International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Regenerating Code Based on Public Auditing Scheme with Security Preserving for Cloud Storage

Dr. K. Sundeep Kumar[1], Anupama P. V[2]

Head of the Department, Computer Science and Engineering, SEACET, Bangalore

M. Tech, Computer Science and Engineering, SEACET, Bangalore.

Abstract: ***To secure outsourced information in distributed storage against defilements, adding adaptation to non-critical failure to distributed storage together with information trustworthiness checking and disappointment reparation gets to be basic. As of late, recovering codes have picked up notoriety because of their lower repair transmission capacity while giving adaptation to non-critical failure. Existing remote checking techniques for recovering coded information just give private reviewing, requiring information proprietors to dependably stay online and handle inspecting, and in addition repairing, which is some of the time unrealistic. In this paper, we propose an open inspecting plan for the recovering code-based distributed storage. To tackle the recovery issue of fizzled authenticators without information proprietors, we present an intermediary, which is favored to recover the authenticators, into the conventional open evaluating framework model. In addition, we plan a novel open irrefutable authenticator, which is produced by a few keys and can be recovered utilizing incomplete keys. In this way, our plan can totally discharge information proprietors from online weight. Moreover, we randomize the encode coefficients with a pseudorandom capacity to safeguard information protection. Broad security examination demonstrates that our plan is provable secure under arbitrary prophet model and test assessment shows that our plan is exceptionally productive and can be attainably incorporated into the recovering code-based distributed storage.***

Keywords: data integrity, bandwidth, public auditing scheme, regenerating-code-based, regeneration problem and authenticator.

1. Introduction

Distributed storage is presently picking up prevalence since it offers an adaptable on-interest information outsourcing administration with engaging advantages: help of the weight for capacity administration, all inclusive information access with area autonomy, and evasion of capital consumption on equipment, programming, and individual systems for upkeeps, and so forth.,. In any case, this new worldview of information facilitating benefit additionally brings new security dangers toward clients information, along these lines making people or enterprisers still feel reluctant.

It is noticed that information proprietors lose extreme control over the destiny of their outsourced information; hence, the rightness, accessibility and trustworthiness of the information are being put at danger. From one viewpoint, the cloud administration is typically confronted with a wide scope of inner/outer enemies, who might malignantly erase or degenerate clients' information; then again, the cloud administration suppliers may act unscrupulously, endeavoring to conceal information misfortune or defilement and asserting that the documents are still accurately put away in the cloud for notoriety or financial reasons. In this manner it bodes well for clients to actualize an effective convention to perform periodical confirmations of their outsourced information to guarantee that the cloud to be sure keeps up their information accurately.

Numerous instruments managing the respectability of outsourced information without a nearby duplicate have been proposed under various framework and security models up to now. The most noteworthy work among these studies are the PDP (provable information ownership) model and POR (evidence of retrievability) model, which were initially proposed for the single-server situation by Ateniese et al. what's more, Juels and Kaliski, individually. Considering that documents are typically striped and needlessly put away crosswise over multi-servers or multi-mists, investigate honesty confirmation plans appropriate for such multi-servers or multi-mists setting with various repetition plans, for example, replication, eradication codes, and, all the more as of late, recovering codes.

In this paper, we concentrate on the honesty check issue in recovering code-based distributed storage, particularly with the utilitarian repair procedure. Comparative studies have been performed by Chen et al. what's more, Chen and Lee independently and freely. amplified the single-server CPOR plan (private form in) to the recovering code-situation; planned and executed an information honesty security (DIP) plan for FMSR-based distributed storage and the plan is adjusted to the flimsy cloud setting.1 However, them two are intended for private review, just the information proprietor is permitted to confirm the uprightness and repair the defective servers. Considering the extensive size of the outsourced information and the client's obliged asset capability, the tasks of auditing and reparation in the cloud can be formidable and expensive for the users. 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) . In particular, users may not want to go through the complexity in verifying and reparation. The auditing schemes in and imply the problem that users need to always stay online, which may impede its adoption in practice, especially for long-term archival storage.

To completely guarantee the information respectability and recovery the clients' calculation assets and in addition online weight, we propose an open reviewing plan for the recovering code-based distributed storage, in which the honesty checking and recovery (of fizzled information squares and authenticators) are executed by an outsider

reviewer and a semi-trusted intermediary independently for the benefit of the information proprietor. Rather than specifically adjusting the current open examining plan to the multi-server setting, we outline a novel authenticator, which is more fitting for recovering codes.

A few difficulties and dangers suddenly emerge in our new framework model with an intermediary, and security examination demonstrates that our plan functions admirably with these issues. In particular, our commitment can be condensed by the accompanying perspectives:

- We plan a novel homomorphism authenticator in view of BLS mark, which can be produced by a few mystery keys and confirmed freely. Using the straight subspace of the recovering codes, the authenticators can be registered effectively. In addition, it can be adjusted for information proprietors outfitted with low end calculation gadgets (e.g. Tablet PC and so forth.) in which they just need to sign the local pieces.
- To the best of our insight, our plan is the first to permit security protecting open inspecting for recovering code-based distributed storage. The coefficients are conceal by a PRF (Pseudorandom Function) amid the Setup stage to keep away from spillage of the first information. This technique is lightweight and does not acquaint any computational overhead with 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.*

2. Preliminaries and Problem Statement

a) Notations and Preliminaries

1) Regenerating Codes: Regenerating codes are first introduced by Dimakis *et al.* 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 MDS2-property. When data corruption at a server is detected, the client will contact _ healthy servers and download *β*_ bits from each server, thus regenerating the corrupted blocks without recovering the entire original file. Dimakis *et al.* [18] showed that the repair bandwidth *γ* _ = _*β*_ can be significantly reduced with _ ≥ *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, _, α_, γ_)*, we obtain:

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

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

Moreover, according to whether the corrupted blocks can be exactly regenerated, there are two versions of repair strategy: exact repair and functional repair. 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.

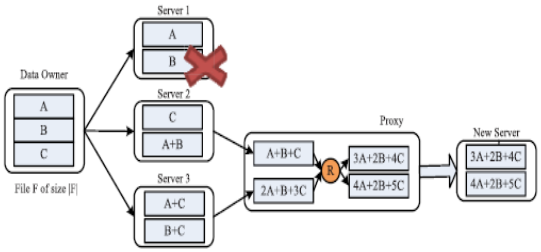

Fig. 1. An example of functional repair regenerating code with parameters $(n = 3, k = 2, \ell = 2, \alpha = 2, \beta = 1)$. The data owner computes six coded blocks as random linear combinations of the native three blocks, and distributes them across three servers. When Server 1 gets corrupted, the 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.

1)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: Before encoding, *F* is split into *m* blocks, and the original *m* s-dimensional vectors (or blocks indistinguishably) {w*i* ∈ *GF(p)*s}*mi* =1 are properly augmented as:

$$w_i = (\overline{w}_{i1}, \overline{w}_{i2}, \ldots, \overline{w}_{is}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{i}) \in GF(p)^{s+m}$$

For each symbol, *wi j* ∈ *GF(p)*. Eq.(3) shows that each original block w*i* is appended with the vector of length *m* containing a single „1" in the *i* th position and is otherwise zero. Then, the augmented vectors are encoded into *nα* coded blocks. Specifically, they are linearly combined and generate coded blocks with randomly chosen coefficients *εi* ∈ *GF (p)*.

$$v = \sum_{i=1}^{m} \varepsilon_i w_i \in GF(p)^{s+m}$$

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

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

Where *(vi*1, *vi*2, . . . , *vis)* 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 w1, w2, . . . , w*m*; all valid coded blocks appended with coding coefficients would belong to subspace *V* .

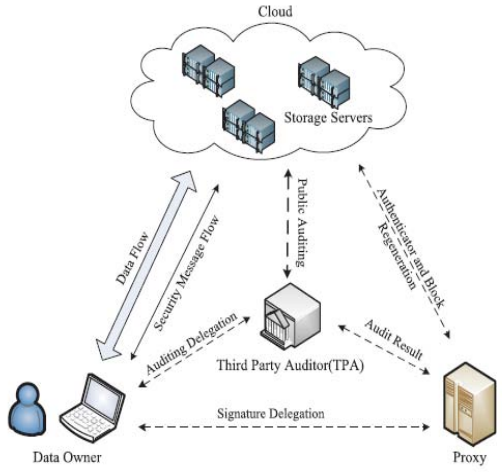

Fig. 2. The system model.

1) ***Bilinear Pairing Map:*** Let *G* and *GT* be multiplicative cyclic groups of the same large prime order *p*. A bilinear pairing map $e : G \times G \rightarrow GT$ is a map with the following properties:
   - **Bilinear:** *e(ua, vb) = e(u, v)ab* for all *u, v* ∈ *G* and *a, b* ∈ Z ∗ *p*, this property can be stretch to the multiplicative property that *e(u*1 · *u*2, *v) = e(u*1, *v)* · *e(u*2, *v)* for any *u*1, *u*2 ∈ *G*;
   - **Non-degenerate:** *e(g, g)* generates the group *GT* when *g* is generator of group *G*;
   - Computability: There exists an efficient algorithm to compute *e(u, v)* for all *u, v* ∈ *G*. Such a bilinear map *e* can be constructed by the modified Weil [21] or Tate pairings on elliptic curves.

3. 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.

4. 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, 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: 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 *Si* 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 *Si* 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 *Sj* , 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 epochs

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.

5. 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 Audit ability:*** 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 manages 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.

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(1κ) → (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) → (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) → (,, 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 , an authenticator set 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(Fin f o) → (C): This algorithm is performed by the TPA with the information of the file *Fin f o* as input and a challenge *C* as output.
Proof Gen(C, ,)→(P): This algorithm is run by each cloud server with input challenge *C*, coded block set and authenticator set , then it outputs a proof *P*.
Veri f y(P, pk, C)→ (0, 1): This algorithm is run by TPA immediately after a proof is received. Taking the proof *P*, public parameter *pk* and the corresponding challenge *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.

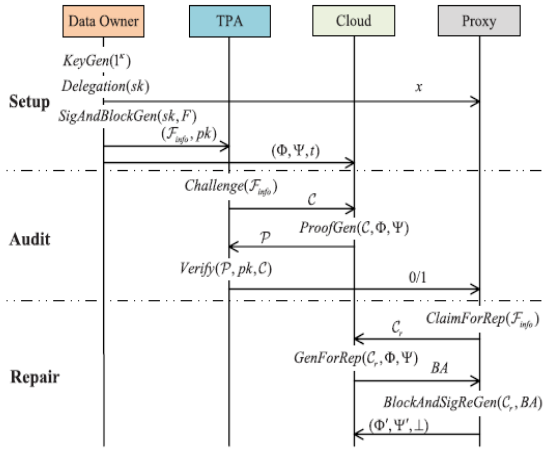

Fig. 3. The sequence chart of our scheme.

Claim for Rep (Fin f o) → (Cr): This algorithm is similar with the *Challenge ()* algorithm in the Audit phase, but outputs a claim for repair *Cr*.

GenFor Rep (Cr…,) → (BA): The cloud servers run this algorithm upon receiving the *Cr* and finally output the block and authenticators set *BA* with another two inputs ,. ***BlockAndSigReGen(Cr , BA) → (_,_,⊥):*** The proxy implements this algorithm with the claim *Cr* and responses *BA* from each server as input, and outputs a new coded block set _ and authenticator set _ if successful, outputting ⊥ if otherwise.

6. 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 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 and 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 maintain 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, 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.

Construction of Our Auditing Scheme

Considering the regenerating-code-based cloud storage with parameters *(n, k, _, α, β)*, we assume $\beta = 1$ for simplicity. Let G and GT be multiplicative cyclic groups of the same large prime order p, and $e: G \times G \rightarrow GT$ be a bilinear pairing map as introduced in the preliminaries. Let g be a generator of G and $H(\cdot) : \{0, 1\}* \rightarrow G$ 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(1κ) → (pk, sk): The data owner generates a random signing key pair *(spk, ssk)*, two random elements x, y $R \leftarrow- Zp$ and computes $pkx \leftarrow gx$, $pky \leftarrow gy$. Then the secret parameter is $sk = (x, y, ssk)$ and the public parameter is $pk = (pkx , pky, spk)$. ***Delegation(sk)→(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) → (,, t): The data owner uniformly chooses a random identifier I D $R \leftarrow- \{0, 1\}*$, a random symbol u $R \leftarrow- G$, one set _ = {w1,w2, . . . wm}

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 $\tilde{v}_i$,and s aggregated authenticators $\{\tilde{\sigma}_{ik}\}_{1\leq k\leq s}$

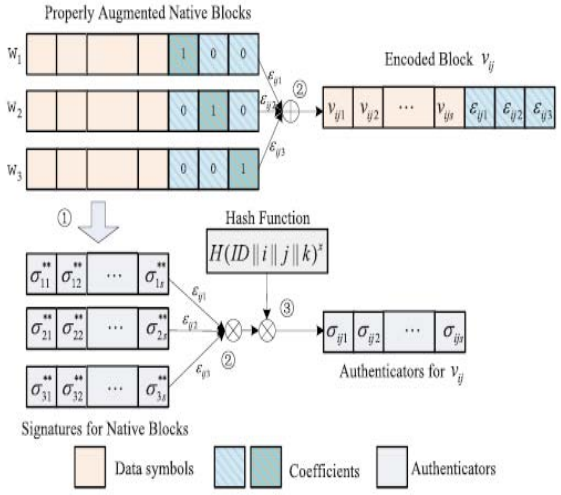

Fig. 4. An example with $m = 3$ for the $SigAndBlockGen(\cdot)$ algorithm.

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-unforgivable and secure guarantee against replay attack.

A. Correctness: There are two verification processes 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 i and accompanied authenticators 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.

7. Evaluation

A. Comparison

Table II lists the features of our proposed mechanism and makes a comparison with other remote data checking schemes, for regenerating-coding-based cloud storage. The security parameter κ is eliminated in the costs estimation for simplicity. While the previously presented 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 where data owners should stay online for faulty server reparation. To localize the faulty server during the audit phase, suggested a traversal method and thus demanded many times of auditing operation with complexity $O(Ck\ n\ (n-k)\alpha)$ 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.

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.

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, 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 $(n, k, \_, \alpha, \beta)$ for regenerating codes is in reference.

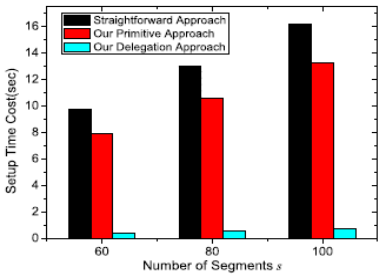

Fig. 5. Time for system setup with different segment number s.

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

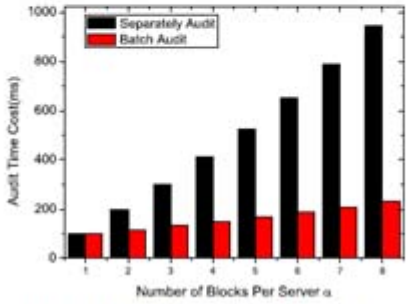

Fig. 6. Time for Audit with different α.

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 proof.

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, _ = 3, α = 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

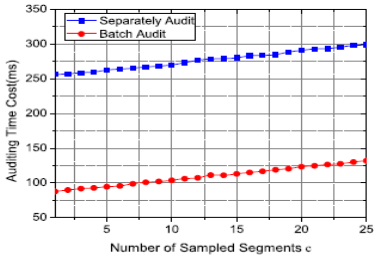

Fig. 7. Time for Audit with different c.

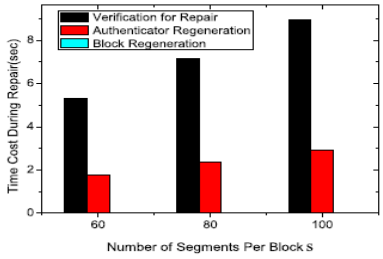

Fig. 8. Time for Repair with different s.

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 $\alpha s(\_+\alpha\_+1)$ modular exponentiations and αs inversions in a finite group across the authenticator regeneration, and only $\alpha s\_$ 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.

8. Related Work

The problem of remote data checking for integrity for integrity was first introduced. Then Ateniese *et al.* [2] and Juels and Kaliski [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, 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.* 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, Wang *et al.* proposed a new method which uses merkle hash tree to support fully dynamic data.

9. 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] M. Armbrust *et al.*, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.
[2] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2007, pp. 598–609.
[3] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 584–597.
[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2008, pp. 411–420.
[5] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 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 geographicallydispersed clouds," *J. Comput. Syst. Sci.*, 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 *Proc. ACM Workshop Cloud Comput. Secur. Workshop*, 2010, pp. 31–42.
[8] H. C. H. Chen and P. P. C. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation," *IEEE Trans. Parallel Distrib. Syst.*, 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," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 9, pp. 1717–1726, Sep. 2013.
[10] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2231–2244, Dec. 2012.
[11] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.

[12] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Advances in Cryptology*. Berlin, Germany: Springer-Verlag, 2008, pp. 90–107.
[13] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang, "NCCloud: Applying network coding for the storage repair in a cloud-of-clouds," in *Proc. USENIX FAST*, 2012, p. 21.
[14] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[15] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013.