A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability

Zhuo Hao, Sheng Zhong, Nenghai Yu

***Abstract*—Remote data integrity checking is a crucial technology in cloud computing. Recently many works focus on providing data dynamics and/or public verifiability to this type of protocols. Existing protocols can support both features with the help of a third party auditor. In a previous work, Sebé et al. [1] propose a remote data integrity checking protocol that supports data dynamics. In this paper, we adapt Sebé et al.'s protocol to support public verifiability. The proposed protocol supports public verifiability without help of a third party auditor. In addition, the proposed protocol does not leak any private information to third party verifiers. Through a formal analysis, we show the correctness and security of the protocol. After that, through theoretical analysis and experimental results, we demonstrate that the proposed protocol has a good performance.**

***Index Terms*—Data integrity, Data dynamics, Public verifiability, Privacy.**

I. Introduction

Storing data in the cloud has become a trend [2], [3]. An increasing number of clients store their important data in remote servers in the cloud, without leaving a copy in their local computers. Sometimes the data stored in the cloud is so important that the clients must ensure it is not lost or corrupted. While it is easy to check data integrity after completely downloading the data to be checked, downloading large amounts of data just for checking data integrity is a waste of communication bandwidth. Hence, a lot of works [1], [4], [5], [6], [7], [8], [9] have been done on designing remote data integrity checking protocols, which allow data integrity to be checked without completely downloading the data.

In remote data integrity checking protocols, the client can challenge the server about the integrity of a certain data file, and the server generates responses proving that it has access to the complete and uncorrupted data. The basic requirements are that the client does not need to access the complete original data file when performing the verification of data integrity, and that the client should be able to verify integrity for an unlimited number of times. Furthermore, the protocol needs to be secure against a malicious server that tries to pass the data integrity verification without access to the complete and uncorrupted data.

This work was supported in part by NSF CNS-0845149 and CCF-0915374.

Zhuo Hao and Nenghai Yu are with the Department of EEIS, University of Science and Technology of China, Hefei, Anhui 230027, China. (email: hzhuo@mail.ustc.edu.cn; ynh@ustc.edu.cn).

Sheng Zhong is with the Department of Computer Science and Engineering, State University of New York at Buffalo, Amherst NY 14260, USA. (email:szhong@buffalo.edu). He is the corresponding author of this paper.

Most previously proposed protocols [1], [4], [5], [6], [7], [8], [9], [10], [11] meet all the above basic requirements. Nevertheless, in addition to these requirements, there are also three advanced requirements: *data dynamics*, *public verifiability* and *privacy against verifiers*. In the following we introduce the motivations and definitions of these advanced requirements.

- *Data dynamics [6], [7], [8], [11].* Data dynamics means after clients store their data at the remote server, they can dynamically update their data at later times. The main operations supported by data dynamics are data insertion, data modification and data deletion. Moreover, when data is updated, the verification metadata also needs to be updated. The updating overhead should be made as small as possible.
- *Public verifiability [4], [8], [9], [11].* Public verifiability allows *anyone* (not just the client) to perform the integrity checking operation. This function avoids disputes between the client and the server regarding data integrity. Whenever there is such a dispute, a third party authority can easily judge whether the data integrity is maintained using the verification protocol.
- *Privacy against verifiers [9], [11].* When the verification is performed by a third party verifier (not by the client), the protocol must ensure that no private information contained in the data is leaked.

In a realistic application, these advanced features may be needed at the same time. For example, consider an online document system, in which the client can create and modify her documents. The client can also cooperate on a document with her partners. The remote data integrity checking can ensure the integrity of the client's documents. When the client or her partners modify the document, the document and the tags need to be updated. This is supported by data dynamics. However, if the file gets corrupted and the server refuses to admit it, then a third party authority can easily prove the data corruption to the court. This is supported by the public verifiability. Furthermore, if the document contains sensitive data, such as business secrets, the client will not want it to be disclosed to the third party authorities. The privacy against third-party verifiers can ensure that.

The previously proposed protocols [1], [4], [5], [6], [7], [12], [8], [9], [10], [11] try to achieve these advanced features. The protocols in [6], [7], [12], [8], [11] support data dynamics at the block level, including block insertion, block modification and block deletion. The protocol of [4] supports data append operation. In addition, the protocol in [1] can be easily extended to support data dynamics.

[9], [10] can be adapted to support data dynamics by using the techniques of [8]. On the other hand, protocols in [4], [8], [9], [10], [11], [13] support public verifiability, by which *anyone* (not just the client) can perform the integrity checking operation. The protocols in [9], [10], [11], [13] support privacy against third party verifiers.

We compare the proposed protocol with selected previous protocols and summarize the results in Table I. The probabilistic guarantee of data integrity is achieved by using the probabilistic checking method proposed in [4]. By using that method, the verifier selectively checks the integrity of c blocks, so that only c blocks are guaranteed at the server side. However, because the c blocks are randomly selected, the detection probability will be high if the server deletes a fraction of all the blocks. The protocol in [1] and the proposed protocol achieve deterministic guarantee of data integrity, because they check the integrity of all the data blocks. However, both the protocol in [1] and the proposed protocol can be easily transformed into a more efficient one by using the probabilistic checking method [4].

In this paper, we have the following main contributions:

- We propose a remote data integrity checking protocol for cloud storage, which can be viewed as an adaptation of Sebé et al.'s protocol [1]. The proposed protocol inherits the support of data dynamics from [1], and supports public verifiability and privacy against third-party verifiers, while at the same time it doesn't need to use a third-party auditor.
- We give a security analysis of the proposed protocol, which shows that it is secure against the untrusted server and private against third party verifiers.
- We have theoretically analyzed and experimentally tested the efficiency of the protocol. Both theoretical and experimental results demonstrate that our protocol is efficient.

The rest of this paper is organized as follows. In Section II, technical preliminaries are presented. In Section III, the proposed remote data integrity checking protocol is presented. In Section IV, a formal analysis of the proposed protocol is presented. In Section V, we describe the support of data dynamics of the proposed protocol. In Section VI, the protocol's complexity is analyzed in the aspects of communication, computation and storage costs; furthermore, experimental results are presented for the efficiency of the protocol. In Section VII, the related work is reviewed. And finally, conclusions and possible future work are presented in Section VIII.

II. Technical Preliminaries

We consider a cloud storage system in which there are a client and an untrusted server. The client stores her data in the server without keeping a local copy. Hence, it is of critical importance that the client should be able to verify the integrity of the data stored in the remote untrusted server. If the server modifies any part of the client's data, the client should be able to detect it; furthermore, *any* third party verifier should also be able to detect it. In case a third party verifier verifies the integrity of the client's data, the data should be kept private against the third party verifier. Below we present a formal statement of the problem.

Problem Formulation Denote by m the file that will be stored in the untrusted server, which is divided into n blocks of equal lengths: $m = m_1 m_2 ... m_n$, where $n = \lceil |m|/l \rceil$. Here l is the length of each file block. Denote by $f_K(\cdot)$ a pseudo-random function which is defined as:

$$f : \{0,1\}^k \times \{0,1\}^{\log_2(n)} \rightarrow \{0,1\}^d,$$

in which k and d are two security parameters. Furthermore, denote the length of N in bits by $|N|$.

We need to design a remote data integrity checking protocol that includes the following five functions: SetUp, TagGen, Challenge, GenProof and CheckProof.

$\mathsf{SetUp}(1^k) \rightarrow (pk, sk)$: Given the security parameter k, this function generates the public key pk and the secret key sk. pk is public to everyone, while sk is kept secret by the client.

$\mathsf{TagGen}(pk, sk, m) \rightarrow \mathcal{D}_m$: Given pk, sk and m, this function computes a verification tag $\mathcal{D}_m$ and makes it publicly known to everyone. This tag will be used for public verification of data integrity.

$\mathsf{Challenge}(pk, \mathcal{D}_m) \rightarrow \mathsf{chal}$: Using this function, the verifier generates a challenge chal to request for the integrity proof of file m. The verifier sends chal to the server.

$\mathsf{GenProof}(pk, \mathcal{D}_m, m, \mathsf{chal}) \rightarrow R$: Using this function, the server computes a response R to the challenge chal. The server sends R back to the verifier.

$\mathsf{CheckProof}(pk, \mathcal{D}_m, \mathsf{chal}, R) \rightarrow$ {"success", "failure"}: The verifier checks the validity of the response R. If it is valid, the function outputs "success", otherwise the function outputs "failure". The secret key sk is not needed in the CheckProof function.

Besides these five functions, if the protocol supports data dynamics, it should also have functions such as block insertion, block deletion, and block modification.

Security Requirements There are two security requirements for the remote data integrity checking protocol: security against the server with public verifiability, and privacy against third party verifiers. We first give the definition of security against the server with public verifiability. In this definition, we have two entities: a challenger that stands for either the client or any third party verifier, and an adversary that stands for the untrusted server.

Definition 1. *(Security against the Server with Public Verifiability [4]) We consider a game between a challenger and an adversary that has four phases: Setup, Query, Challenge and Forge.*

- *Setup: The challenger runs the* SetUp *function, and gets the* (pk, sk)*. The challenger sends* pk *to the adversary and keeps* sk *secret.*
- *Query: The adversary adaptively selects some file blocks* $m_i, i = 1, 2, ..., n$ *and queries the verification tags from the challenger. The challenger computes a*

TABLE I
COMPARISONS BETWEEN THE PROPOSED PROTOCOL AND PREVIOUS PROTOCOLS.

	S-PDP[4]	[1]	[8]	DPDP[7]	[9], [10]	IPDP[11]	The Proposed Protocol
type of guarantee	probabilistic/ deterministic	deterministic	probabilistic				deterministic
public verifiability	Yes	No	Yes	No	Yes	Yes	Yes
With help of TPA	No	No	Yes	No	Yes	Yes	No
data dynamics	append only	Yes	Yes	Yes	No	Yes	Yes
privacy preserving	No	–	No	–	Yes	Yes	Yes
support for sampling	Yes	No	Yes	Yes	Yes	Yes	No
size of verification tags	O(n)						
communication	O(1)		O(clogn)	O(logn)	O(c)	O(s)	O(1)
server block access	O(c)	O(n)	O(c)				O(n)
server computation	O(c)	O(n)	O(clogn)		O(c)	O(c+s)	O(n)
verifier computation	O(c)	O(n)	O(clogn)		O(c)	O(c+s)	O(n)
client storage	O(1)	O(n)	O(1)				O(n)

* n is the block number, c is the sampling block number, and s is the number of sectors in a block.

verification tag D_i for each of these blocks and sends $D_i, i = 1, 2, ..., n$ to the adversary. According to the protocol formulation, $\mathcal{D}_m = \{D_1, D_2, ..., D_n\}$.

- *Challenge: The challenger generates the* chal *for the file blocks $\{m_1, m_2, ..., m_n\}$ and sends it to the adversary.*
- *Forge: The adversary computes a response R to prove the integrity of the requested file blocks.*

If $\mathsf{CheckProof}(pk, \mathcal{D}_m, \mathsf{chal}, R) =$ "$success$"*, then the adversary has won the game. The remote data integrity checking protocol is said to be secure against the server if for any $\mathcal{PPT}$ (probabilistic polynomial time) adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the game on a collection of file blocks is negligibly close to the probability that the challenger can extract these file blocks by a knowledge extractor $\mathcal{E}$.*

When the verifier is not the client herself, the protocol must ensure that no private information about the client's data is leaked to the third party verifier. We formalize this requirement using the simulation paradigm [14].

Before we proceed to the definition of this requirement, we introduce some related notations. Let $f = (f_1, f_2)$ be a $\mathcal{PPT}$ functionality and let Π be a two-party protocol for computing f. During the execution of the protocol Π, denote the view of the first (resp., second) party by $\text{view}_1^\Pi(x, y)$ (resp., $\text{view}_2^\Pi(x, y)$). $\text{view}_1^\Pi(x, y)$ (resp., $\text{view}_2^\Pi(x, y)$) includes $(x, r^1, m_1^1, ..., m_t^1)$ (resp., $(x, r^2, m_1^2, ..., m_t^2)$) where r^1 (resp., r^2) represents the outcome of the first (resp., second) party's internal coin tosses, and m_i^1 (resp., m_i^2) represents the ith message it has received. Denote the output of the first (resp., second) party during the execution of Π on (x, y) by $\text{output}_1^\Pi(x, y)$ (resp., $\text{output}_2^\Pi(x, y)$), which is implicit in the party's own view of the execution. Let $\text{output}^\Pi(x, y) = (\text{output}_1^\Pi(x, y), \text{output}_2^\Pi(x, y))$. We denote the verifier and the server by $\mathcal{V}$ and $\mathcal{P}$ respectively.

Definition 2. *(Privacy against Semi-Honest Behavior) For a functionality f, Π is said to privately compute f if there exist probabilistic polynomial time algorithms, denoted S_1 and S_2, such that*

$$\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{\textit{view}_1^\Pi(x, y)\}_{x,y\in\{0,1\}^*},$$
$$\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{\textit{view}_2^\Pi(x, y)\}_{x,y\in\{0,1\}^*}.$$

Note that $\overset{c}{\equiv}$ denotes computational indistinguishability.

From Definition 2 we define the privacy against third party verifiers, which is given in Definition 3.

Definition 3. *(Privacy against Third Party Verifiers) For the remote data integrity checking protocol Π, if there exists a $\mathcal{PPT}$ simulator $\mathcal{S}_\mathcal{V}$ such that*

$$\{\mathcal{S}_\mathcal{V}(x, f_V(x, y))\}_{x,y\in\{0,1\}^*} \overset{c}{\equiv} \{\textit{view}_V^\Pi(x, y)\}_{x,y\in\{0,1\}^*},$$

then Π is a protocol that ensures privacy against third-party verifiers.

Data Dynamics at Block Level Data dynamics means after clients store their data at the remote server, they can dynamically update their data at later times. At the block level, the main operations are block insertion, block modification and block deletion. Moreover, when data is updated, the verification metadata also needs to be updated. The updating overhead should be made as small as possible.

Homomorphic Verifiable Tags Our construction of the remote data integrity checking protocol uses *homomorphic verifiable tags (HVT)* introduced in [4]. A HVT of a message m is a pre-computed tag which is later used for the integrity checking. Denote the HVT of a message m_i by D_i. The HVT has the following two features:

- Blockless verification: By using HVTs, the server can construct a proof of possession of a certain file blocks, while the client needs not have access to these file blocks.

- Homomorphic property: For any two messages m_i and m_j, the tag for m_i+m_j can be generated by combining D_i and D_j.

In our construction, we use a RSA-based HVT, which is defined as follows. Let $N = pq$ be one publicly known RSA modulus. We know that $\{e : e \in \mathbb{Z}_N \text{ and } \gcd(e, N) = 1\}$ forms a multiplicative group. Denote this group by $\mathbb{Z}_N^*$. Denote an element in $\mathbb{Z}_N^*$ with a large order by g. The RSA-based HVT for message m_i is defined as $\mathsf{Tag}(m_i) = g^{m_i} \bmod N$. Its homomorphic property can be deduced from its definition. When $\mathsf{Tag}(m_i)$ and $\mathsf{Tag}(m_j)$ are tags of m_i and m_j respectively, the tag for m_i+m_j can be generated by computing $\mathsf{Tag}(m_i+m_j) = \mathsf{Tag}(m_i)\cdot\mathsf{Tag}(m_j) = (g^{m_i} \bmod N) \cdot (g^{m_j} \bmod N) = g^{m_i+m_j} \bmod N$.

III. The Proposed Remote Data Integrity Checking Protocol

In this section we describe the proposed remote data integrity checking protocol. Just as mentioned in Section II, the proposed protocol has functions SetUp, TagGen, Challenge, GenProof and CheckProof, as well as functions for data dynamics. In the following we present the former five functions of the proposed protocol. We leave the functions for data dynamics to Section V.

$\mathsf{SetUp}(1^k) \rightarrow (pk, sk)$: Let $N = pq$ be one publicly known RSA modulus, in which $p = 2p' + 1, q = 2q' + 1$ are two large primes. p' and q' are also primes. In addition, all the quadratic residues modulo N form a multiplicative cyclic group, which we denote by QR_N. Denote the generator of QR_N by g.[1] Since the order of QR_N is $p'q'$, the order of g is also $p'q'$. Let $pk = (N, g)$ and $sk = (p, q)$. pk is then released to be publicly known to everyone, and sk is kept secret by the client.

$\mathsf{TagGen}(pk, sk, m) \rightarrow \mathcal{D}_m$: For each file block $m_i, i \in [1, n]$, the client computes the block tag as

$$D_i = (g^{m_i}) \bmod N.$$

Without loss of generality, we assume that each block is unique. If in some particular applications, there exist blocks with the same value, then we differentiate them by adding a random number in each of them. Let $\mathcal{D}_m = \{D_1, D_2, ..., D_n\}$. After finishing computing all the block tags, the client sends the file m to the remote server, and releases $\mathcal{D}_m$ to be publicly known to everyone.

$\mathsf{Challenge}(pk, \mathcal{D}_m) \rightarrow \mathsf{chal}$: In order to verify the integrity of the file m, the verifier generates a random key $r \in [1, 2^k - 1]$ and a random group element $s \in \mathbb{Z}_N \backslash \{0\}$. The verifier then computes $g_s = g^s \bmod N$ and sends $\mathsf{chal} = \langle r, g_s \rangle$ to the server.

$\mathsf{GenProof}(pk, \mathcal{D}_m, m, \mathsf{chal}) \rightarrow R$: When the server receives $\mathsf{chal} = \langle r, g_s \rangle$, it generates a sequence of block indexes $a_1, a_2, ..., a_n$ by calling $f_r(i)$ for $i \in [1, n]$ iteratively. Then the server computes

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

and sends R to the verifier.

$\mathsf{CheckProof}(pk, \mathcal{D}_m, \mathsf{chal}, R) \rightarrow \{\text{“success”}, \text{“failure”}\}$: When the verifier receives R from the server, she computes $\{a_i\}_{i=1,...,n}$ as the server does in the GenProof step. Then the verifier computes P and R' as follows:

$$P = \prod_{i=1}^{n} (D_i^{a_i} \bmod N) \bmod N$$

$$R' = P^s \bmod N$$

After that the verifier checks whether $R' = R$. If $R' = R$, output “success”. Otherwise the verification fails and the verifier outputs “failure”.

Note that in the TagGen function, we make all the blocks distinct by adding random numbers in blocks with the same value. If the server still tries to save its storage space, then the only way is by breaking the prime factorization of N, or equally, getting a multiple of $\phi(N)$. The hardness of breaking large number factorization makes the proposed protocol secure against the untrusted server. We put the formal analysis of the proposed protocol in Section IV.

IV. Correctness and Security Analysis

In this section, we first show that the proposed protocol is correct in the sense that the server can pass the verification of data integrity as long as both the client and the server are honest. Then we show that the protocol is secure against the untrusted server. These two theorems together guarantee that, assuming the client is honest, if and only if the server has access to the complete and uncorrupted data, it can pass the verification process successfully. Finally we show that the proposed protocol is private against third party verifiers.

Theorem 1. *If both the client and the server are honest, then the server can pass the verification successfully.*

Proof: We prove this theorem by showing that R and R' should be equal if all the data blocks are kept completely at the server. From the $\mathsf{TagGen}(m)$ function, we get that $D_i = (g^{m_i}) \bmod N, i \in [1, n]$. Then we get

$$\begin{aligned} P &= \prod_{i=1}^{n} (D_i^{a_i} \bmod N) \bmod N \\ &= \prod_{i=1}^{n} (g^{a_i m_i} \bmod N) \bmod N \\ &= g^{\sum_{i=1}^{n} a_i m_i} \bmod N \end{aligned}$$

Then

$$\begin{aligned} R' &= P^s \bmod N \\ &= g^{s \sum_{i=1}^{n} a_i m_i} \bmod N \\ &= {g_s}^{\sum_{i=1}^{n} a_i m_i} \bmod N \\ &= R \end{aligned}$$

This completes the proof. ∎

Before we proceed to Theorem 2, we first review the KEA1-r assumption, which has been investigated in [15], [16], and adapted to the RSA setting in [4].

[1] A simple way to compute g is to let $g = b^2$, in which $b \xleftarrow{R} \mathbb{Z}_N^*$ and $\gcd(b \pm 1, N) = 1$.

Definition 4. ***KEA1-r****(Knowledge of Exponent Assumption [4]). For any adversary A taking input (N, g, g^s) and returning (C, Y) with $Y = C^s$, there exists "extractor" $\bar{A}$, which given the same input as A returns c such that $C = g^c$.*

Our proof of Theorem 2 needs a lemma from [17] on solving prime factorization when a multiple of $\lambda'(n)$ is known. Denote the prime factorization of N by ${p_1}^{v_1} \cdots {p_t}^{v_t}$. Then $\lambda'(n) = \mathsf{lcm}(p_1 - 1, \cdots, p_t - 1)$, in which lcm denotes least common multiple.

Lemma 1. *[17] Let g be any function that satisfies the following two conditions:*

1) $\lambda'(n) | g(n)$,

2) $|g(n)| = O(|n|^k)$ for some constant k.

Then "prime factorization" is polynomial time reducible to g. Furthermore, the cost of solving the prime factorization of n is $O(|n|^{k+4} M(|n|))$, in which $M(|n|)$ denotes the cost of multiplying two integers of binary length $|n|$.

We refer the readers to [17] for more details of Lemma 1.

Theorem 2. *Under the KEA1-r and the large integer factorization assumptions, the proposed protocol is secure against the untrusted server.*

Proof: Just as in the security formulation, we denote the adversary by $\mathcal{A}$ and the challenger by $\mathcal{B}$. What we want to prove is that for any $\mathcal{PPT}$ adversary who wins the data possession game on some file blocks, the challenger can construct a knowledge extractor $\mathcal{E}$ that extracts these file blocks. Equivalently, if $\mathcal{E}$ cannot extract these file blocks, the challenger can break the integer factorization problem.

For the large integer factorization problem, $\mathcal{B}$ is given a large integer N, which is product of two large primes p and q. Here $p = 2p' + 1$ and $q = 2q' + 1$. $\mathcal{B}$ tries to solve the prime factorization of N.

$\mathcal{B}$ simulates the protocol environment for $\mathcal{A}$ with the following steps:

- Setup: $\mathcal{B}$ generates a random generator of QR_N. Denote the generator by g. $\mathcal{B}$ sends $pk = (N, g)$ to $\mathcal{A}$.
- Query: $\mathcal{A}$ adaptively selects some file blocks $m_i, i = 1, 2, ..., n$ and queries the verification tags from $\mathcal{B}$. $\mathcal{B}$ computes a verification tag $D_i = g^{m_i} \bmod N$ for each of these blocks and sends $\{D_i, i = 1, 2, ..., n\}$ to $\mathcal{A}$. Let $\mathcal{D}_m = \{D_1, D_2, ..., D_n\}$. $\mathcal{D}_m$ is made publicly known to everyone.
- Challenge: $\mathcal{B}$ generates a chal for the file blocks $\{m_1, m_2, ..., m_n\}$ and sends it to $\mathcal{A}$. The generation method is the same with that in the $\mathsf{Challenge}$ function described in section III. Let $\mathsf{chal} = \langle r, g_s \rangle$.
- Forge: $\mathcal{A}$ computes a response R to prove the integrity of the requested file blocks.

If $\mathsf{CheckProof}(pk, \mathcal{D}_m, \mathsf{chal}, R) =$ "$success$", then the adversary has won the game. Note that $\mathcal{A}$ is given (N, g, g^s) as input, and outputs $R = g^{s \sum_{i=1}^{n} a_i m_i} \bmod N$, in which $a_i = f_r(i)$ for $i \in [1, n]$. Because $\mathcal{A}$ can naturally computes $P = g^{\sum_{i=1}^{n} a_i m_i} \bmod N$ from $\mathcal{D}_m$, P is also treated as $\mathcal{A}$'s output. So $\mathcal{A}$ is given (N, g, g^s) as input, and outputs (R, P) that satisfies $R = P^s$. From the KEA1-r assumption, $\mathcal{B}$ can construct an extractor $\bar{A}$, which given the same input as $\mathcal{A}$, outputs c which satisfies $P = g^c \bmod N$. As $P = g^{\sum_{i=1}^{n} a_i m_i} \bmod N$, $\mathcal{B}$ extracts $c = \sum_{i=1}^{n} a_i m_i \bmod p'q'$.

Now $\mathcal{B}$ generates n challenges $\langle r_1, g_{s_1} \rangle$, $\langle r_2, g_{s_2} \rangle$, ..., $\langle r_n, g_{s_n} \rangle$ using the method described in section III. $\mathcal{B}$ computes $a_i^j = f_{r_j}(i)$ for $i \in [1, n]$ and $j \in [1, n]$. Because $\{r_1, r_2, ..., r_n\}$ are chosen by $\mathcal{B}$, now $\mathcal{B}$ chooses them so that $\{a_1^j, a_2^j, ..., a_n^j\}, j = 1, 2, ..., n$ satisfy the following equation:

$$\det \begin{bmatrix} a_1^1 & a_2^1 & ... & a_n^1 \\ a_1^2 & a_2^2 & ... & a_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ a_1^n & a_2^n & ... & a_n^n \end{bmatrix} \neq 0. \tag{1}$$

Here $\det[\cdot]$ denotes the determinant of a matrix. $\mathcal{B}$ challenges $\mathcal{A}$ for n times. On the jth time, $\mathcal{B}$ challenges $\mathcal{A}$ with $\{r_j, g_{s_j}\}$. From the response of $\mathcal{A}$, $\mathcal{B}$ extracts $c_j = a_1^j m_1 + a_2^j m_2 + \cdots + a_n^j m_n \bmod p'q'$.

When equation (1) holds, the following system of linear equations has a unique solution.

$$\begin{cases} a_1^1 m_1 + a_2^1 m_2 + ... + a_n^1 m_n = c_1 \bmod p'q', \\ a_1^2 m_1 + a_2^2 m_2 + ... + a_n^2 m_n = c_2 \bmod p'q', \\ \vdots \\ a_1^n m_1 + a_2^n m_2 + ... + a_n^n m_n = c_n \bmod p'q'. \end{cases} \tag{2}$$

By solving the above equation set, for each file block $m_i, i = 1, 2, ..., n$, $\mathcal{B}$ gets m_i^* which satisfies $m_i^* = m_i \bmod p'q'$. If for each $m_i^*, i = 1, 2, ..., n$, $m_i^* = m_i$, then $\mathcal{B}$ has successfully extracted all the file blocks $m_i, i = 1, 2, ..., n$. However, if for some i, $m_i^* \neq m_i$, then we show that $\mathcal{B}$ can successfully compute the prime factorization of N. Without loss of generality, we assume m_i is larger than m_i^*. Then $\mathcal{B}$ can get a multiple of $\phi(N)$ from $m_i^* = m_i \bmod p'q'$, which we denote by $k_1\phi(N)$. From Lemma 1, $\mathcal{B}$ can solve the prime factorization of N with the cost of $O((|k_1| + |\phi(N)|)|N|^4 M(|N|))$. Because $\mathcal{A}$ is a $\mathcal{PPT}$ adversary, the length of k_1 is bounded by $O(|N|^{k_2})$ for some constant k_2. From the above we can see that if any file block cannot be extracted, then $\mathcal{B}$ can construct a knowledge extractor $\mathcal{E}$ to extract the prime factorization of N in probabilistic polynomial time.

In conclusion, under the KEA1-r and large integer factorization assumptions, the proposed protocol guarantees the data integrity against an untrusted server. ■

Theorem 3. *(Privacy against Third Party Verifiers) Under the semi-honest model [14], a third party verifier cannot get any information about the client's data m from the protocol execution. Hence, the protocol is private against third party verifiers.*

Proof: In this proof we construct a simulator for the view of the verifier, and show that the output of the simulator is computationally indistinguishable with the view

of the verifier. The simulator is provided with the input and the output of the verifier. The input of the verifier includes $\{N, g, \{D_i\}_{i=1,2,...,n}\}$. The output of the verifier is a bit b indicating whether the CheckProof function outputs "success" or "failure". Under the semi-honest model, the server is honest, so the value of b is always 1.

The simulator consists of the following steps:

Step 1: The simulator generates one random element $r_1 \in [1, 2^k - 1]$ and another random element $s_1 \in \mathbb{Z}_N \backslash \{0\}$. The simulator then computes $g_s^1 = g^{s_1} \bmod N$.

Step 2: The simulator generates a sequence of numbers $a_i^1 = f_{r_1}(i), i = 1, 2, ..., n$.

Step 3: The simulator feeds the challenge $\langle r_1, g_s^1 \rangle$ to the server, and gets the output b.

Step 4: Using the knowledge of $\{D_i\}_{i\in[1,n]}$, The simulator computes $P_1 = \prod_{i=1}^{n} (D_i^{a_i^1} \bmod N) \bmod N$ and $R_1^{'} = {P_1}^{s_1} \bmod N$.

Step 5: Let $x = \{N, g, \{D_i\}_{i=1,2,...,n}\}$ denotes the verifier's input. Let $RC = \langle r_1, s_1 \rangle$. Then the simulator outputs $(x, RC, R_1^{'})$.

Denote the verifier's view during the protocol execution as view_v^P. According to the definition in [14], we get view_v^P as follows:

$$\mathsf{view}_v^P = (x, \langle r, s \rangle, (g_s)^{\sum_{i=1}^{n} a_i m_i} \bmod N). \quad (3)$$

$\{D_i\}_{i=1,2,...,n}$ are block tags which are publicly available to anyone, so they are also included in the verifier's view.

In the simulator's output and view_v^P, $\langle r_1, s_1 \rangle$ and $\langle r, s \rangle$ are identically distributed. Because $\langle r_1, s_1 \rangle$ and $\langle r, s \rangle$ are identically distributed, $R_1^{'}$ and $(g_s)^{\sum_{i=1}^{n} a_i m_i} \bmod N$ are also identically distributed.

In summary, the simulator's output and view_v^P have identical distributions, which are also computationally indistinguishable. So the verifier cannot get any information from the received messages, except its input and output. ■

V. Data Dynamics

The proposed protocol supports data dynamics at the block level, which includes block insertion, block modification and block deletion. Our protocol can easily support dynamic data updates because: (1) each block tag depends only on the block content, not on the block position; (2) each block tag D_i depends only on m_i, not on any other blocks. Next we show how our protocol supports these operations.

- **Block Insertion.** Assume the client wants to insert a new block m_x before the block m_i, $1 \le i \le n$ or append m_x after m_n. Then the server updates the stored file to $m' = m_1 m_2 ... m_{i-1} m_x m_i ... m_n$ or $m' = m_1 m_2 ... m_n m_x$. And the client computes block tag for the new block, i.e., $D_x = g^{m_x} \bmod N$ and changes the block tag to $D_1 D_2 ... D_{i-1} D_x D_i ... D_n$ or $D_1 D_2 ... D_n D_x$.
- **Block Modification.** Assume that the client wants to modify the ith block m_i of her file. Denote the modified data block by m_i^*. Then the server updates m_i to m_i^*. Next the client computes a new block tag for the updated block, i.e., $D_i^* = g^{m_i^*} \bmod N$.
- **Block Deletion.** When the client wants to delete one block or several blocks of her file, she can delete these blocks from the server and also delete the corresponding block tags.

From the above we can see that the correspondence relationship between the block and the digest does not change after the data updating, i.e., $D_i = g^{m_i} \bmod N, i = 1, 2, ..., \lceil |m|/l \rceil$. So the data integrity is still protected. If the client wants to make sure that the file has really been updated, she can launch a proof request immediately by sending a challenge to the server. Any block that is updated is given a novel random number, so that each block remains unique. Therefore, the server cannot delete any block without being detected.

VI. Complexity Analysis and Experimental Results

In this section, we first present a complexity analysis of the communication, computation and storage costs of the proposed protocol. After that, we present the experimental results.

A. Communication, Computation and Storage Costs

The communications between the verifier and the server occur in the Challenge and GenProof steps. In the Challenge step, the verifier sends the challenge $\langle r, g_s \rangle$ to the server, which is of binary length $k + |N|$. In the GenProof step, the server sends the response R to the verifier, which is $|N|$ bits. So the total communication cost is $k + 2|N|$ bits.

Next we give an analysis of the computation costs of the client, the server and the verifier. The reason we present analysis for the client and the verifier separately is that the proposed protocol offers the property of public verifiability, so that anybody can be a verifier.

- **Client side.** During the TagGen step, the client computes a tag for each of the file blocks. From Euler Theorem [18], we know that since $\gcd(g, N) = 1$, we should have $g^{\phi(N)} \bmod N = 1$. So the client can compute $m_i \bmod \phi(N)$ before computing $g^{m_i} \bmod N$. As modulo operations are far more efficient than the modular exponentiations, we only consider the latter. The computation cost of the client is upper bounded by $\lceil |m|/l \rceil T_{exp}(|N|, N)$, where $T_{exp}(len, num)$ is the time cost of computing a modular exponentiation with a len-bit long exponent modular num.
- **Server side.** During the protocol, the server needs to perform n pseudorandom number generations and to compute $R = (g_s)^{\sum_{i=1}^{n} a_i m_i} \bmod N$. During the computation of $\sum_{i=1}^{n} a_i m_i$, n large number multiplications are performed. Since a_i and m_i are of d and l bits long respectively, the computation cost of $a_i m_i$ is upper bounded by $(d - 1)$ additions of $(d + l)$-bit integers. Once the values of $a_i m_i, i = 1, 2, ..., n$ have

been computed, the computation cost of their sum is upper bounded by $(n-1)$ additions of $|n|+d+l$-bit integers. In summary, the computation cost of the server is upper bounded by

$$n \cdot T_{prng}(d) + T_{exp}(|n|+d+l, N) + n(d-1)T_{add}(d+l) + (n-1)T_{add}(|n|+d+l), \quad (4)$$

where $T_{prng}(len)$ is the time cost of generating a len-bit pseudo-random number and $T_{add}(len)$ is the time cost of adding two len-bit numbers. Since the cost for computing $T_{add}(d+l)$ is less than for computing $T_{add}(|n|+d+l)$, by replacing n with $\lceil |m|/l \rceil$, we can simplify equation (4) to:

$$\lceil |m|/l \rceil \cdot T_{prng}(d) + T_{exp}(|\lceil |m|/l \rceil| + d + l, N) + d\lceil |m|/l \rceil T_{add}(|\lceil |m|/l \rceil| + d + l).$$

- **Verifier side.** During the Challenge step, the verifier needs to generate two random numbers $\langle r, s \rangle$ and compute $g_s = g^s \bmod N$, whose cost is 2 pseudorandom number generations plus $T_{exp}(|N|, N)$. Then during the CheckProof step, the verifier computes $\{a_i\}_{i=1,...,n}$, P and R' respectively, whose computations include n pseudorandom number generations, $(n+1)$ modular exponentiations and $(n-1)$ modular multiplications. So the total computation cost of the verifier is

$$T_{prng}(|N|) + T_{prng}(k) + (\lceil |m|/l \rceil) \cdot T_{prng}(d) + 2T_{exp}(|N|, N) + nT_{exp}(d, N) + (n-1)T_{mul}(|N|, N). \quad (5)$$

In equation (5), $(n-1)T_{mul}(|N|, N)$ corresponds to the computation of $(n-1)$ multiplications of $|N|$-bit integers modular N. So the computation cost of these multiplications is upper bounded by $(n-1)(|N|-1)$ additions of $|N|$-bit integers. In summary, the computation cost of the verifier is upper bounded by:

$$T_{prng}(|N|) + T_{prng}(k) + (\lceil |m|/l \rceil) \cdot T_{prng}(d) + 2T_{exp}(|N|, N) + \lceil |m|/l \rceil T_{exp}(d, N) + |N|\lceil |m|/l \rceil T_{add}(|N|). \quad (6)$$

In addition, the computation cost for block insertion or block modification is just one modular exponentiation, which is $T_{exp}(|N|, N)$.

Finally, we analyze the storage cost of the server and the verifier. Note that the n block tags $D_1, D_2, ..., D_n$ are made publicly known to everyone. They can be stored at the server side, the client side or the verifier side. If the tags are stored at the server, then traditional integrity protection methods such as digital signatures can be used to protect them from being tampered with by the server. The storage cost of the block tags is upper bounded by $\lceil |m|/l \rceil |N|$ bits. In this case, when the data integrity checking is performed, the tags are transmitted back to the verifier from the server. As the tags have been signed by the client's private key, the server cannot tamper with them. This will incur communication costs that are linear to the number of blocks. However, because the tags are relatively small compared with the original data, the incurred communication costs are acceptable with respect to all the good features the proposed protocol has. If the tags are stored at the verifier or the client, then these communication costs are mitigated. However, this will cause a storage cost of $O(n)$ at the verifier or the client, which is the same as Sebé et al.'s protocol [1].

Besides that, the storage costs of the client, the server and the verifier are analyzed below.

- **Client side.** The client needs to store the public key and the private key. The storage cost is $2|N|+|p|+|q|$ bits.
- **Server side.** The server needs to have complete access to the whole file m, so its storage cost is $|m|$ bits. Note that this is the minimum storage cost possible for any scheme.
- **Verifier side.** The verifier needs to have access to the public key $pk = (N, g)$. So the storage requirement at the verifier is $2|N|$ bits.

B. Experimental Results

In the experiment, we measure the computation costs at the verifier and the server when the file length is fixed and the block size is changed. After that, we measure the computation costs when the file length changes and the block size is fixed. We also measure the client's preprocessing costs.

In the first experiment we use a file m with length 2^{25} bits, or 4MB. We choose the length of N to be 1024 bits. We choose $k = 128$ and $d = 128$. We implement the proposed protocol on a laptop with Intel Core2 Duo 2.00GHz CPU and 1.99GB memory. All the programs are written in the C++ language with the assistance of MIRACL library [19]. We use SHA-1 secure hash algorithm [20] to implement the pseudo-random generator when the length of the random number is less than 160 bits. However, in our protocol, the generation of s requires more than 160 bits, which is implemented by calling the well-implemented pseudo-random number generator in the MIRACL library.

We measure the computation costs at the verifier and the server side, which are shown in Table II. From Table II we can see that when the block length is 2^{18} bits (32KB), the computation cost at the verifier is 173.39 ms, and the computation cost at the server is 2304.39 ms.

TABLE II
COMPUTATION COSTS AT THE VERIFIER AND THE SERVER WITH $|N| = 1024$ AND $|m| = 4MB$.

l(bits)	Verifier (ms)	Server (ms)
65,536(2^{16})	653.37	591.1
131,072(2^{17})	328.81	1161.1
262,144(2^{18})	173.39	2304.39
524,288(2^{19})	95.46	4558.67

On the other hand, we measure the computation costs at the verifier and the server side with different file lengths

and fixed block size. The result is shown in Table III. From Table III, we can see that the computation cost at the server doesn't increase much when the file length increases. But the computation cost at the verifier increases nearly proportionally with the increasing file length. We note that this is consistent with the theoretical analysis in Section VI-A. The exponential operation is the most time-consuming operation at either side. When the file length increases, the times of exponential operations at the verifier increase proportionally. However, the server needs to perform only one exponential operation no matter how large the file is. Therefore, the server's burden is not increased much with larger files. As to the verifier's load, our scheme can be efficient when the file length is not huge. However, when the file length is very large, our protocol can be easily extended into a probabilistic one by using the probabilistic framework proposed in [4]. In that case, the extended protocol will provide probabilistic data possession guarantee while its other good features are still kept.

TABLE III
COMPUTATION COSTS AT THE VERIFIER AND THE SERVER WITH DIFFERENT FILE LENGTHS AND FIXED BLOCK SIZE.

File Length	l(bits)	Verifier (ms)	Server (ms)
1MB	65,536(2^{16})	176.24	568.39
2MB	65,536(2^{16})	332.55	574.2
4MB	65,536(2^{16})	653.37	591.1
8MB	65,536(2^{16})	1281.79	618.04

TABLE IV
THE CLIENT'S PRE-PROCESSING TIME WITH DIFFERENT BLOCK LENGTHS WHEN $|N| = 1024$ AND $|m| = 4MB$.

Block length (bits)	Block number	Processing time (ms)
65,536(2^{16})	512	4,765
131,072(2^{17})	256	2,477
262,144(2^{18})	128	1,328
524,288(2^{19})	64	755

Pre-processing Costs We also measure the client's pre-processing time during the TagGen process, which is shown in Table IV. From Table IV we can see that when the block length is 2^{18} bits (32KB), the client's pre-processing time is 1,328 ms.

In addition, when the block length is 2^{18} bits (32KB), the storage cost at the verifier side is $|N|\lceil|m|/l\rceil$ = $1024 \cdot 128$ $bits$ = 16KB. The communication cost is $k + 2|N|$ = (128+2$\cdot$ 1024)bits = 272 bytes.

VII. RELATED WORK

The problem of remote data integrity checking is first introduced in [21], [22], which independently propose RSA-based methods for solving this problem. After that Shah et al. [23] propose a remote storage auditing method based on pre-computed challenge-response pairs. In [5], Curtmola et al. propose a multiple-replica provable data possession protocol to deal with server collusion attack. In [24], Heitzmann et al. propose a data checking method with use of authenticated skip lists [25]. These protocols all provide good efficiency and security. However, none of them provides public verifiability or data dynamics.

Recently, Ateniese et al. [4] propose two provable data possession (S-PDP, E-PDP) schemes to provide integrity protection for remote data. The S-PDP and E-PDP support data block append operation, and a variant of their main PDP scheme has public verifiability. Sebé et al. [1] propose a remote data possession checking protocol for critical information infrastructures. Their protocol supports unlimited times of file integrity verifications and has a trade off between the running time and the storage cost at the verifier. Their protocol can be easily adapted to support data dynamics, but it doesn't support public verifiability. After that several studies [6], [7], [8] focus on providing data dynamics to provable data possession protocols. Ateniese et al. [6] propose a very efficient PDP protocol which is based on message authentication codes. Their protocol supports block modification, deletion and append. Erway et al. [7] propose two efficient PDP constructions with data dynamics by using rank-based skip lists and RSA trees [26]. Wang et al. [8] propose a method which uses merkle hash tree [27] to support fully data dynamics and uses BLS signature [28] to support public verifiability. Later Wang et al. [9], [10] recognize the need of privacy against third-party verifiers and develop a random masking technique to deal with this problem. Zhu et al. [11] propose a formal framework for interactive provable data possession (IPDP) and a zero-knowledge IPDP solution for private clouds. Their ZK-IPDP protocol achieves probabilistic data possession guarantee, supports fully data dynamics, public verifiability and is also private against the verifiers. Furthermore, they propose an efficient construction of cooperative provable data possession (CPDP), which can be used in hybrid clouds. The proposed schemes in [8], [9], [10], [11] use a third party auditor to perform the verification. A third party auditor has certain special expertise and technical capabilities, which the clients do not have. In addition, Curtmola et al. [29] use the forward error correcting codes to enhance the PDP protocol's robustness. Wang et al. [12] propose a protocol for ensuring remote storage security in the environment of multiple servers, and Ateniese et al. [30] propose a framework that can build a public verifiable proof of storage system from any public key homomorphic linear authenticators. Hao and Yu [13] propose a remote data possession checking protocol for the multiple replica setting, which supports public verifiability and privacy against third party verifiers. The proposed protocol can be viewed as an adaptation of [1] to support more functionalities. It inherits the support of data dynamics from [1], and supports public verifiability and privacy against third-party verifiers, while at the same time it doesn't need to use a third-party auditor.

On the other hand, Proof of Retrievability (PoR) [31] has been proposed as another technology for ensuring remote

data security. PoR puts stronger requirements on the server, which not only needs to prove that the data is stored intact at the server, but also needs to prove that the stored data can be completely retrieved when needed. Several recent studies [32], [33], [34] propose PoR schemes that improve the protocol security and efficiency. Additionally, Bowers et al. [35] propose a remote file integrity checking protocol that provides high availability and integrity.

VIII. Conclusions and Future Work

In this paper we propose a new remote data integrity checking protocol for cloud storage. The proposed protocol is suitable for providing integrity protection of customers' important data. The proposed protocol supports data insertion, modification and deletion at the block level, and also supports public verifiability. The proposed protocol is proved to be secure against an untrusted server. It is also private against third party verifiers. Both theoretical analysis and experimental results demonstrate that the proposed protocol has very good efficiency in the aspects of communication, computation and storage costs.

Currently we are still working on extending the protocol to support data level dynamics. The difficulty is that there is no clear mapping relationship between the data and the tags. In the current construction, data level dynamics can be supported by using block level dynamics. Whenever a piece of data is modified, the corresponding blocks and tags are updated. However, this can bring unnecessary computation and communication costs. We aim to achieve data level dynamics at minimal costs in our future work.

References

[1] F. Sebe, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 20, pp. 1034 –1038, aug. 2008.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009.

[3] C. Cachin, I. Keidar, and A. Shraer, "Trusting the cloud," *SIGACT News*, vol. 40, no. 2, pp. 81–86, 2009.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 598–609, ACM, 2007.

[5] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," in *ICDCS '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, (Washington, DC, USA), pp. 411–420, IEEE Computer Society, 2008.

[6] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication netowrks*, (New York, NY, USA), pp. 1–10, ACM, 2008.

[7] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 213–222, ACM, 2009.

[8] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *14th European Symposium on Research in Computer Security*, pp. 355–370, Springer Berlin / Heidelberg, September 2009.

[9] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *InfoCom2010, IEEE*, March 2010.

[10] C. Wang, S. S.-M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage." Cryptology ePrint Archive, Report 2009/579, 2009. http://eprint.iacr.org/.

[11] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Cooperative provable data possession." Cryptology ePrint Archive, Report 2010/234, 2010. http://eprint.iacr.org/.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Quality of Service, 2009. IWQoS. 17th International Workshop on*, pp. 1 –9, july 2009.

[13] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in *Data, Privacy, and E-Commerce, 2010. The Second International Symposium on*, IEEE, September 2010.

[14] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2004.

[15] I. Damgård, "Towards practical public key systems secure against chosen ciphertext attacks," in *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 445–456, Springer-Verlag, 1992.

[16] M. Bellare and A. Palacio, "The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols," in *Proc. of CRYPTO'04*, pp. 273–289, Springer, 2004.

[17] G. L. Miller, "Riemann's hypothesis and tests for primality," in *STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing*, (New York, NY, USA), pp. 234–239, ACM, 1975.

[18] G. Jones and J. Jones, "Elementary Number Theory," *Springer-Verlag, London*, 1998.

[19] *Multiprecision Integer and Rational Arithmetic C/C++ Library*. http://www.shamus.ie/.

[20] FIPS-PUB-180-1, *Secure Hash Standard*. National Institute of Standards and Technology (NIST), April 1995.

[21] Y. Deswarte and J.-J. Quisquater, "Remote Integrity Checking," in *Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)* (S. J. L. Strous, ed.), pp. 1–11, Kluwer Academic Publishers, 1 2004.

[22] D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer." Cryptology ePrint Archive, Report 2006/150, 2006. http://eprint.iacr.org/.

[23] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *Proc. of HotOS XI.*, Usenix, 2007.

[24] A. Heitzmann, B. Palazzi, C. Papamanthou, and R. Tamassia, "Efficient integrity checking of untrusted network storage," in *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*, (New York, NY, USA), pp. 43–54, ACM, 2008.

[25] M. T. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an authenticated dictionary with skip lists and commutative hashing," *DARPA Information Survivability Conference and Exposition,*, vol. 2, p. 1068, 2001.

[26] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Authenticated hash tables," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 437–448, ACM, 2008.

[27] R. C. Merkle, "Protocols for public key cryptosystems," *Security and Privacy, IEEE Symposium on*, p. 122, 1980.

[28] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004.

[29] R. Curtmola, O. Khan, and R. Burns, "Robust remote data checking," in *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*, (New York, NY, USA), pp. 63–68, ACM, 2008.

[30] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *ASIACRYPT 2009*, pp. 319–333, Springer Berlin / Heidelberg, 2009.

[31] A. Juels and B. S. Kaliski, Jr., "PORs: proofs of retrievability for large files," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 584–597, ACM, 2007.

[32] H. Shacham and B. Waters, "Compact proofs of retrievability," in *ASIACRYPT '08: Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security*, (Berlin, Heidelberg), pp. 90–107, Springer-Verlag, 2008.

[33] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *TCC '09: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, (Berlin, Heidelberg), pp. 109–127, Springer-Verlag, 2009.

[34] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*, (New York, NY, USA), pp. 43–54, ACM, 2009.

[35] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: a high-availability and integrity layer for cloud storage," in *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 187–198, ACM, 2009.