IEEE TRANSACTIONS ON COMPUTERS VOL:62 NO:2 YEAR 2013

Privacy-Preserving Public Auditing for Secure Cloud Storage

Cong Wang, *Student Member, IEEE,* Sherman S.-M. Chow, Qian Wang, *Student Member, IEEE,*
Kui Ren, *Member, IEEE,* and Wenjing Lou, *Member, IEEE*

Abstract—Using Cloud Storage, users can remotely store their data and enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources, without the burden of local data storage and maintenance. However, the fact that users no longer have physical possession of the outsourced data makes the data integrity protection in Cloud Computing a formidable task, especially for users with constrained computing resources. Moreover, users should be able to just use the cloud storage as if it is local, without worrying about the need to verify its integrity. Thus, enabling public auditability for cloud storage is of critical importance so that users can resort to a third party auditor (TPA) to check the integrity of outsourced data and be worry-free. To securely introduce an effective TPA, the auditing process should bring in no new vulnerabilities towards user data privacy, and introduce no additional online burden to user. In this paper, we propose a secure cloud storage system supporting privacy-preserving public auditing. We further extend our result to enable the TPA to perform audits for multiple users simultaneously and efficiently. Extensive security and performance analysis show the proposed schemes are provably secure and highly efficient.

Index Terms—Data storage, privacy-preserving, public auditability, cryptographic protocols, cloud computing.

1 Introduction

Cloud Computing has been envisioned as the next-generation information technology (IT) architecture for enterprises, 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 to the Cloud. From users' perspective, including both individuals and IT enterprises, storing data remotely to 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 Cloud Computing makes these advantages more appealing than ever, it also brings new and challenging security threats towards users' outsourced data. Since cloud service providers (CSP) are separate administrative entities, data outsourcing is actually 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]–[7]. Secondly, there do exist various motivations for CSP to behave unfaithfully towards the cloud users regarding the status of their outsourced data. For examples, CSP might reclaim storage for monetary reasons by discarding data that has not been or is rarely accessed, or even hide data loss incidents so as to maintain a reputation [8]–[10]. In short, although outsourcing data to the cloud is economically attractive for long-term large-scale data storage, it does not immediately 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 cannot be directly adopted [11]. In particular, simply downloading all the data for its integrity verification is not a practical solution due to the expensiveness in I/O and transmission cost across the network. Besides, it is often insufficient to detect the data corruption only when accessing the data, as it does not give users correctness assurance for those unaccessed data and might be too

- *Cong Wang, Qian Wang, and Kui Ren are with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616. E-mail: {cong,qian,kren}@ece.iit.edu.*
- *Sherman S.-M. Chow is with the Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10012. E-mail: schow@cs.nyu.edu.*
- *Wenjing Lou is with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609. E-mail: wjlou@ece.wpi.edu.*

late to recover the data loss or damage. Considering the large size of the outsourced data and the user's constrained resource capability, the tasks of auditing the data correctness in a cloud environment can be formidable and expensive for the cloud users [10], [12]. Moreover, the overhead of using cloud storage should be minimized as much as possible, such that user does not need to perform too many operations to use the data (in additional to retrieving the data). For example, it is desirable that users do not need to worry about the need to verify the integrity of the data before or after the data retrieval. Besides, there may be more than one user accesses the same cloud storage, say in an enterprise setting. For easier management, it is desirable that the cloud server only entertains verification request from a single designated party.

To fully ensure the data integrity and save the cloud users' computation resources as well as online burden, it is of critical importance to enable public auditing service for cloud data storage, so that users may resort to an independent third party auditor (TPA) to audit the outsourced data when needed. The TPA, who has expertise and capabilities that users do not, can periodically check the integrity of all the data stored in the cloud on behalf of the users, which provides a much more easier and affordable way for the users to ensure their storage correctness in the cloud. Moreover, in addition to help users to evaluate the risk of their subscribed cloud data services, the audit result from TPA would also be beneficial for the cloud service providers to improve their cloud based service platform, and even serve for independent arbitration purposes [9]. In a word, enabling public auditing services 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 the cloud.

Recently, the notion of public auditability has been proposed in the context of ensuring remotely stored data integrity under different system and security models [8], [10], [11], [13]. 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 [8], [10], [13] do not consider the privacy protection of users' data against external auditors. Indeed, they may potentially reveal user data information to the auditors, as will be discussed in Section 3.4. This severe 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 [14]. Moreover, there are legal regulations, such as the US Health Insurance Portability and Accountability Act (HIPAA) [15], further demanding the outsourced data not to be leaked to external parties [9]. Exploiting data encryption before outsourcing [11] 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 cannot 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 key management. Unauthorized data leakage still remains a problem due to the potential exposure of decryption 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 the TPA to efficiently perform multiple auditing tasks in a batch manner, i.e., simultaneously.

To address these problems, our work utilizes the technique of public key based homomorphic linear authenticator (or HLA for short) [8], [10], [13], 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 HLA with random masking, our protocol guarantees that the 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 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 enables an external auditor to audit user's outsourced data in the cloud without learning 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. Specifically, 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, and our design goals. 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 the *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 reliability 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. To save the computation resource as well as the online burden, cloud users may resort to TPA for ensuring the storage integrity of their outsourced data, while hoping to keep their data private from TPA.

We consider the existence of a semi-trusted CS as [16] does. Namely, in most of time it behaves properly and does not deviate from the prescribed protocol execution. However, 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. However, it harms the user if the TPA could learn the outsourced data after the audit.

To authorize the CS to respond to the audit delegated to TPA's, 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 guarantees.

1) Public auditability: to allow TPA to verify the correctness of the cloud data on demand without retrieving a copy of the whole data or introducing additional online burden to the cloud users.
2) Storage correctness: to ensure that there exists no cheating cloud server that can pass the TPA's audit without indeed storing users' data intact.
3) Privacy-preserving: to ensure that the TPA cannot 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.

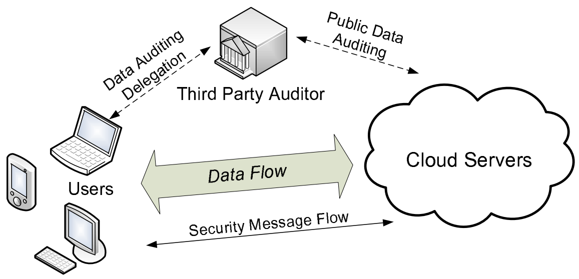


Fig. 1: The architecture of cloud data storage service

3 The Proposed Schemes

This section presents our public auditing scheme which provides a *complete outsourcing* solution of data – not only the data itself, but also its integrity checking. We start from an overview of our public auditing system and discuss two straightforward schemes and their demerits. Then we present our main scheme and show how to extent our main scheme to support batch auditing for the TPA upon delegations from multiple users. Finally, we discuss how to generalize our privacy-preserving public auditing scheme and its support of data dynamics.

3.1 Definitions and Framework

We follow a similar definition of previously proposed schemes in the context of remote data integrity checking [8], [11], [13] 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.

Running a public auditing system consists of 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 and the verification metadata at the cloud server, and deletes its local copy.
  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 and its verification metadata by executing `GenProof`. The TPA then verifies the response via `VerifyProof`.

Our framework assumes the TPA is stateless, which is a desirable property achieved by our proposed solution. It is easy to extend the framework above to capture a stateful auditing system, essentially by spliting the verification metadata into two parts which are stored by the TPA and the cloud server respectively.

Our design does not assume any additional property on the data file. If the user wants to have more error-resiliency, he/she can always first redundantly encodes the data file and then uses our system with the data file that has error-correcting codes integrated.

3.2 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.
- $MAC_{(\cdot)}(\cdot)$ – message authentication code (MAC) function, defined as: $\mathcal{K} \times \{0,1\}^* \rightarrow \{0,1\}^l$ where $\mathcal{K}$ denotes the key space.
- $H(\cdot)$, $h(\cdot)$ – cryptographic hash functions.

We now introduce some necessary cryptographic background for our proposed scheme.

Bilinear Map. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear map is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$. This bilinearity implies that for any $u_1, u_2 \in \mathbb{G}_1$, $v \in \mathbb{G}_2$, $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$. Of course, there exists an efficiently computable algorithm for computing e and the map should be non-trivial, i.e., e is non-degenerate: $e(g_1, g_2) \neq 1$.

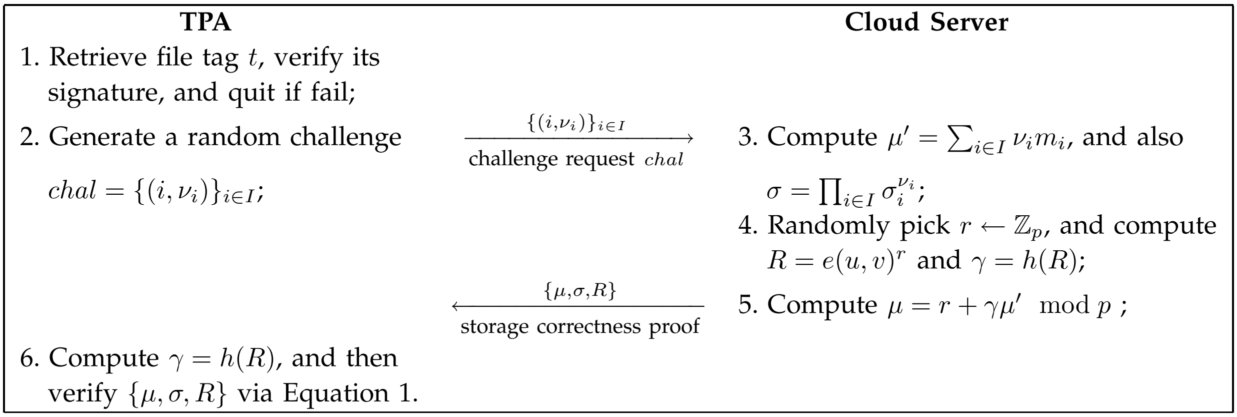
3.3 The Basic Schemes

Before giving our main result, we study two classes of schemes as a warm-up. The first one is a MAC-based solution which suffers from undesirable systematic demerits – bounded usage and stateful verification, which may pose additional online burden to users, in a public auditing setting. This somehow also shows that the auditing problem is still not easy to solve even we have introduced a TPA. The second one is a system based on homomorphic linear authenticators (HLA), which covers many recent proof of storage systems. We will pinpoint the reason why all existing HLA-based systems are not privacy-preserving. The analysis of these basic schemes leads to our main result, which overcomes all these drawbacks. Our main scheme to be presented is based on a specific HLA scheme.

MAC-based Solution. There are two possible ways to make use of MAC to authenticate the data. A trivial way is just uploading the data blocks with their MACs to the server, and sends the corresponding secret key sk to the TPA. Later, the TPA can randomly retrieve blocks with their MACs and check the correctness via sk. Apart from the high (linear in the sampled data size) communication and computation complexities, the TPA requires the knowledge of the data blocks for verification.

To circumvent the requirement of the data in TPA verification, one may restrict the verification to just consist of equality checking. The idea is 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 (deterministic) MACs, $\{MAC_{sk_\tau}(F)\}_{1\leq\tau\leq s}$ for the whole data file F, and publishes these verification metadata (the keys and the MACs) to TPA. The TPA can reveal a secret key sk_τ to the cloud server and ask for a fresh keyed MAC for comparison in each audit. This is privacy-preserving as long as it is impossible to recover F in full given $MAC_{sk_\tau}(F)$ and sk_τ. However, it suffers from the following severe drawbacks: 1) the number of times a particular data file can be audited is limited by the number of secret keys that must be fixed a priori. Once all possible secret keys are exhausted, the user then has to retrieve data in full to re-compute and re-publish new MACs to TPA; 2) The TPA also has to maintain and update state between audits, i.e., keep track on the revealed MAC keys. Considering the potentially large number of audit delegations from multiple users, maintaining such states for TPA can be difficult and error prone; 3) it can only support static data, and cannot efficiently deal with dynamic data at all. However, supporting data dynamics is also of critical importance for cloud storage systems. For the reason of brevity and clarity, our main protocol will be presented based on static data. Section 3.6 will describe how to adapt our protocol for dynamic data.

HLA-based Solution. To effectively support public auditability without having to retrieve the data blocks themselves, the HLA technique [8], [10], [13] can be



TPA

1. Retrieve file tag t, verify its signature, and quit if fail;
2. Generate a random challenge $chal = \{(i, \nu_i)\}_{i \in I}$;

$\xrightarrow[\text{challenge request } chal]{\{(i,\nu_i)\}_{i\in I}}$

Cloud Server

3. Compute $\mu' = \sum_{i \in I} \nu_i m_i$, and also $\sigma = \prod_{i \in I} \sigma_i^{\nu_i}$;
4. Randomly pick $r \leftarrow \mathbb{Z}_p$, and compute $R = e(u, v)^r$ and $\gamma = h(R)$;
5. Compute $\mu = r + \gamma\mu' \mod p$;

$\xleftarrow[\text{storage correctness proof}]{\{\mu,\sigma,R\}}$

6. Compute $\gamma = h(R)$, and then verify $\{\mu, \sigma, R\}$ via Equation 1.

Fig. 2: The privacy-preserving public auditing protocol

used. HLAs, like MACs, are also some unforgeable verification metadata that authenticate the integrity of a data block. The difference is that HLAs can be aggregated. It is possible to compute an aggregated HLA which authenticates a linear combination of the individual data blocks.

At a high level, an HLA-based proof of storage system works as follow. The user still authenticates each element of $F = (m_1, \cdots, m_n)$ by a set of HLAs Φ. The cloud server stores $\{F, \Phi\}$. The TPA verifies the cloud storage by sending a random set of challenge $\{\nu_i\}$. (More precisely, F, Φ and $\{\nu_i\}$ are all vectors, so $\{\nu_i\}$ is an ordered set or $\{i, \nu_i\}$ should be sent). The cloud server then returns $\mu = \sum_i \nu_i \cdot m_i$ and an aggregated authenticator σ (both are computed from F, Φ and $\{\nu_i\}$) that is supposed to authenticate μ.

Though allowing efficient data auditing and consuming only constant bandwidth, the direct adoption of these HLA-based techniques is still not suitable for our purposes. This is because the linear combination of blocks, $\mu = \sum_i \nu_i \cdot m_i$, may potentially reveal user data information to TPA, and violates the privacy-preserving guarantee. Specifically, if an 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.

3.4 Privacy-Preserving Public Auditing Scheme

Overview. To achieve privacy-preserving public auditing, we propose to uniquely integrate the homomorphic linear authenticator with random masking technique. In our protocol, the linear combination of sampled blocks in the server's response is masked with randomness generated the server. With random masking, 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. On the other hand, the correctness validation of the block-authenticator pairs can still be carried out in a new way which will be shown shortly, even with the presence of the randomness. Our design makes use of a public key based HLA, to equip the auditing protocol with public auditability. Specifically, we use the HLA proposed in [13], which is based on the short signature scheme proposed by Boneh, Lynn and Shacham (hereinafter referred as BLS signature) [17].

Scheme Details. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map as introduced in preliminaries. Let g be a generator of $\mathbb{G}_2$. $H(\cdot)$ is a secure map-to-point hash function: $\{0,1\}^* \rightarrow \mathbb{G}_1$, which maps strings uniformly to $\mathbb{G}_1$. Another hash function $h(\cdot) : \mathbb{G}_T \rightarrow \mathbb{Z}_p$ maps group element of $\mathbb{G}_T$ uniformly to $\mathbb{Z}_p$. The proposed scheme is as follows:

`Setup` Phase: The cloud user runs `KeyGen` to generate the public and secret parameters. Specifically, the user chooses a random signing key pair (spk, ssk), a random $x \leftarrow \mathbb{Z}_p$, a random element $u \leftarrow \mathbb{G}_1$, and computes $v \leftarrow g^x$. The secret parameter is $sk = (x, ssk)$ and the public parameters are $pk = (spk, v, g, u, e(u, v))$.

Given a data file $F = (m_1, \ldots, m_n)$, the user runs `SigGen` to compute authenticator σ_i for each block m_i: $\sigma_i \leftarrow (H(W_i) \cdot u^{m_i})^x \in \mathbb{G}_1$. Here $W_i = name||i$ and $name$ is chosen by the user uniformly at random from $\mathbb{Z}_p$ as the identifier of file F. Denote the set of authenticators by $\Phi = \{\sigma_i\}_{1 \leq i \leq n}$.

The last part of `SigGen` is for ensuring the integrity of the unique file identifier $name$. One simple way to do this is to compute $t = name||\mathsf{SSig}_{ssk}(name)$ as the file tag for F, where $\mathsf{SSig}_{ssk}(name)$ is the signature on $name$ under the private key ssk. For simplicity, we assume the TPA knows the number of blocks n. The user then sends F along with the verification metadata (Φ, t) to the server and deletes them from local storage.

`Audit` Phase: The TPA first retrieves the file tag t. With respect to the mechanism we describe in the `Setup` phase, the TPA verifies the signature

$\mathsf{SSig}_{ssk}(name)$ via spk, and quits by emitting FALSE if the verification fails. Otherwise, the TPA recovers $name$.

Now it comes to the "core" part of the auditing process. To generate the challenge message for the audit "$chal$", the TPA picks a random c-element subset $I = \{s_1, \ldots, s_c\}$ of set $[1, n]$. For each element $i \in I$, the TPA also chooses a random value ν_i (of bit length that can be shorter than $|p|$, as explained in [13]). The message "$chal$" specifies the positions of the blocks that are required to be checked. The TPA sends $chal = \{(i, \nu_i)\}_{i \in I}$ to the server.

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 $r \leftarrow \mathbb{Z}_p$, and calculates $R = e(u, v)^r \in \mathbb{G}_T$. 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: $\mu = r + \gamma\mu' \mod p$, where $\gamma = h(R) \in \mathbb{Z}_p$. Meanwhile, the server also calculates an aggregated authenticator $\sigma = \prod_{i \in I} \sigma_i^{\nu_i} \in \mathbb{G}_1$. 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

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

The protocol is illustrated in Fig. 2. The correctness of the above verification equation can be elaborated as follows:

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

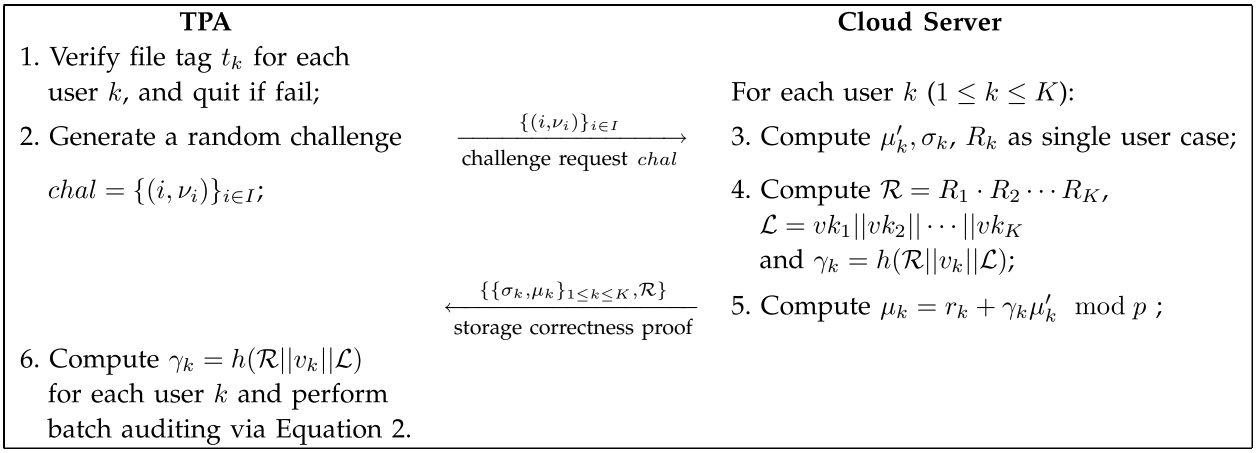
Properties of Our Protocol. It is easy to see that our protocol achieves public auditability. There is no secret keying material or states for the TPA to keep or maintain between audits, and the auditing protocol does not pose any potential online burden on users. This approach ensures the privacy of user data content during the auditing process by employing a random masking r to hide μ, a linear combination of the data blocks. 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. Storage correctness thus follows from that of the underlying protocol [13]. The security of this protocol will be formally proven in Section 4. Besides, the HLA helps achieve the constant communication overhead for server's response during the audit: the size of $\{\sigma, \mu, R\}$ is independent of the number of sampled blocks c.

Previous work [8], [10] 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 examples, if the server is missing 1% of the data F, to detect this misbehavior with probability larger than 95%, the TPA only needs to audit for $c = 300$ (up to $c = 460$ for 99%) randomly chosen blocks of F. Given the huge volume of data outsourced in the cloud, checking a portion of the data file is more affordable and practical for both the TPA and the 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.

3.5 Support for Batch Auditing

With the establishment of privacy-preserving public auditing, the TPA may concurrently handle multiple auditing upon different users' delegation. The individual auditing of these tasks for the 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 the TPA to batch these multiple tasks together and audit at one time. Keeping this natural demand in mind, we slightly modify the protocol in a single user case, and achieves the aggregation of K verification equations (for K auditing tasks) into a single one, as shown in Equation 2. As a result, a secure batch auditing protocol for simultaneous auditing of multiple tasks is obtained. The details are described as follows.

`Setup` Phase: Basically, the users just perform `Setup` independently. Suppose 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 simplicity, we assume each file F_k has the same number of n blocks. For a particular user k, denote his/her secret key as (x_k, ssk_k), and the corresponding public parameter as $(spk_k, v_k, g, u_k, e(u_k, v_k))$ where $v_k = g^{x_k}$. Similar to the single user case, each user k has already randomly chosen a different (with overwhelming probability) name $name_k \in \mathbb{Z}_p$ for his/her file F_k, and has correctly generated the corresponding file tag $t_k = name_k || \mathsf{SSig}_{ssk_k}(name_k)$. Then, each user k runs `SigGen` and computes $\sigma_{k,i}$ for block $m_{k,i}$: $\sigma_{k,i} \leftarrow (H(name_k||i) \cdot u_k^{m_{k,i}})^{x_k} = (H(W_{k,i}) \cdot u_k^{m_{k,i}})^{x_k} \in \mathbb{G}_1$



TPA

1. Verify file tag t_k for each user k, and quit if fail;
2. Generate a random challenge $chal = \{(i, \nu_i)\}_{i \in I}$;

$\xrightarrow[\text{challenge request } chal]{\{(i,\nu_i)\}_{i\in I}}$

Cloud Server

For each user k ($1 \leq k \leq K$):

3. Compute μ'_k, σ_k, R_k as single user case;
4. Compute $\mathcal{R} = R_1 \cdot R_2 \cdots R_K$, $\mathcal{L} = vk_1||vk_2||\cdots||vk_K$ and $\gamma_k = h(\mathcal{R}||v_k||\mathcal{L})$;
5. Compute $\mu_k = r_k + \gamma_k \mu'_k \mod p$;

$\xleftarrow[\text{storage correctness proof}]{\{\{\sigma_k,\mu_k\}_{1\leq k\leq K},\mathcal{R}\}}$

6. Compute $\gamma_k = h(\mathcal{R}||v_k||\mathcal{L})$ for each user k and perform batch auditing via Equation 2.

Fig. 3: The batch auditing protocol

($i \in \{1, \ldots, n\}$), where $W_{k,i} = name_k||i$. Finally, each user k sends file F_k, set of authenticators Φ_k, and tag t_k to the server and deletes them from local storage.

`Audit` Phase: TPA first retrieves and verifies file tag t_k for each user k for later auditing. If the verification fails, TPA quits by emitting FALSE; otherwise, TPA recovers $name_k$. Then 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 $k \in \{1, \ldots, K\}$, the server randomly picks $r_k \in \mathbb{Z}_p$ and computes $R_k = e(u_k, v_k)^{r_k}$. Denote $\mathcal{R} = R_1 \cdot R_2 \cdots R_K$, and $\mathcal{L} = vk_1||vk_2||\cdots||vk_K$, our protocol further requires the server to compute $\gamma_k = h(\mathcal{R}||v_k||\mathcal{L})$. Then, the randomly masked responses can be generated as follows:

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

The server then responses the TPA with $\{\{\sigma_k, \mu_k\}_{1 \leq k \leq K}, \mathcal{R}\}$.

To verify the response, the TPA can first compute $\gamma_k = h(\mathcal{R}||v_k||\mathcal{L})$ for $1 \leq k \leq K$. Next, TPA checks if the following equation holds:

$$\mathcal{R} \cdot e(\prod_{k=1}^{K} {\sigma_k}^{\gamma_k}, g) \overset{?}{=} \prod_{k=1}^{K} e((\prod_{i=s_1}^{s_c} H(W_{k,i})^{\nu_i})^{\gamma_k} \cdot u_k^{\mu_k}, v_k) \tag{2}$$

The batch protocol is illustrated in Fig. 3. Here the left-hand side (LHS) of Equation 2 expands as:

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

which is the right hand side, as required. Note that the last equality follows from Equation 1.

Efficiency Improvement. 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 relatively 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.

Identification of Invalid Responses. The verification equation (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, as we will show in Section 4. In many situations, a response collection may contain invalid responses, especially $\{\mu_k\}_{1 \leq k \leq K}$, 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 [18]. 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. TPA may now require the server to send back all the $\{R_k\}_{1 \leq k \leq K}$, as in individual auditing. In Section 4.2.2, 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.

3.6 Support for Data Dynamics

In Cloud Computing, outsourced data might not only be accessed but also updated frequently by users for various application purposes [10], [19]–

[21]. Hence, supporting data dynamics for privacy-preserving public auditing is also of paramount importance. Now we show how to build upon the existing work [10] and adapt our main scheme to support data dynamics, including block level operations of modification, deletion and insertion.

In [10], data dynamics support is achieved by replacing the index information i with m_i in the computation of block authenticators and using the classic data structure – Merkle hash tree (MHT) [22] for the underlying block sequence enforcement. As a result, the authenticator 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 aux in the MHT. Upon receiving the response, TPA should first use TR_{MHT} and aux 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 $\prod_{s_1 \leq i \leq s_c} H(W_i)^{\nu_i}$ is now replaced by $\prod_{s_1 \leq i \leq s_c} H(m_i)^{\nu_i}$. Data privacy is still preserved due to the random mask. The details of handling dynamic operations are similar to [10] and thus omitted.

3.7 Learning μ' from σ

Though our scheme prevents the TPA from directly deriving μ' from μ, it does not rule out the possibility of offline guessing attack from the TPA using valid σ from the response. Specifically, the TPA can always guess whether the stored data contains certain message $\tilde{m}$, by checking $e(\sigma, g) \overset{?}{=} e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i}) \cdot u^{\tilde{\mu}'}, v)$, where $\tilde{\mu}'$ is constructed from random coefficients chosen by the TPA in the challenge and the guessed message $\tilde{m}$. Thus, our main scheme is not semantically secure yet. However, we must note that $\tilde{\mu}'$ is chosen from $\mathbb{Z}_p$ and $|p|$ is usually larger than 160 bits in practical security settings (see Section 4.2). Therefore, the TPA has to test basically all the values of $\mathbb{Z}_p$ in order to make a successful guess. Given no background information, the success probability of such all-or-nothing guess launched by TPA can be negligible. Nonetheless, for completeness, we will give a provably zero-knowledge based public auditing scheme, which further eliminates the possibilities of above offline guessing attack. The details and corresponding proofs can be found in Appendix A.

3.8 Generalization

As mentioned before, our protocol is based on the HLA in [13]. Recently, it has been shown in [23] that HLA can be constructed by homomorphic identification protocols. One may apply the random masking technique we used to construct the corresponding zero knowledge proof for different homomorphic identification protocols. Therefore, it follows that our privacy-preserving public auditing system for secure cloud storage can be generalized based on other complexity assumptions, such as factoring [23].

4 Evaluation

4.1 Security Analysis

We evaluate the security of the proposed scheme by analyzing its fulfillment of the security guarantee described in Section 2.2, namely, the storage correctness and privacy-preserving property. We start from the single user case, where our main result is originated. Then we show the security guarantee of batch auditing for the TPA in multi-user setting.

4.1.1 Storage Correctness Guarantee

We need to prove that the cloud server cannot generate valid response for the 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: The proof consists of two steps. First, we show that there exists an extractor of μ' in the random oracle model. Once a valid response $\{\sigma, \mu'\}$ are obtained, the correctness of this statement follows from Theorem 4.2 in [13].

Now, the cloud server is treated as an adversary. The extractor controls the random oracle $h(\cdot)$ and answers the hash query issued by the cloud server. For a challenge $\gamma = h(R)$ returned by the extractor, the cloud server outputs $\{\sigma, \mu, R\}$ such that the following equation holds.

$$R \cdot e(\sigma^{\gamma}, g) = e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma} \cdot u^{\mu}, v). \tag{3}$$

Suppose that an extractor can rewind a cloud server in the protocol to the point just before the challenge $h(R)$ is given. Now the extractor sets $h(R)$ to be $\gamma^* \neq \gamma$. The cloud server outputs $\{\sigma, \mu^*, R\}$ such that the following equation holds.

$$R \cdot e(\sigma^{\gamma^*}, g) = e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma^*} \cdot u^{\mu^*}, v). \tag{4}$$

The extractor then obtains $\{\sigma, \mu' = (\mu - \mu^*)/(\gamma - \gamma^*)\}$ as a valid response of the underlying proof of storage system [13]. To see, recall that $\sigma_i = (H(W_i) \cdot u^{m_i})^x$,

TABLE 1: Notation of cryptographic operations.

$Hash^t_{\mathbb{G}_1}$	hash t values into the group $\mathbb{G}_1$.
$Mult^t_{\mathbb{G}}$	t multiplications in group $\mathbb{G}$.
$Exp^t_{\mathbb{G}}(\ell)$	t exponentiations g^{a_i}, for $g \in \mathbb{G}$, $\lvert a_i\rvert = \ell$.
$m\text{-}MultExp^t_{\mathbb{G}}(\ell)$	t m-term exponentiations $\prod_{i=1}^m g^{a_i}$.
$Pair^t_{\mathbb{G}_1,\mathbb{G}_2}$	t pairings $e(u_i, g_i)$, where $u_i \in \mathbb{G}_1$, $g_i \in \mathbb{G}_2$.
$m\text{-}MultPair^t_{\mathbb{G}_1,\mathbb{G}_2}$	t m-term pairings $\prod_{i=1}^m e(u_i, g_i)$.

divide (3) by (4), we have

$$\begin{aligned}
e(\sigma^{\gamma-\gamma^*}, g) &= e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma-\gamma^*} \cdot u^{\mu-\mu^*}, v) \\
e(\sigma^{\gamma-\gamma^*}, g) &= e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma-\gamma^*}, g^x)e(u^{\mu-\mu^*}, g^x) \\
\sigma^{\gamma-\gamma^*} &= (\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{x(\gamma-\gamma^*)} \cdot u^{x(\mu-\mu^*)} \\
(\prod_{i=s_1}^{s_c} \sigma_i^{\nu_i})^{\gamma-\gamma^*} &= (\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{x(\gamma-\gamma^*)} \cdot u^{x(\mu-\mu^*)} \\
u^{x(\mu-\mu^*)} &= (\prod_{i=s_1}^{s_c} (\sigma_i/H(W_i)^x)^{\nu_i})^{\gamma-\gamma^*} \\
u^{x(\mu-\mu^*)} &= (\prod_{i=s_1}^{s_c} (u^{xm_i})^{\nu_i})^{\gamma-\gamma^*} \\
\mu - \mu^* &= (\sum_{i=s_1}^{s_c} m_i\nu_i) \cdot (\gamma - \gamma^*) \\
(\sum_{i=s_1}^{s_c} m_i\nu_i) &= (\mu - \mu^*)/(\gamma - \gamma^*)
\end{aligned}$$

Finally, we remark that this extraction argument and the random oracle paradigm are also used in the proof of the underlying scheme [13].

□

4.1.2 *Privacy Preserving Guarantee*

We want to make sure that the TPA cannot derive users' data content from the information collected during auditing process.

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

Proof: We show the existence of a simulator that can produce a valid response even without the knowledge of μ', in the random oracle model. Now, the TPA is treated as an adversary. Given a valid σ from the cloud server, firstly, randomly pick γ, μ from $\mathbb{Z}_p$, set $R \leftarrow e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma} \cdot u^{\mu}, v)/e(\sigma^{\gamma}, g)$. Finally, backpatch $\gamma = h(R)$ since the simulator is controlling the random oracle $h(\cdot)$. We remark that this backpatching technique in the random oracle model is also used in the proof of the underlying scheme [13]. □

4.1.3 *Security Guarantee for Batch Auditing*

Now we show that our way of extending our result to a multi-user setting will not affect the aforementioned security insurance, as shown in Theorem 3.

TABLE 2: Performance under different number of sampled blocks c for high assurance ($\geq 95\%$) auditing.

	Our Scheme		[13]	
Sampled blocks c	460	300	460	300
Sever comp. time (ms)	411.00	270.20	407.66	265.87
TPA comp. time (ms)	507.79	476.81	504.25	472.55
Comm. cost (Byte)	160	40	160	40

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

Proof: The privacy-preserving guarantee in the multi-user setting is very similar to that of Theorem 2, and thus omitted here. For the storage correctness guarantee, we are going to reduce it to the single-user case. We use the forking technique as in the proof of Theorem 1. However, the verification equation for the batch audits involves K challenges from the random oracle. This time we need to ensure that all the other $K - 1$ challenges are determined before the forking of the concerned random oracle response. This can be done using the idea in [24]. As soon as the adversary issues the very first random oracle query for $\gamma_i = h(\mathcal{R}||v_i||\mathcal{L})$ for any $i \in [1, K]$, the simulator immediately determines the values $\gamma_j = h(\mathcal{R}||v_j||\mathcal{L})$ for all $j \in [1, K]$. This is possible since they are all using the same $\mathcal{R}$ and $\mathcal{L}$. Now, all but one of the γ_k's in Equation 2 are equal, so a valid response can be extracted similar to the single-user case in the proof of Theorem 1.

□

4.2 Performance Analysis

We now assess the performance of the proposed privacy-preserving public auditing schemes to show that they are indeed lightweight. We will focus on the cost of the efficiency of the privacy-preserving protocol and our 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. Our code uses 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.

4.2.1 *Cost of Privacy-Preserving Protocol*

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 challenge

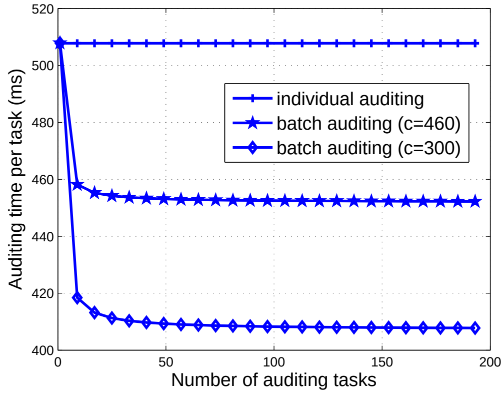


Fig. 4: Comparison on auditing time between batch and individual auditing. Per task auditing time denotes the total auditing time divided by the number of tasks. For clarity reasons, we omit the straight curve for individual auditing when c=300.

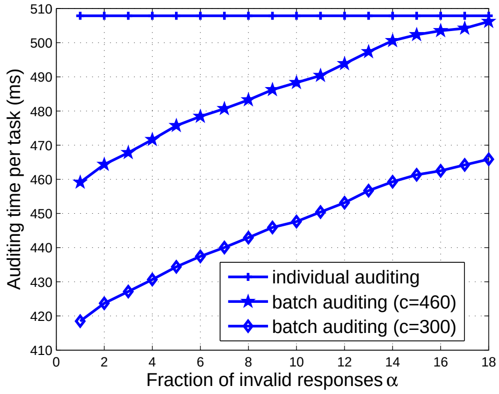


Fig. 5: Comparison on auditing time between batch and individual auditing, when α-fraction of 256 responses are invalid. Per task auditing time denotes the total auditing time divided by the number of tasks.

message $chal$ during the `Audit` phase. Under this setting, we quantify the cost introduced of the privacy-preserving auditing in terms of server computation, auditor computation as well as communication overhead.

On the server side, the generated response includes an aggregated authenticator $\sigma = \prod_{i \in I} \sigma_i^{\nu_i} \in \mathbb{G}_1$, a random factor $R = e(u, v)^r \in \mathbb{G}_T$, 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-$MultExp^1_{\mathbb{G}_1}(|\nu_i|)$, $Exp^1_{\mathbb{G}_T}(|p|)$, and $Hash^1_{\mathbb{Z}_p} + Add^c_{\mathbb{Z}_p} + Mult^{c+1}_{\mathbb{Z}_p}$, respectively. Compared to the existing HLA-based solution for ensuring remote data integrity [13][1], the extra cost for protecting the user privacy, resulted from the random mask R, is only a constant: $Exp^1_{\mathbb{G}_T}(|p|) + Mult^1_{\mathbb{Z}_p} + Hash^1_{\mathbb{Z}_p} + Add^1_{\mathbb{Z}_p}$, which has nothing to do with the number of sampled blocks c. When c is set to be 300 to 460 for high assurance of auditing, as discussed in Section 3.4, 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-$MultExp^1_{\mathbb{G}_1}(|\nu_i|) + Hash^c_{\mathbb{G}_1} + Mult^1_{\mathbb{G}_1} + Mult^1_{\mathbb{G}_T} + Exp^3_{\mathbb{G}_1}(|p|) + Pair^2_{\mathbb{G}_1, \mathbb{G}_2}$, among which only $Hash^1_{\mathbb{Z}_p} + Exp^2_{\mathbb{G}_1}(|p|) + Mult^1_{\mathbb{G}_T}$ 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 [13]. It can be shown that the performance of our scheme is almost the same as that of [13], even if our scheme supports privacy-preserving guarantee while [13] does not. For the extra communication cost of our scheme opposing to [13], the server's response $\{\sigma, R, \mu\}$ contains an additional random element R, which is a group element of $\mathbb{G}_T$ and has the size close to 960 bits.

4.2.2 Batch Auditing Efficiency

Discussion in Section 3.5 gives an asymptotic efficiency analysis on the batch auditing, by considering only the total number of pairing operations. However, on the practical side, there are additional less expensive operations required for batching, such as modular exponentiations and multiplications. Meanwhile, the different sampling strategies, i.e., different number 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 timed batch auditing test, 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 settings $c = 300$ and $c = 460$, the average per task auditing time, which is computed by dividing total auditing time by the number of tasks, is given in Fig. 4 for both batch and individual auditing. It can be shown that compared to individual auditing, batch auditing indeed helps reducing 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.

4.2.3 Sorting out Invalid Responses

Now we use experiment to justify the efficiency of our recursive binary search approach for the TPA to

[1] We refer readers to [13] for a detailed description of their HLA-based solution.

sort out the invalid responses when batch auditing fails, as discussed in Section 3.5. This experiment is tightly pertained to the work in [18], which evaluates the batch verification efficiency of various short signatures.

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. 5. 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 the random distribution of invalid responses within the collection is nearly the worst-case for batch auditing. If invalid responses are grouped together, it is possible to achieve even better results.

5 Related Work

Ateniese *et al.* [8] 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 linear 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.* [11] 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 fixed a 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. Dodis *et al.* [25] give a study on different variants of PoR with private auditability. Shacham *et al.* [13] design an improved PoR scheme built from BLS signatures [17] with full proofs of security in the security model defined in [11]. Similar to the construction in [8], they use publicly verifiable homomorphic linear authenticators that are built from provably secure BLS signatures. Based on the elegant BLS construction, a compact and public verifiable scheme is obtained. Again, their approach does not support privacy-preserving auditing for the same reason as [8]. Shah *et al.* [9], [14] 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 online burden to users when the keyed hashes are used up.

In other related work, Ateniese *et al.* [19] propose a partially dynamic version of the prior PDP scheme, using only symmetric key cryptography but with a bounded number of audits. In [20], Wang *et al.* consider a similar support for partial dynamic data storage in a distributed scenario with additional feature of data error localization. In a subsequent work, Wang *et al.* [10] propose to combine BLS-based HLA with MHT to support both public auditability and full data dynamics. Almost simultaneously, Erway *et al.* [21] developed a skip lists based scheme to enable provable data possession with full dynamics support. However, the verification in these two protocols requires the linear combination of sampled blocks just as [8], [13], and thus does not support privacy-preserving auditing. While all the 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. More importantly, none of these schemes consider batch auditing, which can greatly reduce the computation cost on the TPA when coping with a 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. We utilize the homomorphic linear authenticator and random masking to guarantee that the 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 the TPA can perform multiple auditing tasks in a batch manner for better efficiency. Extensive analysis shows that our schemes are provably secure and highly efficient.

Acknowledgments

This work was supported in part by the US National Science Foundation under grant CNS-0831963, CNS-0626601, CNS-0716306, and CNS-0831628.

Appendix A
Zero Knowledge Public Auditing

Here we present a public auditing scheme with provably zero knowledge leakage. The `setup` phase is similar to our main scheme presented in Section 3.4. The secret parameters are $sk = (x, ssk)$ and the public parameters are $pk = (spk, v, g, u, e(u, v), g_1)$, where $g_1 \in \mathbb{G}_1$ is an additional public group element.

In the `audit` phase, upon receiving challenge $chal = \{(i, \nu_i)\}_{i \in I}$, the server chooses three random elements $r_m, r_\sigma, \rho \leftarrow \mathbb{Z}_p$, and calculates $R = e(g_1, g)^{r_\sigma} \cdot e(u, v)^{r_m} \in \mathbb{G}_T$ and $\gamma = h(R) \in \mathbb{Z}_p$. Let μ' denote the linear combination of sampled blocks $\mu' = \sum_{i \in I} \nu_i m_i$, and σ denote the aggregated authenticator $\sigma = \prod_{i \in I} \sigma_i^{\nu_i} \in \mathbb{G}_1$. To make the auditing scheme with zero knowledge leakage, the server has to blind both μ' and σ. Specifically, the server computes: $\mu = r_m + \gamma\mu' \mod p$, and $\Sigma = \sigma \cdot g_1^\rho$. It then sends $\{\varsigma, \mu, \Sigma, R\}$ as the response proof of storage correctness to the TPA, where $\varsigma = r_\sigma + \gamma\rho \mod p$. 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

$$R \cdot e(\Sigma^\gamma, g) \stackrel{?}{=} e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^\mu, v) \cdot e(g_1, g)^\varsigma \quad (5)$$

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

$$\begin{aligned} R \cdot e(\Sigma^\gamma, g) &= e(g_1, g)^{r_\sigma} \cdot e(u, v)^{r_m} \cdot e((\sigma \cdot g_1^\rho)^\gamma, g) \\ &= e(g_1, g)^{r_\sigma} \cdot e(u, v)^{r_m} \cdot e((\sigma^\gamma, g) \cdot e(g_1^{\rho\gamma}, g) \\ &= e(u, v)^{r_m} \cdot e((\sigma^\gamma, g) \cdot e(g_1, g)^{r_\sigma + \rho\gamma} \\ &= e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^\mu, v) \cdot e(g_1, g)^\varsigma \end{aligned}$$

The last equality follows from the elaboration of Equation 1.

Theorem 4: The above auditing protocol achieves zero-knowledge information leakage to the TPA, and it also ensures the storage correctness guarantee.

Proof: Zero-knowledge is easy to see. Randomly pick γ, μ, ς from $\mathbb{Z}_p$ and Σ from $\mathbb{G}_1$, set $R \leftarrow e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^\gamma \cdot u^\mu, v) \cdot e(g_1, g)^\varsigma / e(\Sigma^\gamma, g)$ and backpatch $\gamma = h(R)$. For proof of storage correctness, we can extract ρ similar to the extraction of μ' as in the proof of Theorem 1. With ρ, σ can be recovered from Σ. To conclude, a valid pair of σ and μ' can be extracted. □

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.

[3] M. Arrington, "Gmail disaster: Reports of mass email deletions," Online at http://www.techcrunch.com/2006/12/28/gmail-disasterreports-of-mass-email-deletions/, December 2006.

[4] J. Kincaid, "MediaMax/TheLinkup Closes Its Doors," Online at http://www.techcrunch.com/2008/07/10/mediamaxthelinkup-closes-its-doors/, July 2008.

[5] Amazon.com, "Amazon s3 availability event: July 20, 2008," Online at http://status.aws.amazon.com/s3-20080720.html, 2008.

[6] S. Wilson, "Appengine outage," Online at http://www.cio-weblog.com/50226711/appengine_outage.php, June 2008.

[7] 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.

[8] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. of CCS'07*, Alexandria, VA, October 2007, pp. 598–609.

[9] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," Cryptology ePrint Archive, Report 2008/186, 2008.

[10] 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, volume 5789 of LNCS*. Springer-Verlag, Sep. 2009, pp. 355–370.

[11] 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.

[12] Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing," 2009, http://www.cloudsecurityalliance.org.

[13] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. of Asiacrypt 2008*, vol. 5350, Dec 2008, pp. 90–107.

[14] 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.

[15] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPPA)," Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.

[16] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained access control in cloud computing," in *Proc. of IEEE INFOCOM'10*, San Diego, CA, USA, March 2010.

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

[18] 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.

[19] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. of SecureComm'08*, 2008, pp. 1–10.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Proc. of IWQoS'09*, July 2009, pp. 1–9.

[21] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. of CCS'09*, 2009, pp. 213–222.

[22] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. of IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 1980.

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

[24] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *ACM Conference on Computer and Communications Security*, 2006, pp. 390–399.

[25] Y. Dodis, S. P. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *TCC*, 2009, pp. 109–127.