Preserving Integrity of Data and Public Auditing for Data Storage Security in Cloud Computing

Abhinandan P Shirahatti, P S Khanagoudar

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) he third party auditing process should bring in no new vulnerabilities towards user data privacy. In this paper, we utilize and uniquely combine the public key based homomorphic authenticator with random masking to achieve the privacy-preserving public cloud data auditing system, which meets all above requirements. 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.**

Index Terms—**Data storage, preserving data integrity, public auditability, cloud computing, batch verification, zero-knowledge**

I. 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 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]–[5]. 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 [6]–[8]. 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 [8], [9]. 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

Abhinandan P Shirahatti is with Department of CSE, VTU Jnana Sangama, Belgaum-590014, India.

P S Khanagoudar is with GIT College of Engineering and Technology, Belgaum-590016, India

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 [7]. 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 [6], [8], [10], [11]. 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 [6], [8], [10] 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 III-C. 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 [12]. Moreover, there are legal regulations, such as the US Health Insurance Portability and Accountability Act (HIPAA) [13], further demanding the outsourced data not to be leaked to external parties [7]. 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 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 [6], [8], [10], 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 masking, 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

II. PROBLEM STATEMENT

A. 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 as [14] does. Namely, in most of time it behaves properly and does not deviate from the prescribed protocol execution. However, during 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.

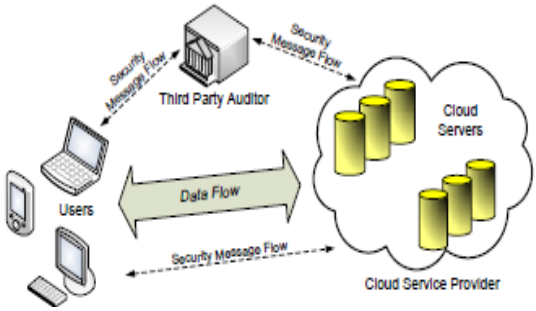


Fig. 1: The architecture of cloud data storage services

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.

B. 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 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.

III. 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. After introducing notations and brief preliminaries, 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.

A. Notation and Preliminaries

• F – the data file to be outsourced, denoted as a sequence of n blocks m1, . . . , mn ∈ Zp for some large prime p.

• fkey(·) – pseudorandom function (PRF), defined as: {0, 1}∗ × key → Zp.

• πkey(·) – pseudorandom permutation (PRP), defined as:{0, 1}log2(n) × key → {0, 1}log2(n).

• MACkey(·) – message authentication code (MAC) function, defined as: {0, 1}∗ × key → {0, 1}l.

• H(·), h(·) – map-to-point hash functions, defined as: {0, 1}∗ → G, where G is some group.

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

Bilinear Map Let G1, G2 and GT be multiplicative cyclic groups of prime order p. Let g1 and g2 be generators of G1 and G2, respectively. A bilinear map is a map e : G1 ×G2 → GT with the following properties [15]: 1) Computable: there exists an efficiently computable algorithm for computing e; 2) Bilinear: for all u ∈ G1, v ∈ G2 and a, b ∈ Zp, e(ua, vb) = e(u, v)ab; 3) Non-degenerate: e(g1, g2) = 1; 4) for any u1, u2 ∈ G1, v ∈ G2, e(u1u2, v) = e(u1, v) · e(u2, v).

B. Definitions and Framework

We follow a similar definition of previously proposed schemes in the context of remote data integrity checking [9], [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 digital signatures. 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. 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 by executing GenProof using *F* and its verification metadata as inputs. The TPA The TPA then verifies the response via Verify Proof. Our framework assumes the TPA is stateless, i.e., TPA does not need to maintain and update state between audits, which is a desirable property especially in the public auditing system [13]. Note that it is easy to extend the framework above to capture a stateful auditing system, essentially by splitting 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-resilience, he can first redundantly

encodes the data file and then uses our system with the data that has error-correcting codes integrated1.

C. *The Basic Schemes*

This is privacy-preserving as long as it is impossible 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 also shows that the auditing problem is still not easy to solve even if 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 $\{sk_\tau\}1\le\tau\le s$, pre computes s (deterministic) MACs, $\{MAC_{sk_\tau}(F)\}1\le\tau\le s$ 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 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 [9], [13], [8] can be used.

TPA		Cloud Server
1. Retrieve file tag t, verify its signature, and quit if fail;		
2. Generate a random challenge $chal = \{(i, \nu_i)\}_{i\in I}$;	$\xrightarrow[\text{challenge request } chal]{\{(i,\nu_i)\}_{i\in I}}$	3. Compute $\mu' = \sum_{i\in I} \nu_i m_i$, and $\sigma = \prod_{i\in I} \sigma_i^{\nu_i}$; 4. Randomly pick $r \leftarrow \mathbb{Z}_p$, and $R = e(u,v)^r$ and $\gamma = h(R)$;
	$\xleftarrow[\text{storage correctness proof}]{\{\mu,\sigma,R\}}$	5. Compute $\mu = r + \gamma\mu' \mod p$;
6. Compute $\gamma = h(R)$, and then verify $\{\mu, \sigma, R\}$ via Equation 1.		

TABLE.1: Privacy preserving Public auditing protocol

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 = {mi} by a set of HLAs Φ. The TPA verifies the cloud storage by sending a random set of challenge {vi}. The cloud server then returns μ = _i vi · mi and its aggregated authenticator σ computed from Φ. 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, μ = _i vi · mi, may potentially reveal user data information to TPA, and violates the privacy-preserving guarantee. Specifically, by challenging the same set of c block m1,m2, . . . , mc using c different sets of random coefficients {vi}, TPA can accumulate c different linear

combinations μ1, . . . , μc. With {μi} and {vi}, TPA can derive the user's data m1,m2, . . . , mc by simply solving a system of linear equations.

D. *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 by 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 (a) [19]Scheme Details. Let G1, G2 and GT be multiplicative cyclic groups of prime order p, and $e : G1 \times G2 \rightarrow GT$ be a bilinear map as introduced in preliminaries. Let g be a generator of G2. H(·) is a secure map-to-point hash function: $\{0, 1\}^* \rightarrow G1$, which maps strings uniformly to G1. Another hash function $h(\cdot) : GT \rightarrow Zp$ maps group element of GT uniformly to Zp. Our 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 ← Zp, a random element u ← G1, and computes v ← gx. 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 = {mi}, the user runs SigGento compute authenticator σi ← (H(Wi) · umi)x ∈ G1 for each i. Here Wi = name‖i and name is chosen by the user uniformly at random from Zp as the identifier of file F. Denote the set of authenticators by Φ = {σi}1≤i≤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‖SSigssk(name) as the file tag for F, where SSigssk(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 SSigssk(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 = {s1, . . . , sc} of set [1, n]. For each element i ∈ I, the TPA also chooses a random value vi (of bit length that can be shorter than |p|, as explained in [13]). The message "chal" specifies the positions of the blocks required to be checked. The TPA sends chal = {(i, vi)}i∈I to the server. Upon receiving challenge chal = {(i, vi)}i∈I , the server runs GenProof to generate a response proof of data storage correctness. Specifically, the server chooses a random element r ← Zp, and calculates R = e(u, v)r ∈ GT . Let μ_ denote the linear combination of sampled blocks specified in chal: μ_ = _i∈I vimi. To blind μ_ with r, the server computes: μ = r + γμ_ mod p, where γ = h(R) ∈ Zp. Meanwhile, the server also calculates

an aggregated authenticator σ = _i∈I σvii ∈ G1. It then sends {μ, σ,R} as the response proof of storage correctness to the TPA. With the response, the TPA runs VerifyProof to validate it by first computing γ = 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 Table 1. The correctness of the above verification equation is elaborated as follows:

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

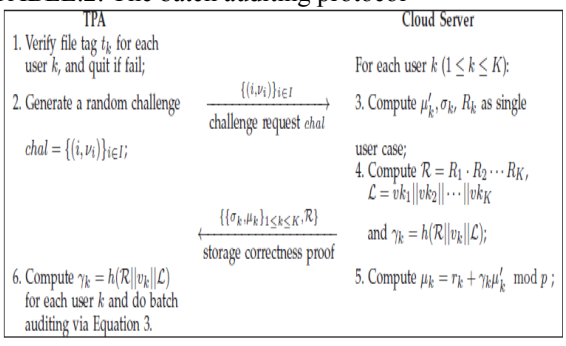
Properties of Our Protocol. It is easy to see that our protocol achieves public auditability. There is no secret keying material or states for the TPA to keep or maintain between audits, and the auditing protocol does not pose any potential online burden on users. This approach ensures the privacy of 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 γ = 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 {σ, μ,R} is independent of the number of sampled blocks c. Previous work [9], [8] showed that if the server I 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). In particular, if t fraction of data is corrupted, then random sampling c blocks would reach the detection probability P = 1−(1−t)c. Here every block is chosen uniformly at random. When t = 1% of the data F, the TPA only needs to audit for c = 300 or 460 randomly chosen blocks of F to detect this misbehavior with probability larger than 95% and 99% 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 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. For some cloud storage providers, it is possible that certain information dispersal algorithms (IDA) may be used to fragment and geographically distribute the user's outsourced data for increased availability. We note that these cloud side operations would not affect the behavior of our proposed mechanism, as long as the IDA is systematic, i.e., it preserves user's data in its original form after encoding with redundancy. This is because from user's perspective, as long as there is a complete yet unchanged copy of his outsourced data in cloud, the precomputed verification metadata (Φ, t) will remain valid. As a result, those metadata can still be utilized in our auditing mechanism to guarantee the correctness of user's outsourced cloud data.

Storage and Communication Tradeoff As described above, each block is accompanied by an authenticator of equal size of |p| bits. This gives about 2× storage overhead on server. However, as noted in [13], we can introduce a parameter s in the authenticator construction to adjust this storage overhead, in the cost of communication overhead in the auditing protocol between TPA and cloud server. In particular, we assume each block mi consists of s sectors {mij} with 1 ≤ j ≤ s, where mij ∈ Zp. The public parameter pk is now (spk, v, g, {uj}, {e(uj, v)}), 1 ≤ j ≤ s, where u1, u2, . . . , us are randomly chosen from G1. The authenticator σi of mi is constructed as: σi ← (H(Wi) · _sj=1 umijj)x ∈ G1.Because we now have one authenticator per block (or peers sectors), we reduce the storage overhead to (1+1/s)×.To respond to the auditing challenge chal ={(i, νi)}i∈I , for 1 ≤ j ≤ s, the cloud server chooses a random elements rj ← Zp, and calculates Rj = e(u, v)rj ∈GT . Then the server blinds each μ_j = _i∈I νimij with rj , and derives the blinded μj = rj + γμ_ j mod p,where γ = h(R1||R2|| . . . ||Rs) ∈ Zp. The aggregated authenticator is still computed as before. It then sends{σ, {μj,Rj}1≤j≤s} as the proof response to TPA. With the proof, TPA first computes γ = h(R1||R2|| . . . ||Rs), and then checks the following verification:

$$R_1 \cdots R_s \cdot e(\sigma^{\gamma}, g) \stackrel{?}{=} e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma} \cdot \prod_{j=1}^{s} u_j^{\mu_j}, v) \quad (2)$$

The correctness elaboration is similar to Eq. (1) and thus omitted. The overall storage overhead is reduced to (1 + 1/s)×, but the proof size now increases roughly s× due to the additional s element pairs {μj,Rj}1≤j≤s that the cloud server has to return. For presentation simplicity, we continue to choose s = 1 in our following scheme description. We will present some experiment results results with larger choice of s in Section 4.

TABLE.2: The batch auditing protocol



E. 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 3. 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 Fk = (mk,1, . . . , mk,n) to be outsourced to the cloud server, where k ∈ {1, . . . , }.For simplicity, we assume each file Fk has the same number of n blocks. For a particular user k, denote his/her secret key as (xk, sskk), and the corresponding public parameter as (spkk, vk, g, uk, e(uk, vk)) where vk = gxk . Similar to the single user case, each user k has already randomly chosen a different (with overwhelming probability) name namek ∈ Zp for his/her file Fk, and has correctly generated the corresponding file tag tk = namek||SSigsskk (namek). Then, each user k runs SigGen and computes σk,i for block mk,i: σk,i ← (H(name k ||i) · umk,I k)xk = (H(Wk,i) · umk,I k)xk ∈ G1 (i ∈ {1, . . . , n}), where Wk,i = namek||i. Finally, each user k sends file Fk, set of authenticators Φk, and tag tk to the server and deletes them from local storage.

Audit Phase: TPA first retrieves and verifies file tag tk for each user k for later auditing. If the verification fails, TPA quits by emitting FALSE. Otherwise, TPA recovers namek and sends the audit challenge chal = {(i, vi)}i∈I to the server for auditing data files of all K users. Upon receiving chal, for each user k ∈ {1, . . . , K}, the server randomly picks rk ∈ Zp and computes Rk = e(uk, vk)rk . Denote R = R1 · R2 · · · RK, and L = vk1||vk2|| · · · ||vkK, our protocol further requires the server to compute γk = h(R||vk||L). Then, the randomly masked responses can be generated as follows:

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

The server then responds with {{σk, μk}1≤k≤K,R}. To verify the response, the TPA can first compute γk = h(R||vk||L) for 1 ≤ k ≤ K. Next, TPA checks if the following equation holds:

$$\mathcal{R} \cdot e(\prod_{k=1}^{K} \sigma_k^{\gamma_k}, g) \stackrel{?}{=} \prod_{k=1}^{K} e((\prod_{i=s_1}^{s_c} H(W_{k,i})^{\nu_i})^{\gamma_k} \cdot u_k^{\mu_k}, v_k) \quad (3)$$

The batch protocol is illustrated in able 2. Here the left-hand side (LHS) of Equation 3 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 3, 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, which saves a considerable amount of auditing time.

Identification of Invalid Responses. The verification equation (Equation 3) 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 {μ k}1≤k≤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 [20]. Specifically, if the batch auditing fails, we can simply divide the collection of responses into two halves, and repeat the auditing on halves via Equation 3. TPA may now require the server to send back all the {R k}1≤k≤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 20% of responses are invalid, batch auditing still performs faster than individual verification

F. Support for Data Dynamics

In Cloud Computing, outsourced data might not only be accessed but also updated frequently by users for various application purposes [21], [8], [22], [23]. Hence, supporting data dynamics for privacy-preserving public auditing is also of paramount importance. Now we show how to build upon the existing work [8] and adapt our main scheme to support data dynamics, including block level operations of modification, deletion and insertion. In [8], data dynamics support is achieved by replacing the index information i with mi in the computation of block authenticators and using the classic data structure – Merkle hash tree (MHT) [24] for the underlying block sequence enforcement. As a result, the authenticator for each block is changed to σi = (H(mi) · umi)x. We can adopt this technique in our design to achieve privacy-preserving public auditing with support of data dynamics. Specifically, in the Setup phase, the user has to generate and send the tree root TRMHT to TPA as additional metadata, where the leaf nodes of MHT are values of H(mi). In the Audit phase, besides {μ, σ,R}, the server's response should also include {H(mi)}i∈I and their corresponding auxiliary authentication information aux in the MHT. Upon receiving the response, TPA should first use TRMHT and aux to authenticate {H(mi)}i∈I computed by the server. Once {H(mi)}i∈I are authenticated, TPA can then perform the auditing on {μ, σ,R, {H(mi)}i∈I} via Equation 1, where _s1≤i≤sc H(Wi)vi is now replaced by _s1≤i≤sc H(mi)vi . All these changes does not interfere with the proposed random masking technique, so data privacy is still preserved. To support data dynamics, each data update would require the user to generate a new tree root TRMHT , which is later sent to TPA as the new metadata for storage auditing task. The details of handling dynamic operations are similar to [8] and thus omitted.

G. Application to Version Control System

The above scheme allows TPA to always keep the new tree root for auditing the updated data file. But it is worth noting that our mechanism can be easily extended to work with version control system, where both current and previous versions of the data file F and the corresponding authenticators are stored and need to be audited on demand. One possible way is to require TPA to keep tracks of both the current and previous tree roots generated by the user, denoted as {TR1 MHT , TR2 MHT , . . . , TRV MHT}. Here V is the number of file versions and TRV MHT is the root related to the most current version of the data file F. Then, whenever an designated version v (1 ≤ v ≤ V) of data file is to be audited, the TPA just uses the corresponding TRv MHT to perform the auditing. The cloud server should also keep track of all the versions of data file F and their authenticators, in order to correctly answer the auditing request from TPA. Note that cloud server does not need to replicate every block of data file in every version, as many of them are the same after updates. However, how to efficiently manage such block storage in cloud is not within the scope of our paper

H. Generalization

As mentioned before, our protocol is based on the HLA in [13]. It has been shown in [25] that HLA can be constructed by homomorphic identification protocols. One may apply the masking technique we used to construct the corresponding zero knowledge proof for different homomorphic identification protocols. Therefore, our privacy-preserving public auditing system for secure cloud storage can be generalized based on other complexity assumptions, such as factoring [25].

IV. Evaluation

A. 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.

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, it must indeed possess the specified data intact as it is.

Proof: We show that there exists an extractor of μ_ in the random oracle model. With valid {σ, μ_}, our theorem follows from Theorem 4.2 in [13]. The extractor controls the random oracle h(·) and answers the hash query issued by the cloud server, which is treated as an adversary here. For a challenge γ = h(R) returned by the extractor, the cloud server outputs {σ, μ, R} such that the following equation holds.

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

Suppose that our extractor can rewind a cloud server in the execution of the protocol to the point just before the challenge h(R) is given. Now the extractor sets h(R) to be γ* _= γ. The

cloud server outputs {σ, μ*,R} such that the following equation holds.

$$R \cdot e(\sigma^{\gamma^*}, g) = e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma^*} \cdot u^{\mu^*}, v). \qquad (5)$$

The extractor then obtains {σ, μ_ = (μ – μ*)/ (γ – γ*)} as a valid response of the underlying proof of storage system [13]. To see why, recall that σi = (H (Wi) ⋅ umi)x. If we divide (4) by (5), 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].

2) Privacy Preserving Guarantee

The below theorem shows that TPA cannot derive users' data from the information collected during auditing.

Theorem 2: From the server's response {σ, μ, 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 Zp, set R ←e((_sc i=s1

H(Wi)νi)γ ⋅ uμ, v)/e(σγ, g). Finally, back patch γ = h(R) since the simulator is controlling the random oracle h(⋅). We remark that this back patching technique in the random oracle model is also used in the proof of the underlying scheme [13]

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.

$Hash^t_{\mathbb{G}_1}$	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 $\prod_{i=1}^{m} e(u_i, g_i)$.

TABLE.3:Notation of cryptographic operations

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 [26]. As soon as the adversary issues the very first random oracle query for γi = h(R‖vi‖L)for any i ∈ [1,K], the simulator immediately determines the values γj = h(R‖vj‖L) for all j ∈ [1,K]. This is possible since they are all using the same R and L. Now, all but one of the γk's in Equation 3 are equal, so a valid response can be extracted similar to the single-user case in the proof of Theorem 1.

B. Performance Analysis

We now report some performance results of our experiments. We consider our auditing mechanism happens between a dedicated TPA and some cloud storage node, where user's data is outsourced to. In our experiment, the TPA/user side process is implemented on a workstation 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. The cloud server side process is implemented on Amazon Elastic Computing Cloud (EC2) with a large instance type [27], which has 4 EC2 Compute Units, 7.5 GB memory, and 850 GB instance storage. The randomly generated test data is of 1 GB size. All algorithms are implemented using C language. Our code uses the Pairing-Based Cryptography (PBC) library version 0.4.21. The elliptic curve utilized in the experiment is an 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 |vi| = 80 and |p| = 160. All experimental results represent the mean of 20 trials. Because the cloud is a pay-per-use model, users have to pay both the storage cost and the bandwidth cost (for data transfer) when using the cloud storage auditing. Thus, when implementing our mechanism, we have to take into consideration both factors. In particular, we conducts the experiment with two different sets of storage/ communication tradeoff parameter s as introduced in Section 3.4. When s = 1, the mechanism incurs extra storage cost as large as the data itself, but only takes very small auditing bandwidth cost. Such a mechanism can be adopted when the auditing has to happen very frequently (e.g., checking the storage correctness every few minutes [21]), because the resulting data transfer charge could be dominant in the pay-per-use-model. On the other hand, we also choose a properly larger s = 10, which reduces the extra storage cost to only 10% of the original data but increases the bandwidth cost roughly 10 times larger than the choice of s = 1. Such a case is relatively more desirable if the auditing does not need to happen frequently. In short, users can flexibly choose the storage/communication tradeoff parameter s for their different system application scenarios.

On our not-so-powerful workstation, the measurement shows that the user setup phase (i.e., generating authenticators) achieves a throughput of around 9.0 KB/s and 17.2 KB/s when

s = 1 and s = 10 respectively. These results are not very fast due to the expensive modular exponentiation operations for each 20 byte block sector in the authenticator computation. (See [28] for some similar experimental results.) Note that for each data file to be outsourced, such setup phase happens once only. Further, since the authenticator generation on each block is independent, these one-time operations can be easily parallelized by using multithreading technique on the modern multi-core systems. Therefore, various optimization techniques can be applied to speed up the user side setup phase. As our paper focuses on privacy preserving storage auditing performance, in the following, we will primarily assess the performance of the proposed auditing schemes on both TPA side and cloud server side, and show they are indeed lightweight.

s = 1	Our Scheme		[13]	
Sampled blocks c	460	300	460	300
Sever comp. time (ms)	335.17	219.27	333.23	217.33
TPA comp. time (ms)	530.60	357.53	526.77	353.70
Comm. cost (Byte)	160	160	40	40
s = 10	Our Scheme		[13]	
Sampled blocks c	460	300	460	300
Sever comp. time (ms)	361.56	242.29	342.91	223.64
TPA comp. time (ms)	547.39	374.32	543.35	370.29
Comm. cost (Byte)	1420	1420	220	220

TABLE 4: Performance under different number of sampled blocks c for high assurance (≥ 95%) auditing

We will focus on the cost of the privacy-preserving protocol and our proposed batch auditing technique. 4.2.1 Cost of Privacy-Preserving Protocol We begin by estimating the cost in terms of basic cryptographic operations (refer to Table 3 for notations). Suppose there are c random blocks specified in the challenge message chal during the Audit phase. Under this setting, we quantify the cost introduced by the privacy preserving auditing in terms of server computation, auditor computation as well as communication overhead. Since the difference for choices on s has been discussed previously, in the following privacy-preserving cost analysis we only give the atomic operation analysis for the case s = 1 for simplicity. The analysis for the case of s = 10 follows similarly and is thus omitted. On the server side, the generated response includes an aggregated authenticator σ = _i∈ I σvii∈ G1, a random factor R = e(u, v)r ∈ GT , and a blinded linear combination of sampled blocks μ = γ_i∈ I νimi + r ∈ Zp, where γ = h(R) ∈ Zp. The corresponding computation cost is c-MultExp1 G1 (|vi|), Exp1 GT (|p|), and Hash1 Zp + Addc Zp + Multc+1 Zp , respectively. Compared to the existing HLA based solution for ensuring remote data integrity [13], the extra cost resulted from the random mask R is only a constant: Exp1 GT (|p|)+Mult1 Zp +Hash1 Zp +Add1 Zp , 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 on the server side for privacy-preserving guarantee would be negligible against the total server computation for response generation. Similarly, on the auditor side, upon receiving the response {σ,R, μ}, the corresponding computation cost for response validation is Hash1 Zp +c-MultExp1 G1(|vi|)+ Hashc G1 + Mult1G1+ Mult1 GT + Exp3G1(|p|) + Pair2G1,G2 ,among which only Hash1 Zp+ Exp2 G1 (|p|) + Mult1 GT 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 4 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 when compared with [13], the server's response {σ,R, μ} contains an additional random element R, which is a group element of GT and has the size close to 960 bits.

1) 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. Thus, whether the benefits of removing pairings significantly outweighs these additional operations remains 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

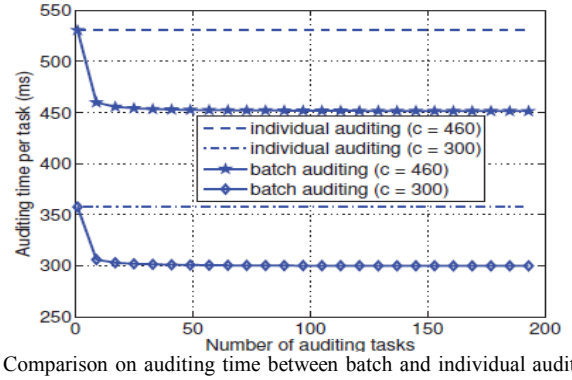


Fig. 2: Comparison on auditing time between batch and individual auditing: Per task auditing time denotes the total auditing time divided by the number of tasks.

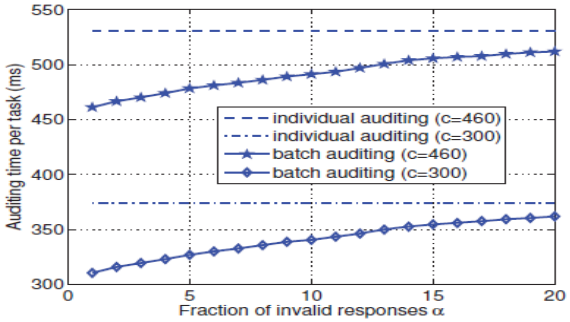


Fig. 3: Comparison on auditing time between batch and individual auditing, when α-fraction of 256 responses areinvalid: Per task auditing time denotes the total auditing time divided by the number of tasks. of 8.

Note that we only focus on the choice of s = 1 here, from which similar performance results can be directly obtained for the choice of s = 10. The performance of the corresponding non-batched (individual) auditing is provided as a baseline for the measurement. Following the same 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. 2 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 15% of per-task auditing time is saved.

2) Sorting out Invalid Responses

Now we use experiment to justify the efficiency of our recursive binary search approach for the TPA to sort out the invalid responses for negative batch auditing result, as discussed in Section 3.5. This experiment is tightly pertained to the work in [20], which evaluates the batch verification of various short signatures. The feasibility of the recursive approach is evaluated under the choice of s = 1, which is consistent with the experiment settings in Section 4.2.2. We do not duplicate evaluation of the recursive binary search methodology for s = 10, because similar results can be easily deduced from the choice of s = 1. 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 20%, 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 when the number of invalid responses exceeds 18% 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, even better results can be expected.

V. ZERO KNOWLEDGE PUBLIC AUDITING

Though our scheme prevents the TPA from directly deriving μ_ from μ, it does not rule out the possibility of offline guessing threat by TPA using valid σ from the response. Specifically, the TPA can always guess whether μ_ ?= ˜μ_, by checking e(σ, g) ?=e((_sc i=s1

H (Wi)vi) · u ˜μ_, v), where ˜μ_ is constructed from random coefficients chosen by the TPA in the challenge and the guessed message {m˜ i}s1≤i≤sc . However, we must note that ˜μ_ is chosen from Zp and |p| is usually larger than 160 bits in practical security settings (see Section 4.2). Given no background information, the success of this all-or-nothing guess on μ_ launched by TPA over such a large space Zp can be very difficult. Besides, because TPA must at least make c successful guesses on the same set of blocks to derive {mi}s1≤i≤sc from the system of c linear equations, we can specify c to be large enough in the protocol (e.g., as discussed in Section 3.4, a strict choice of c should be at least larger than 460), which can significantly decrease the TPA's successful guessing probability. In addition, we can also restrict the number of re-auditing on exactly the same set of blocks (e.g., to limit the repeated auditing times on exactly the same set of blocks to be always less than c). In this way, TPA can be kept from accumulating successful guesses on μ_ for the same set of blocks, which further diminishes the chance for TPA to solve for {mi}s1≤i≤sc . In short, by appropriate choices of parameter c and group size Zp, we can effectively defeat such potential offline guessing threat. Nevertheless, we present a public auditing scheme with provably zero knowledge leakage. This scheme can completely eliminate the possibilities of above offline guessing attack, but at the cost of a little higher communication and computation overhead. The setup phase is similar to our main scheme presented in Section 3.4. The12 secret parameters are sk = (x, ssk) and the public parameters is an additional public group element. In the audit are pk = (spk, v, g, u, e(u, v), g1), where g1 ∈ G1 phase, upon receiving challenge chal = {(i, vi)}i∈I , the server chooses three random elements rm, rσ, ρ ← Zp, and calculates R = e(g1, g)rσ · e(u, v)rm ∈ GT and γ = h(R) ∈ Zp. Let μ_ denote the linear combination of sampled blocks μ_ = _i∈I vimi, and σ denote the aggregated authenticator σ = _i∈I σvii ∈ G1. To ensure the auditing leaks zero knowledge, the server has to blind both μ_ and σ. Specifically, the server computes: μ = rm + γμ_ mod p, and Σ = σ · gρ 1 . It then sends {ς,μ,Σ,R} as the response proof of storage correctness to the TPA, where ς = rσ +γρ mod p. With the response from the server, the TPA runs VerifyProof to validate the response by first computing γ = h(R) and then checking the verification equation

$$R \cdot e(\Sigma^{\gamma}, g) \stackrel{?}{=} e((\prod_{i=s_1}^{s_c} H(W_i)^{\nu_i})^{\gamma} \cdot u^{\mu}, v) \cdot e(g_1, g)^{\varsigma} \quad (6)$$

To see the correctness of the above equation, we have:

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

The last equality follows from the elaboration of Equation1 in Section 3.4.Theorem 4: The above auditing protocol achieves zero knowledge information leakage to the TPA, and it also ensures the storage correctness guarantee.

VI. 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-usersetting, 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. Our preliminary experiment conducted on Amazon EC2 instance further demonstrates the fast performance of our design on both the cloud and the auditor side. We leave the

full-fledged implementation of the mechanism on commercial public cloud as an important future extension, which is expected to robustly cope with very large scale data and thus encourage users to adopt cloud storage services more confidently.

REFERENCES

[1] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for storage security in cloud computing," in Proc. of IEEE INFOCOM'10, March 2010.

[2] P. Mell and T. Grance, "Draft NIST working definition of cloud computing," Referenced on June. 3rd, 2009. http://csrc.nist.gov/groups/SNS/cloudcomputing/index.html.

[3] 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. UCBEECS-2009-28, Feb 2009.

[4] Cloud Security Alliance, "Top threats to cloud computing,"2010, http://www.cloudsecurityalliance.org.

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

[6] J.Kincaid, "MediaMax/TheLinkup closes its doors,"July 2008,http://www.techcrunch.com/2008/07/10/mediamaxthelinkup-closes-its-doors/.

[7] Amazon.com, "Amazon s3 availability event: July 20,2008,"http://status.aws.amazon.com/s320080720.html, 2008.

[8] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 5, pp. 847–859, 2011.

[9] 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, 2007, pp. 598–609.

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

[11] A. Juels and J. Burton S. Kaliski, "PORs: Proofs of retrievability for large files," in Proc. of CCS'07, 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, vol. 5350, Dec 2008, pp. 90–107.

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

[15] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to keep online storage services honest," in Proc. of HotOS'07, 2007, pp. 1–6.

[16] 104th United States Congress, "Health Insurance Portability and Accountability Act of 1996 (HIPPA)," Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.

[17] R. Curtmola, O. Khan, and R. Burns, "Robust remote data checking," in Proc. of the 4th ACM international workshop on Storage security and survivability (StorageSS'08), 2008, pp. 63–68.

[18] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in Proc. of ACM workshop on Cloud Computing security (CCSW'09), 2009, pp. 43–54.

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

[20] A. L. Ferrara, M. Green, S. Hohenberger, and M. Pedersen, "Practical short signature batch verification," in Proc. of CT-RSA, volume 5473 of LNCS. Springer-Verlag, 2009, pp. 309–324.

[21] 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.

[22] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards secure and dependable storage services in cloud computing," IEEE Transactions on Service Computing, 2011, to appear.

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

[24] R. C. Merkle, "Protocols for public key cryptosystems," in Proc. of IEEE Symposium on Security and Privacy, 1980.

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

[26] M. Bellare and G. Neven, "Multi-signatures in the plain publickey model and a general forking lemma," in Proc. of CCS, 2006, pp. 390–399.

[27] Amazon.com, "Amazon elastic compute cloud," http://aws. amazon.com/ec2/, 2009.

[28] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. Yau, "Efficient provable data possession for hybrid clouds," Cryptology ePrint Archive, Report 2010/234, 2010.

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

[30] F. Sebe, J. Domingo-Ferrer, A. Mart´ınez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient remote data possession checking in critical information infrastructures," IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 8, pp. 1034–1038, August 2008.

[31] T. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in Proc. of ICDCS'06, 2006.

[32] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in Proc. of ICDCS'08. IEEE Computer Society, 2008, pp. 411–420.

[33] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in Proc. of CCS'09, 2009.