\rho_1 - \xi A} h^{\rho_2 - \xi B} \left(\prod_{j=1}^{c} g^{a_j m_{i_j}} h^{a_j H_1(m_{i_j},t)}\right)^{\xi} \\
&= g^{\rho_1 - \xi A + \sum_{j=1}^{c} a_j m_{i_j} \xi} h^{\rho_2 - \xi B + \sum_{j=1}^{c} a_j h_j \xi} \\
&= g^{\rho_1 + \xi(\sum_{j=1}^{c} a_j m_{i_j} - A)} h^{\rho_2 + \xi(\sum_{j=1}^{c} a_j h_j - B)} \\
&= g^{\rho_1} h^{\rho_2} \\
&= T \pmod N.
\end{aligned}$$

Thus, the following verification holds.

$$\begin{aligned}
\xi &= H_2(T, chal) \\
&= H_2\left(g^{z_1} h^{z_2} \left(\prod_{j=1}^{c} D_{i_j}^{a_j}\right)^{\xi} \pmod N), chal\right).
\end{aligned}$$

5.2 Soundness

If a malicious server can convince a verifier that he stores the data using the proposed new scheme, we can construct a simulator to extract the data in the random oracle model. Otherwise, an instance of the factorization problem can be solved.

Proof What we are going to prove here is that for any PPT adversary $\mathcal{A}$ who wins the soundness game of some file blocks, there exists a challenger $\mathcal{B}$ that can construct a simulator $\mathcal{S}$ to extract these blocks. Otherwise, the challenger can solve an instance of the factorization problem. $\mathcal{B}$ is given a large integer N, the product of two large primes p and q, and simulates the environment as follows.

- Setup. $\mathcal{B}$ generates two random generators g, h of QR_N and sends $pk = (N, g, h)$ as the public key to the adversary $\mathcal{A}$.
- Queries. $\mathcal{A}$ adaptively selects some blocks m_i and a tag salt t_j to query $\mathcal{B}$ the tags of these blocks. $\mathcal{B}$ computes $h_{ij} = H_1(m_i, t_j)$ and $D_i = g^{m_i} h^{h_{ij}}$, and sends D_i to $\mathcal{A}$.
- Challenge. $\mathcal{B}$ picks an integer c, two keys $k_1, k_2 \in \{0,1\}^k$ and generates a challenge $chal = (c, k_1, k_2)$ for c file blocks and sends it to $\mathcal{A}$.
- Response. $\mathcal{A}$ generates a response $R = (\xi, z_1, z_2)$ and sends it back to $\mathcal{B}$ as integrity proof of the requested blocks.

□

If the response can pass the proof check, i.e.,

$$\xi = H_2(g^{z_1} h^{z_2} \left(\prod_{j=1}^{c} D_{i_j}^{a_j}\right)^{\xi} \pmod N), chal),$$

based on the well-known cryptographic techniques, the oracle replay technique and the forking lemma due to Pointcheval and Stern [27], the challenger can replay the oracle H_2 with the same randomness and a different oracle to generate a new response $R' = (\xi', z_1', z_2')$. The challenger can obtain two pairs of collision for H_2 as $(T, chal)$ and $(T', chal)$. Thus, we have

$$g^{z_1}h^{z_2}\left(\prod_{j=1}^{c} D_{i_j}^{a_j}\right)^{\xi} = g^{z'_1}h^{z'_2}\left(\prod_{j=1}^{c} D_{i_j}^{a_j}\right)^{\xi'} \pmod N.$$

That is

$$\left(g^{\sum_{j=1}^{c} a_j m_{i_j}} h^{\sum_{j=1}^{x} H_1(m_{i_j},t)a_j}\right)^{\xi-\xi'} = g^{z'_1-z_1}h^{z'_2-z_2} \pmod N.$$

According to the elegant conclusion due to Damgård and Fujisaki [28], $\xi - \xi'$ divides both $z'_1 - z_1$ and $z'_2 - z_2$ with significant probability. As a consequence, we have

$$g^{\sum_{j=1}^{c} a_j m_{i_j}} h^{\sum_{j=1}^{x} H_1(m_{i_j},t,i)a_j} = g^{\frac{z'_1-z_1}{\xi-\xi'}} h^{\frac{z'_2-z_2}{\xi-\xi'}} \pmod N.$$

Based on the discrete logarithm assumption, $\mathcal{B}$ can extract

$$\sum_{j=1}^{c} a_j m_{i_j} = \frac{z'_1 - z_1}{\xi - \xi'}.$$

$\mathcal{B}$ then generates c challenges $(c, k_1^1, k_2), \ldots, (c, k_1^c, k_2)$ to challenge the same blocks using the approach described above. For each challenge, $\mathcal{B}$ can get an equation of the blocks. $\mathcal{B}$ can select such $k_1^i (1 \le i \le c)$ that

$$\begin{vmatrix} a_1^1 & a_2^1 & \cdots & a_c^1 \\ a_1^2 & a_2^2 & \cdots & a_c^2 \\ \vdots & \vdots & \vdots & \vdots \\ a_1^c & a_2^c & \cdots & a_c^c \end{vmatrix} \ne 0$$

where $a_j^i = f_{k_i}(j)(1 \le i, j \le c)$.

When the determinant is not zero, the following system of linear equations has a unique solution.

$$\begin{cases} a_1^1 m_{1_j} + a_2^1 m_{2_j} + \cdots + a_c^1 m_{c_j} = \frac{z_1{}^{1'} - z_1^1}{\xi^1 - \xi^{1'}} & \pmod{p'q'} \\ a_1^2 m_{1_j} + a_2^2 m_{2_j} + \cdots + a_c^2 m_{c_j} = \frac{z_1{}^{2'} - z_1^2}{\xi^2 - \xi^{2'}} & \pmod{p'q'} \\ \vdots \\ a_1^c m_{1_j} + a_2^c m_{2_j} + \cdots + a_c^c m_{c_j} = \frac{z_1{}^{c'} - z_1^c}{\xi^c - \xi^{c'}} & \pmod{p'q'} \end{cases} \quad (1)$$

The simulator can get each challenged block $m^*_{i_j} = m_{i_j} \pmod{p'q'}$ by solving the system of linear equations. If $m^*_{i_j} = m_{i_j}$ for all $a_{i_j} = f_{k_i}(j)$, the simulator has successfully extracted all the changed blocks. We show that all the challenged blocks are indeed extracted by the simulator. Otherwise, assume $a_{i_j} \ne f_{k_i}(j)$ but $m^*_{i_j} = m_{i_j} + tp'q'$, where t is a nonzero integer, the challenger can make use of $m^*_{i_j} - m_{i_j}$ to factorize N [29], which violates the factorization assumption.

5.3 Zero-knowledge privacy

In the random oracle model, the new protocol achieves "zero-knowledge privacy". To prove this property, we construct a simulator $\mathcal{S}$ who plays the role of the challenger in the game. For any public key (N, g, h) and system parameters (k, l, H_1, H_2, f, π), $\mathcal{S}$ interacts with the adversary $\mathcal{A}$ as follows.

- Setup. $\mathcal{S}$ forwards the public key to the adversary.
- Query. The adversary submits two equal-length files, $m_0 = (m_{0,1}, \ldots, m_{0,n})$ and $m_1 = (m_{1,1}, \ldots, m_{1,n})$ to $\mathcal{S}$. For $i = 1$ to n, $\mathcal{S}$ picks $D_i \in_R \mathbb{N}^*$ and returns $\{D_i\}$ to $\mathcal{A}$.
- Challenge. The adversary sends $chal = (c, k_1, k_2)$ as the challenge to $\mathcal{S}$.
- GenProof. $\mathcal{S}$ simulates the response as follows.
    1. Pick a random tag-salt $t \in Z_N$ for m and for each block m_i in m, compute

       $$D_i = g^{m_i}h^{H_1(m_i,t)} \pmod N,$$

       as the tag of m_i.
    2. For a challenge (c, k_1, k_2), compute the indices $i_j = \pi_{k_2}(j)$ of the challenged blocks and coefficients $a_j = f_{k_1}(j)$.
    3. Pick a random $\xi^* \in Z_n$ and two integers z_1^* and z_2^*, and compute $T^* = g^{z_1^*}h^{z_2^*}(\prod_{j=1}^{c} D_{i_j}^{a_j})^{\xi^*} \pmod N$.
    4. Take H_2 as a random oracle and set the

       $$H_2(T^*, chal) \to \xi^*.$$

    5. Out $R^* = (\xi^*, z_1^*, z_2^*)$ as the response.
- Guess. The adversary outputs a guess bit b'.

We argue that the probability of $b' = b$ must be 0.5 since the tags $\{D_i\}$ and response R^* are independent of b. It remains to show the tags and response given to $\mathcal{A}$ are distributed correctly. Firstly, for any value of $m_{b,i}$ there exists a value $r_{b,i}$ such that $D_i = g^{m_i}h^{r_{b,i}}$. The above simulation implicitly sets $r_{b,i} = H_1(m_{b,i}, t)$.[3] Since t is never given to the adversary, this simulation is perfect. As for the response, note that for any file m_b and the, respectively, A_b, B_b, there exists $\rho_{b,1}, \rho_{b,2}$ such that $z_1 = \rho_{b,1} - \xi^* A_b$ and $z_2 = \rho_{b,2} - \xi^* B_b$. In addition, z_1, z_2 are statistically indistinguishable with the actually responses produced using m_0 or m_1. Thus, the simulation provided to $\mathcal{A}$ is correctly distributed. This completes the proof.

[3] This is allowed in the random oracle model where H_1 is modeled as an ideal random function.

6 Data dynamics operations

In the improved protocol, the tag generation depends only on the block content and is independent of the index of this block and the content of other blocks. As a result, the protocol fully supports data dynamics at the block level including block insertion, block modification and block deletion. The detailed descriptions are as follows.

- **Block Insertion** When needing to insert a new block m_x before $m_i (1 \leq i \leq n)$, the data owner generates a tag for m_x as $D_x = g^{m_x} h^{H_1(m_x,t)} \pmod N$ and then constructs an update message "*update* $= (I, i, m_x, D_x)$" and sends it to the cloud server, where I denotes the block insertion operation. Upon receiving the request, the server updates the stored file to $m' = m_1 \cdots m_{i-1} m_x m_i \cdots m_n$ and updates the block tags to $D_1 \cdots D_{i-1} D_x D_i \cdots D_n$.
- **Block Deletion** When needing to delete block m_i, the data owner constructs an update message "*update*$=(D, i, m_i, D_i)$" and sends it to the cloud server, where D denotes the block deletion operation. The server updates the stored file to $m' = m_1 \cdots m_{i-1} m_{i+1} \cdots m_n$ and the block tags to $D_1 \cdots D_{i-1} D_{i+1} \cdots D_n$.
- **Block Modification.** When needing to modify m_i to m_i^*, the data owner generates a tag for m_i^* as $D_i^* = g^{m_i^*} h^{H_1(m_i^*,t)} \pmod N$ and constructs an update message "*update* $= (M, i, m_i^*, D_i^*)$" and sends it to the cloud server, where M denotes the block modification operation. The server updates the stored file to $m' = m_1 \cdots m_{i-1} m_i^* m_{i+1} \cdots m_n$ and updates the block tags to $D_1 \cdots D_{i-1} D_i^* D_{i+1} \cdots D_n$.

7 Performance analysis and implementation

In this section, we firstly report the complexity analysis of communication, computation and storage costs of the improved protocol and then describe the experimental results.

7.1 Complexity analysis

Communication cost In the challenge phase, the verifier sends (c, k_1, k_2) to the server, which is of binary length $\log_2 c + 2k$. In the response phase, the server returns $R = (\xi, z_1, z_2)$ as the response to the verifier, which is $\log_2 N + \log_2 z_1 + \log_2 z_2$.

Computation cost We present the computation cost from the viewpoint of the data owner, the server and the verifier. Let $T_{prf}(len)$, $T_{prp}(len)$ denote the time cost of generating a len-bit pseudo-random number or performing a permutation of len-bit number. $T_{add}(len)$ stands for the time cost of adding two len-bit numbers, and $T_{exp}(len, num)$ represents the time cost of computing a modular exponentiation of a len-bit long exponent modular num.

The dominated computation of the data owner is generating tags for file blocks as $D_i = g^{m_i} h^{H_1(m_i,t)} \pmod N$. According to the Euler Theorem, since $\gcd(g, N) = 1$ and $\gcd(h, N) = 1$, we have $g^{\phi(N)} = 1 \pmod N$ and $h^{\phi(N)} = 1 \pmod N$. Thus, the data owner can compute $g^{m_i \pmod{\phi(N)}} h^{H_1(m_i,t) \pmod{\phi(N)}} \pmod N$ instead of computing $g^{m_i} h^{H_1(m_i,t)} \pmod N$ directly, which will save significant computation cost since modulo operations are far more efficient than modular exponentiations. To generate a proof, the server needs to perform c pseudo-random functions and c pseudo-random permutations to determine the indices of the challenged blocks and the corresponding coefficients. The main cost is computing the sum $\sum_{j=1}^{c} a_j m_{i_j}$, $\sum_{j=1}^{c} a_j h_j$ and exponentiations g^{ρ_1}, g^{ρ_2}. a_j is of l bits long and assume each block is d bits long, the cost of computing $a_j m_{i_j}$ is upper bounded by $l-1$ additions of $d+l$-bit integers. The computation cost of summing them is upper bounded by $c - 1$ additions of $\log_2 c + d + l$- bit integers. Similarly, The computation cost of computing $\sum_{j=1}^{c} a_j h_j$ is upper bounded by $c - 1$ additions of $\log_2 c + \log_2 N + l$-bit integers. Thus the total computation cost of the cloud server is upper bounded by

$$\begin{aligned} &cT_{prf}(\log_2 c) + cT_{prp}(\log_2 c) + 2T_{exp}(\log_2 N, N) \\ &\quad +(c-1)T_{add}(\log_2 c + d + l) \\ &\quad +(c-1)T_{add}(\log_2 c + \log_2 N + l). \end{aligned}$$

In the verification phase, the main computation is $g^{z_1} h^{z_2} (\prod_{j=1}^{c} D_{i_j}^{a_j})^{\xi}$, and so the total computation cost of the verifier is

$$\begin{aligned} &cT_{prf}(\log_2 c) + cT_{prp}(\log_2 c) + T_{exp}(\log_2 z_1, N) \\ &\quad +T_{exp}(\log_2 z_2, N) + cT_{exp}(l + \log_2 N, N). \end{aligned}$$

Storage cost Regarding the storage cost of the cloud server and the verifier, since we need the property of public verifiability, both the data and the tags are stored at the server side. Traditional integrity protection methods, say a secure digital signature mechanism, can be employed to protect the tags from being tampered by external and internal adversaries. In this case, what stored on the cloud are as follows.

Data fils||Tag salt	Tags	Signature of tags

The storage cost of the block tags is upper bounded by $\lceil \log_2(m)/d \rceil \log_2 N$ bits. When performing an auditing task now, the tags are transmitted back to the verifier from the cloud server, which will incur communication costs that are linear to the number of blocks. Fortunately, due to the employment of the modular of composite order, the tags can be relatively much smaller compared with the original files.

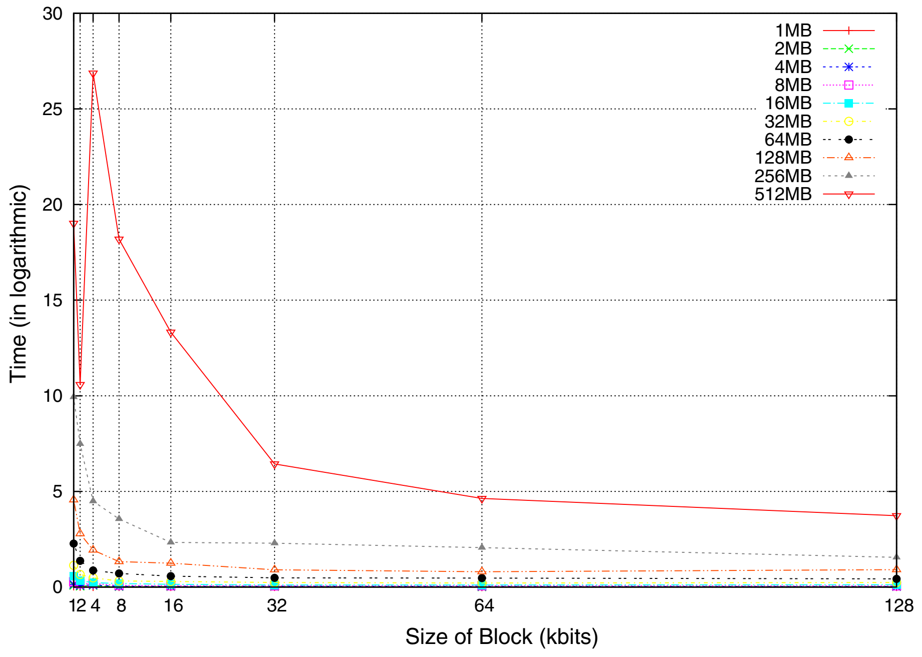


Fig. 3 Total time for TagGen versus size of blocks

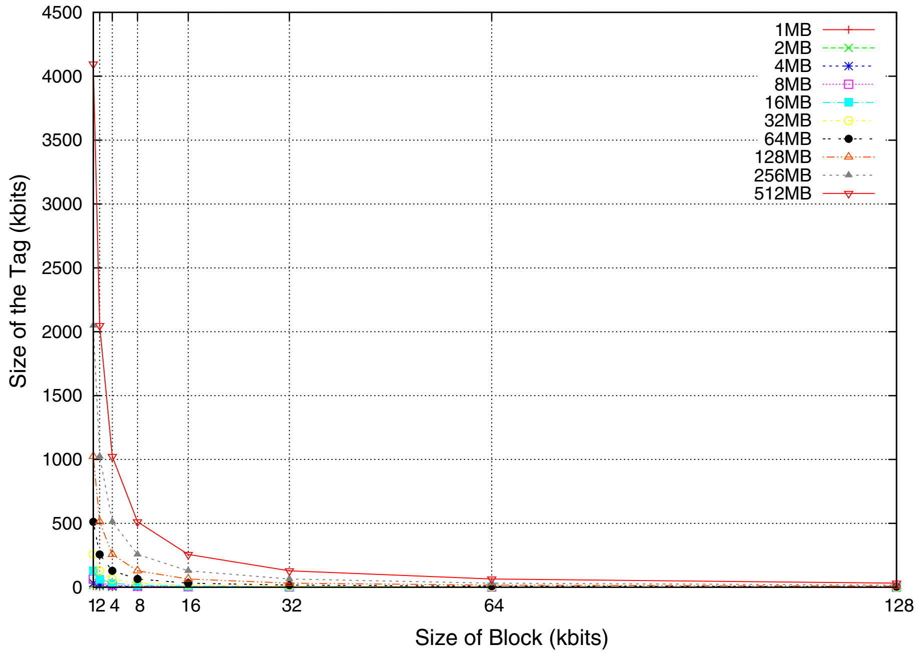


Fig. 4 Total time for CheckProof versus size of blocks

The data owner only needs to store the public key N, g, the private key (p, q) and the tag-salt t, so the storage cost of the data owner is $3\log_2 N + \log_2 p + \log_2 q$ bits. A verifier needs to store the public key N, g, thus, his storage cost is $2\log_2 N$ bits.

7.2 Implementation results

The implementation was conducted with MAGMA [30] on Xeon E5640 CPUs @ 2.66GHz. The memory is always sufficient since the scheme only requires a polynomial space. In our implementation, we uses RSA-1024, where N is of 1,024 bits, p and q are 512 bits each. Our tests aim to determine the cost of the following algorithms: TagGen, ProofGen and CheckProof. We note that the time for the Setup and Challenge steps are not shown in the result. Since the Setup is run for one time only, which incurs a cost of approximately 300 s, while essentially the Challenge step only requires an exponentiation operation over $\mathbb{Z}_N$, and hence, the timing is negligible.

To observe the impact of the number of blocks and the block sizes, we proceed our tests as follows: we use different

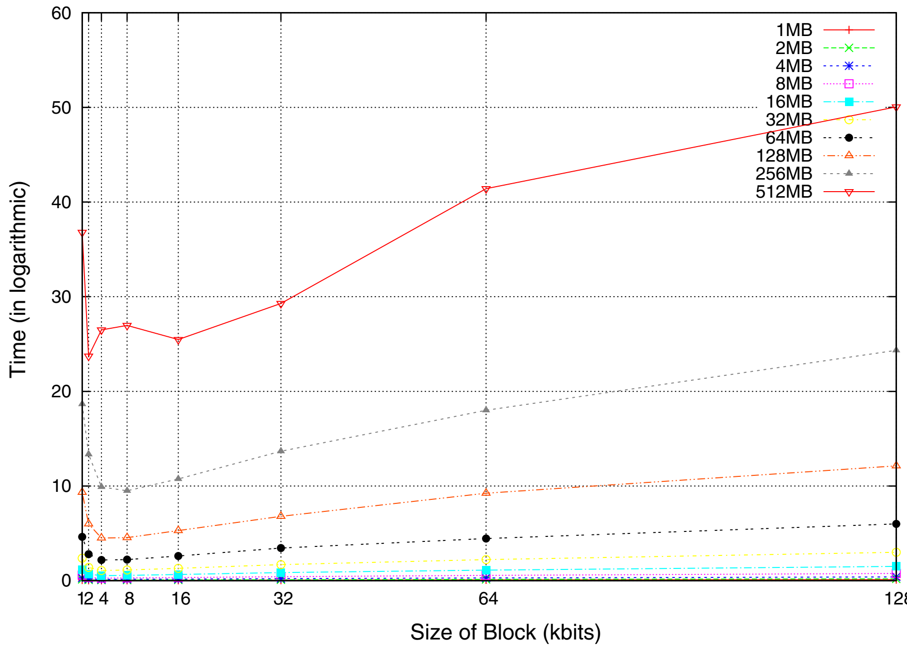


Fig. 5 Total time for ProofGen versus size of blocks

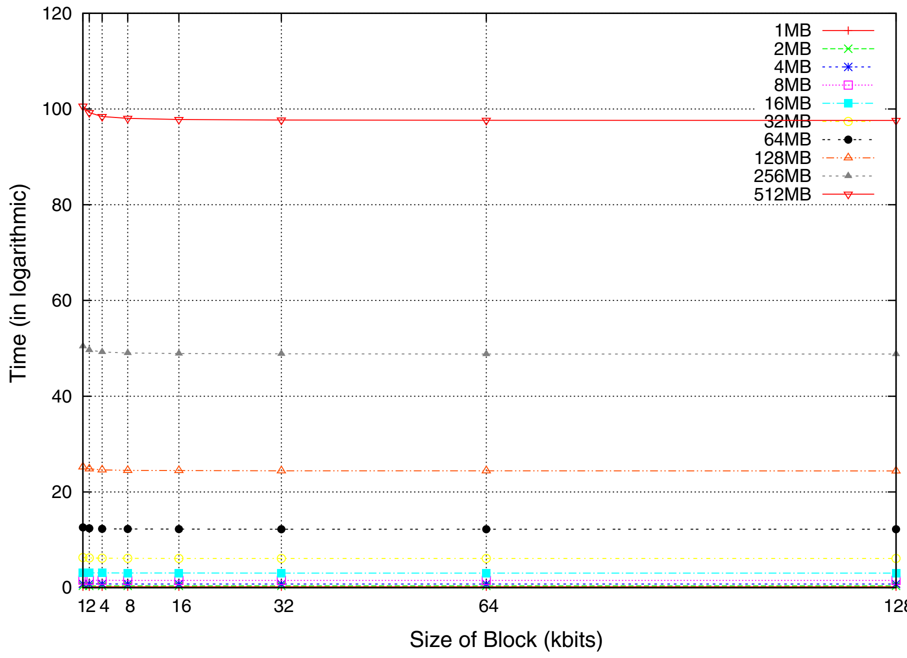


Fig. 6 Total time for CheckProof versus size of blocks

size of files for our tests, from 1 to 512 MB. Then, for each file size, we tested different block sizes, from 1 to 128 KB. Specifically, the following figures illustrate the results of our tests in details.

The tag generation procedure is efficient in general. This is due to the fact that the owner knows $\phi(N)$, and subsequently, exponentiations are performed with respect to relatively small exponents (large exponents can be computed modulo $\phi(N)$ first). On the other hand, proof generation, performed by the server, is a bit more costly, since one needs to perform two exponentiations with large exponent (z_1 and z_2). The CheckProof procedure, performed by the auditor, is most costly, since in this process, one needs to do polynomial in c number of exponentiations.

In more details, Fig. 3 shows that the total time for TagGen is almost linear with the increment of the size of the block, i.e., for a same block size, the TagGen time is doubled when one increases the size of the file by two. This is natural since doubling the file size implies doubling the total number of blocks. In the meantime, for a constant file size, it is also linear with the inverse of the size of the blocks. This is in accordance with our empirical analysis since block size is inversely proportional to the total number of blocks and as discussed, for TagGen, increasing the block size has negligible effect on the cost of an exponentiation for the owner since exponents can always be computed modulo $\phi(N)$. Likewise, when one increases the size of the blocks, the generation time for the tags is decreasing.

Figure 4 shows that the size of the tags is linear with the increment of the size of the blocks and hence is linear with the increase of the number of the blocks. This is because for each block, the size of the tag is constant that depends only on N.

The timing of ProofGen is the product of two components: the number of the tags and the cost to generate each tag. The latter component is depending on the size of the block. Therefore, from Fig. 5 one can see that it takes more time when the size of the block is either very small (this implies large number of tags) or very big (this implies to generate one tag is very costly). The best scenario takes place when the size of the block is 4 or 8 KB.

It is also observed in Fig. 6 that with the increment of the size of the blocks, the time to check the proof decreases as well. However, the decrement was merely negligible.

Finally, we suggest that in practice, the best size of the block is between 4 and 8 KB, which delivers the best performance for our system.

8 Conclusion

In this paper, we investigated data privacy issues in remote data integrity-checking protocols. We showed that the existing privacy-preserving remote data integrity-checking protocol [24] could not achieve the desired goal of "leaking no information to a third party". We formalized the notion of "zero-knowledge privacy" and proposed an improved version of the protocol in [24] to achieve this property. In addition, we proved that our protocol satisfied other security requirements. Finally, both the performance analysis and the implementation showed that our improvement was practical.

Acknowledgments This work was supported by National Natural Science Foundation of China (U1135004, 61170080), Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011), and High-level Talents Project of Guangdong Institutions of Higher Education (2012).

References

1. Fernandes, D.A.B., Soares, L.F.B., Gomes, J.V., Freire, M.M., Incio, P.R.M.: Security issues in cloud environments: a survey. Int. J. Inf. Secur. 1–58 (2013). doi:10.1007/s10207-013-208-7
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
3. Wei, L., Zhu, H., Cao, Z., Dong, X., Jia, W., Chen, Y., Vasilakos, A.V.: Security and privacy for storage and computation in cloud computing. Inf. Sci. **258**(10), 371–386 (2014)
4. Cloud Security Alliance, Top threats to cloud computing (2010). http://www.cloudsecurityalliance.org
5. Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: Proceedings 32nd Annual Symposium on Foundations of Computer Science (FOCS 1991), pp. 90–99 (1991)
6. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song,D.X.: Provable data possession at untrusted stores. In: Proceedings 14th ACM Conference on Computer and Communications Security (ACM CCS 2007), pp. 598–609 (2007)
7. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Khan, O., Kissner, L., Peterson, Z.N.J., Song, D.: Remote data checking using provable data possession. ACM Trans. Inf. Syst. Secur. **14**, 1–34 (2011)
8. Juels, A., Pors Jr, B.S.K.: Proofs of retrievability for large files. In: Proceedings 14th ACM Conference on Computer and Communications Security (ACM CCS 2007), pp. 584–597 (2007)
9. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Proceedings 14th Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2008), pp. 90–107 (2008)
10. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling public verifiability and data dynamics for storage security in cloud computing. In: Proceedings 14th European Symposium on Research in Computer Security (ESORDICS 2009), pp. 355–370 (2009)
11. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings 29th Conference on Computer Communications (IEEE INFOCOM 2010), pp. 525–533 (2010)
12. Wang, C., Ren, K., Lou, W., Li, J.: Toward publicly auditable secure cloud data storage services. IEEE Netw. **24**(4), 19–24 (2010). doi:10.1109/MNET.2010.5510914
13. Wang, Q., Wang, C., Ren, K., Lou, W., Li, J.: Enabling public audibility and data dynamics for storage security in cloud computing. IEEE Trans. Parallel Distrib. Syst. **22**(5), 847–859 (2011)
14. Wang, C., Chow, S.S., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. IEEE Trans. Comput. **62**(2), 362–375 (2013)
15. Yang, K., Jia, X.: An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE Trans. Parallel Distrib. Syst. **24**(9), 1717–1726 (2013)
16. Zhu, Y., Ahn, G.-J., Hu, H., Yau, S.S., An, H.G., Hu, C.-J.: Dynamic audit services for outsourced storages in clouds. IEEE Trans. Serv. Comput. **6**(2), 227–238 (2013)
17. Zhu, Y., Hu, H., Ahn, G.-J., Yau, S.S.: Efficient audit service outsourcing for data integrity in clouds. J. Syst. Softw. **85**(5), 1083–1095 (2012)
18. Wang, H., Zhang, Y.: On the knowledge soundness of a cooperative provable data possession scheme in multicloud storage. IEEE Trans. Parallel Distrib. Syst. (2013). doi:10.1109/TPDS.2013.16
19. Ateniese, G., Pietro,R.D., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings 4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008) (2008). doi:10.1145/1460877.1460889
20. Wang, C., Wang, Q., Ren, K., Lou, W.: Ensuring data storage security in cloud computing. In: Proceedings 17th International Workshop on Quality of Service (IWQoS 2009) (2009)
21. Erway, C., Kupcu, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceedings 16th ACM Conference on Computer and Communications Security (ACM CCS 2009) (2009)
22. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptol. **17**, 297–319 (2004)

23. Sebé, F., Domingo-Ferrer, J., Martinez-Balleste, A., Deswarte, Y., Quisquater, J.J.: Efficient remote data possession checking in critical information infrastructures. IEEE Trans. Knowl. Data Eng. **20**(8), 1034–1038 (2008)
24. Hao, Z., Zhong, S., Yu, N.: A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. IEEE Trans. Knowl. Data Eng. **23**(9), 1432–1437 (2011)
25. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Commun. ACM **21**, 120–126 (1978)
26. Lin, S., Costello, D.J.: Error Control Coding, 2nd edn. Prentice-Hall, Upper Saddle River (2004)
27. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptol. **13**(3), 361–396 (2000)
28. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Proceedings 8th Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2002), pp. 125–142(2002)
29. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, Cambridge (2008)
30. Bosma, W., Cannon, J., Playoust, C.: The MAGMA algebra system. I. The user language. J. Symb. Comput. **24**(3–4), 235–265 (1997)