Secure Cloud Storage Hits Distributed String Equality Checking: More Efficient, Conceptually Simpler, and Provably Secure

Fei Chen*¶, Tao Xiang†, Yuanyuan Yang‡, Cong Wang§, Shengyu Zhang¶

*Department of Computer Science and Engineering, Shenzhen University. Email:feichenn@gmail.com.
†College of Computer Science, Chongqing University. Email: txiang@cqu.edu.cn.
‡Department of Electrical and Computer Engineering, Stony Brook University. Email: yuanyuan.yang@stonybrook.edu.
§Department of Computer Science, City University of Hong Kong. Email: congwang@cityu.edu.hk.
¶Department of Computer Science and Engineering, The Chinese University of Hong Kong. Email:syzhang@cse.cuhk.edu.hk.

***Abstract*—Cloud storage has gained a remarkable success in recent years with an increasing number of consumers and enterprises outsourcing their data to the cloud. To assure the availability and integrity of the outsourced data, several protocols have been proposed to audit cloud storage. Despite the formally guaranteed security, the constructions employed heavy cryptographic operations as well as advanced concepts (e.g., bilinear maps over elliptic curves and digital signatures), and thus are inefficient to admit wide applicability in practice. In this paper, we design a novel secure cloud storage protocol, which is conceptually and technically simpler and significantly more efficient than previous constructions. Inspired by a classic string equality checking protocol in distributed computing, our protocol uses only basic integer arithmetic (without advanced techniques and concepts). As simple as the protocol is, it supports both randomized and deterministic auditing to fit different applications. We further extend the proposed protocol to support data dynamics, i.e., adding, deleting and modifying data, using a novel technique. As a further contribution, we find a systematic way to design secure cloud storage protocols based on verifiable computation protocols. Theoretical and experimental analyses validate the efficacy of our protocol.**

I. Introduction

With the increasing popularity of cloud computing, the security of cloud storage has recently drawn considerable attention from the research community [1]–[7]. Various protocols with different strengths and weaknesses which can check the integrity of outsourced data have been proposed for securing cloud storage [1]–[7]. One can categorize the existing protocols from different perspectives. When functionality is concerned, some protocols do not support full data dynamics [1]–[3], [7], while the others do [4]–[6]. From the security perspective, some protocols rely their security on standard model [1], [2], [6], while others assume the random oracle model (ROM) [3]–[5]. From the technical perspective, some protocols employ only basic number theoretic operations [1]–[3], [7], while others build heavily on bilinear parings over elliptic curves [4], [5]; these different techniques incur different efficiency.

For practical applicability of cloud storage with protection for users against potentially malicious clouds, one desires protocols without heavy cryptographic operations yet still efficient and able to support data dynamics and third-party auditing. In this paper, we provide a provably secure cloud storage protocol with conceptual and technical novelty. At the same time, the protocol does not need any heavy cryptographic operation, yet supports data dynamics under a malicious cloud.

Our protocol is inspired by a classic protocol solving the string equality checking problem in communication complexity of distributed computing [8], [9]. Our protocol runs basically as follows: 1) we model the data as a vector in some vector space; 2) when the user checks the availability and integrity of the outsourced data, the user asks the cloud to compute an inner product of the data with some challenge vector in a "*verifiable*" way; 3) the verifiability of the inner product ensures the integrity of the cloud storage.

Our protocol enjoys several merits. First, it is very *efficient* because it involves only integer additions and multiplications, which enables a fast implementation. Besides, our protocol is conceptually and technically very *simple* and does not rely on heavy cryptograhic operations. The proposed protocol only uses inner product of the input string and some randomly chosen strings together with pseudorandom functions. Furthermore, our protocol is *provably secure* under a formal definition with real-world motivations of security.

We further propose a novel method for supporting data dynamics. In the protocol design, we associate with each data block a unique sequence number to ensure verifiability of outsourced data in the cloud. Supporting data insertion and deletion is pretty challenging when modeling the cloud as malicious. To solve the data dynamics problem, we employ two ideas: the first is to increase the sequence number all the time whenever new data is inserted; the second is to re-normalize the sequence number periodically such that no

The corresponding author is Tao Xiang (txiang@cqu.edu.cn).

sequence number gap exists when data blocks are deleted. The re-normalization operation is achieved by sending carefully designed update messages. This method for supporting data dynamics is novel and efficient.

Next, we step further and find a generic framework for designing secure cloud storage protocols in a systematic way based on verifiable computation protocols [10]. This can be used to automatically transform previous protocols to new ones with desirable security properties.

To summarize, we make the following contributions:

1) We propose a novel and efficient secure cloud storage protocol which enjoys both conceptual and technical simplicity. We connect the secure cloud storage problem to the canonical distributed string equality checking problem.
2) We introduce a new technique to support data dynamics, including adding, deleting and modifying data blocks, which only requires constant-size user-side cache. The communication cost incurred can be amortized for frequent data updates.
3) We formally prove the security of the our protocol under a practical security definition. More importantly, we find a systematic framework to construct secure cloud storage protocols based on verifiable computation protocols. Theoretical and experimental analyses validate the efficacy of our protocol.

The remaining of the paper proceeds as follows. Section II first models the secure cloud storage problem, and then proposes a solution framework and a formal security definition. Section III details the design of the secure cloud storage protocol, which is followed by the security and performance analysis in Section IV. Section V shows how to extend the protocol to support data dynamics. Section VI shows a systematic way to design secure cloud storage protocols based on verifiable computation protocols. Section VII presents the experimental evaluation of the proposed protocol. Section VIII reviews related work. Finally, Section IX concludes the paper.

II. Problem Formulation

A. System Model & Threat Model

Abstracting from the real-world usage of a cloud, we model the secure cloud storage problem as in Fig. 1. Two entities are involved: user and cloud. A secure cloud storage system runs as follows: The user outsources the data to the cloud. Later, to verify whether the outsourced data remains intact on the cloud, the user may send audit queries any time to the cloud to verify the integrity of the outsourced data. On each audit query, the cloud is supposed to "prove" to the user compactly that the data is well stored in the cloud. Then, the user verifies whether the cloud's "proof" is indeed correct to know whether the data is damaged.

Similar to previous work [1]–[3], [6], [7], we model the cloud as malicious due to internal/external attacks, aiming at stronger security and wider applicability. A malicious cloud can deviate the protocol in arbitrary ways. We focus

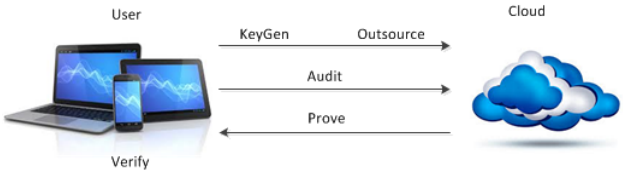

Fig. 1. System Model for Verifiable Cloud Storage.

on integrity of the outsourced data. For confidentiality and authentication, we assume they are well protected using other mechanisms, e.g., secure encryption algorithms and MACs. Our main goal is to devise novel, efficient, elementary, and provably secure protocols to ensure the integrity of the outsourced data.

B. Solution Framework & Security Definition

We employ the same framework and security definition for a basic secure cloud storage protocol as previous work [1]–[7].

Let an integer λ denote the security level and D denote the data to be outsourced. A secure cloud storage protocol, denoted as SCS, comprises five algorithm components (KeyGen, Outsource, Audit, Prove, Verify). It runs as follows:

- $\mathsf{KeyGen}(1^\lambda) \mapsto K$: On input a security level λ, the user runs this algorithm to generate a secret key K.
- $\mathsf{Outsource}(D; K) \mapsto D'$: On input the data D to be outsourced and the secret key K, the user runs this algorithm to obtain the outsourced data D'.
- $\mathsf{Audit}(K) \mapsto \chi$: The user runs this algorithm to generate an audit query χ and sends it to the cloud.
- $\mathsf{Prove}(D', \chi) \mapsto \Gamma$: On receiving an audit query χ, the cloud runs this algorithm to output a "proof" Γ, which was then sent to the user, with the aim to convince the user that the data is well stored in the cloud.
- $\mathsf{Verify}(\chi, \Gamma; K) \mapsto \{0, 1\}$: On receiving a proof Γ from the cloud, the user runs this algorithm to check whether the returned proof is correct. The algorithm is supposed to output 1 if the data remains intact, and to output 0 otherwise.

We now formally define the security for a SCS protocol. We say a pair (χ^*, Γ^*) is a forgery if Γ^* is not computed using D' for the query χ^*. A forgery models the malicious cloud's behavior. Let $\mathcal{A}$ denote a malicious cloud and $\Pr[\mathsf{Cheat}]$ denote the probability of a malicious cloud that successfully finds a forgery after sufficient learning of the protocol. Then, $\Pr[\mathsf{Cheat}]$ can be computed as

$$\Pr\left[\begin{array}{c}\mathsf{KeyGen}(1^\lambda) \to K \\ \mathsf{Outsource}(D; K) \to D' \\ \mathsf{Audit}(K) \to \chi_i \\ \mathsf{Prove}(D', \chi_i) \to \Gamma_i \\ \mathsf{Verify}(\chi_i, \Gamma_i; K) \to \delta_i \\ i = 1, \ldots, \mathrm{poly}(\lambda)\end{array} : \begin{array}{c}\mathcal{A}(D', \chi_i, \Gamma_i, \delta_i) \\ \text{outputs a} \\ \text{forgery } (\chi^*, \Gamma^*)\end{array}\right] \quad (1)$$

where $\mathrm{poly}(\lambda)$ is an arbitrary but fixed polynomial in the security level λ. The left part before the colon ":" of Eq. (1) denotes the capability of the cloud and the right part captures a malicious cloud's malicious behavior. Let $\mathcal{B}$ be a user that

wants to recover the original data from the interaction with the malicious cloud. Let $\Pr[\mathsf{Recover}]$ denote the probability of a successful recovery of the data. Then, $\Pr[\mathsf{Recover}]$ can be computed similarly

$$\Pr\left[\begin{array}{c}\mathsf{KeyGen}(1^\lambda)\to K\\ \mathsf{Outsource}(D;K)\to D'\\ \mathsf{Audit}(K)\to\chi_i\\ \mathsf{Prove}(D',\chi_i)\to\Gamma_i\\ \mathsf{Verify}(\chi_i,\Gamma_i;K)\to\delta_i\\ i=1,\ldots,\mathrm{poly}(\lambda)\end{array} : \begin{array}{c}\mathcal{B}(K,\chi_i,\Gamma_i,\delta_i)\\ \text{outputs } F^*\\ \text{and } F^*=F\end{array}\right] \quad (2)$$

where the left part before ":" denotes the interaction between the user and the malicious cloud and the right part models a successful data recovery. We introduce a useful concept before presenting the formal security definition. A function $f(\lambda)$ is negligible in λ if $f(\lambda) < \frac{1}{\mathrm{poly}(\lambda)}$ for every polynomial in λ as λ tends to positive infinity, denoted as $f(\lambda)=\mathrm{negl}(\lambda)$.

Definition 1. *A secure cloud storage protocol SCS is secure if* $\Pr[\mathsf{Cheat}]=\mathrm{negl}(\lambda)$ *and* $\Pr[\mathsf{Recover}]=1-\mathrm{negl}(\lambda)$.

C. Preliminary

In the design of our secure cloud storage protocol, we use a tool *pseudorandom functions* which is the only cryptographic tool employed in this paper. Let $\mathsf{PRF}=\{F_K(\cdot)\}$ be a set of deterministic functions indexed by K, mapping some input domain X to output image Y. Intuitively, a deterministic function $\mathsf{PRF}=\{F_K(\cdot)\}$ is said to be pseudorandom if any polynomial time algorithm cannot distinguish it from a truly random function. Please refer to [11] for more details on security definition of pseudorandom functions. In practice, a secure encryption algorithm or a keyed hash function can be used as a pseudorandom function.

We also fix some notations for use later. Let p denote a prime number and $\mathbb{Z}_p$ denote the set $\{0,1,\ldots,p-1\}$ and $\mathbb{Z}_p^*$ the set $\{1,\ldots,p-1\}$. For any $g\in\mathbb{Z}_p^*$, it holds that $g^{p-1}=1 \bmod p$, which is a basic fact in number theory. Let D denote the data to be outsourced and $F_K(\cdot)$ denote a pseudorandom function mapping integers to $\mathbb{Z}_p$.

III. Distributed String Equality Checking Based Protocol

In this section, we detail the design of an efficient, simple, and provable secure cloud storage protocol by leveraging a traditional distributed string equality checking protocol. By efficiency, we seek that the user's and cloud's computation and communication cost should be as small as possible. By simplicity, we aim that all the operations in the protocol should be elementary and no bilinear parings or digital signatures are needed. Most importantly, we expect a protocol with confident security guarantee. We first present the basic idea of the proposed protocol; then we show the details of the protocol.

A. The Basic Idea

The basic idea of our protocol design is to model the secure cloud storage problem as a variant of the distributed string equality checking problem, and then transform a classical solution for the latter to a solution for secure cloud storage.

1) Distributed String Equality Checking Primitive: A distributed string equality protocol can enable two parties possessing two strings (e.g., messages) in a network to find whether their strings are equal [8] with minimum communication cost. The problem setup is as follows. Alice and Bob are two entities in a distributed network. Alice has an n-bit string $x\in\{0,1\}^n$ and Bob also has such a string $y\in\{0,1\}^n$. To complete a distributed computation task cooperatively, Alice and Bob want to know whether their two stings are equal by communicating to each other. The question is that how many communication bits are at least needed to finish the task.

We now present a classical protocol for enabling distributed string equality checking. This protocol is very simple since it only involves inner product computations. Suppose there is a public random string pool $Z\subseteq\{0,1\}^n$ with size $O(n)$. A classical protocol with $O(\log n)$ communication bits works as follows [9]:

- Alice chooses a random string $z\in Z$ in the pool and sends the index to Bob. Alice also sends the inner produce $\langle x,z\rangle \bmod 2$.
- Bob also computes $\langle y,z\rangle \bmod 2$ and checks whether Bob's result is equal to Alice's result $\langle x,z\rangle \bmod 2$. If not, Bob tells Alice that the two strings are not equal and the protocol aborts. Otherwise, the protocol continues.
- Alice and Bob repeat this process 100 times. If $\langle x,z\rangle=\langle y,z\rangle$ all the time, Bob reports that the two strings are equal and the protocol stops.

The basic rationale of the above protocol is as follows: If $x=y$, then $\langle x,z\rangle=\langle y,z\rangle$ always holds and the protocol is correct. If $x\neq y$, in a single run we have $\Pr_z[\langle x,z\rangle=\langle y,z\rangle]=\frac{1}{2}$. After running 100 times, we can find that x is not equal to y if $\langle x,z\rangle\neq\langle y,z\rangle$ for some z. We can only make a mistake when $\langle x,z\rangle=\langle y,z\rangle$ for all 100 z's. However, this bad even only occurs with probability $\frac{1}{2^{100}}$, which is very small.

2) Adaptation: In this subsection, we construct a secure cloud storage protocol by adapting the above distributed string equality checking protocol. Our idea is as follows: we take the user as Alice possessing x and the cloud as Bob possessing y; then cloud storage checking can be modeled as and solved by distributed string equality checking. The cloud can compute $\langle y,z\rangle$ straightly. Nonetheless, one challenge remains: the user does not know how to compute $\langle x,z\rangle$ since the user does not possess x anymore which is outsourced to the cloud.

Before solving the remaining challenge, we first make two improvements in order to achieve good efficiency: 1) instead of working over $\{0,1\}^n$, we model the data $D=(d_1,\ldots,d_n)$ to be outsourced as a vector in $\mathbb{Z}_p^n$ where $d_i\in\mathbb{Z}_p$ is a data block and n is the total number of data blocks, which can decrease false positive probability significantly and promote efficiency; 2) instead of using a public random string pool, we send a random string directly to the cloud which can be generated by a pseudorandom function to save communication cost.

We now solve the remaining challenge by leveraging the cloud to compute $\langle x,z\rangle$ for the user in a verifiable way.

It works as follows: when outsourcing $D = (d_1, \ldots, d_n)$, the user also outsources an integer $s_i = g^{r \cdot d_i + F_K(i)} \bmod p$ using some secret information for each data block d_i, where $g, r \in \mathbb{Z}_p^*$ are randomly chosen and $F_K(i)$ is the output of a pseudorandom function $F_K(\cdot)$ on input i. The values r and $F_K(i)$ are kept secret from the cloud. Let e_i be the random sequence sent to the cloud for computing the inner product. The cloud computes

$$\alpha = \sum_i d_i \cdot e_i \bmod (p-1) \tag{3}$$

and

$$\beta = \prod_i s_i^{e_i} = g^{r \cdot (\sum_i d_i e_i) + \sum_i e_i F_K(i)} \bmod p. \tag{4}$$

Equivalently, α contains the inner product of the cloud side, which is similar to $\langle y, z \rangle$ for distributed string equality; and β contains the information on the inner production of the user side, which is similar to the functionality of $\langle x, z \rangle$ for distributed string equality. We can check whether the inner products are equal by checking whether $\beta = g^{r\alpha + \sum_i e_i F_K(i)}$ holds, where $r, e_i, F_K(i)$ are known by the user.

We present some intuition on why our adaptation is correct and secure. Please refer to Section IV for more strict argument. Suppose a data block d_{j^*} for some j^* is damaged. When the user audits this block, the cloud needs to return the inner produces α and β. When the user checks $\beta = g^{r\alpha + \sum_i e_i F_K(i)}$, the secret information $F_K(j^*)$ is only known to the user. If the cloud wants to cheat the user that the data is not damaged, the cloud needs to know the secret value of $F_K(j^*)$, for which we employ a pseudorandom function here to keep it private.

B. Detailed Protocol Design

In this section, we combine the basic ideas above to design a secure cloud storage protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) in detail by showing all the design choices for each algorithm in SCS.

KEY GENERATION. In the key generation algorithm, we choose parameters for the protocol according to a security level; we also generate secret keys. The parameters include p, g, r and $F_K(\cdot)$. The prime p is public and can be generated randomly. However, the bit length of p should be at least the same as the security demand, the reason of which can be found in the security analysis in Section IV. For example, if the security level is 128 bit, then p should be a prime number whose bit length is greater or equal to 128. After p is chosen, the numbers g and r can be chosen as any random number in $\mathbb{Z}_p^*$. The pseudorandom function $F_{K_1}(\cdot)$ can be any secure one in the pseudorandom function family $\{F_{K_1}(\cdot)\}$ indexed by a secret key K_1 [11]. Thus, the public key is (p, g); and the private key is (r, K_1).

OUTSOURCE. To outsource the data D to the cloud, we first divide it into blocks $(d_1, \ldots, d_n)$ where $d_i \in \mathbb{Z}_p$ and n is the total number of blocks. In order to enable future audit queries that can check the integrity of the outsourced data, we embed some secret information in the data using the secret key (r, K_1). For each d_i, we compute $s_i = g^{r \cdot d_i + F_{K_1}(i)} \bmod p$ and outsource (d_i, s_i) to the cloud, where $F_{K_1}(i)$ is the output by the pseudorandom function on input i. Implicitly, we contain an index information in the outsourced data. This index information is very important for the security of the secure cloud storage protocol since it can prevent a malicious cloud from cheating the user.

AUDIT. Our protocol supports two types of auditing mechanisms: deterministic and randomized. For deterministic auditing, the user can send a challenge sequence $(e_1, \ldots, e_n)$ to ask the cloud for computing the inner product as in Eq. (3) and its proof as in Eq. (4). The challenge sequence can be generated using a pseudorandom function with a new secret key K_2. The user only needs to send a constant-length string K_2 to the cloud and the challenge sequence can be obtained as $e_i = F_{K_2}(i)$.

For randomized auditing, the user can send a much shorter challenge sequence $(e_{i_1}, \ldots, e_{i_L})$ and their indices $(i_1, i_2, \ldots, i_L)$, where $i_1, i_2, \ldots, i_L \in \{1, 2, \ldots, n\}$ are randomly chosen and L is far smaller than n. In the randomized setting, the cloud only retrieves small parts of the data, which may not touch the damaged data blocks. In order to achieve a good accuracy, the user may repeatedly send multiple independent queries.

PROVE. After the cloud receives the challenge sequence, the cloud simply computes the inner product $\alpha = \sum_i d_i \cdot e_i \bmod p - 1$ and $\beta = \prod_i s_i^{e_i}$ which equals $g^{r \cdot (\sum_i d_i e_i) + \sum_i e_i F_K(i)} \bmod p$. Then (α, β) is sent back as a proof.

VERIFY. After the cloud returns a proof (α, β), the user simply checks whether $\beta = g^{r\alpha + \sum_i e_i F_{K_1}(i)}$. It is easy to find that $g^{r\alpha + \sum_i e_i F_{K_1}(i)}$ can be computed efficiently by first computing $t = r\alpha + \sum_i e_i F_K(i) \bmod p - 1$ and then computing the exponent $g^t \bmod p$. All operations are very efficient. Especially, p could be as small as a 128-bit prime. The user accepts the proof if all returned (α, β)'s are correct for both randomized and deterministic audits, i.e., the outsourced data remains intact and available. Otherwise, the outsourced data is damaged and then the user needs to take further actions which is out of the scope of the current paper and thus is a future research direction.

C. Protocol Specification

We summarize our protocol in a compact form. The proposed secure cloud storage protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) based on distributed string equality checking runs as follows:

- $\mathsf{KeyGen}(1^\lambda) \to K$: On input a security level λ, the user generates a prime p with bit length at least λ. The user also generates two integers $g, r \in \mathbb{Z}_p^*$, and a λ-bit key K_1 for the pseudorandom function $F_{K_1}(\cdot)$ with input domain integers and output image $\mathbb{Z}_p$. We require that g is a generator of the group $\mathbb{Z}_p^*$. The secret key is $K = (r, K_1)$ and the public key is (p, g).
- $\mathsf{Outsource}(D; K) \to D'$: On input the data D to be outsourced, the user divides D into blocks $(d_1, \ldots, d_n)$ where $d_i \in \mathbb{Z}_p$ and n is the total number of blocks. For

each i, the user also computes $s_i = g^{r \cdot d_i + F_{K_1}(i)} \bmod p$. The user sends all (d_i, s_i)'s to the cloud.

- $\mathsf{Audit}(K) \rightarrow \chi$: The user can audit the cloud's storage either in a deterministic way or a randomized way.
  - For deterministic auditing, the user generates a new secret λ-bit key K_2 for the pseudorandom function $F_{K_2}(\cdot)$ and then the user sends K_2 to the cloud. The audit query is $\chi = K_2$.
  - For randomized auditing, the user generates L indices $(i_1, i_2, \ldots, i_L)$ in $\{1, 2, \ldots, n\}$ and a vector $(e_{i_1}, \ldots, e_{i_L})$ over $\mathbb{Z}_p^L$ to the cloud. The audit query is $\chi = [(i_1, i_2, \ldots, i_L), (e_{i_1}, \ldots, e_{i_L})]$.
- $\mathsf{Prove}(D', \chi) \rightarrow \Gamma$: Depending on whether it is a deterministic audit or a randomized one, there are two cases.
  - For deterministic auditing, the cloud uses $\chi = K_2$ to generate a vector $(e_1, \ldots, e_n)$ over $\mathbb{Z}_p^n$. The cloud computes $\alpha = \sum_{i=1}^{n} d_i \cdot e_i \bmod (p-1)$ and $\beta = \prod_i s_i^{e_i} \bmod p$. The cloud sends $\Gamma = (\alpha, \beta)$ back to the user as a proof.
  - For randomized auditing, the cloud computes $\alpha = \sum_{j=1}^{L} d_{i_j} \cdot e_{i_j} \bmod p - 1$ and $\beta = \prod_{j=1}^{L} s_{i_j}^{e_{i_j}} \bmod p$ using the audit query $\chi = [(i_1, i_2, \ldots, i_L), (e_{i_1}, \ldots, e_{i_L})]$. The cloud also sends $\Gamma = (\alpha, \beta)$ back as a proof.
- $\mathsf{Verify}(\chi, \Gamma; K) \rightarrow \{0.1\}$: On receiving a proof $\Gamma = (\alpha, \beta)$, the user checks whether $\beta = g^{r\alpha + \sum_i e_i F_{K_1}(i)}$ is true using the audit query χ and the secret key K. If yes, the user accepts the proof; otherwise, rejects it.

IV. Correctness, Security and Performance Analysis

A. Correctness

Our protocol is correct if both parities follow the protocol. The user checks whether $\beta = g^{r\alpha + \sum_i e_i F_{K_1}(i)}$ holds on a proof $(\alpha = \sum_i d_i \cdot e_i \bmod p - 1, \beta = \prod_i s_i^{e_i})$ from the cloud. Note that $\beta = \prod_i s_i^{e_i} = \prod_i (g^{r \cdot d_i + F_K(i)})^{e_i}$, which is exactly $g^{r\alpha + \sum_i e_i F_{K_1}(i)}$. Thus, an honest cloud with intact data can always proves to the user that the data remains undamaged.

B. Security

We now show a cloud cannot cheat the user on any data damage. We analyze security in two steps, focusing on two probabilities according to Definition 1, respectively. We first argue $\Pr[\mathsf{Cheat}]$ is negligible by showing a cloud with damaged data cannot compute the inner product correctly. The proof idea is similar to the the Cramer-Shoup public-key encryption algorithm [12]. We then argue $\Pr[\mathsf{Recover}]$ is close to 1 by proposing a data reconstruction algorithm.

Theorem 2. *The protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) specified in Section III-C is secure if the pseudorandom function is secure.*

Proof Sketch. We show the basic idea of our security argument. The detailed proof will be shown in an extended paper in future, due to page limits.

TABLE I
Theoretical Performance of the Proposed Protocol

	Computation	Communication	Storage
User	$O(l)$	$O(l)$	$O(1)$
Cloud	$O(l)$	$O(1)$	$O(n)$

Step I. We show a cloud with damaged data cannot cheat with a non-negligible probability. First, suppose we use a truly random function in our protocol and let $\Pr[\mathsf{Unbound}]$ be the probability of a successful cheating by a cloud. Then we can show $\Pr[\mathsf{Unbound}] = \mathrm{negl}(\lambda)$ using a linear algebra argument. Second, we return to the original protocol using a pseudorandom function. The only difference lies in the pseudorandom function. Thus, we have $\mathrm{Adv}[\mathsf{PRF}] \geq |\Pr[\mathsf{Cheat}] - \Pr[\mathsf{Unbound}]|$ because the malicious cloud can be employed to distinguish a pseudorandom function from a truly random function. Therefore, $\Pr[\mathsf{Cheat}] \leq \mathrm{Adv}[\mathsf{PRF}] + \mathrm{negl}(\lambda)$.

Step II. We show if $\Pr[\mathsf{Cheat}]$ is non-negligible, the user can reconstruct the data with probability $\Pr[\mathsf{Recover}] = 1 - \mathrm{negl}(\lambda)$ by proposing a reconstructing algorithm. The basic idea is that the user can obtain a linear system of equations of the inner products of the outsourced data and the challenge vectors, and then solve it to recover the outsourced data. We remark that such similar treatment has also been used in earlier proofs of storage works, e.g. [4], [13].

□

C. Performance

Tables I and II list the theoretical performance of our protocol and comparisons with previous work. Let n denote the total number of blocks of the outsourced data, λ denote the security parameter, and l denote the length of the audit query.

Theoretical Performance. For the user, four algorithms are involved, i.e. SCS.KeyGen, SCS.Outsource, SCS.Audit, and SCS.Verify. The key generation algorithm and the data outsourcing algorithm are only executed once when the data is outsourced. Thus, all computation in them can be amortized in the following many audit queries. When the user audits the cloud for data integrity and availability, generating the audit query takes $O(l)$ computation and $O(l)$ communication as well. When the user verifies the proof from the cloud, it also takes $O(l)$ computation to compute $\sum_i e_i F_{K_1}(i)$ and check $\beta = g^{r\alpha + \sum_i e_i F_{K_1}(i)}$. Thus, the total computation cost is $O(l)$. For communication, the user sends an audit challenge to the cloud. Since the audit challenge has length $O(l)$, the communication cost is then also $O(l)$. For storage, the user only needs to store the secret key $K = (r, K_1)$. The random r is just a random element in $\mathbb{Z}_p$ and the key K_1 is only a constant-length bit string. Thus, the storage cost is $O(1)$. The same analysis also applies to the cloud.

Performance Comparison. We compare the performance of the our protocol with recent protocols [1], [2], [4]–[7], [14] in Table II to show their strength and their advantages. For computation, storage, and communication cost, we account

TABLE II
COMPARISON OF DIFFERENT PROTOCOLS

Protocols	Computation	Storage	Communication	Heavy Crypto	Data Dynamics	Security Model
Juels and Kaliski [1]	$O(\lambda l)$ / $O(\lambda l)$	$O(\lambda)$ / $O(n)$	$O(l)$ / $O(\lambda l)$	NO	NO	Standard
Ateniese et al. [2]	$O(\lambda l)$ / $O(\lambda l)$	$O(\lambda)$ / $O(n)$	$O(l)$ / $O(\lambda l)$	YES	NO	ROM
Shacham and Waters [14]	$O(\lambda l)$ / $O(\lambda l)$	$O(\lambda)$ / $O(n)$	$O(l)$ / $O(\lambda l)$	YES	NO	ROM
Yang and Jia [5]	$O(\lambda l)$ / $O(\lambda l)$	$O(\lambda), O(n)$ / $O(n)$	$O(l)$ / $O(\lambda l)$	YES	YES	ROM
Wang et al. [4]	$O(\lambda l)$ / $O(\lambda l)$	$O(\lambda)$ / $O(n)$	$O(l)$ / $O(\lambda l)$	YES	YES	ROM
Shi et al. [6]	$O(\lambda l), O(n)$ / $O(\lambda l), O(n)$	$O(\lambda)$ / $O(n)$	$O(l), O(n)$ / $O(\lambda l), O(n)$	YES	YES	-
Chen et al. [7]	$O(\lambda l)$ / $O(\lambda l)$	$O(\lambda)$ / $O(n)$	$O(l)$ / $O(\lambda l)$	YES	NO	Standard
This Work	$O(\lambda l), O(n)$ / $O(\lambda l)$	$O(\lambda + 1)$ / $O(n)$	$O(l), O(n)$ / $O(\lambda l)$	NO	YES	Standard

for both the user and the cloud by dividing the cost in two parts, e.g., $O(l)$ / $O(\lambda l)$ in the second row of Table II where $O(l)$ is for the user and $O(\lambda l)$ is for the cloud. When data dynamics is supported, the cost may be different between auditing and supporting data dynamics, e.g., $O(\lambda l), O(n)$ in the seventh row of Table II where $O(\lambda l)$ is for auditing and $O(n)$ is for supporting data dynamics. We find that different protocols are designed with different theoretical foundations; thus different protocols may fit different application scenarios depending on the requirement of the user and the cloud. Most of the protocols employ heavy cryptography [2], [4]–[7], [14], e.g., RSA signatures, elliptic curve cryptography, etc. In contrast, our work only employs light-weight cryptographic operations. The asymptotic performance is roughly the same as the protocols in terms of $O(\lambda), O(l)$ and $O(n)$. It is worth noting that the big-O notation hides different constants depending on the block size of different protocols. Our protocol has a small constant in the big-O notation for the computation cost because no heavy crypto is employed; thus our protocol is also very efficient. Our protocol can employ a modulus p with 128 bits, which is much smaller and thus more efficient than RSA and other signatures. The works [4]–[6] and this work all support data dynamics; however, they differ in assumptions and efficiency when supporting data dynamics, which is discussed in detail in Section V. Some protocols also base on their security on the random oracle model (ROM) while ROM is not needed for other protocols, including the protocol in this paper. The main advantage of our protocol in this work includes: 1) its simplicity both in techniques and concepts and its functionalities; 2) its efficiency due to simplicity; 3) supporting simple yet efficient data dynamics; and 4) no ROM assumption is needed. Finally, we stress that different protocols may fit different applications.

V. SUPPORTING DATA DYNAMICS

We show our technique to support data dynamics in this section. Data dynamics include three different types of operations: deleting a data block, inserting a data block and modifying a data block. The difficulty when handling data dynamics lies in the fact that the secret information accompanying each data block is correlated with the index of the block. If we modify a data block while not changing the index, the cloud can replay previous data blocks to avoid detecting of any data loss. Adding a new data block and deleting a data block both requires to tackle the embedded secret information and the indices. To solve this challenge, we keep assigning new indices whenever a data block is changed, either inserted, deleted or modified and we keep a local cache recording the changes. This idea is similar to the approach in [5], but requires a smaller cache since we only cache the necessary updated information but not all information on all the data blocks. However, the challenge is not solved completely. When there are considerable data dynamic operations, the local cache in the user side may increase to a very large size. In order to solve this cache size issue, we propose an algorithm to dynamically empty the cache, which is achieved by re-normalizing all the data blocks and sending new secret embedded information with consecutive indices.

Therefore, we support data dynamics as follows:

- The user keeps a state recording the total number of data blocks n, the maximal index max_index, the current index $current\_index$, the deleted indices del_index_array. Initially, it holds that $max\_index = current\_index = n$ and del_index_array is empty.
- When a data block is deleted, the user adds the corresponding index to del_index_array and decreases the total number of blocks n by 1.
- When a data block is inserted, the user assigns its index as $current\_index + 1$ and embeds the secret information as $s_{current\_index+1} = g^{r \cdot d_i + F_{K_1}(current\_index+1)}$. The user also increases $current\_index, max\_index$ and the total number of blocks n by 1.
- When a data block is modified, the user first deletes this data block and insert the modified value as a new data block using the methods described above.
- When the local state of the cache achieves to a threshold, the user resets all the secret information through an algorithm as discussed later such that the data blocks have consecutive indices from 1 to n again. The threshold for user's local cache can be set according to the data access patterns of a particular application scenario. After resetting, we have $max\_index = current\_index = n$ and empty del_index_array again. This procedure is executed periodically whenever the cache exceeds the threshold.

User auditing still works after data dynamics happens. When the user sends an audit query to the cloud, the challenged data blocks are with indices in $[1, max\_index]$. Since we know those indices that are invalid from del_index_array, the proposed protocol can still work.

The idea for periodic emptying of the cache is that we can

embed new secret information in the outsourced data using another independent secret key for a pseudorandom function. The user knows both secret keys for the pseudorandom functions; thus, the user can send update messages to modify the embedded secret information. Details are as follows: Suppose when the cache is full, the cloud has n data blocks and they are $(d_j, s_j = g^{r \cdot d_j + F_{K_1}(i_j)} \bmod p)$ where $i_j \in [1, max\_index]$ and $j = 1, \ldots, n$. We have max_index is much larger than n and the indices i_j's could not be consecutive. However, the user indeed knows the detailed indices for all data blocks since the user keeps all the state information locally. In order to reset all the indices for the data in the cloud from 1 to n, the user can choose a new secret key K_3 for the pseudorandom function and then change each secret information from $s_j = g^{r \cdot d_j + F_{K_1}(i_j)} \bmod p$ to $s_j = g^{r \cdot d_j + F_{K_3}(j)} \bmod p$ for $j = 1, \ldots, n$. This is achieved by sending a reset message $g^{F_{K_3}(j) - F_{K_1}(i_j)}$ for $j = 1, \ldots, n$ to cloud. The cloud can obtain the new embedded secret information using $s_j = g^{r \cdot d_j + F_{K_1}(i_j)}$ and $g^{F_{K_3}(j) - F_{K_1}(i_j)}$ by multiplying them together. The security proof still holds in this case since the cloud doesn't get new information due to the pseudorandomness of $g^{F_{K_3}(j) - F_{K_1}(i_j)}$. In this way, the cache is emptied and the state $current\_index, max\_index, del\_index\_array$ can return to the initial values.

Our technique advances the design of secure cloud storage protocols that can support data dynamics for *untrusted* clouds. To the best knowledge of the authors, three techniques supporting data dynamics have been employed in the community for secure cloud storage [4]–[6]. However, a challenge still remains, i.e., how to support data dynamics in a scalable and efficient way in front of a malicious cloud. The technique in [4] avoids the use of an index to identify a data block when supporting data dynamics assuming a *semi-honest* cloud. The technique in [5] requires the user to store a cache with size $\Omega(n)$ where n is the size of the total number of data blocks. The technique in [6] instead requires the cloud to store a cache with size $\Omega(n)$; the cache is later emptied periodically. Further, [6] requires the user to download authentication information of the updated data blocks and verify the correctness of the data updates by the cloud. We solve the scalable data dynamics challenge partially using a *constant* size cache in the user side and a special technique. Compared with [6], the computation cost of data updates in our approach is much lower both for the user and the cloud by trading off some communication cost.

VI. Secure Cloud Storage from Verifiable Computation

In this section, we propose a detailed framework to design secure cloud storage protocols in a *systematic* way based on any verifiable computation protocol, which is inspired by reconciling our protocol proposed in this paper. We find that the core of our protocol is the ability to compute inner products verifiably, which turns to be a special case of a much broader area, i.e., *verifiable computation*. The areas of the secure cloud storage [1], [2], [4], [5] and verifiable computation [10], [15] are currently being developed independently. We hope our work here can make both areas benefit from each other. One advantage of our systematic design is we automatically have many secure cloud storage protocols based on previous work on verifiable computation. In contrast, the previous ad-hoc approaches designing secure cloud storage protocols [1], [2], [4], [5] are more time-consuming and require good intuitions. Another advantage of the systematic design is that we can have a better and more in-depth understanding of the secure cloud storage problem. Furthermore, many previous secure cloud storage protocols can be modeled using the proposed framework.

A verifiable computation protocol VC = (KeyGen, ProbGen, Compute, Verify) contains a computationally weak user and a powerful worker and runs as follows:

- $\mathsf{KeyGen}(1^\lambda, f) \to (PK, SK)$: On input a security level λ and the function f to be computed, the user generates a public key PK and a secret key SK. The public key, which is given to the worker, contains the information about the function to be outsourced and the secret key is only known by the user.
- $\mathsf{ProbGen}(x; SK) \to x'$: On input a point x on which the user wants to evaluate f, the user generates an input x' using the secret key SK.
- $\mathsf{Compute}(PK, x') \to \Gamma$: On input the processed input x', the worker computes f on x' and outputs a value y' and a proof σ_y. The value y' contains the information about $f(x)$ which is wanted by the user. The worker returns $\Gamma = (y', \sigma_y)$ to the user.
- $\mathsf{Verify}(x, \Gamma; SK) \to \{y, \perp\}$: On receiving the computation result $\Gamma = (y', \sigma_y)$, the user finds the value $y = f(x)$ through y'. Then, the user verifies whether this value y is correct using x, y, σ_y, SK. If it is indeed correct, the user outputs y; else outputs $\perp$ denoting that the answer is not accepted.

We now show how to design a secure cloud storage protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) from any verifiable computation protocol VC = (KeyGen, ProbGen, Compute, Verify). First, the user determines a function using the data blocks to be outsourced. Later, the user asks the cloud to compute some output of the determined function in a verifiable way using a verifiable computation protocol. The function is chosen in a way that the data blocks can be reconstructed from the many outputs of the function on many inputs. For example, the function could be an inner product function as our protocol in Section III. The detailed design, in parallel to our proposed protocol, is thus as follows:

- $\mathsf{KeyGen}(1^\lambda, D) \to K$: On input a security level λ, the user divides the data to be outsourced into blocks $(d_1, \ldots, d_n)$ and determines a function $f_{d_1, \ldots, d_n}(\cdot)$ on these blocks. The user invokes $\mathsf{VC.KeyGen}(1^\lambda, f_{d_1, \ldots, d_n})$ to generate (PK, SK). The user keeps SK as the secret key and $K = SK$.
- $\mathsf{Outsource}(D; K) \to D'$: On input the data D to be outsourced, the user gives $D' = PK$ to the cloud. The

TABLE III
WIKIPEDIA DATA SET AND STORAGE COST

Benchmark	File Size	Randomized	Deterministic
#1	2.27KB	3.35KB	3.35KB
#2	1.45MB	2.0MB	2.0MB
#3	23.5MB	32.7MB	32.7MB
#4	121MB	162.7MB	162.8MB

value PK is generated in the key generation algorithm and it contains the information about all data blocks as in $f_{d_1,\ldots,d_n}$.

- $\mathsf{Audit}(K) \rightarrow \chi$: To audit the availability and integrity of the outsourced data, the user first generates an input x in the domain of $f_{d_1,\ldots,d_n}$, then invokes $\mathsf{VC.ProbGen}(x; SK)$ to generate a processed value x'. The user sends $\chi = x'$ as the audit query. The user also keeps x locally as a part of the secret key K. The value of x determines whether the auditing is deterministic or randomized.
- $\mathsf{Prove}(D', \chi) \rightarrow \Gamma$: On input an audit query, the cloud invokes $\mathsf{VC.Compute}(PK, x')$ to compute $\Gamma = (y', \sigma_y)$ for the value $f_{d_1,\ldots,d_n}(\cdot)$ on input x', where $D' = PK$ and $\chi = x'$. The cloud returns Γ.
- $\mathsf{Verify}(\chi, \Gamma; K) \rightarrow \{0.1\}$: On receiving a proof $\Gamma = (y', \sigma_y)$, the user invokes $\mathsf{VC.Verify}(x, \Gamma; SK)$ to check whether the returned result is correct. If yes, the user accepts the proof and the outsourced data remains intact; else reject the returned result and report the damage of the outsourced data.

VII. EXPERIMENTAL EVALUATION

Methodology. We prototype the proposed both the deterministic and randomized protocols in Section III to evaluate its performance. The user side and the cloud side are both implemented using Java 1.7. We focus on the storage, computation and communication cost of the proposed protocol for performance evaluation. For each performance indicator, we carry out the experiment for 20 times and then average the results to get a stable performance result. We employ a public data set [16] as in Table III for performance evaluation. We also open-source our prototype for potential follow-up works [17].

Results. We employ four test cases [16] using data '*-oldimage.sql.gz', '*-pages-articles-multistream-index.txt.bz2', '*-stub-meta-current.xml.gz', and '*-pages-meta-current.xml.bz2' respectively, where '*' denotes 'simplewiki-20130608'. The bit length of the big-integer arithmetic is set to be 1024 and the challenge length is set to be 10. The experimental results are detailed as follows.

Storage Cost. For both the randomized protocol and the deterministic protocol, the storage cost is shown as in Table III. The cost is roughly the same for both randomized and deterministic protocols. The maximal difference is 33232 bytes for test case 4, which is 0.026% of the size of the outsourced data. This is as expected since the two protocols only differ in the way how the data is audited.

TABLE IV
COMPUTATION COST FOR RANDOMIZED AUDITING IN MILLISECONDS

Benchmark	KeyGen	Outsource	Audit	Prove	Verify
#1	2108	331	0.05	170	17.6
#2	331	211679	0.12	178	18.1
#3	842	3516962	0.08	191	20.3
#4	297	17038286	5.97	168	17.0

TABLE V
COMPUTATION COST FOR DETERMINISTIC AUDITING IN MILLISECONDS

Benchmark	KeyGen	Outsource	Audit	Prove	Verify
#1	1628.6	330	0.05	1106	1484
#2	395.6	210833	0.07	202864	2300
#3	1507	3653035	1.00	3542552	11139
#4	478	17285442	0.08	16964576	56586

Communication Cost. For the randomized protocol, the communication consists of two parts: one is the indices and coefficients of the challenged data blocks; the other is the proof returned from the cloud. The former depends on the challenge length, which is 10 in the experiments; the latter is just two big integers. The total cost is roughly 2000bytes. For the deterministic protocol, it is much smaller: the size of the audit query is just a key for the pseudorandom function and the size of the proof is just two big integers. The total cost is about 400bytes in the experiments.

Computation Cost. Tables IV and V contain the detailed computation cost for the randomized protocol and the deterministic protocol. For key generation, the computation cost is very small. The maximal time is 2.1 seconds and the average cost is smaller than 1 second. For outsourcing, it takes considerable computation cost; however, this is a one-time cost and it can be amortized in the following audit and proof interactions. For auditing, the time is quite small. The computation cost for the cloud answering a query is quite different for the randomized and deterministic protocols. For the former, the time cost only depends on the length of the audit query. The maximal time cost is 0.2 seconds, which is very small. For the deterministic protocol, the time cost is very large. It is roughly the same time as that of outsourcing, which is as expected since all the data in the cloud needs to be processed as in the theoretical analysis. The same observation also holds for the verification process. The verification time is also very small for both the randomized protocol and the deterministic protocol.

It is worth noting that the most frequent operations are $\mathsf{SCS.Audit}$, $\mathsf{SCS.Prove}$ and $\mathsf{SCS.Verify}$ for data auditing. These operations cost very small time as proved in Tables IV and V. Therefore, the experimental results suggest that the proposed protocol is effective in practice.

VIII. RELATED WORK

This section reviews the most relevant work. Juels et al. [1] and Ateniese et al. [2] first proposed the secure cloud storage problem for a malicious cloud. These two initial attempts however cannot support data dynamics. Shacham et al. also

proposed two protocols based on MACs and bilinear parings [14]. Xu et al. [3] also improved these protocols by suggesting to use polynomial computations. The data dynamics is also not well solved. Wang et al. and Yang et al. [4], [5] also employed bilinear maps to audit cloud storage. Wang. et al. also reviewed several cloud storage auditing protocols in [18]. Wang et al. [19] proposed a protocol to support fast user revocation. Chen et al. [7] found a general design of secure cloud storage protocols based on secure network coding protocols. Shi et al. [6], [13], [20] showed how to support data dynamics using skip lists, B+ trees, etc. Researchers also extend single cloud storage auditing to multiple cloud auditing to further ensure data availability [21]–[23].

It is worth noting that Benabbas et al. [24] studied special verifiable polynomial computation delegation protocols. Indeed, such a protocol can be applied to cloud storage auditing. While this protocol is pretty beautiful in theory, the resulting cloud storage auditing protocol is not useful at all and is rather inefficient, mainly because only deterministic auditing is supported and too many big-integer operations are required. Besides, data dynamics is also not supported while this property is highly expected by cloud storage users, and the security foundation highly depends on the DDH assumption.

Compared with previous work, the proposed protocol in this paper supports data dynamics, and furthermore is efficient, provably secure, and especially pretty simple both conceptually and technically. More importantly, our protocol shows the connection between cloud storage auditing and distributed string equality checking; our protocol also gives insights on designing cloud storage auditing protocols systematically from any verifiable computation protocols.

IX. Conclusion

In this paper, we propose an efficient protocol for verifying the integrity and availability of outsourced data on a cloud. The proposed protocol is designed from a classic protocol checking whether two strings are equal in distributed computing. Compared with previous protocols, the proposed protocol is simple both technically and conceptually with no heavy cryptography required. The proposed protocol employs a new technique to support data dynamics, including adding, deleting and modifying data. Further, we generalize the idea of the proposed protocol. As a result, we propose a general framework to design a secure cloud storage protocol systemically from a verifiable computation protocol. We thus can have many secure cloud storage protocols automatically. Experimental results validate the effectiveness of the proposed protocol. The connections of distributed string equality checking, secure cloud storage, and verifiable computation are very interesting and may find more applications in other problems related to cloud storage. To design secure cloud storage protocols based on such connections is a very interesting future work.

Acknowledgement

The authors are thankful to the reviewers for their insightful comments. Tao Xiang's research was partially supported by the Natural Science Foundation Project of CQ CSTC (No. cstc2013jcyjA40001) and the Fundamental Research Funds for the Central Universities (No. CDJZR13185501); Yuanyuan Yang's research was partially supported by the National Natural Science Foundation of China (No. 61373178); Cong Wang's research was partially supported by Research Grants Council of Hong Kong (Project no. CityU 138513); and Shengyu Zhang's research was partially supported by Research Grants Council of the Hong Kong S.A.R. (Project no. CUHK419413).

References

[1] A. Juels and B. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proc. of ACM CCS*, 2007.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. of ACM CCS*, 2007.

[3] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proc. of ACM ASIACCS*, 2012.

[4] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.

[5] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2013.

[6] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *Proc. of ACM CCS*, 2013.

[7] F. Chen, T. Xiang, Y. Yang, and S. S. M. Chow, "Secure cloud storage meets with secure network coding," in *Proc. of IEEE INFOCOM*, 2014.

[8] A. C.-C. Yao, "Some complexity questions related to distributive computing (preliminary report)," in *Proc. of ACM STOC*, 1979.

[9] Wikipedia, "Communication complexity," http://en.wikipedia.org/wiki/Communication_complexity, 2014.

[10] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," *Proc. of CRYPTO*, pp. 465–482, 2010.

[11] O. Goldreich, "Foundation of cryptography (in two volumes: Basic tools and basic applications)," 2001.

[12] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *Proc. of CRYPTO*. Springer, 1998, pp. 13–25.

[13] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. of ACM CCS*, 2009.

[14] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. of ASIACRYPT*, 2008, pp. 90–107.

[15] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. of IEEE S&P*, 2013.

[16] Wikipedia, "Wikipedia dump service," http://dumps.wikimedia.org/simplewiki/20130608/, 2013.

[17] F. Chen, "Source code," https://sites.google.com/site/chenfeiorange/secure-cloud-storage-hits-distributed-string-equality-checking, 2014.

[18] C. Wang, K. Ren, W. Lou, and J. Li, "Toward publicly auditable secure cloud data storage services," *IEEE Network Magazine*, vol. 24, no. 4, pp. 19–24, 2010.

[19] B. Wang, B. Li, and H. Li, "Public auditing for shared data with efficient user revocation in the cloud," in *Proc. of IEEE INFOCOM*, 2013.

[20] Z. Mo, Y. Zhou, and S. Chen, "A dynamic Proof of Retrievability (PoR) scheme with $O(logn)$ complexity," in *Proc. of IEEE ICC*, 2012.

[21] T. Schwarz and E. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *Proc. of IEEE ICDCS*, 2006.

[22] K. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proc. of ACM CCS*, 2009.

[23] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Transactions On Services Computing*, vol. 5, no. 2, pp. 220–232, 2012.

[24] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," *Proc. of CRYPTO*, pp. 111–131, 2011.