© 2015 IJSRSET | Volume 1 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Security Challenges for Public Cloud

Raghavendran R, Parthasarathi L, Deepa N
Computer Science & Engineering,GKM College of Engineering & Technology, Chennai, Tamilnadu, India

ABSTRACT

The integrity of data in cloud storage is subject to skepticism and scrutiny, as data stored in the cloud can easily be corrupted due to the inevitable hardware/software failures and human errors. Therefore, the integrity of cloud data should be verified before any data utilization, such as search or computation over cloud data. The traditional approach for checking data correctness is to retrieve the entire data from the cloud, and then verify data integrity by checking the correctness of signatures or hash values of the entire data. Certainly, this conventional approach is able to successfully check the correctness of cloud data. However, the efficiency of using this traditional approach on cloud data is in doubt. The main reason is that the size of cloud data is large in general. Downloading the entire cloud data to verify data integrity will cost or even waste user amounts of computation and communication resources, especially when data have been corrupted in the cloud.

Keywords: Public Auditing, Privacy Preserving, Shared data, Cloud Computing

I. INTRODUCTION

Cloud service providers offer users efficient and scalable data storage services with a much lower marginal cost than traditional approaches [2]. Nowadays it has become common for the users that they share their data with the group members, which also includes Dropbox, iCloud, etc., Both the integrity and the reliability of the data stored in the cloud is fragile, as data stored in a cloud can easily be lost or corrupted due to software failure or hardware failure, and human errors [3], [4]. In a certain situation, cloud service providers may be reluctant to update the users about such data errors in order to maintain their reputation and avoid losing profits [5]. Integrity and correctness check should be made prior to any data utilization in the cloud [6].

The traditional approach is not handy as it involves checking for data correctness by retrieving the entire data from the cloud and verifying the data integrity by evaluating the correctness of the signatures (e.g., RSA [7]) or Hash values (e.g., MD5 [8]) of the entire data. Although it successfully Checks the correctness of the data, the efficiency of using this approach on cloud is in doubt [9].

It involves downloading voluminous data from the cloud which will cost more or even waste user amounts of computation and communication resources, especially when data have been corrupted in the cloud. Recently many mechanisms [9], [10], [11], [12], [13], [14], [15], [16], [17] have been deployed to allow not only the data owner but also public verifier to effectively perform integrity checking without downloading the entire data from the cloud, which is referred to as public auditing [5]. In these mechanisms, data is divided into many small blocks, where each block is independently signed by the owner; and a random combination of all blocks instead of the whole data is retrieved during integrity checking [9]. A public verifier or a third party auditor (TPA) can provide expert integrity checking services [18]. Unfortunately, current public auditing solutions mentioned above only focus on personal data in the cloud [1].

We believe that sharing data among multiple users is perhaps one of the most engaging features that motivate cloud storage. Therefore, it is also necessary to ensure the integrity of shared data in the cloud is correct. However, a new significant privacy issue introduced in the case of shared data with the use of existing

mechanisms is the leakage of identity privacy to public verifiers [1].

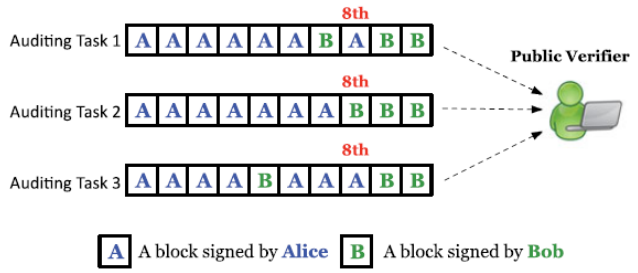


Fig. 1. Alice and Bob share a data file in the cloud, and a public verifier audits shared data integrity with existing mechanisms.

For instance, Alice and Bob work together as a group and share a file in the cloud (as in fig.1). The shared file is divided into a number of small blocks, where each block is independently signed by one of the two users with existing public auditing solutions. Once a block in this shared file is modified by a user, this user needs to sign the new block using their private key. Eventually, different blocks are signed by different users due to the modification introduced by these two different users. Then, in order to correctly audit the integrity of the entire data, a public verifier needs to choose the appropriate public key for each block. Hence, this public auditor will gain knowledge about the identity of the signer on each block due to unique binding between an identity and a public key via digital certificates under public key infrastructure (PKI). Significant confidential information will be revealed as a result of improper identity privacy on shared data during public auditing.

In this paper, to solve the above privacy issue on shared data, we propose a novel privacy preserving public auditing mechanism. We utilize ring signatures [21] to construct homomorphic authenticators [10] so that a public verifier is able to verify the integrity of the shared data without retrieving the entire data while the identity of the signer on each block is kept private from the public verifier.

Table 1: Comparison among Different Mechanisms

	PDP [9]	WWRL [5]	SCP
Public Auditing	✓	✓	✓
Data Privacy	-	✓	✓
Identity Privacy	-	-	✓

We extend our mechanism to support batch auditing, which can perform multiple auditing tasks simultaneously and improve the efficiency of verification for multiple auditing tasks.

II. METHODS AND MATERIAL

2.1 Problem Statement

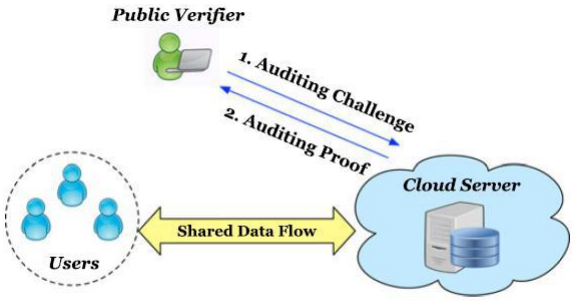


Fig. 2. Our system model includes the cloud server, a group of users and a public verifier.

A. System Model

As illustrated in Fig.2, the system model in this paper involves three parties: The cloud server, a group of users and a Public verifier. There are two types of users in a group: the original user and a number of group users. Both the original user and the group users are members of the group such that they are allowed to modify and access the shared data. Shared data and its verification metadata are also stored in the cloud server. A TPA or a public verifier intending to utilize shared data is able to publicly verify the integrity of shared data in the cloud server.

When a public verifier wishes to check the integrity of shared data an auditing challenge is first sent to the cloud server. After attaining it, the cloud server responds to the public verifier with an auditing proof of the possession of shared data. Then, the TPA checks the correctness of the entire data by verifying the correctness of the auditing proof. Essentially, the process of public auditing is a challenge and responsive protocol between a public verifier and the cloud server [9].

B. Threat Model

Integrity Threats. Two kinds of threats related to the integrity of shared data are possible. First, an adversary may corrupt the integrity of shared data. Second, the cloud service provider may inadvertently corrupt data in its storage due to hardware failures or human errors.

Privacy Threats. The identity of signer on each block in shared data is private and confidential to the group.

During auditing, TPA may try to reveal the identity of the signer based on verification metadata. Once he achieves he can easily distinguish a high value target from others.

2.2 Preliminaries

In this section, we briefly introduce cryptographic primitives and their corresponding properties that we implement.

A. Bilinear Maps

Let Ǫ1, Ǫ2 and ǪT be three multiplicative cyclic groups of prime order Þ, ǵ1 be a generator of Ǫ1 and ǵ2 be a generator of Ǫ2. A bilinear map ҿ: Ǫ1 x Ǫ2 → ǪT with the following properties:

- *Computability:* There exists an efficiently computable algorithm for computing map ҿ.
- *Bilinearity:* for all υ € Ǫ1, ν € Ǫ2 and a,b € Ƶp , ҿ (υa, νb) = ҿ(υ, ν) ab
- *NonDegeneracy:* ҿ(ǵ1, ǵ2) ≠ 1. Bilinear maps can be generally constructed from certain elliptical curves [27]. Readers do not learn the technical details about how to build linear maps from certain elliptic curves. Understanding the properties of bilinear maps described above is sufficient enough for readers to access the design of our mechanism.

B. Ring Signatures

With ring signatures, a verifier is convinced that a signature is computed using one of group member's private keys, but the verifier is not able to determine which one. By having a ring signature and a group of d users, a verifier cannot distinguish the signer's identity with a probability more than 1/d.

C. Homomorphic Authenticators

These are basic tools to construct public auditing mechanisms [1], [5], [9], [10], [12], [15]. Homomorphic Authenticable signature scheme, which denotes a homomorphic authenticator based on signatures, should satisfy the following properties:

Let (pk, sk) denotes the signer's public/private key pair, ǿ1 denote a signature on block m1 € Ƶp, ǿ2 denote a signature on

a block m2 € Ƶp.

- *Blockless Verifiability:* Given ǿ1 and ǿ2, two random values ɑ1, ɑ2 € Ƶp and a block m'= ɑ1m1 + ɑ2m2 € Ƶp, a verifier is able to check the correctness of the block m' without knowing block m1 and m2.
- *Non Malleability:* Given ǿ1 and ǿ2, two random values ɑ1, ɑ2 € Ƶp and a block m'= ɑ1m1 + ɑ2m2 € Ƶp, a user who does not have private key sk, is not able to generate a valid signature ǿ' on block m' by linearly combining signatures ǿ1 and ǿ2.

Blockless verifiability allows the verifier to audit the correctness of data stored in cloud server with a special block, which is a linear combination of all the blocks in the data. If the integrity of the combined block is correct, then the verifier believes that the integrity if the entire data is correct. Non Malleability indicates that an adversary cannot generate valid signatures on arbitrary block by linearly combining existing signatures.

2.3 Traditional Ring Signature Scheme

A. Overview

We design a new homomorphic authenticable ring signature (HARS) scheme, which is extended from a classic ring signature scheme [21]. The ring signatures generated by HARS are not only able to preserve identity privacy but also able to support blockless verifiability. We will show how to build the privacy preserving public auditing mechanism for shared data in the cloud based on this new ring signature scheme in the next section.

B. Construction of HARS

HARS contains three algorithms: KeyGen, RingSign and RingVerify. In KeyGen, each user in the group generates his/her public key and private key. In RingSign, a user in the group is able to generate a signature on a block and its block identifier with his/her private key and all the group members' public keys. A block identifier is a string that can distinguish the corresponding block from others. A verifier is able to check whether a given block is signed by a group member in RingVerify.

2.4 Key Generation Scheme

A. Overview

The RSA scheme is a block cipher in which the plain text and cipher text are integers between 0 and n1 for some n. A typical size for 'n' is 1024 bits, or 309 decimal digits. That is, n are less than 21024.

B. Description of RSA

The scheme developed by Rivest, Shamir, and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number 'n'. That is, the block size must be less than or equal to log2(n); in practice, the block size is *i* bits, where $2i<n\leq 2i+1$. Encryption and decryption are of the following form, for some plaintext block *M* and cipher text block *C*:

$C = Me \bmod n$

$M = Cd \bmod n = (Me)d \bmod n$

Both sender and the receiver must know the value of *n*. The sender knows the value of *e*, and only the receiver knows the value of *d*. Thus, this is a public key encryption algorithm with a public key of $PU = \{e,n\}$ and a private key of $PR = \{d,n\}$.

For example, the keys were generated as follows:

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate $n = pq = 17 * 11 = 187$
3. Calculate $\Phi(n) = (p-1) (q-1) = 16 * 10 = 160$.
4. Select *e* such that *e* is relatively prime to $\Phi(n) = 160$ and less than $\Phi(n)$; We choose $e = 7$
5. Determine *d* such that $de = 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 * 7 = 161 = 10 * 160 + 1$; *d* can be calculated using the extended Euclid's algorithm.

The resulting keys are public key $PU = \{7 , 187 \}$ and private key $PR = \{23 , 187\}$. The example shows the use of these keys for a plaintext input of $M = 88$. For encryp-tion, we need to calculate $C = 887 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows :

887 mod 187 = [(884 mod 187) * (882 mod 187) * (881 mod 187)] mod 187

881 mod 187 = 88

882 mod 187 = 7744 mod 187 = 77

884 mod 187 = 59, 969, 536 mod 187 = 132

887 mod 187 = (88 * 77 * 132) mod 187 = 894, 432 mod 187 = 11

For decryption, we calculate

$M = 1123 \bmod 187$

1123 mod 187 = [(111 mod 187) * (112 mod 187) * (114 mod 187) * (118 mod 187)] mod 187

111 mod 187 = 11

112 mod 187 = 121

114 mod 187 = 124, 641 mod 187 = 55

118 mod 187 = 214, 358, 881 mod 187 = 33

1123 mod 187 = (11 * 121 * 55 * 33 * 33) mod 187 = 79, 720, 245 mod 187 = 88

C. Security of RSA

Four possible approaches to attacking the RSA algorithm are as follows :

- *Brute Force* : This involves trying all possible private keys.
- *Mathematical attacks* : There are several approaches, all equivalent in effort to factoring the product of two primes.
- *Timing attacks* : These depend on the running time of the decryption algorithm.
- *Chosen ciphertext attacks* : This type of attack exploits properties of the RSA algorithm.

The defense against the bruteforce approach is the same for RSA as for other cryptosystems, namely,

use a large key space. Thus the larger the number of bits in *d*, the better it is secure. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run

.

D. Implementation

We implement the RSA algorithm in our methodology in the registration phase of both the group users and group owners. An individual public and private key is generated for every individual in the group user and the group owner. These keys are further used whenever a user wants to share his/her data. At the time of sharing the user's data is divided into several blocks and each block is generated a unique signature which is based upon the private and public key of the group owner and the public key of the group user.

2.5 New Ring Signature Scheme

A. Overview

A hash function such as SHA was not designed for use as a Message Authentication Code (MAC) and cannot be directly used for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing has algorithm. The approach that has received the most support is HMAC which has been chosen as the mandatory to implement MAC for IP security, and is used in other internet protocols, such as SSL.

B. Design Objectives

RFC 2104 lists the following design objectives for HMAC:

- To use, without modifications, available has functions. In particular, hash functions that perform well in software, and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

The first two objectives are important to the acceptability of HMAC. It treats the hash function as a "black box". This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one.

C. HMAC algorithm

H = embedded hash function
IV = initial value input to hash function
M = message input to HMAC
Yi = *i*th block of *M*, $0 \leq i \leq (L - 1)$
L = number of blocks in *M*
b = number of bits in a block
n = length of hash code produced by embedded hash function
K = secret key recommended length is > n; if key length is greater than *b;* the key is input to the hash function to produce an *n*-bit key
K+ = *K* padded with zeros on the left so that the result is *b* bits in length.
ipad = 00110110
opad = 01011100
Then HMAC can be expressed as follow:

HMAC (*K, M)*= H[(*K*+ø opad) || H [(*K*+ ø ipad) || M]]

D. Security of HMAC

The security of any MAC function based on an embedded
Hash function depends in some way on the cryptographic strength of the underlying hash function. The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC.

The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of the time spent by the forger and a given number of messageMAC pairs created with the same key. In essence, it is provided in that for a given level of effort on messages generated by a legitimate user and seen by the attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function :

1. The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker.
2. The attacker finds collision in the hash function even when the IV is random and secret.

In the first attack, we can view the compression function as equivalent to the hash function applied to a message consisting of a single *b*bit block. For this attack, the IV of the hash function is replaced by a secret, random value of *n* bits. An attack on this hash function requires either a bruteforce attack on the key, which is a level of effort on the order of 2n, or a birthday attack, which is a special case of the second attack.
In second attack, the attacker is looking for two messages *M* and *M'* that produce hash: H(*M*)=H(*M'*). This is birthday attack. Thus, if speed is a concern, it is fully acceptable to use MD5 rather than SHA-1 as the embedded hash function for HMAC.

E. Implementation of HMAC

In our methodology we implement HMAC algorithm during the generation of ring signatures. After dividing the data into several blocks, each block is signed by the user with his own private and public key where they get appended along with the HMAC algorithm and finally generate the ring signature. The ring signatures generated by HMAC algorithm are not only able to preserve identity privacy but also able to support blockless verifiability. Hence, the public verifier need

not download the entire data, which may be inefficient as it costs memory wastage and time, instead he can just verify the ring signatures on each block and check for the correctness of the data.

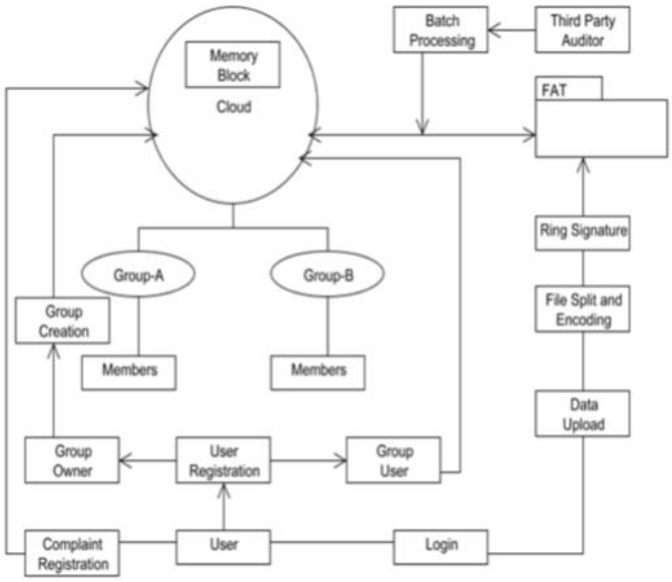


Fig 3: Architecture Diagram

2.5 Batch Auditing

At times, a public verifier may need to verify the integrity of multiple auditing tasks in a very short time. Directly verifying these tasks separately would be inefficient. By using the properties of bilinear maps, we can further extend our implementation to support batch auditing, which can verify the correctness of multiple auditing tasks simultaneously and improve the efficiency of public auditing. Based on the correctness of Equation (6), the correctness of batch auditing in Equation (7) can be presented as

$$\begin{aligned}
&\left(\prod_{b=1}^{B}\prod_{i=1}^{d_b} e(\phi_{b,i}, w_{b,i})\right)\cdot e\left(\prod_{b=1}^{B}\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right)\\
&=\prod_{b=1}^{B}\left(\left(\prod_{i=1}^{d_b} e(\phi_{b,i}, w_{b,i})\right)\cdot e\left(\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right)\right)\\
&=\prod_{b=1}^{B} e\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}, g_2\right)\\
&=e\left(\prod_{b=1}^{B}\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}\right), g_2\right).
\end{aligned}$$

If all the B shared data are from the same group, the public verifier can further improve the efficiency of batch auditing by verifying

$$\begin{aligned}
&e\left(\prod_{b=1}^{B}\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}\right), g_2\right)\\
&\stackrel{?}{=}\left(\prod_{i=1}^{d} e(\prod_{b=1}^{B}\phi_{b,i}, w_i)\right)\cdot e\left(\prod_{b=1}^{B}\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right),
\end{aligned}\tag{8}$$

which can save the public verifier about (d -1)B pairing operations in total compared to Equation (7). Note that batch auditing will fail if at least one incorrect auditing proof exists in all the B auditing proofs. To allow most of auditing proofs to still pass the verification when there exists only a small number of incorrect auditing proofs, we can utilize binary search [5] during batch auditing. More specifically, once the batch auditing of the B auditing proofs fails, the public verifier divides the set of all the B auditing proofs into two subsets, where each subset contains a number of B=2 auditing proofs. Then the public verifier rechecks the correctness of auditing proofs in each subset using batch auditing. If the verification result of one subset is correct, then all the auditing proofs in this subset are all correct. Otherwise, this subset is further divided into two sub subsets, and the public verifier rechecks the correctness of auditing proofs in each sub subset with batch auditing until all the incorrect auditing proofs are found. Clearly, when the number of incorrect auditing proofs increases, the public verifier needs more time to distinguish all the incorrect auditing proofs, and the efficiency of batch auditing will be reduced. Experimental result in B shows that, when less than 12 percent of all the B auditing proofs are incorrect, batching auditing is still more efficient than verifying all the B auditing proofs one by one.

III. RESULTS AND DISCUSSION

PERFORMANCE

In this section, we first analyze the computation and communication costs of our implementation, and then evaluate the performance of the same in experiments.

A. Computation Cost

During an auditing task, the public verifier first generates some random values to construct an auditing challenge, which only introduces a small cost in computation. Then, after receiving the auditing challenge, the cloud server needs to compute an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\Phi}, \{idj\}j\epsilon J\}$. Based on the description in Section 5, the computation cost of calculating an auditing proof is about (k+dc)ExpG1 + dcMulG1 + ckMulZp + kHashZp , where ExpG1 denotes the cost of computing one exponentiation in G1, MulG1 denotes the cost of computing one multiplication in G1, MulZp and HashZp respectively denote the cost of computing one

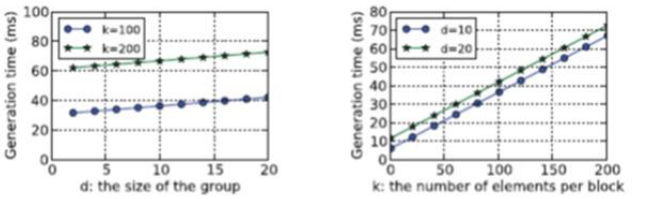
multiplication and one hashing operation in Zp. To check the correctness of an auditing proof *{λ, μ, Φ, {idj}j€J*, a public verifier audits it with Equation (6). The total cost of verifying this auditing proof is about *(2k + c)*ExpG1 + *(2k + c)* MulG1 + *d*MulGT + *c*HashG1 + *(d + 2)*Pair. We use Pair to denote the cost of computing one pairing operation on *e* : G1 * G2 → GT .

B. Communication Cost

The communication cost is mainly introduced by two aspects: the auditing challenge and auditing proof. For each auditing challenge {j, yj}j€J , the communication cost is c(|q| + |n|) bits, where |q| is the length of an element of Zq and |n| is the length of an index. Each auditing proof *{λ, μ, Φ, {idj}j€J}* contains (*k* + *d*) elements of G1, *k* elements of Zp and *c* elements of Zq, therefore the communication cost of one auditing proof is (2*k* + d)|p| + c|q| bits.

C. Experimental Results

We now evaluate the efficiency of our method. In our experiments, we utilize the GNU Multiple Precision Arithmetic (GMP) library and Pairing Based Cryptography (PBC) library. All the following experiments are based on C and tested on a 2.26 GHz Linux system over 1,000 times. Because our implementation needs more exponentiations than pairing operations during the process of auditing, the elliptic curve we choose in our experiments is an MNT curve with a base field size of 159 bits, which has a better performance than other curves on computing exponentiations. We choose |p| = 160 bits and |q| = 80 bits. We assume the total number of blocks in shared data is n = 1,000, 000 and |n| = 20 bits. The size of shared data is 2GB. To keep the detection probability greater than 99 percent, we set the number of selected blocks in an auditing task as c = 460 [9]. If only 300 blocks are selected, the detection probability is greater than 95 percent. We also assume the size of the group *d* € [2, 20] in the following experiments. Certainly, if a larger group size is used, the total computation cost will increase due to the increasing number of exponentiations and pairing operations.

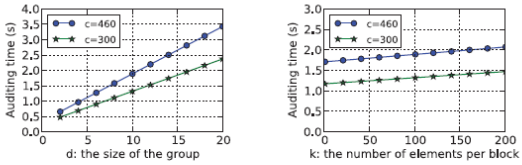


(a) Impact of d on signature generation time (ms). (b) Impact of k on signature generation time (ms).

Fig 4: Performance of signature generation

Performance of Signature Generation. According to Section 5, the generation time of a ring signature on a block is determined by the number of users in the group and the number of elements in each block. As illustrated in Figs. 10a and 10b, when k is fixed, the generation time of a ring signature is linearly increasing with the size of the group; when d is fixed, the generation time of a ring signature is linearly increasing with the number of elements in each block. Specifically, when d = 10 and k = 100, a user in the group requires about 37 mill seconds to compute a ring signature on a block in shared data.

Performance of Auditing. Based on our proceeding analyses, the auditing performance under different detection probabilities is illustrated in Figs. 11a and 12b, and Table 2. As shown in Fig. 11a, the auditing time is linearly increasing with the size of the group. When c = 300, if there are two users sharing data in the cloud, the auditing time is only about 0:5 seconds; when the number of group member increases to 20, it takes about 2:5 seconds to finish the same auditing task. The communication cost of an auditing task under different parameters is presented in Figs. 12a and 12b. Compared to the size of entire shared data, the communication cost that a public verifier consumes in an auditing task is very small. It is clear in Table 2 that when maintaining a higher detection probability, a public verifier needs to consume more computation and communication overhead to finish the auditing task. Specifically, when c = 300, it takes a public verifier 1:32 seconds to audit the correctness of shared data, where the size of shared data is 2GB; when c = 460, a public verifier needs 1:94 seconds to verify the integrity of the same shared data.



(a) Impact of d on auditing time (second), where $k = 100$. (b) Impact of k on auditing time (second), where $d = 10$.

Fig 5 : Performance of auditing time

As we discussed in the previous section, the privacy performance of our mechanism depends on the number of members in the group. Given a block in shared data, the probability that a public verifier fails to reveal the identity of the signer is 1 – 1/d, where d ≤ 2. Clearly, when the number of group members is larger, our mechanism has a better performance in terms of privacy. As we can see from Fig. 13a, this privacy performance increases with an increase of the size of the group.

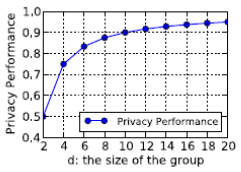
Performance of Batch Auditing. As we discussed in Section 5, when there are multiple auditing proofs, the public verifier can improve the efficiency of verification by performing batch auditing. In the following experiments, we choose c = 300, k = 100 and d = 10. Compared to verifying a number of B auditing proofs one by one, if these B auditing proofs are for different groups, batching auditing can save 2:1 percent of the auditing time per auditing proof on average (as shown in Fig. 14a). If these B auditing tasks are for the same group, batching auditing can save 12:6 percent of the average auditing time per auditing proof (as shown in Fig. 14b). Now we evaluate the performance of batch auditing when incorrect auditing proofs exist among the B auditing proofs. As we mentioned in Section 5, we can use binary search in batch auditing, so that we can distinguish the incorrect ones from the B auditing proofs.

However, the increasing number of incorrect auditing proofs will reduce the efficiency of batch auditing. It is important for us to find out the maximal number of incorrect auditing proofs exist in the B auditing proofs, where the batch auditing is still more efficient than separate auditing.

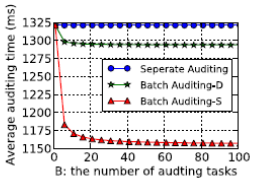
In this experiment, we assume the total number of auditing proofs in the batch auditing is B = 128 (because we leverage binary search, it is better to set B as a power of 2), the number of elements in each block is k = 100 and the number of users in the group is d = 10. Let A denote the number of incorrect auditing proofs. In addition, we also assume that it always requires the worst case algorithm to detect the incorrect auditing proofs in the experiment.

TABLE 2
Performance of Auditing

System Parameters	$k = 100$, $d = 10$,	
Storage Usage	2GB + 200MB (data + signatures)	
Selected Blocks c	460	300
Communication Cost	14.55KB	10.95KB
Auditing Time	1.94s	1.32s



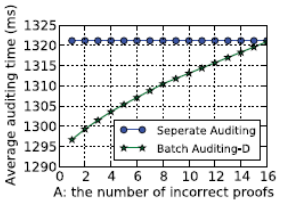
(a) Impact of d on privacy performance.



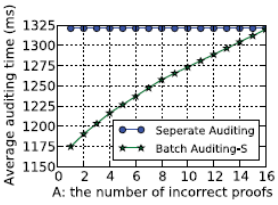
(b) Impact of B on the efficiency of batch auditing, where $k = 100$ and $d = 10$.

Fig 6 : Performance of privacy and batch auditing

According to Equation (7) and (8), the extra computation cost in binary search is mainly introduced by extra pairing operations. As shown in Fig. 14a, if all the 128 auditing proofs are for different groups, when the number of incorrect auditing proofs is less than 16 (12 percent of all the auditing proofs), batching auditing is still more efficient than separate auditing.
Similarly, in Fig. 14b, if all the auditing proofs are for the same group, when the number of incorrect auditing proofs is more than 16, batching auditing is less efficient than verifying these auditing proofs separately.



(a) Impact of A on the efficiency of batch auditing, where $B = 128$.



(b) Impact of A on the efficiency of batch auditing, where $B = 128$.

Fig 7: Efficiency of batch auditing with incorrect proofs

Provable data possession (PDP), proposed by Ateniese et al. [9], allows a verifier to check the correctness of a client's data stored at an untrusted server. By utilizing RSAbased homomorphic authenticators and sampling strategies, the verifier is able to publicly audit the integrity of data without retrieving the entire data, which is referred to as public auditing. Unfortunately, their mechanism is only suitable for auditing the integrity of personal data. Juels and Kaliski [32] defined another similar model called Proofs of Retrievability (POR), which is also able to check the correctness of data on an untrusted server. The original file is added with a set of randomly valued check blocks called sentinels. The verifier challenges the untrusted server by specifying the

positions of a collection of sentinels and asking the untrusted server to return the associated sentinel values. Shacham and Waters [10] designed two improved schemes. The first scheme is built from BLS signatures [27], and the second one is based on pseudorandom functions.

To support dynamic data, Ateniese et al. [33] presented an efficient PDP mechanism based on symmetric keys. This mechanism can support update and delete operations on data, however, insert operations are not available in this mechanism. Because it exploits symmetric keys to verify the integrity of data, it is not public verifiable and only provides a user with a limited number of verification requests. Wang et al. [12] utilized Merkle Hash Tree and BLS signatures [27] to support dynamic data in a public auditing mechanism. Erway et al. [11] introduced dynamic provable data possession (DPDP) by using authenticated dictionaries, which are based on rank information. Zhu et al. [15] exploited the fragment structure to reduce the storage of signatures in their public auditing mechanism. In addition, they also used index hash tables to provide dynamic operations on data. The public mechanism proposed by Wang et al. [5] and its journal version [18] are able to preserve users' confidential data from a public verifier by using random maskings. In addition, to operate multiple auditing tasks from different users efficiently, they extended their mechanism to enable batch auditing by leveraging aggregate signatures [21].

Wang et al. [13] leveraged homomorphic tokens to ensure the correctness of erasure codesbased data distributed on multiple servers. This mechanism is able not only to support dynamic data, but also to identify misbehaved servers. To minimize communication overhead in the phase of data repair, Chen et al. [14] also introduced a mechanism for auditing the correctness of data under the multiserver scenario, where these data are encoded by network coding instead of using erasure codes. More recently, Cao et al. [16] constructed an LT codesbased secure and reliable cloud storage mechanism. Compare to previous work [13], [14], this mechanism can avoid high decoding computation cost for data users and save computation resource for online data owners during data repair.

Related Works

By utilizing RSAbased homomorphic authenticators and sampling strategies, the verifier is able to publicly audit the integrity of data without retrieving the entire data, which is referred to as public auditing. Unfortunately, their mechanism is only suitable for auditing the integrity of personal data.

IV. CONCLUSION

In this paper, we propose our method, a privacypreserving public auditing mechanism for shared data in the cloud. We utilize ring signatures to construct homomorphic authenticators, so that a public verifier is able to audit shared data integrity without retrieving the entire data, yet it cannot distinguish who is the signer on each block. To improve the efficiency of verifying multiple auditing tasks, we further extend our mechanism to support batch auditing.

V. REFERENCES

[1] B. Wang B Li, and H. Li, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud" Proc IEEE Fifth Int'l Conf. Cloud Computing pp, 295-302, 2012

[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, "A View of Cloud Computing" Comm ACM, Vol 53, no. 4, pp. 50,58, April 2010

[3] K. Ren C. Wand abd Q. Wang, "Security Challanges for the Public Cloud" IEEE Internet Computing, Vol. 16, no. 1, pp. 69-73, 2012

[4] F. Song E. Shi, I. Fischer and U. Shankar, "Cloud Data Protection for the Masses", Computer, vol.45, no. 1. 39-45,2012

[5] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing" Proc. IEEE INFOCOM, pp. 525-533,2010.

[6] B. Wang, M. Li, S.S. Chow, and H. Li, "Computing Encrypted Cloud Data Efficiently under Multiple Keys" Proc. IEEE Conf. Comm. and Network Security(CNS'13), pp. 90-99, 2013

[7] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems" Comm. ACM, vol.21m no2, pp.120-126,1978

[8] The MD5 Message-Digest Algorithm (RFC1321). https://tools.ieft.org/html/rfc1321,2014

[9] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song "Provable Data Possession at Untrusted Stores" Proc. 14th ACM Conf. Computer and Comm Security (CCS ' 07), pp. 598-610, 2007

[10] H Shacham and B. Waters, "Compact Proofs of Retrivability" Proc. 14th Int'l Conf. THeory and Application of Crytology and Information Security: Advances in Cryptology (ASIACRYPT'08), pp.90-107, 2008.