IEEE Transactions on Information Forensics and Security,(Volume:10 , Issue: 7),July 2015.

Privacy-Preserving Public Auditing for Regenerating-Code-Based Cloud Storage

Jian Liu, Kun Huang, Hong Rong, Huimei Wang and Ming Xian

***Abstract*—To protect outsourced data in cloud storage against corruptions, adding fault tolerance to cloud storage together with data integrity checking and failure reparation becomes critical. Recently, regenerating codes have gained popularity due to their lower repair bandwidth while providing fault tolerance. Existing remote checking methods for regenerating-coded data only provide private auditing, requiring data owners to always stay online and handle auditing, as well as repairing, which is sometimes impractical. In this paper, we propose a public auditing scheme for the regenerating-code-based cloud storage. To solve the regeneration problem of failed authenticators in the absence of data owners, we introduce a proxy, which is privileged to regenerate the authenticators, into the traditional public auditing system model. Moreover, we design a novel public verifiable authenticator, which is generated by a couple of keys and can be regenerated using partial keys. Thus, our scheme can completely release data owners from online burden. In addition, we randomize the encode coefficients with a pseudorandom function to preserve data privacy. Extensive security analysis shows that our scheme is provable secure under random oracle model and experimental evaluation indicates that our scheme is highly efficient and can be feasibly integrated into the regenerating-code-based cloud storage.**

***Index Terms*—Cloud storage, regenerating codes, public audit, privacy preserving, authenticator regeneration, proxy, privileged, provable secure.**

I. Introduction

CLOUD storage is now gaining popularity because it offers a flexible on-demand data outsourcing service with appealing benefits: relief of the burden for storage management, universal data access with location independence, and avoidance of capital expenditure on hardware, software, and personal maintenances,etc., [1]. Nevertheless, this new paradigm of data hosting service also brings new security threats toward users data, thus making individuals or enterprisers still feel hesitant.

It is noted that data owners lose ultimate control over the fate of their outsourced data; thus, the correctness, availability and integrity of the data are being put at risk. On the one hand, the cloud service is usually faced with a broad range of internal/external adversaries, who would maliciously delete or corrupt users' data; on the other hand, the cloud service providers may act dishonestly, attempting to hide data loss or corruption and claiming that the files are still correctly stored in the cloud for reputation or monetary reasons. Thus it makes great sense for users to implement an efficient protocol to perform periodical verifications of their outsourced data to ensure that the cloud indeed maintains their data correctly.

Many mechanisms dealing with the integrity of outsourced data without a local copy have been proposed under different system and security models up to now. The most significant work among these studies are the PDP (*provable data possession*) model and POR (*proof of retrievability*) model, which were originally proposed for the **single-server** scenario by Ateniese *et al.* [2] and Juels *et. al.* [3], respectively. Considering that files are usually striped and redundantly stored across multi-servers or multi-clouds, [4]–[10] explore integrity verification schemes suitable for such **multi-servers** or **multi-clouds** setting with different redundancy schemes, such as *replication*, *erasure codes*, and, more recently, *regenerating codes*.

In this paper, we focus on the integrity verification problem in **regenerating-code-based cloud storage**, especially with the functional repair strategy [11]. Similar studies have been performed by Bo Chen *et al.* [7] and H. Chen *el al.* [8] separately and independently. [7] extended the single-server CPOR scheme(private version in [12]) to the regenerating-code-scenario; [8] designed and implemented a data integrity protection(DIP) scheme for FMSR [13]-based cloud storage and the scheme is adapted to the thin-cloud setting[1]. However, both of them are designed for private audit, only the data owner is allowed to verify the integrity and repair the faulty servers. Considering the large size of the outsourced data and the user's constrained resource capability, the tasks of auditing and reparation in the cloud can be formidable and expensive for the users [14]. The overhead of using cloud storage should be minimized as much as possible such that a user does not need to perform too many operations to their outsourced data (in additional to retrieving it) [15]. In particular, users may not want to go through the complexity in verifying and reparation. The auditing schemes in [7], [8] imply the problem that users need to always stay online, which may impede its adoption in practice, especially for long-term archival storage.

To fully ensure the data integrity and save the users' computation resources as well as online burden, we propose a public auditing scheme for the regenerating-code-based cloud storage, in which the integrity checking and regeneration (of failed

Copyright (c) 2013 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

Jian Liu, Kun Huang, Hong Rong, Huimei Wang and Ming Xian are with the State Key Laboratory of Complex Electromagnetic Environment Effects on Electronic and Information System, National University of Defense Technology, Changsha, 410073, China. Email: (ljabc730@gmail.com, khuangresearch923@gmail.com, ronghong01@gmail.com, freshcdwhm@163.com and qwertmingx@sina.com)

[1]Indicating that the cloud servers are only provided with the RESTful interface.

data blocks and authenticators) are implemented by a third-party auditor and a semi-trusted proxy separately on behalf of the data owner. Instead of directly adapting the existing public auditing scheme [12] to the multi-server setting, we design a novel authenticator, which is more appropriate for regenerating codes. Besides, we "*encrypt*" the coefficients to protect data privacy against the auditor, which is more lightweight than applying the proof blind technique in [14], [15] and data blind method in [16]. Several challenges and threats spontaneously arise in our new system model with a proxy (Section II-C), and security analysis shows that our scheme works well with these problems. Specifically, our contribution can be summarized by the following aspects:

- We design a novel homomorphic authenticator based on BLS signature [17], which can be generated by a couple of secret keys and verified publicly. Utilizing the linear subspace of the regenerating codes, the authenticators can be computed efficiently. Besides, it can be adapted for data owners equipped with low end computation devices(e.g. Tablet PC etc.) in which they only need to sign the native blocks.
- "To the best of our knowledge, our scheme is the first to Moreover, according to whether the corrupted blocks can be allow privacy-preserving public auditing for regenerating-code-based cloud storage. The coefficients are masked by a PRF(Pseudorandom Function) during the Setup phase to avoid leakage of the original data. This method is lightweight and does not introduce any computational overhead to the cloud servers or TPA.
- Our scheme completely releases data owners from online burden for the regeneration of blocks and authenticators at faulty servers and it provides the privilege to a proxy for the reparation.
- Optimization measures are taken to improve the flexibility and efficiency of our auditing scheme; thus, the storage overhead of servers, the computational overhead of the data owner and communication overhead during the audit phase can be effectively reduced.
- Our scheme is provable secure under *random oracle model* against adversaries illustrated in Section II-C. Moreover, we make a comparison with the state of the art and experimentally evaluate the performance of our scheme.

The rest of this paper is organized as follows: Section II introduces some preliminaries, the system model, threat model, design goals and formal definition of our auditing scheme. Then we provide the detailed description of our scheme in Section III; Section IV analyzes its security and Section V evaluates its performance. Section VI presents a review of the related work on the auditing schemes in cloud storage. Finally, we conclude this paper in Section VII.

II. Preliminaries and Problem Statement

A. Notations and Preliminaries

1) Regenerating Codes: Regenerating codes are first introduced by A. G. Dimakis *et al.* [18] for distributed storage to reduce the repair bandwidth. Viewing cloud storage to be a collection of n storage servers, data file F is encoded and stored redundantly across these servers. Then F can be retrieved by connecting to any k-out-of-n servers, which is termed the MDS[2]-property. When data corruption at a server is detected, the client will contact ℓ healthy servers and download β' bits from each server, thus regenerating the corrupted blocks without recovering the entire original file. Dimakis [18] showed that the repair bandwidth $\gamma' = \ell\beta'$ can be significantly reduced with $\ell \geq k$. Furthermore, they analyzed the fundamental tradeoff between the storage cost α' and the repair bandwidth γ', then presented two extreme and practically relevant points on the optimal tradeoff curve: *the minimum bandwidth regenerating*(MBR) point, which represents the operating point with the least possible repair bandwidth, and *the minimum storage regenerating*(MSR) point, which corresponds to the least possible storage cost on the servers. Denoted by the parameter tuple $(n, k, \ell, \alpha', \gamma')$, we obtain:

$$(\alpha'_{MSR}, \gamma'_{MSR}) = \left(\frac{|F|}{k}, \frac{|F|\ell}{k(\ell - k + 1)}\right) \quad (1)$$

$$(\alpha'_{MBR}, \gamma'_{MBR}) = \left(\frac{2|F|\ell}{2k\ell - k^2 + k}, \frac{2|F|\ell}{2k\ell - k^2 + k}\right) \quad (2)$$

Moreover, according to whether the corrupted blocks can be exactly regenerated, there are two versions of repair strategy: exact repair and functional repair [11]. Exact repair strategy requires the repaired server to store an exact replica of the corrupted blocks, while functional repair indicates that the newly generated blocks are different from the corrupted ones with high probability. As one basis of our work, the functional repair regenerating codes are non-systematic and do not perform as well for read operation as systematic codes, but they really make sense for the scenario in which data repair occurs much more often than read, such as regulatory storage, data escrow and long-term archival storage [7].

The regenerating codes with functional repair strategy can be achieved using the random network coding. Given a file represented by m blocks $\{\overline{\mathrm{w}}_i\}_{i=1}^m$, with each $\overline{\mathrm{w}}_i = (\overline{w}_{i1}, \overline{w}_{i2}, ..., \overline{w}_{is})$, where $\overline{w}_{ij}$'s belong to the finite field $GF(2^\omega)$ and are referred to as symbols, the data owner then generates coded blocks from these native blocks; specifically, m coding coefficients $\{\varepsilon_i\}_{i=1}^m$ are chosen randomly from $GF(2^\omega)$ and computed out $\overline{v} = \sum_{i=1}^m \varepsilon_i \cdot \overline{\mathrm{w}}_i$, where all algebraic operations work over $GF(2^\omega)$. When the size of the finite field $GF(2^\omega)$, where the coefficients are drawn, is large enough, any m coded blocks will be linearly independent with high probability [19]; thus, the original file can be reconstructed from any m coded blocks by solving a set of m equations. The coded blocks will be distributed across n storage servers, with each server storing α blocks. Any k-out-of-n servers can be used to recover the original data file when the inequality $k\alpha \geqslant m$ is maintained, thus satisfying the MDS property. When the periodic auditing process detect data corruption at one server, the repair procedure will contact ℓ servers and obtain β blocks from each to regenerate the corrupted α blocks. Fig.1 shows an example of functional repair regenerating code. The guidelines for choosing parameter tuple $(n, k, \ell, \alpha, \beta)$ can be found in [7], [18].

[2]Maximum Distance Separable

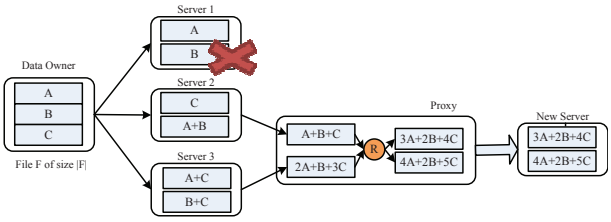

Fig. 1. An example of functional repair regenerating code with parameters $(n = 3, k = 2, \ell = 2, \alpha = 2, \beta = 1)$. The data owner computes six coded blocks as random linear combinations of the native three blocks, and distributes them across three servers. When Server 1 gets corrupted, the proxy contacts the remaining two servers and retrieves one block(obtained also by linear combination) from each, then it linearly combines them to generate two new coded blocks. Finally, the new coded blocks are sent to a new healthy server. The resulting storage system turns out to satisfy the $(3, 2)$MDS property.

2) Linear Subspace from Regenerating code: As mentioned above, each coded block represents the linear combination of m native blocks in the functional repair regenerating code scenario. Thus, we can generate a linear subspace with dimension m for file F in the following way [20]:

Before encoding, F is split into m blocks, and the original m s-dimensional vectors (or blocks indistinguishably) $\{\overline{\mathtt{w}}_i \in GF(p)^s\}_{i=1}^m$ are properly augmented as:

$$\mathtt{w}_i = (\overline{w}_{i1}, \overline{w}_{i2}, ..., \overline{w}_{is}, \overbrace{\underbrace{0, ..., 0, 1}_{i}, 0, ..., 0}^{m}) \in GF(p)^{s+m} \quad (3)$$

for each symbol, $\overline{w}_{ij} \in GF(p)$. Eq.(3) shows that each original block $\overline{\mathtt{w}}_i$ is appended with the vector of length m containing a single '1' in the ith position and is otherwise zero.

Then, the augmented vectors are encoded into $n\alpha$ coded blocks. Specifically, they are linearly combined and generate coded blocks with randomly chosen coefficients $\varepsilon_i \in GF(p)$.

$$v = \sum_{i=1}^{m} \varepsilon_i \mathtt{w}_i \in GF(p)^{s+m} \quad (4)$$

Obviously, the latter additional m elements in the vector v keep track of the ε values of the corresponding blocks, i.e.,

$$v = (\underbrace{v_{i1}, v_{i2}, ..., v_{is}}_{data}, \underbrace{\varepsilon_1, ..., \varepsilon_m}_{coefficients}) \in GF(p)^{s+m} \quad (5)$$

where $(v_{i1}, v_{i2}, ..., v_{is})$ are the data, and the remaining elements indicate the coding coefficients. Notice that the blocks regenerated in the repair phase also meet the form of Eq.(5), thus we can construct a linear subspace V of m-dimension by spanning the base vectors $\mathtt{w}_1, \mathtt{w}_2, ..., \mathtt{w}_m$; all valid coded blocks appended with coding coefficients would belong to subspace V.

Under the construction of linear subspace V, we can generate tags for vectors in V efficiently, i.e., we only need to sign the m base vectors in the beginning. Such a signature scheme can be viewed as similar with signing on the subspace V [20]. We will further introduce this procedure and its better performance in the following section.

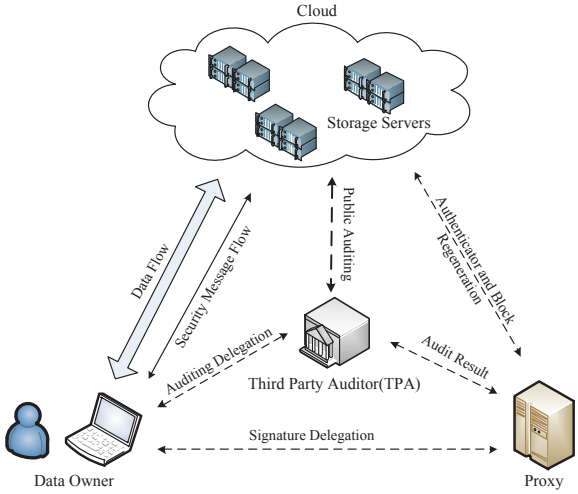

Fig. 2. The system model

3) Bilinear Pairing Map: Let G and G_T be multiplicative cyclic groups of the same large prime order p. A bilinear pairing map $e : G \times G \to G_T$ is a map with the following properties:

- Bilinear: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u, v \in G$ and $a, b \in \mathbb{Z}_p^*$, this property can be stretch to the multiplicative property that $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$ for any $u_1, u_2 \in G$;
- Non-degenerate: $e(g, g)$ generates the group G_T when g is generator of group G;
- Computability: There exists an efficient algorithm to compute $e(u, v)$ for all $u, v \in G$.

Such a bilinear map e can be constructed by the modified Weil [21] or Tate pairings [22] on elliptic curves.

B. System Model

We consider the auditing system model for Regenerating-Code-based cloud storage as Fig.2, which involves four entities: *the data owner*, who owns large amounts of data files to be stored in the cloud; *the cloud*, which are managed by the cloud service provider, provide storage service and have significant computational resources; *the third party auditor*(TPA), who has expertise and capabilities to conduct public audits on the coded data in the cloud, the TPA is trusted and its audit result is unbiased for both data owners and cloud servers; and *a proxy agent*, who is semi-trusted and acts on behalf of the data owner to regenerate authenticators and data blocks on the failed servers during the repair procedure. Notice that the data owner is restricted in computational and storage resources compared to other entities and may becomes off-line even after the data upload procedure. The proxy, who would always be online, is supposed to be much more powerful than the data owner but less than the cloud servers in terms of computation and memory capacity. To save resources as well as the online burden potentially brought by the periodic auditing and accidental repairing, the data owners resort to the TPA for integrity verification and delegate the reparation to the proxy.

Compared with the traditional public auditing system model, our system model involves an additional proxy agent. In order to reveal the rationality of our design and make our following description in Section III to be more clear and concrete, we consider such a reference scenario: A company employs a commercial regenerating-code-based public cloud and provides long-term archival storage service for its staffs, the staffs are equipped with low end computation devices(e.g., Laptop PC, Tablet PC, etc.) and will be frequently off-line. For public data auditing, the company relies on a trusted third party organization to check the data integrity; Similarly, to release the staffs from heavy online burden for data and authenticator regeneration, the company supply a powerful workstation(or cluster) as the proxy and provide proxy reparation service for the staffs' data.

C. Threat Model

Apparently, threat in our scheme comes from the compromised servers, curious TPA and semi-trusted proxy.

In terms of compromised servers, we adopt a *mobile adversary* under the multi-servers setting, similar with [5], who can compromise at most $n-k$ out of the n servers in any epoch, subject to the (n,k)-MDS fault tolerance requirement. To avoid creeping-corruption which may lead to the unrecoverable of the stored data, the repair procedure will be triggered at the end of each epoch once some corruption is detected. There are some differences in our model compared with the one in [5]: First, the adversary can corrupt not only the data blocks but also the coding coefficients stored in the compromised servers; and second, the compromised server may act honestly for auditing but maliciously for reparation. We assume that some blocks stored in server S_i are corrupted at some time, the adversary may launch the following attacks in order to prevent the auditor from detecting the corruption:

- *Replace attack*: The server S_i may choose another valid and intact pair of data block and authenticator to replace the corrupted pair, or even simply store the blocks and authenticators at another healthy server S_j ,thus successfully passing the integrity check.
- *Replay attack*: The server may generate the proof from an old coded block and corresponding authenticator to pass the verification, thus leading to a reduction of data redundancy to the point that the original data becomes unrecoverable[3].
- *Forge attack*: The server may forge an authenticator for modified data block and deceive the auditor.
- *Pollution attack*: The server may use correct data to avoid detection in the audit procedure but provide corrupted data for repairing; thus the corrupted data may pollute all the data blocks after several epoches.

With respect to the TPA, we assume it to be honest but curious. It performs honestly during the whole auditing procedure but is curious about the data stored in the cloud.

The proxy agent in our system model is assumed to be semi-trusted. It will not collude with the servers but might attempt to forge authenticators for some specified invalid blocks to pass the following verification.

[3]We refer readers to [7] for the details of such replay attack.

D. Design Goals

To correctly and efficiently verify the integrity of data and keep the stored file available for cloud storage, our proposed auditing scheme should achieve the following properties:

- *Public Auditability*: to allow TPA to verify the intactness of the data in the cloud on demand without introducing additional online burden to the data owner.
- *Storage Soundness*: to ensure that the cloud server can never pass the auditing procedure except when it indeed manage the owner's data intact.
- *Privacy Preserving*: to ensure that neither the auditor nor the proxy can derive users' data content from the auditing and reparation process.
- *Authenticator Regeneration*: the authenticator of the repaired blocks can be correctly regenerated in the absence of the data owner.
- *Error Location*: to ensure that the wrong server can be quickly indicated when data corruption is detected.

E. Definitions of Our Auditing Scheme

Our auditing scheme consists of three procedures: **Setup**, **Audit** and **Repair**. Each procedure contains certain polynomial-time algorithms as follows:

Setup: The data owner maintains this procedure to initialize the auditing scheme.

$KeyGen(1^\kappa) \rightarrow (pk, sk)$: This polynomial-time algorithm is run by the data owner to initialize its public and secret parameters by taking a security parameter κ as input.

$Degelation(sk) \rightarrow (x)$: This algorithm represents the interaction between the data owner and proxy. The data owner delivers partial secret key x to the proxy through a secure approach.

$SigAndBlockGen(sk, F) \rightarrow (\Phi, \Psi, t)$: This polynomial time algorithm is run by the data owner and takes the secret parameter sk and the original file F as input, and then outputs a coded block set Ψ, an authenticator set Φ and a file tag t.

Audit: The cloud servers and TPA interact with one another to take a random sample on the blocks and check the data intactness in this procedure.

$Challenge(\mathcal{F}_{info}) \rightarrow (\mathcal{C})$: This algorithm is performed by the TPA with the information of the file $\mathcal{F}_{info}$ as input and a challenge $\mathcal{C}$ as output.

$ProofGen(\mathcal{C}, \Phi, \Psi) \rightarrow (\mathcal{P})$: This algorithm is run by each cloud server with input challenge $\mathcal{C}$, coded block set Ψ and authenticator set Φ, then it outputs a proof $\mathcal{P}$.

$Verify(\mathcal{P}, pk, \mathcal{C}) \rightarrow (0, 1)$: This algorithm is run by TPA immediately after a proof is received. Taking the proof $\mathcal{P}$, public parameter pk and the corresponding challenge $\mathcal{C}$ as input, it outputs 1 if the verification passed and 0 otherwise.

Repair: In the absence of the data owner, the proxy interacts with the cloud servers during this procedure to repair the wrong server detected by the auditing process.

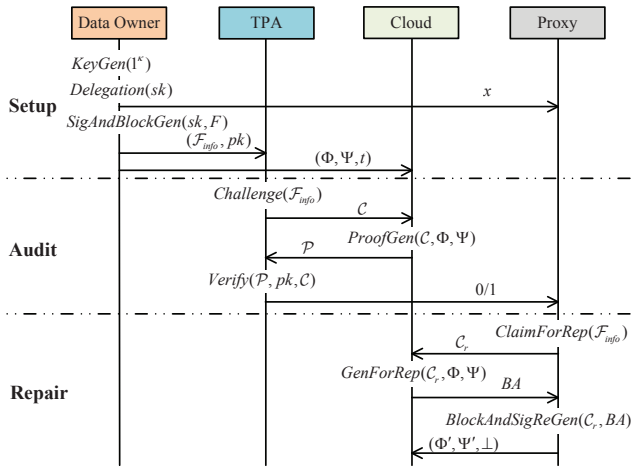

Fig. 3. The Sequence Chart of Our Scheme

$ClaimForRep(\mathcal{F}_{info}) \rightarrow (\mathcal{C}_r)$: This algorithm is similar with the $Challenge()$ algorithm in the Audit phase, but outputs a claim for repair $\mathcal{C}_r$.

$GenForRep(\mathcal{C}_r, \Phi, \Psi) \rightarrow (BA)$:The cloud servers run this algorithm upon receiving the $\mathcal{C}_r$ and finally output the block and authenticators set BA with another two inputs Φ, Ψ.

$BlockAndSigReGen(\mathcal{C}_r, BA) \rightarrow (\Phi', \Psi', \perp)$: The proxy implements this algorithm with the claim $\mathcal{C}_r$ and responses BA from each server as input, and outputs a new coded block set Ψ' and authenticator set Φ' if successful, outputting $\perp$ if otherwise.

The sequence chart of our scheme is shown in Fig.3.

III. The Proposed Scheme

In this section we start from an overview of our auditing scheme, and then describe the main scheme and discuss how to generalize our privacy-preserving public auditing scheme. Furthermore, we illustrate some optimized methods to improve its performance.

A. Overview

Although [7], [8] introduced private remote data checking schemes for regenerating-code-based cloud storage, there are still some other challenges for us to design a public auditable version.

First, although a direct extension of the techniques in [2], [12], [15] can realize public verifiability in the multi-servers setting by viewing each block as a set of segments and performing spot checking on them, such a straightforward method makes the data owner generate tags for all segments independently, thus resulting in high computational overhead. Considering that data owners commonly maintains limited computation and memory capacity, it is quite significant for us to reduce those overheads. Second, unlike cloud storage based on traditional erasure code or replication, a fixed file layout does not exist in the regenerating-code-based cloud storage. During the repair phase, it computes out new blocks, which are totally different from the faulty ones, with high probability. Thus, a problem arises when trying to determine how to regenerate authenticators for the repaired blocks. A direct solution, which is adopted in [7], is to make data owners handle the regeneration. However, this solution is not practical because the data owners will not always remain online through the life-cycle of their data in the cloud, more typically, it becomes off-line even after data uploading. Another challenge is brought in by the proxy in our system model(see Section II-C).

TABLE I
Some Primary Notations in Our Scheme Description

Notation	Description
m	the number of native data blocks
s	the number of segment in a native data block
$\overline{w}_{ik}$	the kth segment of native block w_i
v_{ij}	the jth coded block at server i
v_{ijk}	the kth segment of coded block v_{ij}
$\varepsilon_{ij\lambda}$	the λth coefficient for coded block v_{ij}
t	file tag which contains a identifier ID and random symbols $u, w_1, w_2, ..., w_m$
σ_{ijk}	the authenticator for segment v_{ijk}
Φ_i	the authenticator set for blocks in server i
Ψ_i	the coded block set for server i
$\mathcal{C}$	the challenge for audit which contains an index-coefficient pair set Q_i and a random symbol set Λ_i for server i
$\mathcal{P}$	the proof for audit which contains: μ_i-the aggregated segment, σ_i-the aggregated authenticator and $\{\rho_{i1}, \rho_{i2}, ..., \rho_{im}\}$-the auxiliary values for coefficients checking
$\mathcal{C}_r$	the claim for reparation which contains a random coefficient set Λ_i for server i
BA	the response from cloud server for reparation which contains combined block $\widetilde{v}_i$,and s aggregated authenticators $\{\widetilde{\sigma}_{ik}\}_{1\leq k\leq s}$

The following parts of this section shows our solution to the problems above. First, we construct a BLS-based [17] authenticator, which consists of two parts for each segment of coded blocks. Utilizing its homomorphic property and the linearity relation amongst the coded blocks, the data owner is able to generate those authenticators in a new method, which is more efficient compared to the straightforward approach. Our authenticator contains the information of encoding coefficients to avoid data pollution in the reparation with wrong coefficients. To reduce the bandwidth cost during the audit phase, we perform a batch verification over all α blocks at a certain server rather than checking the integrity of each block separately as [7] does. Moreover, to make our scheme secure against the replace attack and replay attack, information about the indexes of the server, blocks, and segments are all embedded into the authenticator. Besides, our primitive scheme can be easily improved to support privacy-preserving through the masking of the coding coefficients with a keyed PRF.

All the algebraic operations in our scheme work over the field $GF(p)$ of modulo p, where p is a large prime (at least 80 bits)[4].

B. Construction of Our Auditing Scheme

Considering the regenerating-code-based cloud storage with parameters $(n, k, \ell, \alpha, \beta)$, we assume $\beta = 1$ for simplicity. Let

[4]Analysis in [23] showed that the successful decoding probability for the functional repair regenerating code over $GF(2^\omega)$ and over $GF(p)$ is similar while the two fields have about the same order (*e.g.*,when $\omega = 8$, p is 257).

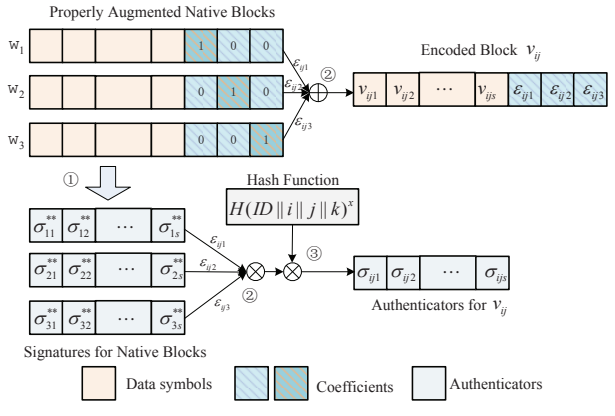

Fig. 4. An example with $m = 3$ for the $SigAndBlockGen(\cdot)$ algorithm.

G and G_T be multiplicative cyclic groups of the same large prime order p, and $e : G \times G \to G_T$ be a bilinear pairing map as introduced in the preliminaries. Let g be a generator of G and $H(\cdot) : \{0,1\}^* \to G$ be a secure hash function that maps strings uniformly into group G. Table I list the primary notations and terminologies used in our scheme description.

■ **Setup**:The audit scheme related parameters are initialized in this procedure.

$KeyGen(1^\kappa) \to (pk, sk)$: The data owner generates a random signing key pair (spk, ssk), two random elements $x, y \xleftarrow{R} \mathbb{Z}_p$ and computes $pk_x \leftarrow g^x, pk_y \leftarrow g^y$. Then the secret parameter is $sk = (x, y, ssk)$ and the public parameter is $pk = (pk_x, pk_y, spk)$.

$Delegation(sk) \to (x)$: The data owner sends encrypted x to the proxy using the proxy's public key, then the proxy decrypts and stores it locally upon receiving.

$SigAndBlockGen(sk, F) \to (\Phi, \Psi, t)$: The data owner uniformly chooses a random identifier $ID \xleftarrow{R} \{0,1\}^*$, a random symbol $u \xleftarrow{R} G$, one set $\Gamma = \{w_1, w_2, ...w_m\}$ with elements $w_i \xleftarrow{R} G$, and a file tag $t = (ID||u||w_1||...||w_m)||Sig_{ssk}(ID||u||w_1||...||w_m)$ for F. $Sig()$ means a standard signature scheme. Recall that the original file F is split into m blocks, $\{\overline{\mathtt{w}}_i\}_{i=1}^m$; the client computes and stores $n\alpha$ coded blocks amongst n cloud servers. Viewing each segment of the blocks as a single symbol for simplicity, our signature is generated simultaneously with the encoding process as follows (Fig.4 shows an instance of this algorithm with $m = 3$.):

► Augmentation: The data owner properly augments the native m blocks as Eq.(3).

► Signing for Native Blocks: The data owner views the data parts of the augmented blocks as a set of segments and computes authenticator for each segment, i.e.,

$$\sigma^{**}_{j_o k_o} = \left(u^{\overline{w}_{j_o k_o}} \cdot \prod_{\lambda=1}^{m} w_\lambda^{\overline{w}_{j_o(s+\lambda)}} \right)^y = \left(u^{\overline{w}_{j_o k_o}} \cdot w_{j_o} \right)^y \quad (6)$$

where $1 \le j_o \le m, 1 \le k_o \le s$.

► Combination and Aggregation: The data owner randomly chooses m elements $\{\varepsilon_{ij\lambda}\}_{\lambda=1}^m$ from $GF(p)$ to be coefficients and linearly combines the augmented native blocks to generate coded blocks $v_{ij} (1 \le i \le n, 1 \le j \le \alpha)$, as follows:

$$v_{ij} = \sum_{\lambda=1}^{m} \varepsilon_{ij\lambda} \mathtt{w}_\lambda \in GF(p)^{s+m} \quad (7)$$

and apparently each symbol can be obtained by

$$v_{ijk} = \sum_{\lambda=1}^{m} \varepsilon_{ij\lambda} \overline{w}_{\lambda k} \in GF(p) \quad (8)$$

with $1 \le k \le s+m$. Then we can get the aggregated tags as:

$$\sigma^*_{ijk} = \prod_{\lambda=1}^{m} (\sigma^{**}_{\lambda k})^{\varepsilon_{ij\lambda}} = \left(u^{v_{ijk}} \cdot \prod_{\lambda=1}^{m} w_\lambda^{\varepsilon_{ij\lambda}} \right)^y \quad (9)$$

with $1 \le k \le s$.

► Authenticator Generation: The data owner generates an authenticator for v_{ijk} as:

$$\begin{aligned} \sigma_{ijk} =& H(ID||i||j||k)^x \cdot \sigma^*_{ijk} \\ =& H(ID||i||j||k)^x \left(u^{v_{ijk}} \prod_{\lambda=1}^{m} w_\lambda^{\varepsilon_{ij\lambda}} \right)^y \end{aligned} \quad (10)$$

where the symbols i, j, k denote the index of the server, the index of the block at a server and the index of a segment in a certain coded block.

Then, authenticator set $\Phi = \{\Phi_i = \{\sigma_{ijk}\}_{\substack{1 \le j \le \alpha \\ 1 \le k \le s}}\}_{1 \le i \le n}$ and coded block set $\Psi = \{\Psi_i = \{v_{ij}\}_{1 \le j \le \alpha}\}_{1 \le i \le n}$ are obtained. Finally the data owner distributes these two sets across n cloud servers, specifically sending $\{\Phi_i, \Psi_i, t\}$ to server i and delete them from local storage.

■**Audit**: For each of the n servers, TPA verifies the possession of α coded blocks by randomly checking samples of segments of every block and performing a batch verification.

$Challenge(\mathcal{F}_{info}) \to (\mathcal{C})$: Taking the information of file $\mathcal{F}_{info}$ as input, TPA randomly generates a c-element set $Q_i = \{(k^*_\xi, a^*_\xi)\}_{1 \le \xi \le c}$ for server i $(1 \le i \le n)$ with the sampled segment indexes $k^* \in_R [1, s]$ and corresponding randomness $a^* \xleftarrow{R} GF(p)$. Furthermore, to execute a batch auditing, TPA picks another random symbol set $\Lambda_i = \{a_\xi\}_{1 \le \xi \le \alpha}$ with each $a_\xi \xleftarrow{R} GF(p)$. TPA sends $\mathcal{C} = \{Q_i, \Lambda_i\}$ to the server i.

$ProofGen(\mathcal{C}, \Phi, \Psi) \to (\mathcal{P})$: Upon receiving the challenge $\mathcal{C} = \{Q_i, \Lambda_i\}$, server i first computes

$$\mu_{ij} = \sum_{\tau=1}^{c} a^*_\tau v_{ijk^*_\tau} \quad , \quad \sigma_{ij} = \prod_{\tau=1}^{c} \sigma_{ijk^*_\tau}{}^{a^*_\tau} \quad (11)$$

for coded block $j \in [1, \alpha]$ stored on itself, and then it generates an aggregated proof using Λ_i, i.e.,

$$\mu_i = \sum_{j=1}^{\alpha} a_j \mu_{ij} \quad , \quad \sigma_i = \prod_{j=1}^{\alpha} \sigma_{ij}{}^{a_j} \quad (12)$$

For the coefficients checking, server i generates m symbols as

$$\rho_{i\lambda} = \left(\sum_{j=1}^{\alpha} a_j \varepsilon_{ij\lambda} \right) \cdot \sum_{\tau=1}^{c} a^*_\tau \quad (13)$$

where $1 \le \lambda \le m$. Let $\Theta_i = \{\rho_{i1}, \rho_{i2}, ..., \rho_{im}\}$.

Thus the ith server responds to the TPA with proof $\mathcal{P} = \{\mu_i, \sigma_i, \Theta_i, t\}$.

$Verify(\mathcal{P}, pk, \mathcal{C}) \rightarrow (0,1)$: Upon receiving a proof $\mathcal{P}$ from server i, TPA uses spk to verify the signature on t to check whether or not it is the delegated file that needs auditing, retrieves u and $\Gamma = \{w_1, w_2, ...w_m\}$ if successful and aborts if otherwise.

Then, TPA computes:

$$RHS = e\left(\prod_{j=1}^{\alpha}\prod_{\tau=1}^{c} H_{ijk_\tau^*}^{a_j a_\tau^*}, pk_x\right) \cdot e\left(u^{\mu_i}, pk_y\right) \tag{14}$$

where $H_{ijk_\tau^*} = H(ID||i||j||k_\tau^*)$. Finally, TPA verify if the following equation holds:

$$e\left(\prod_{\lambda=1}^{m} w_\lambda^{-\rho_{i\lambda}}, pk_y\right) \cdot e\left(\sigma_i, g\right) \stackrel{?}{=} RHS \tag{15}$$

if so, output 1; otherwise, output 0.

■**Repair**: As previously introduced, the data owner empowers a proxy to take charge of faulty server reparation and moves off-line once it completes its upload procedure. When TPA detects a server corruption, an alert will be sent to proxy and then a repair procedure will be triggered. Moreover, to avoid pollution attack, the ℓ blocks downloaded from servers will be first verified before they are used for block regeneration. Without loss of generality, we assume that the TPA has identified a certain faulty server η.

$ClaimForRep(\mathcal{F}_{info}) \rightarrow (\mathcal{C}_r)$: The proxy randomly contacts ℓ healthy servers $\{i_\theta\}_{1\le\theta\le\ell}$. For each server $i \in \{i_\theta\}_{1\le\theta\le\ell}$, the proxy draws a coefficient set $\Lambda_i = \{a_\xi\}_{1\le\xi\le\alpha}$ with $a_\xi \in GF(p)$. Then it sends claim $\mathcal{C}_r = \{\Lambda_i\}$ to server i.

$GenForRep(\mathcal{C}_r, \Phi, \Psi) \rightarrow (BA)$: When server i receives the repair claim $\mathcal{C}_r$, it linearly combines α local coded blocks into one single block and then generates tags for verification:

$$\widetilde{v}_i = \sum_{j=1}^{\alpha} a_j v_{ij} \quad , \quad \widetilde{\sigma}_{ik} = \prod_{j=1}^{\alpha} {\sigma_{ijk}}^{a_j} \tag{16}$$

where $(1 \le k \le s)$. Then server i responds with the block and authenticators set $BA_i = \{\widetilde{v}_i, \{\widetilde{\sigma}_{ik}\}_{1\le k\le s}\}$.

$BlockAndSigReGen(\mathcal{C}_r, BA) \rightarrow (\Phi', \Psi', \perp)$: Before regeneration, the proxy will execute batch verification for the received blocks. By viewing each $\widetilde{v}_i$ as $(\widetilde{v}_{i1}, ..., \widetilde{v}_{is}, \widetilde{\varepsilon}_{i1}, ..., \widetilde{\varepsilon}_{im})$ consulting Eq.(5), the proxy verifies whether the k following equations hold,

$$\begin{aligned} e(\prod_{i\in\{i_\theta\}} \widetilde{\sigma}_{ik}, g) \stackrel{?}{=} & e\left(\prod_{i\in\{i_\theta\}}\prod_{j=1}^{\alpha} H_{ijk}^{a_j}, pk_x\right) \cdot \\ & e\left(u^{\sum_{i\in\{i_\theta\}} \widetilde{v}_{ik}}, pk_y\right) \cdot \\ & e\left(\prod_{\lambda=1}^{m} w_\lambda^{\sum_{i\in\{i_\theta\}} \widetilde{\varepsilon}_{i\lambda}}, pk_y\right) \end{aligned} \tag{17}$$

where $1 \le k \le s$ and $H_{ijk} = H(ID||i||j||k)$. If verification fails, which means that some of the ℓ servers connected are malicious for repair, proxy aborts the reparation; otherwise, it continues to generate new coded blocks and authenticators.

Assuming that the repaired coded blocks would be stored at a new added server η', the proxy rebuilds α blocks as follows: The proxy picks up ℓ random coefficients $z_i \stackrel{R}{\leftarrow} GF(p)$ for each $i \in \{i_\theta\}_{1\le\theta\le\ell}$ and then computes a linear combination

$$v_{\eta' j} = \sum_{i\in\{i_\theta\}} z_i \widetilde{v}_i \tag{18}$$

and regenerate authenticators for each segment,

$$\sigma_{\eta' jk} = \mathcal{T}_{jk}^{x} \cdot \prod_{i\in\{i_\theta\}} (\widetilde{\sigma}_{ik})^{z_i} \tag{19}$$

where $1 \le j \le \alpha, 1 \le k \le s$ and the transform operator $\mathcal{T}_{jk}$ denotes

$$\mathcal{T}_{jk} = \frac{H(ID||\eta'||j||k)}{\prod_{i\in\{i_\theta\}}\prod_{j^*=1}^{\alpha} H(ID||i||j^*||k)^{a_{j^*} z_i}} \tag{20}$$

Finally, the regenerated block set $\Psi' = \{v_{\eta' j}\}_{1\le j\le\alpha}$ and authenticator set $\Phi' = \{\sigma_{\eta' jk}\}_{\substack{1\le k\le s \\ 1\le j\le\alpha}}$ are sent to server η'. Thus, the repair process is finished.

Example Description: To make our contribution easier to follow, we briefly introduce our above scheme under the reference scenario in Section II-B: The staffs (i.e., cloud users) first generate their public and private keys, and then delegate the authenticator regeneration to a proxy(a cluster or powerful workstation provided by the company) by sharing partial private key. After producing encoded blocks and authenticators, the staffs upload and distribute them to the cloud servers. Since that the staffs will be frequently off-line, the company employs a trust third party(the TPA) to interact with the cloud and perform periodical verification on the staffs' data blocks in a sampling mode. Once some data corruption is detected, the proxy is informed, it will act on behalf of the staffs to regenerate the data blocks as well as corresponding authenticators in a secure approach. So we could see that our scheme guarantees that the staffs can use the regenerating-code-based cloud in a practical and lightweight way, which completely releases the staffs from online burden for data auditing and reparation.

Enabling Privacy-Preserving Auditable: The privacy protection of the owner's data can be easily achieved through integrating with the random proof blind technique [15] or other technique [9]. However, all these privacy-preservation methods introduce additional computation overhead to the auditor, who usually needs to audit for many clouds and a large number of data owners; thus, this could possibly make it create a performance bottleneck. Therefore, we prefer to present a novel method, which is more light-weight, to mitigate private data leakage to the auditor. Notice that in a regenerating-code-based cloud storage, data blocks stored at servers are coded as linear combinations of the original blocks $\{\overline{w}_i\}_{i=1}^{m}$ with random coefficients. Supposing that the curious TPA has recovered m coded blocks by elaborately performing *Challenge-Response* procedures and solving systems of linear equations [14], the TPA still requies to solve another group of m linearly independent equations to derive the m native blocks. We can utilize a keyed pseudorandom function $f_{key}(\cdot) : \{0,1\}^* \times K \rightarrow GF(p)$ to mask the coding coefficients and thus prevent the TPA from correctly obtaining

the original data. Specifically, the data owner maintains a secret key κ for $f_\kappa(\cdot)$ in the beginning of the **Setup** procedure and augments m original data blocks as

$$\mathtt{w}_i = (\overline{w}_{i1}, \overline{w}_{i2}, ..., \overline{w}_{is}, \overbrace{\underbrace{0, ..., 0, f_\kappa(i)}_{i}, 0, ..., 0}^{m}) \tag{21}$$

thus every coded blocks can be represented as

$$v = (\underbrace{v_{i1}, v_{i2}, ..., v_{is}}_{\substack{data \\ part}}, \underbrace{\varepsilon'_1, ..., \varepsilon'_m}_{\substack{masked \\ coefficients}}) \in GF(p)^{s+m} \tag{22}$$

where $\varepsilon'_i = f_\kappa(i) \cdot \varepsilon_i$, i.e.,the ith coefficient of a coded block is masked with $f_\kappa(i)$. The TPA is unable to recover correct coding coefficients as the key κ is kept secret by the data owner. Without the knowledge of coding coefficients, the TPA can not retrieve the original data blocks $\{\overline{\mathtt{w}}_i\}_{i=1}^m$ correctly, thus realizing the privacy-preserve. Specifically, this method also guarantees that the proxy can not obtain the private original data either.

Apparently, our novel privacy-preserve method can be easily integrated with the above primitive auditing scheme. We only need to modify the "Augmentation" step of $SigAndBlockGen(\cdot)$ as Eq.(21) and leave the rest unchanged. The experiment in Section V-B shows the efficiency of this method.

Mitigating The Overhead Of Data Owner : Despite that the data owner has been released from online burden for auditing and repairing, it still makes sense to reduce its computation overhead in the **Setup** phase because data owners usually maintain very limited computational and memory resources. As previously described, authenticators are generated in a new method which can reduce the computational complexity of the owner to some extent; however, there exists a much more efficient method to introduce further reduction.

Considering that there are so many modular exponent arithmetic operations during the authenticator generation, the data owner can securely delegate part of its computing task to the proxy in the following way: The data owner first properly augments the m native blocks, signs for them as Eq.(6), and thus obtains the $\{\sigma^{**}_{j_o k_o}\}_{\substack{1 \le j_o \le m \\ 1 \le k_o \le s}}$, then it sends the augmented native blocks and $\{\sigma^{**}_{j_o k_o}\}$ to the proxy. After receiving from the data owner, the proxy implements the last two steps of *SigAndBlockGen(·)* and finally generates entire authenticators σ_{ijk} for each segment v_{ijk} with secret value x. In this way, the data owner can migrate the expensive encoding and authenticator generation task to the proxy while itself maintaining only the first two lightweight steps; thus, the workload of data owner can be greatly mitigated. A detailed analysis will be shown in Section V. Besides, Theorem 4 in Section IV argues that the proxy can not forge valid authenticators for invalid segments with non-negligible probability.

A Tradeoff Between Storage And Communication: In our auditing scheme described above, we assume that each segment contains only one symbol for simplicity and is accompanied by an authenticator of equal length. This approach gives a storage overhead twice as much as the length of the data block(*i.e.*,$2 \cdot s \cdot |p|$ bits), and the server's response in each epoch is $(m+2) \cdot |p|$ bits. As noted in paper [12], we can introduce a parameter ζ that gives a tradeoff between storage overhead and response length: By denoting each segment as a collection of ζ symbols and computing an authenticator for each segment, the server storage overhead can be reduced by a factor of ζ (*i.e.*,$(1 + \frac{1}{\zeta}) \cdot s \cdot |p|$) bits, while the communication overhead will be increased to $(m + \zeta + 1) \cdot |p|$ bits. We omit a detailed description of the variant scheme due to space limitations.

Detection Probability: The TPA performs random spot checking on each coded block to improve the efficiency of our auditing scheme, while still achieving detection of faulty servers with high probability [2]. Supposing that an adversary is able to corrupt data blocks as well as coefficients at a certain server with probability p_1 and p_2 respectively, we can compute the probability of detection as $\mathcal{P}_\mathcal{D} = 1-(1-p_1)^{c\alpha}(1-p_2)^{m\alpha}$, where c denotes the number of segments selected from each coded block during each Audit phase. More specifically, if the corruption proportion is set to be $p_1 = 0.5\%, p_2 = 0.2\%$, and the system parameter is chosen as $m = 15, \alpha = 5$, then the TPA can detect the corrupted server with a probability 99% or 95%, where the number of sampled segments c in each coded block is 185 or 120, respectively.

IV. Security Analysis

In this section, we first elaborate on the correctness of verification in our auditing scheme and then formally evaluate the security by analyzing its fulfillment of soundness, regeneration-unforgeable and secure guarantee against replay attack.

A. Correctness

There are two verification process in our scheme, one for spot checking during the **Audit** phase and another for block integrity checking during the **Repair** phase.

Theorem 1*: Given a cloud server i storing data blocks Ψ_i and accompanied authenticators Φ_i, TPA is able to correctly verify its possession of those data blocks during audit phase, and the proxy can correctly check the integrity of downloaded blocks during repair phase.*

Proof: Proving the correctness of our auditing scheme is equivalent of proving that Eq.(15) and Eq.(17) is correct. The correctness of Eq.(15) is shown in Appendix A and Eq.(17) in Appendix B.□

B. Soundness

Following from paper [12], we say that our auditing protocol is sound if any cheating server that convinces the verification algorithm that it is storing the coded blocks and corresponding coefficients is actually storing them.

Before proving the soundness, we will first show that the authenticator as Eq.(10) is unforgeable against malicious cloud servers(referred to as *adversary* in the following definition) under the *random oracle model*. Similar to the standard notion of security for a signature scheme [24], we give the formal

definition of security model for our authenticator in Definition 1[5].

Definition 1: *Our authenticator as Eq.(10) is existentially unforgeable under adaptive chosen message attacks if no PPT adversary has a non-negligible advantage in the following game-**Game 0:***

1. Setup: The challenger runs the KeyGen() algorithm on input 1^κ and gives the public parameter (pk_x, pk_y) to adversary $\mathcal{A}$.

2. Queries: The challenger maintains the following oracles which can be queried by the adversary $\mathcal{A}$:

- *Hash Oracle($\mathcal{O}_{hash}$): provide result of $H(\cdot)$ for hash queries.*
- *UW Oracle($\mathcal{O}_{uw}$): produce the random parameters u, $w_\lambda (1 \le \lambda \le m)$ for signature or forgery.*
- *Sign Oracle($\mathcal{O}_{sign}$): produce authenticators on adaptively chosen tuple $\{ID, i, j, k, v_{ijk}, \{\varepsilon_{ij\lambda}\}_{\lambda=1}^m\}$, and return an authenticator σ_{ijk}.*

3. Output: Eventually, the adversary $\mathcal{A}$ produces a tuple $\{ID^*, i,^* j^*, k^*, v^*, \{\varepsilon_\lambda^*\}_{\lambda=1}^m, \sigma^*\}$, and we say that $\mathcal{A}$ wins the game if Eq.(23) holds while the input tuple $\{ID^*, i,^* j^*, k^*, v^*, \{\varepsilon_\lambda^*\}_{\lambda=1}^m\}$ was not submitted to the Sign Oracle $\mathcal{O}_{sign}$.

$$e(\sigma^*, g) = e\left(H_{i^*j^*k^*}, pk_x\right) \cdot e\left(u^{v^*} \prod_{\lambda=1}^{m} w_\lambda^{\varepsilon_\lambda^*}, pk_y\right) \quad (23)$$

In the following theorem, we will show that our authenticator is secure under the CDH(Computational Diffie-Hellman) assumption.

Theorem 2: *If the adversary $\mathcal{A}$ is able to win Game 0 with non-negligible advantage ϵ after at most q_h hash queries and q_{uw} UW queries($q_{uw} = q_h$ implied by our assumption in footnote 5), then there exist a PPT algorithm $\mathcal{S}$ that can solve the CDH problem with non-negligible probability ϵ'.*[6]

Proof: See Appendix C.□

The formal definition of auditing soundness is shown in Definition 2.

Definition 2: *Our auditing scheme is sound if no PPT adversary has a non-negligible advantage in the following game-**Game 1:***

1. Setup: The challenger runs the KeyGen() algorithm on input 1^κ and gives the public parameter (pk_x, pk_y) to adversary $\mathcal{A}$

2. Store Query: The adversary $\mathcal{A}$ queries store oracle on data F^*, the challenger implements the SigAndBlockGen() algorithm of our protocol and returns $\{\Phi_i^*, \Psi_i^*, t\}$ to $\mathcal{A}$.

3. Challenge: For any F^* on which $\mathcal{A}$ previously made a store query, the challenger generates a challenge $\mathcal{C} = \{Q_i, \Lambda_i\}$ and requests $\mathcal{A}$ to provide a proof of possession for the selected segments and corresponding coefficients.

4. Forge: Suppose the expected proof $\mathcal{P}$ should be $\{\mu_i, \sigma_i, \{\rho_{i\lambda}\}_{\lambda=1}^m\}$, which can pass the verification with Eq.(15). However, the adversary $\mathcal{A}$ generates a forgery of the proof as $\mathcal{P}' = \{\mu_i', \sigma_i', \{\rho_{i\lambda}'\}_{\lambda=1}^m\}$. If the $\mathcal{P}'$ can still pass the verification in the following two case:

- *Case 1: $\sigma_i' \neq \sigma_i$;*
- *Case 2: $\sigma_i' = \sigma_i$, but at least one of the following inequalities should be satisfied: $\mu_i' \neq \mu_i$, $\rho_{i\lambda}' \neq \rho_{i\lambda}$ with $1 \le \lambda \le m$;*

then adversary $\mathcal{A}$ wins this game, otherwise, it fails.

Theorem 3: *If the authenticator scheme is existentially unforgeable and adversary $\mathcal{A}$ is able to win Game 1 with non-negligible probability ϵ, then there exist a PPT algorithm $\mathcal{S}$ that can solve the CDH problem or DL(Discrete Logarithm Problem) problem with non-negligible probability ϵ'.*

Proof: See Appendix D.□

C. Regeneration-unforgeable

Noting that the semi-trusted proxy handles regeneration of authenticators in our model, we say our authenticator is *regeneration-unforgeable* if it satisfies the following theorem.

Theorem 4: *The adversary(or proxy) can only regenerate a forgery of authenticator for invalid segment from certain coded block(augmented) and pass the next verification with negligible probability, except that it implements the Repair procedure correctly.*

Proof: See Appendix E.□

D. Resistant to Replay Attack

Theorem 5: *Our public auditing scheme is resistant to replay attack mentioned in [[7], Appendix B.], since the repaired server maintains identifier η' which is different with the corrupted server η.*

V. Evaluation

A. Comparison

Table II lists the features of our proposed mechanism and makes a comparison with other remote data checking schemes [7], [8] for regenerating-coding-based cloud storage. The security parameter κ is eliminated in the costs estimation for simplicity. While the previously presented schemes in [7] and [8] are designed for private verification, ours allows anyone to challenge the servers for data possession while preserving the privacy of the original data. Moreover, our scheme can completely release the owners from online burden compared with schemes in [7], [8] where data owners should stay online for faulty server reparation. To localize the faulty server during the audit phase, [8] suggested a traversal method and thus demanded many times of auditing operation with complexity $O(C_n^k(n-k)\alpha)$ before the auditor can pick up the wrong server, while our scheme is able to localize the faulty server by a one-time auditing procedure.

For the storage overhead of each cloud server, our scheme performs slightly better than Chen Bo's as the servers do not store the "*repair tags*" [7], but computes them on the fly. For communication overhead during the audit phase, both

[5]Notice that in the definition, we modeled the randomness u, $w_\lambda (1 \le \lambda \le m)$ as a random oracle $\mathcal{O}_{uw}$ just as Dan Boneh *et al.* did in [20](Section 4). Besides, we assume that $\mathcal{A}$ is well-behaved that it always queries the oracles $\mathcal{O}_{hash}$ and $\mathcal{O}_{uw}$ before it requests a signature for adaptively chosen input tuple, and that it always query the $\mathcal{O}_{hash}$ and $\mathcal{O}_{uw}$ before it outputs the forgery [17].

[6]In this paper we only give a sketch proof due to space limitation, a rigorous version can be conducted similar with the proof in [17](Section 4).

TABLE II
COMPARISON OF DIFFERENT AUDIT SCHEMES FOR REGENERATING-CODE-BASED CLOUD STORAGE

Items	Chen Bo [7]	Henry C. H. Chen [8]	Ours
Public Auditability	No	No	Yes
Privacy Preserving	Yes	Yes	Yes
Owners off-line support	No	No	Yes
Time for Faulty server localization	$O(1)$	$O(C_n^k(n-k)\alpha)$	$O(1)$
Server Storage Overhead	$(s\alpha+m\alpha+\alpha)\|p\|$	$(nm\alpha+s(\frac{n'-k'}{k'})\alpha+\alpha)\|p\|^\sharp$	$(s\alpha+m\alpha)\|p\|$
Communication Overhead(Audit)	$(m+2)\alpha\|p\|$	$(nm+c)\alpha\|p\|$	$(m+2)\|p\|$
Communication Overhead(Repair)	$(m\alpha+s+1)\|p\|$	$(nm\alpha+s+1)\|p\|$	$(2s+m)\|p\|$

s is the number of segments of each block(we assume one segment contains one symbol for simplicity); m is the number of native blocks that the file splits into; α is the number of blocks stored in a server; c is the sampled segments in each block; k' and n' are parameters for AECC(Adversary Error Correcting Code); $\sharp$ denotes that the storage cost of [8] is for the parities of (n', k')-AECC and the metadata file which mainly contains all the coefficients, and MACs for each block.

[7] and ours generate aggregated proofs for blocks verification, thus significantly reducing the bandwidth compared with [8]. Besides, the communication cost of our scheme is lower than Chen Bo's [7] by a factor of α, because we generate a single aggregated proof for all α blocks at each server, while [7] checks the integrity of each one separately. Nevertheless, the bandwidth cost during the repair procedure of our scheme is higher; this comes from the transmission of authenticators(total number s) for each segment of the block generated by a helper server. These authenticators are not only for verifying the integrity of the received blocks but also for the regeneration of authenticators for repaired blocks; this type of overhead can be reduced by reducing the number of authenticators which can be achieved through increasing the parameter $\zeta \geq 1$.

B. Performance Analysis

We focus on evaluating the performance of our privacy-preserving public audit scheme during the Setup, Audit and Repair procedure. In our experiments, all the codes are written in C++ language on an OpenSUSE 12.2 Linux platform with kernel version 3.4.6-2-desktop and compiler version g++ 4.7.1. All entities in our prototype (as shown in Fig.2) are represented by PCs with Intel Core i5-2450 2.5GHz, 8G DDR3 RAM and a 7200 RPM Hitachi 500G SATA drive. The implementation of our algorithms uses open source PBC (Pairing-Based Cryptography) Library version 0.5.14, GMP version 5.13 and Openssl version 1.0.1e. The elliptic curve utilized here is Barreto-Naehrig curve [25], with base field size of 160 bits and embedding degree 12. The security level is chosen to be 80 bits and thus $|p| = 160$. All the experimental results represent the mean of 10 trials. In addition, the choice of parameters $(n, k, \ell, \alpha, \beta)$ for regenerating codes is in reference [7]. Notice that we only focus on the choice of $\beta = 1$ and $\zeta = 1$ here, from which similar performance results can be easily obtained for other choices of β and ζ.

1) Setup computational complexity: During the Setup phase, the authenticators are generated in a novel method instead of computing an authenticator for each segment of every coded block independently(meaning that the file is first encoded and then authenticator is directly computed for each segment as Eq.(10), we call this the straightforward approach hereafter). Fixing the parameters $(n = 10, k = 3, \ell = 3, \alpha = 3, \beta = 1)$, we assume that the original data file is split into $m = 6$ native blocks and would be encoded into $n\alpha = 30$ coded blocks. First we evaluate the running time of the data owner for auditing system setup. We mainly measure the time cost of the block and authenticator generation in this experiment.

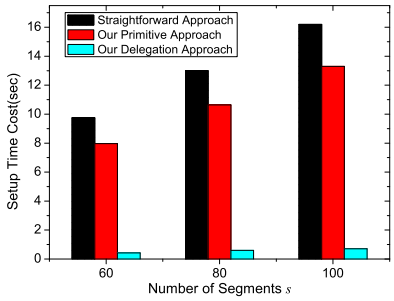

Fig. 5. Time for system setup with different segment number s

Fig.5 compares the running time of setup phase utilizing three different authenticator generation methods(with variable $s = 60, 80, 100$): the straightforward approach, our primitive approach(described in Section III-B "**Setup**") and our delegation approach(described in Section III-B "**Mitigating The Overhead Of Data Owner**"). Apparently, all the three approaches introduce higher time cost with larger s, as there are more units need to sign; nevertheless, ours is always more efficient than the straightforward approach. The reason is that our primitive generation method decreases the times of the time-consuming modular exponent operations to $n\alpha s(m+1)+2m$ compared to the straightforward approach, which requires $n\alpha s(m+3)$ operations totally. However, it is still intensive resource consuming and even unaffordable when data owners use resource limited equipment(*e.g.,* Tablet PC *et al.*) to sign their data blocks and upload them. Thus, we introduce another delegation method to solve this problem. Experimental result shows that our delegation method releases the data owner from heavy computational overhead; the computational complexity of data owner is reduced to about $1/18$ of that with our primitive method, which makes our schemes more flexible and practical for low power cloud users.

Moreover, we evaluate the efficiency of our privacy-

TABLE III
COMPUTATIONAL COST INTRODUCED BY THE PRIVACY-PRESERVING METHOD

	Without privacy preserving	With privacy preserving
$s = 60$	7966 ms	7994 ms
$s = 80$	10642 ms	10653 ms
$s = 100$	1329 6ms	13301 ms

preserving method and the results in Table III show that our design is perfectly lightweight for the data owner to execute. Because our privacy-preservation method is implemented only once during the whole life of a user's file, while the random blind process in [15] [9] would be performed in each Audit instance, apparently our scheme is much more efficient and thus we do not experimentally compare their performance. In addition, [16] introduced a similar privacy preserve method for their public auditing protocol. Both their scheme and ours utilize random mask in linear-code-based distributed storage to avoid the auditor getting enough information for original data retrieve. However, there is one significant difference between the two, *i.e.*, [16]'s method used the PRF to blind the data blocks before the encoding process, while ours choose to mask the coding coefficients instead. Numberically comparing them, we can see that they need to execute PRF for $s \cdot m$ times and m times during the **Setup** phase, separately. Thus our privacy preserve method is more efficient and effective than [16].

2) Audit computational complexity: Considering that the cloud servers are usually powerful in computing capacity, we focus on analyzing the computational overhead on the auditor side and omit those on the cloud side. Theoretically, verification for an aggregated proof requires less pairing operations, which is the most expensive operation compared to modular exponentiations and multiplication, than for separate α proofs. To get a complete view of the batching efficiency, we conduct two timed batch auditing tests, the first is implemented with variable α from 1 to 8, and other parameters are set as $n = 10$, $c = 25$, $s = 60$, $k = \alpha$ and $m = (k^2 + k)/2$, while the second one maintains the number of spot segment per block c from 1 to 25 with fixed $n = 10$, $s = 60$, $k = 3$, $m = 6$ and $\alpha = 3$. Fig.6 and Fig.7 show the time cost of auditing tasks for a single server, and it can be found that the batch auditing indeed helps reduce the TPA's computational complexity by a factor of approximately α.

3) Repair computational complexity: The regeneration of the faulty blocks and authenticators is delegated to a proxy in our auditing scheme; we now experimentally evaluate its performance here. Fixing the parameters as $n = 10, k = 3, \ell = 3, \alpha = 3$ and letting s vary to be 60, 80 and 100, we measure the running time of three sub-processes: Verification for Repair, Regeneration for Blocks and Regeneration for Authenticators, thus obtaining the results in Fig.8. Obviously, it takes the proxy much more time to verify the received blocks for repair, less to regenerate the authenticators, and negligible to regenerate the faulty blocks. This situation occurs mainly because there are a mass of expensive pairing operations and less expensive modular exponentiations during the verification, thus leading to the most time consuming sub-process. In contrast, there are only asymptotic $\alpha s(\ell + \alpha\ell + 1)$ modular exponentiations and αs inversions in a finite group across the authenticator regeneration, and only $\alpha s\ell$ multiplications during the block regeneration. However, the efficiency of the verification can be significantly improved using a batch method, where we can randomly choose s weights for each received blocks, aggregate the segments and authenticators, and then verify them all in one, thus we can avoid applying so many pairing operations.

VI. RELATED WORK

The problem of remote data checking for integrity was first introduced in [26], [27]. Then Ateniese *et al.* [2] and Juels *et al.* [3] gave rise to the similar notions *provable data possession (PDP)* and *proof of retrievability (POR)*, respectively. Ateniese *et al.* [2] proposed a formal definition of the PDP model for ensuring possession of files on untrusted storage, introduced the concept of RSA-based homomorphic tags and suggested randomly sampling a few blocks of the file. In their subsequent work [28], they proposed a dynamic version of the prior PDP scheme based on MAC, which allows very basic block operations with limited functionality but block insertions. Simultaneously, Erway *et al.* [29] gave a formal framework for dynamic PDP and provided the first fully dynamic solution to support provable updates to stored data using rank-based authenticated skit lists and RSA trees. To improve the efficiency of dynamic PDP, Q. Wang [30] proposed a new method which uses merkle hash tree to support fully dynamic data.

To release the data owner from online burden for verification, [2] considered the public auditability in the PDP model for the first time. However, their variant protocol exposes the linear combination of samples and thus gives no data privacy guarantee. Then C. Wang *et al.* [14], [15] developed a random blind technique to address this problem in their BLS signature based public auditing scheme. Similarly, Solomon *et al.* [31] introduced another privacy-preserving method, which is more efficient since it avoids involving a computationally intensive pairing operation for the sake of data blinding. Yang *et al.* [9] presented a public PDP scheme, where the data privacy is provided through combining the cryptography method with the bilinearity property of bilinear pairing. [16] utilized random mask to blind data blocks in error-correcting coded data for privacy-preserving auditing with TPA. Zhu *et al.* [10] proposed a formal framework for interactive provable data possession(IPDP) and a zero-knowledge IPDP solution for private clouds. Their ZK-IPDP protocol supports fully data dynamics, public verifiability and is also privacy-preserving against the verifiers.

Considering that the PDP model does not guarantee the retrievability of outsourced data, Juels *et al.* [3] described a POR model, where spot-checking and error correcting codes are used to ensure both "possession" and "retrievability" of data files on remote archive service systems. Later, Browers *et al.* [32] proposed an improved framework for POR protocols that generalizes Juels' work. Dodis *et al.* [33] also gave a study on different variants of POR with private auditability. A representative work upon the POR model is the CPOR

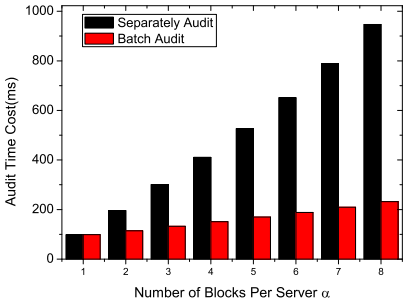

Fig. 6. Time for Audit with different α

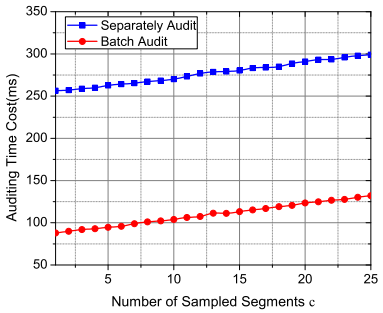

Fig. 7. Time for Audit with different c

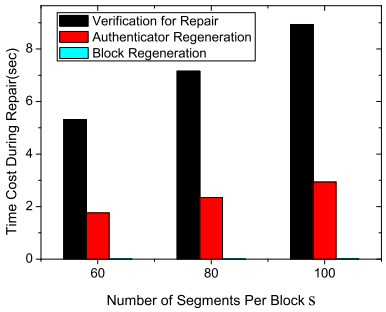

Fig. 8. Time for Repair with different s

presented by Shacham and Waters [12] with full proofs of security in the security model defined in [3]. They utilize the publicly verifiable homomorphic linear authenticator built from BLS signatures to achieve public auditing. However, their approach is not privacy preserving due to the same reason as [2].

To introduce fault tolerance in practical cloud storage, outsourced data files are commonly striped and redundantly stored across multi-servers or even multi-clouds. It is desired to design efficient auditing protocols for such settings. Specifically, [4]–[6] extend those integrity checking schemes for the single-server setting to the multi-servers setting under replication and erasure coding respectively. Both B. Chen *et al.* [7] and H. Chen *et al.* [8] make great effort to design auditing schemes for regenerating-code-based cloud storage, which is similar to our contribution except that ours release the data owner from online burden for verification and regeneration. Furthermore, Zhu *et al.* [10] proposed an efficient construction of cooperative provable data possession(CPDP) which can be used in multi-clouds, and [9] extend their primitive auditing protocol to support batch auditing for both multiple owners and multiple clouds.

VII. Conclusion

In this paper, we propose a public auditing scheme for the regenerating-code-based cloud storage system, where the data owners are privileged to delegate TPA for their data validity checking. To protect the original data privacy against the TPA, we randomize the coefficients in the beginning rather than applying the blind technique during the auditing process. Considering that the data owner cannot always stay online in practise, in order to keep the storage available and verifiable after a malicious corruption, we introduce a semi-trusted proxy into the system model and provide a privilege for the proxy to handle the reparation of the coded blocks and authenticators. To better appropriate for the regenerating-code-scenario, we design our authenticator based on the BLS signature. This authenticator can be efficiently generated by the data owner simultaneously with the encoding procedure. Extensive analysis shows that our scheme is provable secure, and the performance evaluation shows that our scheme is highly efficient and can be feasibly integrated into a regenerating-code-based cloud storage system.

References

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, p. 13, 2009.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 598–609.

[3] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 584–597.

[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: Multiple-replica provable data possession," in *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*. IEEE, 2008, pp. 411–420.

[5] K. D. Bowers, A. Juels, and A. Oprea, "Hail: a high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 187–198.

[6] J. He, Y. Zhang, G. Huang, Y. Shi, and J. Cao, "Distributed data possession checking for securing multiple replicas in geographically-dispersed clouds," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1345–1358, 2012.

[7] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*. ACM, 2010, pp. 31–42.

[8] H. Chen and P. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 2, pp. 407–416, Feb 2014.

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

[10] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 12, pp. 2231–2244, 2012.

[11] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

[12] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Advances in Cryptology-ASIACRYPT 2008*. Springer, 2008, pp. 90–107.

[13] Y. Hu, H. C. Chen, P. P. Lee, and Y. Tang, "Nccloud: Applying network coding for the storage repair in a cloud-of-clouds," in *USENIX FAST*, 2012.

[14] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

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

[16] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards secure and dependable storage services in cloud computing," *Service Computing, IEEE Transactions on*, vol. 5, no. 2, pp. 220–232, May 2012.
[17] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004.
[18] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.
[19] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, 2006.
[20] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *Public Key Cryptography–PKC 2009*. Springer, 2009, pp. 68–87.
[21] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology‖CRYPTO 2001*. Springer, 2001, pp. 213–229.
[22] A. Miyaji, M. Nakabayashi, and S. Takano, "New explicit conditions of elliptic curve traces for fr-reduction," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 84, no. 5, pp. 1234–1243, 2001.
[23] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin, "Secure network coding over the integers," in *Public Key Cryptography–PKC 2010*. Springer, 2010, pp. 142–160.
[24] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen message attacks," *SIAM Journal of Computing*, vol. 17, no. 2, pp. 281–308, 1988.
[25] P. S. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Selected areas in cryptography*. Springer, 2006, pp. 319–331.
[26] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking," in *Integrity and Internal Control in Information Systems VI*. Springer, 2004, pp. 1–11.
[27] D. L. Gazzoni Filho and P. S. L. M. Barreto, "Demonstrating data possession and uncheatable data transfer." *IACR Cryptology ePrint Archive*, vol. 2006, p. 150, 2006.
[28] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*. ACM, 2008, p. 9.
[29] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 213–222.
[30] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Computer Security–ESORICS 2009*. Springer, 2009, pp. 355–370.
[31] S. G. Worku, C. Xu, J. Zhao, and X. He, "Secure and efficient privacy-preserving public auditing scheme for cloud storage," *Computers & Electrical Engineering*, 2013.
[32] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 43–54.
[33] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Theory of Cryptography*. Springer, 2009, pp. 109–127.

Jian Liu received the Bachelor's degree and Master's degree from the National University of Defense Technology in 2009 and 2011 respectively. He is currently working toward the PhD degree in the CEMEE State Key Laboratory at the National University of Defense Technology. His research interests include secure storage and computation outsourcing in cloud computing and security of distributed systems.

Kun Huang received the Bachelor's degree and Master's degree from the National University of Defense Technology in 2010 and 2012 respectively. He is working toward the PhD degree in the CEMEE State Key Laboratory at the National University of Defense Technology and is currently studying abroad at University of Melbourne with advisor Professor Udaya Parampalli. His research interests include applied cryptography, network encoding and cloud computing.

Hong Rong received the Bachelor's degree from South China University of Technology in 2011 and Master's degree from the National University of Defense Technology in 2013. He is currently working toward the PhD degree in the CEMEE State Key Laboratory at the National University of Defense Technology. His research interests include privacy preservation of big data, cloud computing and virtualization.

Huimei Wang received the Bachelor's degree from Southwest Jiaotong University in 2004 and Master's degree from the National University of Defense Technology in 2007. She got her Ph.D degree in Information and Communication Engineering in National University of Defense Technology in 2012. She is currently working in the CEMEE State Key Laboratory at the National University of Defense Technology as a lecturer. Her research interests include network security, cloud computing and distributed systems.

Ming Xian received the Bachelor's degree(1991), Master's degree(1995) and PhD degree(1998) from the National University of Defense Technology. Now he is a professor in the CEMEE State Key Laboratory at the National University of Defense Technology. His research focus on cryptography and information security, cloud computing, distributed systems, wireless sensor network and system modeling, simulation and evaluation.