Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing

Cong Wang[1], Qian Wang[1], Kui Ren[1], and Wenjing Lou[2]

[1] Illinois Institute of Technology, Chicago IL 60616, USA,
{cong,qian,kren}@ece.iit.edu
[2] Worcester Polytechnic Institute, Worcester MA 01609, USA,
{wjlou}@ece.wpi.edu

Abstract. Cloud Computing is the long dreamed vision of computing as a utility, where users can remotely store their data into the cloud so as to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources. By data outsourcing, users can be relieved from the burden of local data storage and maintenance. However, the fact that users no longer have physical possession of the possibly large size of outsourced data makes the data integrity protection in Cloud Computing a very challenging and potentially formidable task, especially for users with constrained computing resources and capabilities. Thus, enabling public auditability for cloud data storage security is of critical importance so that users can resort to an external audit party to check the integrity of outsourced data when needed. To securely introduce an effective third party auditor (TPA), the following two fundamental requirements have to be met: 1) TPA should be able to efficiently audit the cloud data storage without demanding the local copy of data, and introduce no additional on-line burden to the cloud user; 2) The third party auditing process should bring in no new vulnerabilities towards user data privacy. In this paper, we utilize the public key based homomorphic authenticator and uniquely integrate it with random mask technique to achieve a privacy-preserving public auditing system for cloud data storage security while keeping all above requirements in mind. To support efficient handling of multiple auditing tasks, we further explore the technique of bilinear aggregate signature to extend our main result into a multi-user setting, where TPA can perform multiple auditing tasks simultaneously. Extensive security and performance analysis shows the proposed schemes are provably secure and highly efficient.

1 Introduction

Cloud Computing has been envisioned as the next-generation architecture of IT enterprise, due to its long list of unprecedented advantages in the IT history: on-demand self-service, ubiquitous network access, location independent resource pooling, rapid resource elasticity, usage-based pricing and transference of risk [1]. As a disruptive technology with profound implications, Cloud Computing is transforming the very nature of how businesses use information technology. One fundamental aspect of this paradigm shifting is that data is being centralized or outsourced into the Cloud. From users' perspective, including both individuals and IT enterprises, storing data remotely into the cloud in a flexible on-demand manner brings appealing benefits: relief of the burden for storage management, universal data access with independent geographical locations, and avoidance of capital expenditure on hardware, software, and personnel maintenances, etc [2].

While these advantages of using clouds are unarguable, due to the opaqueness of the Cloud—as separate administrative entities, the internal operation details of cloud service providers (CSP) may not be known by cloud users—data outsourcing is also relinquishing user's ultimate control over the fate of their data. As a result, the correctness of the data in the cloud is being put at risk due to the following reasons. First of all, although the infrastructures under the cloud are much more powerful and reliable than personal computing devices, they are still facing the broad range of both internal and external threats for data integrity. Examples of outages and security breaches of noteworthy cloud services appear from time to time [3–6]. Secondly, for the benefits of their own, there do exist various motivations for cloud service providers to behave unfaithfully towards the cloud users regarding the status of their outsourced data. Examples include cloud service providers, for monetary reasons, reclaiming storage by discarding data that has not been or is rarely accessed, or even hiding data loss incidents so as to maintain a reputation [7–9]. In short, although outsourcing data into the cloud is economically attractive for the cost and complexity of long-term large-scale data storage, it does not offer any guarantee on data integrity and availability. This problem, if not properly addressed, may impede the successful deployment of the cloud architecture.

As users no longer physically possess the storage of their data, traditional cryptographic primitives for the purpose of data security protection can not be directly adopted. Thus, how to efficiently verify the correctness of outsourced cloud data without the local copy of data files becomes a big challenge for data storage security in Cloud Computing. Note that simply downloading the data for its integrity verification is not a practical solution due to the expensiveness in I/O cost and transmitting the file across the network. Besides, it is often insufficient to detect the data corruption when accessing the data, as it might be too late for recover the data loss or damage. Considering the large size of the outsourced data and the user's constrained resource capability, the ability to audit the correctness of the data in a cloud environment can be formidable and expensive for the cloud users [9,10]. Therefore, to fully ensure the data security and save the cloud users' computation resources, it is of critical importance to enable public auditability for cloud data storage so that the users may resort to a third party auditor (TPA), who has expertise and capabilities that the users do not, to audit the outsourced data when needed. Based on the audit result, TPA could release an audit report, which would not only help users to evaluate the risk of their subscribed cloud data services, but also be beneficial for the cloud service provider to improve their cloud based service platform [8]. In a word, enabling public risk auditing protocols will play an important role for this nascent cloud economy to become fully established, where users will need ways to assess risk and gain trust in Cloud.

Recently, the notion of public auditability has been proposed in the context of ensuring remotely stored data integrity under different systems and security models [7, 9, 11, 12]. Public auditability allows an external party, in addition to the user himself, to verify the correctness of remotely stored data. However, most of these schemes [7, 9, 11] do not support the privacy protection of users' data against external auditors, i.e., they may potentially reveal user data information to the auditors, as will be discussed in Section 3.3. This drawback greatly affects the security of these protocols in Cloud Computing. From the perspective of protecting data privacy, the users, who own the data and rely on TPA just for the storage security of their data, do not want this auditing process introducing new vulnerabilities of unauthorized information leakage towards their data security [13]. Moreover, there are legal regulations, such as the US Health Insurance Portability and Accountability Act (HIPAA) [14], further demanding the outsourced data not to be leaked to external parties [8]. Exploiting data encryption before outsourcing [12] is one way to mitigate this privacy concern, but it is only complementary to the privacy-preserving public auditing scheme to be proposed in this paper. Without a properly designed auditing protocol, encryption itself can not prevent data from "flowing away" towards external parties during the auditing process. Thus, it does not completely solve the problem of protecting data privacy but just reduces it to the one of managing the encryption keys. Unauthorized data leakage still remains a problem due to the potential exposure of encryption keys.

Therefore, how to enable a privacy-preserving third-party auditing protocol, independent to data encryption, is the problem we are going to tackle in this paper. Our work is among the first few ones to support privacy-preserving public auditing in Cloud Computing, with a focus on data storage. Besides, with the prevalence of Cloud Computing, a foreseeable increase of auditing tasks from different users may be delegated to TPA. As the individual auditing of these growing tasks can be tedious and cumbersome, a natural demand is then how to enable TPA to efficiently perform the multiple auditing tasks in a batch manner, i.e., simultaneously. To address these problems, our work utilizes the technique of public key based homomorphic authenticator [7, 9, 11], which enables TPA to perform the auditing without demanding the local copy of data and thus drastically reduces the communication and computation overhead as compared to the straightforward data auditing approaches. By integrating the homomorphic authenticator with random mask technique, our protocol guarantees that TPA could not learn any knowledge about the data content stored in the cloud server during the efficient auditing process. The aggregation and algebraic properties of the authenticator further benefit our design for the batch auditing. Specifically, our contribution in this work can be summarized as the following three aspects:

1) We motivate the public auditing system of data storage security in Cloud Computing and provide a privacy-preserving auditing protocol, i.e., our scheme supports an external auditor to audit user's outsourced data in the cloud without learning knowledge on the data content.

2) To the best of our knowledge, our scheme is the first to support scalable and efficient public auditing in the Cloud Computing. In particular, our scheme achieves batch auditing where multiple delegated auditing tasks from different users can be performed simultaneously by the TPA.

3) We prove the security and justify the performance of our proposed schemes through concrete experiments and comparisons with the state-of-the-art.

The rest of the paper is organized as follows. Section II introduces the system and threat model, our design goals, notations and preliminaries. Then we provide the detailed description of our scheme in Section III. Section IV gives the security analysis and performance evaluation, followed by Section V which overviews the related work. Finally, Section VI gives the concluding remark of the whole paper.

2 Problem Statement

2.1 The System and Threat Model

We consider a cloud data storage service involving three different entities, as illustrated in Fig. 1: the *cloud user* (U), who has large amount of data files to be stored in the cloud; the *cloud server* (CS), which is managed by *cloud service provider* (CSP) to provide data storage service and has significant storage space and computation resources (we will not differentiate CS and CSP hereafter.); the *third party auditor* (TPA), who has expertise and capabilities that cloud users do not have and is trusted to assess the cloud storage service security on behalf of the user upon request.

Users rely on the CS for cloud data storage and maintenance. They may also dynamically interact with the CS to access and update their stored data for various application purposes. The users may resort to TPA for ensuring the storage security of their outsourced data, while hoping to keep their data private from TPA. We consider the existence of a semi-trusted CS in the sense that in most of time it behaves properly and does not deviate from the prescribed protocol execution. While providing the cloud data storage based services, for their own benefits the CS might neglect to keep or deliberately delete rarely accessed data files which belong to ordinary cloud users. Moreover, the CS may decide to hide the data corruptions caused by server hacks or Byzantine failures to maintain reputation. We assume the TPA, who is in the business of auditing, is reliable and independent, and thus has no incentive to collude with either the CS or the users during the auditing process. TPA should be able to efficiently audit the cloud data storage without local copy of data and without bringing in additional on-line burden to cloud users. However, any possible leakage of user's outsourced data towards TPA through the auditing protocol should be prohibited.

Note that to achieve the audit delegation and authorize CS to respond to TPA's audits, the user can sign a certificate granting audit rights to the TPA's public key, and all audits from the TPA are authenticated against such a certificate. These authentication handshakes are omitted in the following presentation.

2.2 Design Goals

To enable privacy-preserving public auditing for cloud data storage under the aforementioned model, our protocol design should achieve the following security and performance guarantee: 1) Public auditability: to allow TPA to verify the correctness of the cloud data on demand without

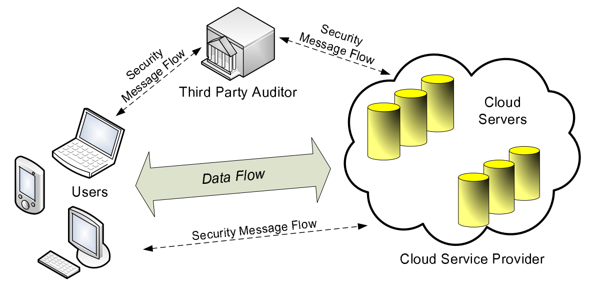

Fig. 1: The architecture of cloud data storage service

retrieving a copy of the whole data or introducing additional on-line burden to the cloud users; 2) Storage correctness: to ensure that there exists no cheating cloud server that can pass the audit from TPA without indeed storing users' data intact; 3) Privacy-preserving: to ensure that there exists no way for TPA to derive users' data content from the information collected during the auditing process; 4) Batch auditing: to enable TPA with secure and efficient auditing capability to cope with multiple auditing delegations from possibly large number of different users simultaneously; 5) Lightweight: to allow TPA to perform auditing with minimum communication and computation overhead.

2.3 Notation and Preliminaries

- F – the data file to be outsourced, denoted as a sequence of n blocks $m_1, \ldots, m_n \in \mathbb{Z}_p$ for some large prime p.
- $f_{key}(\cdot)$ – pseudorandom function (PRF), defined as: $\{0,1\}^* \times key \to \mathbb{Z}_p$.
- $\pi_{key}(\cdot)$ – pseudorandom permutation (PRP), defined as: $\{0,1\}^{\log_2(n)} \times key \to \{0,1\}^{\log_2(n)}$.
- $MAC_{key}(\cdot)$ – message authentication code (MAC) function, defined as: $\{0,1\}^* \times key \to \{0,1\}^l$.
- $H(\cdot)$, $h(\cdot)$ – map-to-point hash functions, defined as: $\{0,1\}^* \to G$, where G is some group.

We now introduce some necessary cryptographic background for our proposed scheme.
Bilinear Map Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p. Let g_1 and g_2 be generators of G_1 and G_2, respectively. A bilinear map is a map $e : G_1 \times G_2 \to G_T$ with the following properties [15, 16]: 1) Computable: there exists an efficiently computable algorithm for computing e; 2) Bilinear: for all $u \in G_1$, $v \in G_2$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$; 3) Non-degenerate: $e(g_1, g_2) \neq 1$; 4) for any $u_1, u_2 \in G_1$, $v \in G_2$, $e(u_1u_2, v) = e(u_1, v) \cdot e(u_2, v)$.

3 The Proposed Schemes

In the introduction we motivated the public auditability with achieving economies of scale for cloud computing. This section presents our public auditing scheme for cloud data storage security. We start from the overview of our public auditing system and discuss two straightforward schemes and their demerits. Then we present our main result for privacy-preserving public auditing to achieve the aforementioned design goals. We also show how to extent our main scheme to support batch auditing for TPA upon delegations from multi-users. Finally, we discuss how to adapt our main result to support data dynamics.

3.1 Definitions and Framework of Public Auditing System

We follow the similar definition of previously proposed schemes in the context of remote data integrity checking [7, 11, 12] and adapt the framework for our privacy-preserving public auditing system.

A public auditing scheme consists of four algorithms (`KeyGen`, `SigGen`, `GenProof`, `VerifyProof`). `KeyGen` is a key generation algorithm that is run by the user to setup the scheme. `SigGen` is used by the user to generate verification metadata, which may consist of MAC, signatures, or other related information that will be used for auditing. `GenProof` is run by the cloud server to generate a proof of data storage correctness, while `VerifyProof` is run by the TPA to audit the proof from the cloud server.

Our public auditing system can be constructed from the above auditing scheme in two phases, `Setup` and `Audit`:

– `Setup`: The user initializes the public and secret parameters of the system by executing `KeyGen`, and pre-processes the data file F by using `SigGen` to generate the verification metadata. The user then stores the data file F at the cloud server, deletes its local copy, and publishes the verification metadata to TPA for later audit. As part of pre-processing, the user may alter the data file F by expanding it or including additional metadata to be stored at server.
– `Audit`: The TPA issues an audit message or challenge to the cloud server to make sure that the cloud server has retained the data file F properly at the time of the audit. The cloud server will derive a response message from a function of the stored data file F by executing `GenProof`. Using the verification metadata, the TPA verifies the response via `VerifyProof`.

Note that in our design, we do not assume any additional property on the data file, and thus regard error-correcting codes as orthogonal to our system. If the user wants to have more error-resiliency, he/she can first redundantly encode the data file and then provide us with the data file that has error-correcting codes integrated.

3.2 The Basic Schemes

Before giving our main result, we first start with two warmup schemes. The first one does not ensure privacy-preserving guarantee and is not as lightweight as we would like. The second one overcomes the first one, but suffers from other undesirable systematic demerits for public auditing: bounded usage and auditor statefulness, which may pose additional on-line burden to users as will be elaborated shortly. We believe the analysis of these basic schemes will lead us to our main result, which overcomes all these drawbacks.

Basic Scheme I The cloud user pre-computes MACs $\sigma_i = MAC_{sk}(i||m_i)$ of each block m_i ($i \in \{1, \ldots, n\}$), sends both the data file F and the MACs $\{\sigma_i\}_{1 \leq i \leq n}$ onto the cloud server, and releases the secret key sk to TPA. During the `Audit` phase, the TPA requests from the cloud server a number of randomly selected blocks and their corresponding MACs to verify the correctness of the data file. The insight behind this approach is that auditing most of the file is much easier than the whole of it. However, this simple solution suffers from the following severe drawbacks: 1) The audit from TPA demands retrieval of users' data, which should be prohibitive because it violates the privacy-preserving guarantee; 2) Its communication and computation complexity are both linear with respect to the sampled data size, which may result in large communication overhead and time delay, especially when the bandwidth available between the TPA and the cloud server is limited.

Basic Scheme II To avoid retrieving data from the cloud server, one may improve the above solution as follows: Before data outsourcing, the cloud user chooses s random message authentication code keys $\{sk_\tau\}_{1 \leq \tau \leq s}$, pre-computes s MACs, $\{MAC_{sk_\tau}(F)\}_{1 \leq \tau \leq s}$ for the whole data file F, and publishes these verification metadata to TPA. The TPA can each time reveal a secret key sk_τ to the cloud server and ask for a fresh keyed MAC for comparison, thus achieving privacy-preserving auditing. However, in this method: 1) the number of times a particular data file can be audited is limited by the number of secret keys that must be a fixed priori. Once all possible secret keys are exhausted, cloud user then has to retrieve data from the server in order to re-compute and re-publish new MACs to TPA. 2) The TPA has to maintain and update state between audits, i.e., keep a track on the possessed MAC keys. Considering the potentially large number of audit delegations from multiple users, maintaining such states for TPA can be difficult and error prone.

Note that another common drawback of the above basic schemes is that they can only support the case of static data, and none of them can deal with data dynamics. For the reason of brevity and clarity, our main result will focus on the static data, too. In Section 3.5, we will show how to adapt our main result to support dynamic data update.

3.3 The Privacy-Preserving Public Auditing Scheme

To effectively support public auditability without having to retrieve the data blocks themselves, we resort to the homomorphic authenticator technique [7, 9, 11]. Homomorphic authenticators are unforgeable verification metadata generated from individual data blocks, which can be securely aggregated in such a way to assure an auditor that a linear combination of data blocks is correctly computed by verifying only the aggregated authenticator. However, the direct adoption of these techniques is not suitable for our purposes, since the linear combination of blocks may potentially reveal user data information, thus violating the privacy-preserving guarantee. Specifically, if enough number of the linear combinations of the same blocks are collected, the TPA can simply derive the user's data content by solving a system of linear equations.

Overview To achieve privacy-preserving public auditing, we propose to uniquely integrate the homomorphic authenticator with random mask technique. In our protocol, the linear combination of sampled blocks in the server's response is masked with randomness generated by a pseudo random function (PRF). With random mask, the TPA no longer has all the necessary information to build up a correct group of linear equations and therefore cannot derive the user's data content, no matter how many linear combinations of the same set of file blocks can be collected. Meanwhile, due to the algebraic property of the homomorphic authenticator, the correctness validation of the block-authenticator pairs will not be affected by the randomness generated from a PRF, which will be shown shortly. Note that in our design, we use public key based homomorphic authenticator, specifically, the one in [11] which is based on BLS signature [16], to equip the auditing protocol with public auditability. Its flexibility in signature aggregation will further benefit us for the multi-task auditing.

Scheme Details Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p, and $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map as introduced in preliminaries. Let g be the generator of G_2. $H(\cdot)$ is a secure map-to-point hash function: $\{0,1\}^* \rightarrow G_1$, which maps strings uniformly to G_1. Another hash function $h(\cdot) : G_T \rightarrow \mathbb{Z}_p$ maps group element of G_T uniformly to $\mathbb{Z}_p$. The proposed scheme is as follows:

`Setup Phase`:

1) The cloud user runs `KeyGen` to generate the system's public and secret parameters. He chooses a random $x \leftarrow \mathbb{Z}_p$, a random element $u \leftarrow G_1$, and computes $v \leftarrow g^x$. The secret parameter is $sk = (x)$ and the public parameters are $pk = (v, g, u, e(u, v))$. Given data file $F = (m_1, \ldots, m_n)$, the user runs `SigGen` to compute signature σ_i for each block m_i: $\sigma_i \leftarrow (H(i) \cdot u^{m_i})^x \in G_1$ $(i = 1, \ldots, n)$. Denote the set of signatures by $\Phi = \{\sigma_i\}_{1 \leq i \leq n}$. The user then sends $\{F, \Phi\}$ to the server and deletes them from its local storage.

`Audit Phase`:

2) During the auditing process, to generate the audit message "*chal*", the TPA picks a random c-element subset $I = \{s_1, \ldots, s_c\}$ of set $[1, n]$, where $s_q = \pi_{k_{prp}}(q)$ for $1 \leq q \leq c$ and k_{prp} is the randomly chosen permutation key by TPA for each auditing. We assume that $s_1 \leq \cdots \leq s_c$. For each element $i \in I$, the TPA also chooses a random value ν_i (of a relative small bit length compared to $|p|$). The message "*chal*" specifies the positions of the blocks that are required to be checked in this `Audit` phase. The TPA sends the $chal = \{(i, \nu_i)\}_{i \in I}$ to the server.

3) Upon receiving challenge $chal = \{(i, \nu_i)\}_{i \in I}$, the server runs `GenProof` to generate a response proof of data storage correctness. Specifically, the server chooses a random element $r \leftarrow \mathbb{Z}_p$ via $r = f_{k_{prf}}(chal)$, where k_{prf} is the randomly chosen PRF key by server for each auditing, and calculates $R = e(u, v)^r$. Let μ' denote the linear combination of sampled blocks specified in *chal*: $\mu' = \sum_{i \in I} \nu_i m_i$. To blind μ' with r, the server computes: $\mu = r + \gamma\mu' \bmod p$, where $\gamma = h(R)$. Meanwhile, the server also calculates an aggregated signature $\sigma = \prod_{i \in I} \sigma_i^{\nu_i} \in G_1$. It then sends $\{\mu, \sigma, R\}$ as the response proof of storage correctness to the TPA. With the response from the

server, the TPA runs `VerifyProof` to validate the response by first computing $\gamma = h(R)$ and then checking the verification equation

$$R \cdot e(\sigma^{\gamma}, g) \overset{?}{=} e((\prod_{i=s_1}^{s_c} H(i)^{\nu_i})^{\gamma} \cdot u^{\mu}, v) \tag{1}$$

The correctness of the above verification equation can be elaborated as follows:

$$\begin{aligned} R \cdot e(\sigma^{\gamma}, g) &= e(u, v)^r \cdot e((\prod_{i=s_1}^{s_c} (H(i) \cdot u^{m_i})^{x \cdot \nu_i})^{\gamma}, g) \\ &= e(u^r, v) \cdot e((\prod_{i=s_1}^{s_c} (H(i)^{\nu_i} \cdot u^{\nu_i m_i})^{\gamma}, g)^x \\ &= e(u^r, v) \cdot e((\prod_{i=s_1}^{s_c} H(i)^{\nu_i})^{\gamma} \cdot u^{\mu' \gamma}, v) \\ &= e((\prod_{i=s_1}^{s_c} H(i)^{\nu_i})^{\gamma} \cdot u^{\mu' \gamma + r}, v) \\ &= e((\prod_{i=s_1}^{s_c} H(i)^{\nu_i})^{\gamma} \cdot u^{\mu}, v) \end{aligned}$$

It is clear that the random mask r and related $R = e(u, v)^r$ has no effect on the validity of the checking result. The security of this protocol will be proved in Section 4.

Discussion As analyzed at the beginning of this section, this approach ensures the privacy of user data content during the auditing process. Meanwhile, the homomorphic authenticator helps achieve the constant communication overhead for server's response during the audit: the size of $\{\sigma, \mu, R\}$ is fixed and has nothing to do with the number of sampled blocks c. Note that there is no secret keying material or states for TPA to keep or maintain between audits, and the auditing protocol does not pose any potential on-line burden toward users. Since the TPA could "re-generate" the random c-element subset $I = \{s_1, \ldots, s_c\}$ of set $[1, n]$, where $s_q = \pi_{k_{prp}}(q)$, for $1 \leq q \leq c$, unbounded usage is also achieved.

Previous work [7,9] showed that if the server is missing a fraction of the data, then the number of blocks that needs to be checked in order to detect server misbehavior with high probability is in the order of $O(1)$. For example, if the server is missing 1% of the data F, the TPA only needs to audit for $c = 460$ or 300 randomly chosen blocks of F so as to detect this misbehavior with probability larger than 99% or 95%, respectively. Given the huge volume of data outsourced in the cloud, checking a portion of the data file is more affordable and practical for both TPA and cloud server than checking all the data, as long as the sampling strategies provides high probability assurance. In Section 4, we will present the experiment result based on these sampling strategies.

3.4 Support for Batch Auditing

With the establishment of privacy-preserving public auditing in Cloud Computing, TPA may concurrently handle multiple auditing delegations upon different users' requests. The individual auditing of these tasks for TPA can be tedious and very inefficient. Given K auditing delegations on K distinct data files from K different users, it is more advantageous for TPA to batch these multiple tasks together and audit at one time. Keeping this natural demand in mind, we propose to explore the technique of bilinear aggregate signature [15], which supports the aggregation of multiple signatures by distinct signers on distinct messages into a single signature and thus provides efficient verification for the authenticity of all messages. Using this signature aggregation technique and bilinear property, we can now aggregate K verification equations (for K auditing tasks) into a single one, as shown in equation 2, so that the simultaneous auditing of multiple tasks can be achieved.

The details of extending our main result to this multi-user setting is described as follows: Assume there are K users in the system, and each user k has a data file $F_k = (m_{k,1}, \ldots, m_{k,n})$ to be outsourced to the cloud server, where $k \in \{1, \ldots, K\}$. For a particular user k, denote his secret key as $x_k \leftarrow \mathbb{Z}_p$, and the corresponding public parameter as $(v_k, g, u_k, e(u_k, v_k))$ where $v_k = g^{x_k}$. In the `Setup` phase, each user k runs `SigGen` and computes signature $\sigma_{k,i} \leftarrow [H(k||i) \cdot u_k^{m_{k,i}}]^{x_k} \in G_1$ for block $m_{k,i}$ ($i \in \{1, \ldots, n\}$). In the `Audit` phase, the TPA sends the audit challenge $chal = \{(i, \nu_i)\}_{i \in I}$ to the server for auditing data files of all K users. Upon receiving $chal$, for each user k ($k \in \{1, \ldots, K\}$), the server randomly picks $r_k \in \mathbb{Z}_p$ and computes

$$\mu_k = \gamma_k \sum_{i=s_1}^{s_c} \nu_i m_{k,i} + r_k \bmod p \quad \text{and} \quad \sigma_k = \prod_{i=s_1}^{s_c} \sigma_{k,i}^{\nu_i},$$

where $\gamma_k = h(R_k) = h(e(u_k, v_k)^{r_k})$. The server then responses the TPA with $\{\sigma_k, \mu_k, R_k\}_{1 \le k \le K}$. Similar as the single user case, using the properties of the bilinear map, the TPA can first compute $\gamma_k = h(R_k)$ for $1 \le k \le K$ and then check if the following equation holds:

$$R_1 \cdot R_2 \cdots R_K \cdot e(\prod_{k=1}^{K} {\sigma_k}^{\gamma_k}, g) \overset{?}{=} \prod_{k=1}^{K} e((\prod_{i=s_1}^{s_c} H(k||i)^{\nu_i})^{\gamma_k} \cdot u_k^{\mu_k}, v_k) \qquad (2)$$

The left-hand side (LHS) of the equation expands as:

$$\begin{aligned} \text{LHS} &= R_1 \cdot R_2 \cdots R_K \cdot \prod_{k=1}^{K} e({\sigma_k}^{\gamma_k}, g) \\ &= \prod_{k=1}^{K} R_k \cdot e({\sigma_k}^{\gamma_k}, g) \\ &= \prod_{k=1}^{K} e((\prod_{i=s_1}^{s_c} H(k||i)^{\nu_i})^{\gamma_k} \cdot u_k^{\mu_k}, v_k) \end{aligned}$$

which is the right hand side, as required. Note that the last equality follows from equation 1.
Discussion As shown in equation 2, batch auditing not only allows TPA to perform the multiple auditing tasks simultaneously, but also greatly reduces the computation cost on the TPA side. This is because aggregating K verification equations into one helps reduce the number of expensive pairing operations from $2K$, as required in the individual auditing, to $K + 1$. Thus, a considerable amount of auditing time is expected to be saved. Note that the verification equation 2 only holds when all the responses are valid, and fails with high probability when there is even one single invalid response in the batch auditing. In many situations, a response collection may contain invalid responses, especially $\{\mu_k\}_{1 \le k \le K}$, caused by accidental data corruption, or possibly malicious activity by a cloud server. The ratio of invalid responses to the valid could be quite small, and yet a standard batch auditor will reject the entire collection. To further sort out these invalid responses in the batch auditing, we can utilize a recursive divide-and-conquer approach (binary search), as suggested by [17]. Specifically, if the batch auditing fails, we can simply divide the collection of responses into two halves, and recurse the auditing on halves via equation 2. In Section 4.2, we show through carefully designed experiment that using this recursive binary search approach, even if up to 18% of responses are invalid, batch auditing still performs faster than individual verification.

3.5 Support for Data Dynamics

In Cloud Computing, outsourced data might not only be accessed but also updated frequently by users for various application purposes [9, 18, 19]. Hence, supporting data dynamics for privacy-preserving public risk auditing is also of paramount importance. Now we show how our main scheme can be adapted to build upon the existing work [9] to support data dynamics, including block level operations of modification, deletion and insertion.

In [9], data dynamics support is achieved by replacing the index information i with m_i in the computation of block signatures and using the classic data structure-Merkle hash tree (MHT) [20] for the underlying block sequence enforcement. As a result, the signature for each block is changed to $\sigma_i = (H(m_i) \cdot u^{m_i})^x$. We can adopt this technique in our design to achieve privacy-preserving public risk auditing with support of data dynamics. Specifically, in the `Setup` phase, the user has to generate and send the tree root TR_{MHT} to TPA as additional metadata, where the leaf nodes of MHT are values of $H(m_i)$. In the `Audit` phase, besides $\{\mu, \sigma, R\}$, the server's response should also include $\{H(m_i)\}_{i \in I}$ and their corresponding auxiliary authentication information (AAI) in the MHT. Upon receiving the response, TPA should first use TR_{MHT} and the AAI to authenticate $\{H(m_i)\}_{i \in I}$ computed by the server. Once $\{H(m_i)\}_{i \in I}$ are authenticated, TPA can then perform the auditing on $\{\mu, \sigma, R, \{H(m_i)\}_{i \in I}\}$ via equation 1, where $\prod_{s_1 \leq i \leq s_c} H(i)^{\nu_i}$ is now replaced by $\prod_{s_1 \leq i \leq s_c} H(m_i)^{\nu_i}$. Note that data privacy is still preserved due to the random mask R. The details of handling dynamic operations are similar to [9] and thus omitted.

4 Security Analysis and Performance Evaluation

4.1 Security Proofs

We evaluate the security of the proposed scheme by analyzing its fulfillment of the security guarantee described in Section 2, namely, the storage correctness and privacy-preserving. We start from the single user case, where our main result is originated. Then we show how to extend the security guarantee to a multi-user setting, where batch auditing for TPA is enabled. All proofs are derived on the probabilistic base, i.e., with high probability assurance, which we omit writing explicitly.

Storage Correctness Guarantee We need to prove that the cloud server can not generate valid response toward TPA without faithfully storing the data, as captured by Theorem 1.

Theorem 1. *If the cloud server passes the* `Audit` *phase, then it must indeed possess the specified data intact as it is.*

Proof (Proof Sketch). The proof consists of three steps. First, we show that there exists no malicious server that can forge a valid response $\{\sigma, \mu, R\}$ to pass the verification equation 1. The correctness of this statement follows from the Theorem 4.2 proposed in [11]. Note that the value R in our protocol, which enables the privacy-preserving guarantee, will not affect the validity of the equation, due to the circular relationship between R and γ in $\gamma = h(R)$ and the verification equation.

Next, we show that if the response $\{\sigma, \mu, R\}$ is valid, where $\mu = \gamma\mu' + r$ and $\gamma = h(R) = h(e(u, v)^r)$, then the underlying μ' must be valid too. Indeed, we can extract μ' from the protocol in the random oracle model.

Finally, similar to the argument in [11], we show that the validity of μ' implies the correctness of $\{m_i\}_{i \in I}$ where $\mu' = \sum_{i \in I} \nu_i m_i$. Here we utilize the small exponent (SE) test technique of batch verification in [21]. Because $\{\nu_i\}$ are picked up randomly by the TPA in each `Audit` phase, $\{\nu_i\}$ can be viewed similarly as the random chosen exponents in the SE test [21]. Therefore, the correctness of individual sampled blocks is ensured. All above sums up to the storage correctness guarantee.

Privacy Preserving Guarantee We want to make sure that TPA can not derive users' data content from the information collected during auditing process. This is equivalent to prove the Theorem 2. Note that if μ' can be derived by TPA, then $\{m_i\}_{i \in I}$ can be easily obtained by solving a group of linear equations when enough combinations of the same blocks are collected.

Theorem 2. *From the server's response $\{\sigma, \mu, R\}$, TPA cannot recover μ'.*

Proof (Proof Sketch). Again, we argue in three steps. First, recall the relationship between μ' and σ, where

$$\begin{aligned}\sigma &= \prod_{i\in I} \sigma_i^{\nu_i} = \prod_{i\in I} (H(i)\cdot u^{m_i})^{x\cdot\nu_i}\\ &= [\prod_{i\in I} H(i)^{\nu_i}\cdot u^{\mu'}]^x = [\prod_{i\in I} H(i)^{\nu_i}]^x \cdot [(u^{\mu'})^x].\end{aligned}$$

This can be analyzed as follows: $(u^{\mu'})^x$ is blinded by $[\prod_{i\in I} H(i)^{\nu_i}]^x$. However, to reconstruct $[\prod_{i\in I} H(i)^{\nu_i}]^x$ from $\{\nu_i\}_{i\in I}$, $\{H(i)\}_{i\in I}$, and g^x, which is the only information TPA can utilize, is the same as forging a BLS signature [16]. Therefore, TPA cannot derive the value of $(u^{\mu'})^x$, let alone μ', which requires solving discrete-log problems.

Second, we consider how to learn μ' from μ. Note that μ is blinded by r as $\mu = \gamma\mu' + r$ and $R = e(u, v)^r$, where r is chosen randomly by cloud server and is unknown to TPA. Even with R, due to the hardness of discrete-log assumption, the value r is still hidden against TPA. Thus, privacy of μ' is guaranteed from μ.

Finally, all that remains is to prove from $\{\sigma, \mu, R\}$, still no information on μ' can be obtained by TPA. Recall that r is a random private value chosen by the server and $\mu = \gamma\mu' + r$, where $\gamma = h(e(u, v)^r)$. Following the same technique of Schnorr signature [22], our auditing protocol between TPA and cloud server can be regarded as a provably secure honest zero knowledge identification scheme [23], by viewing μ' as a secret key and γ as a challenge value, which implies no information on μ' can be leaked. Indeed, it is easy to simulate valid response $\{\mu, R\}$ without knowing μ' in the random oracle model. This completes the proof of Theorem 2.

Security Guarantee for Batch Auditing Now we show that extending our main result to a multi-user setting will not affect the aforementioned security insurance, as shown in Theorem 3:

Theorem 3. *Our batch auditing protocol achieves the same storage correctness and privacy preserving guarantee as in the single-user case.*

Proof (Proof Sketch). We only prove the storage correctness guarantee, as the privacy-preserving guarantee in the multi-user setting is similar to that of Theorem 2, and thus omitted here. The proposed batch auditing protocol is built upon the aggregate signature scheme proposed in [15]. According to the security strength of aggregate signature [15], in our multi-user setting, there exists no malicious cloud servers that can forge valid $\mu_1, \ldots, \mu_k$ in the responses to pass the verification equation 2. Actually, the equation 2 functions as a kind of screening test as proposed in [21]. While the screening test may not guarantee the validity of each individual σ_k, it does ensure the authenticity of μ_k in the batch auditing protocol, which is adequate for the rationale in our case. Once the validity of $\mu_1, \ldots, \mu_k$ is guaranteed, from the proof of Theorem 1, the storage correctness guarantee in the multi-user setting is achieved.

Table 1: Notation summary of cryptographic operations.

$Hash^t_{\mathbb{G}}$	hash t values into the group $\mathbb{G}$.
$Add^t_{\mathbb{G}}$	t additions in group $\mathbb{G}$.
$Mult^t_{\mathbb{G}}$	t multiplications in group $\mathbb{G}$.
$Exp^t_{\mathbb{G}}(\ell)$	t exponentiations g^{a_i}, where $g \in \mathbb{G}$, $\|a_i\| = \ell$.
$m\text{-}MultExp^t_{\mathbb{G}}(\ell)$	t m-term exponentiations $\prod_{i=1}^m g^{a_i}$.
$Pair^t_{\mathbb{G},\mathbb{H}}$	t pairings $e(g_i, h_i)$, where $g_i \in \mathbb{G}$, $h_i \in \mathbb{H}$.
$m\text{-}MultPair^t_{\mathbb{G},\mathbb{H}}$	t m-term pairings $\prod_{i=1}^m e(g_i, h_i)$.

Table 2: Performance comparison under different number of sampled blocks c for high assurance auditing.

	Our Scheme		[11]	
Sampled blocks c	460	300	460	300
Sever compt. time (ms)	411.00	270.20	407.66	265.87
TPA compt. time (ms)	507.79	476.81	504.25	472.55
Comm. cost (Byte)	160	40	160	40

4.2 Performance Analysis

We now assess the performance of the proposed privacy-preserving public auditing scheme. We will focus on the extra cost introduced by the privacy-preserving guarantee and the efficiency of the proposed batch auditing technique. The experiment is conducted using C on a Linux system with an Intel Core 2 processor running at 1.86 GHz, 2048 MB of RAM, and a 7200 RPM Western Digital 250 GB Serial ATA drive with an 8 MB buffer. Algorithms use the Pairing-Based Cryptography (PBC) library version 0.4.18. The elliptic curve utilized in the experiment is a MNT curve, with base field size of 159 bits and the embedding degree 6. The security level is chosen to be 80 bit, which means $|\nu_i| = 80$ and $|p| = 160$. All experimental results represent the mean of 20 trials.

Cost of Privacy-preserving Guarantee We begin by estimating the cost in terms of basic cryptographic operations, as notated in Table 1. Suppose there are c random blocks specified in the *chal* during the `Audit` phase. Under this setting, we quantify the extra cost introduced by the support of privacy-preserving into server computation, auditor computation as well as communication overhead. On the server side, the generated response includes an aggregated signature $\sigma = \prod_{i \in I} \sigma_i^{\nu_i} \in G_1$, a random metadata $R = e(u, v)^r \in G_T$, and a blinded linear combination of sampled blocks $\mu = \gamma \sum_{i \in I} \nu_i m_i + r \in \mathbb{Z}_p$, where $\gamma = h(R) \in \mathbb{Z}_p$. The corresponding computation cost is $c\text{-}MultExp^1_{G_1}(|\nu_i|)$, $Exp^1_{G_T}(|p|)$, and $Hash^1_{\mathbb{Z}_p} + Add^c_{\mathbb{Z}_p} + Mult^{c+1}_{\mathbb{Z}_p}$, respectively. Compared to the existing homomorphic authenticator based solution for ensuring remote data integrity [11][3], the extra cost for protecting the user privacy, resulted from the random mask R, is only a constant: $Exp^1_{G_T}(|p|) + Mult^1_{\mathbb{Z}_p} + Hash^1_{\mathbb{Z}_p} + Add^1_{\mathbb{Z}_p}$, which has nothing to do with the number of sampled blocks c. When c is set to be 460 or 300 for high assurance of auditing, as discussed in Section 3.3, the extra cost for privacy-preserving guarantee on the server side would be negligible against the total server computation for response generation.

Similarly, on the auditor side, upon receiving the response $\{\sigma, R, \mu\}$, the corresponding computation cost for response validation is $Hash^1_{\mathbb{Z}_p} + c\text{-}MultExp^1_{G_1}(|\nu_i|) + Hash^c_{G_1} + Mult^1_{G_1} + Mult^1_{G_T} + Exp^3_{G_1}(|p|) + Pair^2_{G_1,G_2}$, among which only $Hash^1_{\mathbb{Z}_p} + Exp^2_{G_1}(|p|) + Mult^1_{G_T}$ account for the additional constant computation cost. For $c = 460$ or 300, and considering the relatively expensive pairing operations, this extra cost imposes little overhead on the overall cost of response validation, and thus can be ignored. For the sake of completeness, Table 2 gives the experiment result on performance comparison between our scheme and the state-of-the-art [11]. It can be shown that the performance of our scheme is almost the same as that of [11], even if our scheme supports privacy-preserving guarantee while [11] does not. Note that in our scheme, the server's response $\{\sigma, R, \mu\}$ contains an additional random element R, which is a group element of G_T and has the size close to 960 bits. This explains the extra communication cost of our scheme opposing to [11].

Batch Auditing Efficiency Discussion in Section 3.4 gives an asymptotic efficiency analysis on the batch auditing, by considering only total number of expensive pairing operations. However, on the practical side, there are additional operations required for batching, such as modular exponentiations and multiplications. Meanwhile, the different sampling strategies, i.e., different number

[3] We refer readers to [11] for detailed description of homomorphic authenticator based solutions.

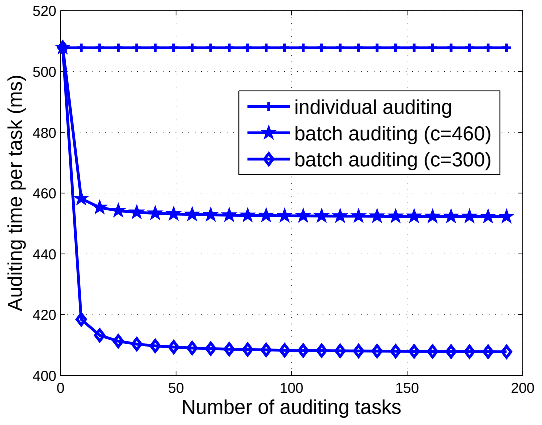

Fig. 2: Comparison on auditing time between batch auditing and individual auditing. Per task auditing time denotes the total auditing time divided by the number of tasks. For clarity reasons, we omit the straight curve for individual auditing when c=300.

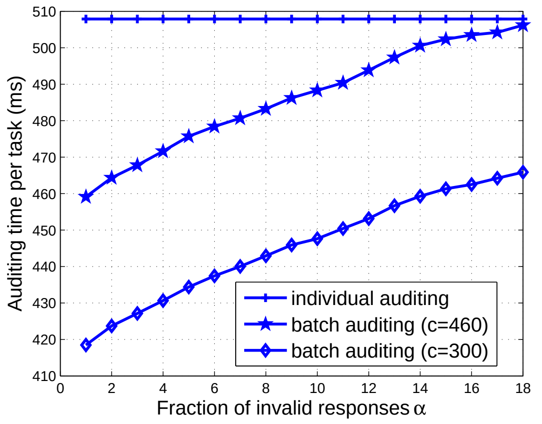

Fig. 3: Comparison on auditing time between batch auditing and individual auditing, when α-fraction of 256 responses are invalid. Per task auditing time denotes the total auditing time divided by the number of tasks.

of sampled blocks c, is also a variable factor that affects the batching efficiency. Thus, whether the benefits of removing pairings significantly outweighs these additional operations is remained to be verified. To get a complete view of batching efficiency, we conduct a similar timed batch auditing test as in [17], where the number of auditing tasks is increased from 1 to approximately 200 with intervals of 8. The performance of the corresponding non-batched (individual) auditing is provided as a baseline for the measurement. Following the same experimental setting as $c = 460$ and 300, the average per task auditing time for both batch auditing and the individual auditing is shown in Fig. 2, where the per task auditing time is computed by dividing total auditing time by the number of tasks. It can be shown that compared to individual auditing, batch auditing indeed helps reduce the TPA's computation cost, as more than 11% and 14% of per-task auditing time is saved, when c is set to be 460 and 300, respectively.

Sorting out Invalid Responses Now we use experiment to justify the efficiency of our recursive binary search approach for TPA to sort out the invalid responses when batch auditing fails, as discussed in Section 3.4. Note that this experiment is tightly pertained to works by [17, 24], which evaluates the batch verification efficiency of various short signature schemes.

To evaluate the feasibility of the recursive approach, we first generate a collection of 256 valid responses, which implies the TPA may concurrently handle 256 different auditing delegations. We then conduct the tests repeatedly while randomly corrupting an α-fraction, ranging from 0 to 18%, by replacing them with random values. The average auditing time per task against the individual auditing approach is presented in Fig. 3. The result shows that even the number of invalid responses exceeds 15% of the total batch size, the performance of batch auditing can still be safely concluded as more preferable than the straightforward individual auditing. Note that this is consistent with the experiment results derived in [17].

5 Related Work

Ateniese *et al.* [7] are the first to consider public auditability in their defined "provable data possession" (PDP) model for ensuring possession of data files on untrusted storages. Their scheme utilizes the RSA-based homomorphic authenticators for auditing outsourced data and suggests randomly sampling a few blocks of the file. However, the public auditability in their scheme demands the linear combination of sampled blocks exposed to external auditor. When used directly, their protocol is not provably privacy preserving, and thus may leak user data information to the auditor. Juels *et al.* [12] describe a "proof of retrievability" (PoR) model, where spot-checking and error-correcting codes are used to ensure both "possession" and "retrievability" of data files on remote archive service systems. However, the number of audit challenges a user can perform is a fixed priori, and public auditability is not supported in their main scheme. Although they describe a straightforward Merkle-tree construction for public PoRs, this approach only works with encrypted data. Shacham *et al.* [11] design an improved PoR scheme built from BLS signatures with full proofs of security in the security model defined in [12]. Similar to the construction in [7], they use publicly verifiable homomorphic authenticators that are built from provably secure BLS signatures. Based on the elegant BLS construction, public retrievability is achieved. Again, their approach does not support privacy-preserving auditing for the same reason as [7]. Shah *et al.* [8,13] propose allowing a TPA to keep online storage honest by first encrypting the data then sending a number of pre-computed symmetric-keyed hashes over the encrypted data to the auditor. The auditor verifies both the integrity of the data file and the server's possession of a previously committed decryption key. This scheme only works for encrypted files, and it suffers from the auditor statefulness and bounded usage, which may potentially bring in on-line burden to users when the keyed hashes are used up.

In other related work, Ateniese *et al.* [25] propose a partially dynamic version of the prior PDP scheme that uses only symmetric key cryptography. However, the system imposes a priori bound on the number of audits and does not support public auditability. In [18], Wang *et al.* consider a similar support for partial dynamic data storage in distributed scenario. The proposed challenge-response protocol can both determine the data correctness and locate possible errors. In a subsequent work, Wang *et al.* [9] propose to combine BLS based homomorphic authenticator with MHT to support both public auditability and fully data dynamics. Almost simultaneously, Erway *et al.* [19] developed a skip lists based scheme to enable provable data possession with fully dynamics support. However, all their protocol requires the linear combination of sampled blocks just as [7, 11], and thus does not support privacy-preserving auditing on user's outsourced data.

While all above schemes provide methods for efficient auditing and provable assurance on the correctness of remotely stored data, none of them meet all the requirements for privacy-preserving public auditing in Cloud Computing, as supported in our result. More importantly, none of these schemes consider batch auditing, which will greatly reduce the computation cost on the TPA when coping with large number of audit delegations.

6 Conclusion

In this paper, we propose a privacy-preserving public auditing system for data storage security in Cloud Computing, where TPA can perform the storage auditing without demanding the local copy of data. We utilize the homomorphic authenticator and random mask technique to guarantee that TPA would not learn any knowledge about the data content stored on the cloud server during the efficient auditing process, which not only eliminates the burden of cloud user from the tedious and possibly expensive auditing task, but also alleviates the users' fear of their outsourced data leakage. Considering TPA may concurrently handle multiple audit sessions from different users for their outsourced data files, we further extend our privacy-preserving public auditing protocol into a multi-user setting, where TPA can perform the multiple auditing tasks in a batch manner, i.e., simultaneously. Extensive security and performance analysis shows that the proposed schemes are provably secure and highly efficient. We believe all these advantages of the proposed schemes will shed light on economies of scale for Cloud Computing.

Acknowledgement

This work was supported in part by the US National Science Foundation under grant CNS-0831963, CNS-0626601, CNS-0716306, and CNS-0831628. The authors would like to thank anonymous reviewers and Sherman S.M. Chow for their useful comments on the preparation of the full version.

References

1. P. Mell and T. Grance, "Draft NIST working definition of cloud computing," Referenced on June. 3rd, 2009 Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2009.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28, Feb 2009.
3. N. Gohring, "Amazon's s3 down for several hours," Online at http://www.pcworld.com/businesscenter/article/142549/amazons_s3_down_for_several_hours.html, 2008.
4. Amazon.com, "Amazon s3 availability event: July 20, 2008," Online at http://status.aws.amazon.com/s3-20080720.html, July 2008.
5. S. Wilson, "Appengine outage," Online at http://www.cio-weblog.com/50226711/appengine_outage.php, June 2008.
6. B. Krebs, "Payment Processor Breach May Be Largest Ever," Online at http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html, Jan. 2009.
7. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," Cryptology ePrint Archive, Report 2007/202, 2007, http://eprint.iacr.org/.
8. M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," Cryptology ePrint Archive, Report 2008/186, 2008, http://eprint.iacr.org/.
9. Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. of ESORICS'09*, Saint Malo, France, Sep. 2009.
10. Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing," 2009, http://www.cloudsecurityalliance.org.
11. H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. of Asiacrypt 2008*, vol. 5350, Dec 2008, pp. 90–107.
12. A. Juels and J. Burton S. Kaliski, "Pors: Proofs of retrievability for large files," in *Proc. of CCS'07*, Alexandria, VA, October 2007, pp. 584–597.
13. M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in *Proc. of HotOS'07*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–6.
14. 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPPA)," Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996, last access: July 16, 2009.
15. D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. of Eurocrypt 2003, volume 2656 of LNCS*. Springer-Verlag, 2003, pp. 416–432.

16. D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” in *Proc. of ASIACRYPT’01*. London, UK: Springer-Verlag, 2001, pp. 514–532.
17. A. L. Ferrara, M. Greeny, S. Hohenberger, and M. Pedersen, “Practical short signature batch verification,” in *Proceedings of CT-RSA, volume 5473 of LNCS*. Springer-Verlag, 2009, pp. 309–324.
18. C. Wang, Q. Wang, K. Ren, and W. Lou, “Ensuring data storage security in cloud computing,” in *Proc. of IWQoS’09*, July 2009.
19. C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Dynamic provable data possession,” in *Proc. of CCS’09*, 2009.
20. R. C. Merkle, “Protocols for public key cryptosystems,” in *Proc. of IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 1980.
21. M. Bellare, J. Garay, and T. Rabin, “Fast batch verification for modular exponentiation and digital signatures,” in *Proceedings of Eurocrypt 1998, volume 1403 of LNCS*. Springer-Verlag, 1998, pp. 236–250.
22. C.-P. Schnorr, “Efficient signature generation by smart cards,” *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.
23. D. Pointcheval and J. Stern, “Security arguments for digital signatures and blind signatures,” *J. Cryptology*, vol. 13, no. 3, pp. 361–396, 2000.
24. J. Camenisch, S. Hohenberger, and M. Pedersen, “Batch verification of short signatures,” in *Proceedings of Eurocrypt EUROCRYPT, volume 4515 of LNCS*. Springer-Verlag, 2007, pp. 243–263.
25. G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, “Scalable and efficient provable data possession,” in *Proc. of SecureComm’08*, 2008.