Dynamic Audit Services for Integrity Verification of Outsourced Storages in Clouds

Yan Zhu[1,2], Huaixi Wang[3], Zexing Hu[1], Gail-Joon Ahn[4], Hongxin Hu[4], Stephen S. Yau[4]
[1]Institute of Computer Science and Technology, Peking University, Beijing 100871, China
[2]Key Laboratory of Network and Software Security Assurance (Peking University), Ministry of Education
[3]School of Mathematical Sciences, Peking University, Beijing 100871, China
[4]School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University, Tempe, AZ 85287, USA
{yan.zhu,wanghx,huzx}@pku.edu.cn, {gahn,hxhu,yau}@asu.edu

ABSTRACT

In this paper, we propose a dynamic audit service for verifying the integrity of untrusted and outsourced storage. Our audit service, constructed based on the techniques, fragment structure, random sampling and index-hash table, can support provable updates to outsourced data, and timely abnormal detection. In addition, we propose an efficient approach based on probabilistic query and periodic verification for improving the performance of audit services. Our experimental results not only validate the effectiveness of our approaches, but also show our audit system has a lower computation overhead, as well as a shorter extra storage for audit metedata.

Categories and Subject Descriptors

H.3.2 [**Information Storage and Retrieval**]: Information Storage; E.3 [**Data**]: Data Encryption

General Terms

Design, Performance, Security

Keywords

Dynamic Audit, Storage Security, Integrity Verification

1. INTRODUCTION

Cloud computing provides a scalability environment for growing amounts of data and processes that work on various applications and services by means of on-demand self-service. One of the strength of cloud computing is that data are being centralized and outsourced in clouds. This kind of outsourced storage in clouds has become a new profit growth point by providing a comparably low-cost, scalable, location-independent platform for managing clients' data. The cloud storage service (CSS) relieves the burden for storage management and maintenance. However, if such an important service is vulnerable to attacks or failures, it would bring irretrievable losses to the clients since their data or archives are stored in an uncertain storage pool outside the enterprises. These security risks come from the following reasons: the cloud infrastructures are much more powerful and reliable than personal computing devices. However, they are still facing all kinds of internal and external threats; for the benefits of their possession, there exist various motivations for cloud service providers (CSP) to behave unfaithfully towards the cloud users; furthermore, the dispute occasionally suffers from a lack of trust on CSP. Consequently, their behaviors may not be known by the cloud users, even if this dispute may result from the users' own improper operations. Therefore, it is necessary for cloud service providers to offer an efficient audit service to check the integrity and availability of the stored data [10].

Security audit is an important solution enabling tracking and analysis of any activities including data accesses, security breaches, application activities, and so on. Data security tracking is crucial for all organizations that must be able to comply with a range of federal laws including the Sarbanes-Oxley Act, Basel II, HIPAA and other regulations[1]. Furthermore, compared to the common audit, the audit service for cloud storages should provide clients with a more efficient proof of the integrity of stored data.

In this paper, we introduce a dynamic audit service for integrity verification of untrusted and outsourced storages. Our audit system, based on a novel audit system architecture, can support dynamic data operations and timely abnormal detection with the help of several effective techniques, such as fragment structure, random sampling, and index-hash table. Furthermore, we propose an efficient approach based on probabilistic query and periodic verification for improving the performance of audit services. A proof-of-concept prototype is also implemented to evaluate the feasibility and viability of our proposed approaches. Our experimental results not only validate the effectiveness of our approaches, but also show our system has a lower computation cost, as well as a shorter extra storage for integrity verification.

The rest of the paper is organized as follows. Section 2 describes the research background and related work. Section 3 and 4 address our audit system architecture and main techniques, and the construction of corresponding algorithms, respectively. In Section 5 we present the performance of our schemes and the experimental results. Finally, we conclude this paper in Section 6.

[1]http://www.hhs.gov/ocr/privacy/.

2. BACKGROUND AND RELATED WORK

The traditional cryptographic technologies for data integrity and availability, based on Hash functions and signature schemes [4, 11, 13], cannot work on the outsoucred data without a local copy of data. In addition, it is not a practical solution for data validation by downloading them due to the expensive communications, especially for large-size files. Moreover, the ability to audit the correctness of the data in a cloud environment can be formidable and expensive for the cloud users. Therefore, it is crucial to realize public auditability for CSS, so that data owners may resort to a third party auditor (TPA), who has expertise and capabilities that a common user does not have, for periodically auditing the outsourced data. This audit service is significantly important for digital forensics and credibility in clouds.

To implement public auditability, the notions of proof of retrievability (POR) [5] and provable data possession (PDP) [1] have been proposed by some researchers. Their approach was based on a probabilistic proof technique for a storage provider to prove that clients' data remain intact. For ease of use, some POR/PDP schemes work on a publicly verifiable way, so that anyone can use the verification protocol to prove the availability of the stored data. Hence, this provides us an effective approach to accommodate the requirements from public auditability. POR/PDP schemes evolved around an untrusted storage offer a publicly accessible remote interface to check the tremendous amount of data.

There exist some solutions for audit services on outsourced data. For example, Xie *et al.* [9] proposed an efficient method on content comparability for outsourced database, but it wasn't suited for irregular data. Wang *et al.* [8] also provided a similar architecture for public audit services. To support their architecture, a public audit scheme was proposed with privacy-preserving property. However, lack of rigorous performance analysis for constructed audit system greatly affects the practical application of this scheme. For instance, in this scheme an outsourced file is directly split into n blocks, and then each block generates a verification tag. In order to maintain security, the length of block must be equal to the size of cryptosystem, that is, 160-bit=20-Bytes. This means that 1M-Bytes file is split into 50,000 blocks and generates 50,000 tags [7], and the storage of tags is at least 1M-Bytes. It is clearly inefficient to build an audit system based on this scheme. To address such a problem, a fragment technique is introduced in this paper to improve performance and reduce extra storage (see Section 3.1).

Another major concern is the security issue of dynamic data operations for public audit services. In clouds, one of the core design principles is to provide dynamic scalability for various applications. This means that remotely stored data might be not only accessed but also dynamically updated by the clients, for instance, through block operations such as modification, deletion and insertion. However, these operations may raise security issues in most of existing schemes, e.g., the forgery of the verification metadata (called as tags) generated by data owners and the leakage of the user's secret key. Hence, it is crucial to develop a more efficient and secure mechanism for dynamic audit services, in which possible adversary's advantage through dynamic data operations should be prohibited.

Note that this paper only addresses the problems of integrity checking and auditing. Other security services, such as user authentication and data encryption, are orthogonal to and compatible with audit services.

3. ARCHITECTURE AND TECHNIQUES

We introduce an audit system architecture for outsourced data in clouds as shown in Figure 1. In this architecture, we consider a data storage service involving four entities: data owner (DO), who has a large amount of data to be stored in the cloud; cloud service provider (CSP), who provides data storage service and has enough storage space and computation resources; third party auditor (TPA), who has capabilities to manage or monitor the outsourced data under the delegation of data owner; and authorized applications (AA), who have the right to access and manipulate stored data. Finally, application users can enjoy various cloud application services via these authorized applications.

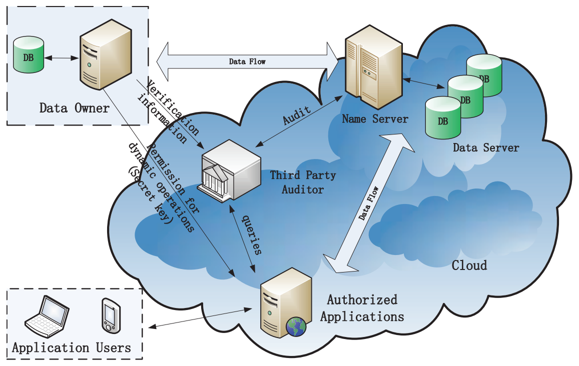


Figure 1: The audit system architecture.

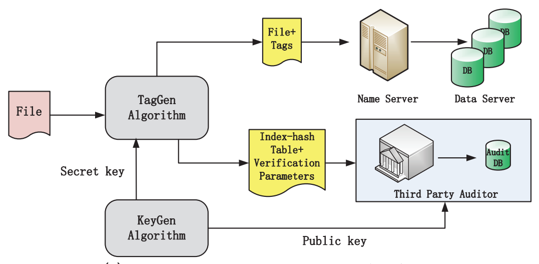
We assume the TPA is reliable and independent through the following audit functions: TPA should be able to make regular checks on the integrity and availability of the delegated data at appropriate intervals; TPA should be able to organize, manage, and maintain the outsourced data instead of data owners, and support the dynamic data operations for authorized applications; and TPA should be able to take the evidences for disputes about the inconsistency of data in terms of authentic records for all data operations.

To realize these functions, our audit service is comprised of three processes:

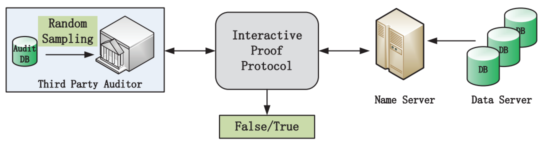
Tag Generation: the client (data owner) uses the secret key sk to pre-process a file, which consists of a collection of n blocks, generates a set of public verification parameters (PVP) and index-hash table (IHT) that are stored in TPA, transmits the file and some verification tags to CSP, and may delete its local copy (see Figure 2(a));

Periodic Sampling Audit: by using an interactive proof protocol of retrievability, TPA (or other applications) issues a "Random Sampling" challenge to audit the integrity and availability of outsourced data in terms of the verification information (involves PVP and IHT) stored in TPA (see Figure 2(b));

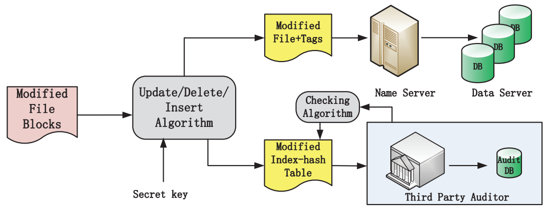
Audit for Dynamic Operations: An authorized applications, who hold data owner's secret key sk, can manipulate the outsourced data and update the associated index-hash table (IHT) stored in TPA. The privacy of sk and the checking algorithm ensure that the storage server cannot cheat the authorized applications and forge the valid audit records (see Figure 2(c)).



(a) Tag generation for outsourced file



(b) Periodic sampling audit



(c) Dynamic data operations and audit

Figure 2: Three processes of audit system.

In general, the authorized applications should be cloud application services inside clouds for various application purposes, but they must be specifically authorized by data owners for manipulating the outsourced data. Since the acceptable operations require that the authorized applications must present authentication information for TPA, any unauthorized modifications for data will be detected in audit processes or verification processes. Based on this kind of strong authorization-verification mechanism, we neither assume that CSP is trust to guarantee the security of stored data, nor assume that a date owner has the capability to collect the evidence of CSP's faults after errors have been found.

The ultimate goal of this audit infrastructure is to enhance the credibility of cloud storage services, but not to increase data owner's burden and overheads. For this purpose, TPA should be constructed in clouds and maintained by a cloud storage provider (CSP). In order to ensure the trust and security, TPA must be secure enough to resist malicious attacks, and it also should be strictly controlled to prevent unauthorized access even for internal members in clouds. A more practical way is that TPA in clouds should be mandated by a trusted third party (TTP). This mechanism not only improves the performance of audit services, but also provides the data owner with a maximum access transparency. This means that data owners are entitled to utilize the audit service without further costs besides storing a secret-key and some secret information.

The above processes involve some procedures: KeyGen, TagGen, Update, Delete, Insert algorithms, as well as an interactive proof protocol of retrievability (see Appendix A). In order to improve security and performance, we make use of following techniques to construct corresponding algorithms and protocols.

3.1 Fragment Structure and Secure Tags

To maximize the storage efficiency and audit performance, a general fragment structure is introduced into our audit system for outsourced storage. An instance for this framework which is used in this scheme is showed in Figure 3: an outsourced file F is split into n blocks $\{m_1, m_2, \cdots, m_n\}$, and each block m_i is split into s sectors $\{m_{i,1}, m_{i,2}, \cdots, m_{i,s}\}$. The fragment framework consists of n block-tag pair (m_i, σ_i), where σ_i is a signature tag of block m_i generated by some secrets $\tau = (\tau_1, \tau_2, \cdots, \tau_s)$. Finally, these block-tag pairs are stored in CSP and the encryption of the secrets τ (called as PVP) are in TTP. Although this fragment structure is simple and straightforward, but the file is split into $n \times s$ sectors and each block (s sectors) corresponds to a tag, so that the storage of signature tags can be reduced with increase of s. Hence, this structure can reduce extra storage for tags and improve the audit performance.

There exist some schemes to convergence s blocks to generate a secure signature tag, e.g., MAC-based, ECC or RSA schemes [1, 7]. These schemes, built from collision-resistance signatures (see Appendix A) and the random oracle model, have higher scalability, performance and security.

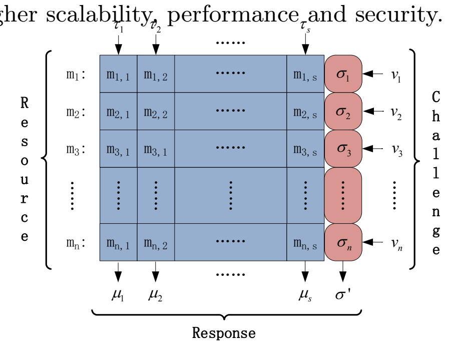


Figure 3: Fragment structure and sampling audit.

3.2 Periodic Sampling Audit

In contract with "whole" checking, random "sampling" checking greatly reduces the workload of audit services, while still achieve an effective detection of misbehavior. Thus, the probabilistic audit on sampling checking is preferable to realize the abnormality detection in a timely manner, as well as rationally allocate resources. The fragment structure shown in Figure 3 can provide the support of probabilistic audit as well: given a random chosen challenge (or query) $Q = \{(i, v_i)\}_{i \in I}$, where I is a subset of the block indices and v_i is a random coefficient, an efficient algorithm is used to produce a constant-size response $(\mu_1, \mu_2, \cdots, \mu_s, \sigma')$, where μ_i comes from all $\{m_{k,i}, v_k\}_{k \in I}$ and σ' is from all $\{\sigma_k, v_k\}_{k \in I}$. Generally, this algorithm relies on homomorphic properties to aggregate data and tags into a constant size response, which minimizes network communication.

Since the single sampling checking may overlook a very small number of data abnormality, we propose a periodic sampling approach to audit outsourcing data, which is called as *Periodic Sampling Audit*. In this way, the audit activities are efficiently scheduled in an audit period, and a TPA needs merely access small portions of file to perform audit in each activity. Therefore, this method can detect the exceptions in time, and reduce the sampling numbers in each audit.

3.3 Index-Hash Table

In order to support dynamic data operations, we introduce a simple index-hash table (IHT) to record the changes of file blocks, as well as generate the hash value of block in the verification process. The structure of our index-hash table

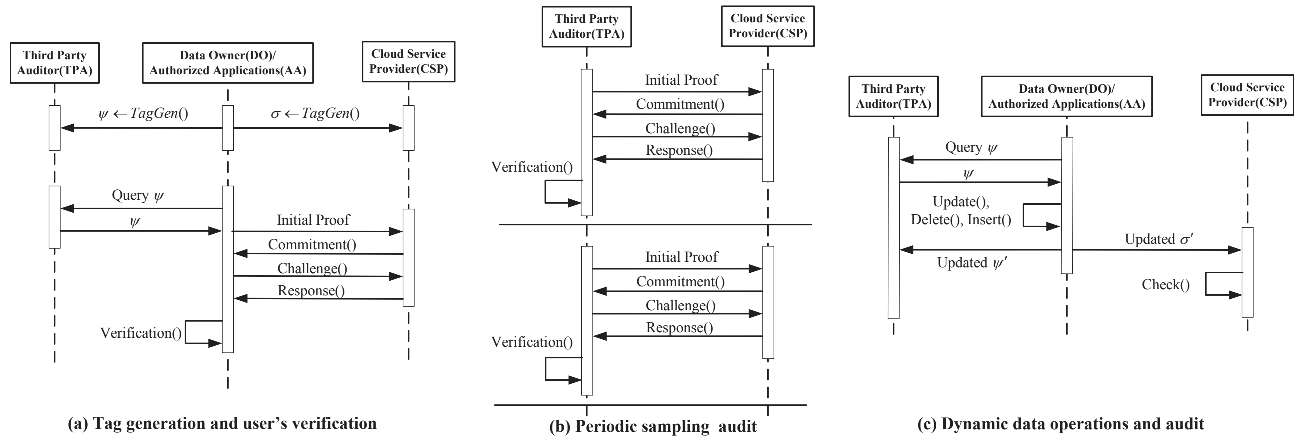

Third Party Auditor(TPA)
Data Owner(DO)/ Authorized Applications(AA)
Cloud Service Provider(CSP)
ψ ← TagGen()
σ ← TagGen()
Query ψ
ψ
Initial Proof
Commitment()
Challenge()
Response()
Verification()
(a) Tag generation and user's verification
(b) Periodic sampling audit
Update(), Delete(), Insert()
Updated σ′
Updated ψ′
Check()
(c) Dynamic data operations and audit

Figure 4: The workflow of audit system.

is similar to that of file block allocation table in file systems. Generally, the index-hash table χ consists of serial number, block number, version number, random integer, and so on (see Table 1 in Appendix A). Different from the common index table, we must assure that all records in this kind of table differ from one another to prevent the forgery of data blocks and tags. In addition to record data changes, each record χ_i in table is used to generate a unique Hash value, which in turn is used for the construction of signature tag σ_i by the secret key sk. This kind of relationship between χ_i and σ_i must be cryptographic secure, and we can make use of it to design our verification protocol depicted in Figure 2(b) and the checking algorithm in Figure 2(c).

Although the index-hash table may increases the complexity of an audit system, it provides the strongest assurance to monitor the behavior of untrusted CSP, as well as efficient evidence of computer forensics, due to the reason that anyone cannot forge the valid χ_i (in TPA) and σ_i (in CSP) without the secret key sk. In practical applications, the designer should consider that the index-hash table is kept into the virtualization infrastructure of cloud-based storage services.

4. ALGORITHMS FOR AUDIT SYSTEM

In this section we describe the construction of algorithms in our audit architecture. A more detailed descriptions of the our can be found in Appendix A. Firstly, we present the definition of two algorithms for the tag generation process as follows:

$\mathcal{K}eyGen$ (1^{κ}): takes a security parameter κ as input, and returns a public/secret keypair (pk, sk);

$\mathcal{T}agGen$ (sk, F): takes as inputs the secret key sk and a file F, and returns the triple (τ, ψ, σ), where τ denotes the secret used to generate the verification tags, ψ is a set of public verification parameters u and index-hash table χ, i.e., $\psi = (u, \chi)$, and σ denotes the set of tags.

Data owner or authorized applications only need to save the secret key sk, moreover, sk would not be necessary for the verification/audit process. The secret of the processed file τ can be discarded after tags are generated due to public verification parameters u.

In Figure 4 demonstrates the workflow of our audit system. Suppose a data owner wants to store a file in a storage server, and maintains a corresponding authenticated index structure at a TPA. In Figure 4 (a), we describe this process as follows: firstly, using $KeyGen()$, the owner generates a public/secret keypair (pk, sk) by himself or the system manager, and then sends his public key pk to TPA. Note that TPA cannot obtain the client's secret key sk; secondly, the owner chooses the random secret τ and then invokes the algorithm $TagGen()$ to produce public verification information $\psi = (u, \chi)$ and signature tags σ, where τ is unique for each file. Finally, the owner sends ψ and (F, σ) to TPA and CSP, respectively, where χ is an index-hash table.

4.1 Supporting Periodic Sampling Audit

At any time, TPA can check the integrity of F as follows: TPA first queries database to obtain the verification information ψ; and then it initializes an interactive protocol $Proof(CSP, Client)$ and performs a 3-move proof protocol in a random sampling way: *Commitment*, *Challenge*, and *Response*; finally, TPA verifies the interactive data to get the results. In fact, since our scheme is a publicly verifiable protocol, anyone can run this protocol, but s/he is unable to get any advantage to break the cryptosystem, even if TPA and CSP cooperate for an attack. Let $P(x)$ denotes the subject P holds the secret x and $\langle P, V\rangle(x)$ denotes both parties P and V share a common data x in a protocol. This process can be defined as follows:

$\mathcal{P}roof$ (CSP, TPA): is an interactive proof protocol between CSP and TPA, that is $\langle CSP(F, \sigma), TPA\rangle(pk, \psi)$, where a public key pk and a set of public parameters ψ are the common inputs between TPA and CSP, and CSP takes the inputs, a file F and a set of tags σ. At the end of the protocol running, TPA returns $\{0|1\}$, where 1 means the file is correctly stored on the server.

An audit service executes the verification process periodically by using above-mentioned protocol. Figure 4(b) shows such a two-party protocol between TPA and CSP, i.e., $Proof(CSP, TPA)$, without the involvement of a client (DO or AA). In Figure 4 (b) shows two verification processes. To improve the efficiency of verification process, TPA should work in a probabilistic sampling way. We articulate the relationship among the detection probability, the corruption probability, and the sampling number in Section 5.

4.2 Supporting Dynamic Data Operations

In order to meet the requirements from dynamic scenarios, we introduce following definitions for dynamic algorithms:

$\mathcal{U}pdate(sk, \psi, m_i')$: is an algorithm run by AA to update the block of file m_i' at the index i by using sk, and it returns a new verification metadata (ψ', σ');

$\mathcal{D}elete(sk, \psi, m_i)$: is an algorithm run by AA to delete the block m_i of file at the index i by using sk, and it returns a new verification metadata (ψ');

$\mathcal{I}nsert(sk, \psi, m_i)$: is an algorithm run by AA to insert the block of file m_i at the index i by using sk, and it returns a new verification metadata (ψ', σ').

To ensure the security, dynamic data operations are only available to data owners or authorized applications, who hold the secret key sk. Here, all operations are based on data blocks. Moreover, in order to implement audit services, applications need to update the index-hash table. It is necessary for TPA and CSP to check the validity of updated data. In Figure 4(c), we describe the process of dynamic data operations and audit. First, the authorized application obtains the public verification information ψ from TPA. Second, the application invokes the $Update$, $Delete$, and $Insert$ algorithms, and then sends the new ψ' and σ' to TPA and CSP, respectively. Finally, the CSP makes use of an efficient algorithm $Check$ to verify the validity of updated data. Note that, the Check algorithm is important to ensure the effectiveness of the audit materials.

5. PERFORMANCE AND EVALUATION

No doubt too frequent audit activities will increase the computation and communication overheads of audit services. However, less frequent activities may not detect abnormality timely. Hence, the scheduling of audit activities is significant for improving the quality of audit services. In order to detect abnormality in a low-overhead and timely manner, we optimize the audit performance from two aspects: performance evaluation of probabilistic queries and schedule of periodic verification. Our basic idea is to achieve an overhead balance by verification dispatching, which is one of efficient strategies to improve the performance of audit systems.

5.1 Probabilistic Queries Evaluation

The audit service achieves the detection of CSP servers misbehavior in a random sampling mode in order to reduce the workload on the server. The detection probability P of disrupted blocks is an important parameter to guarantee that these blocks can be detected in time. Assume the TPA modifies e blocks out of the n-block file. The probability of disrupted blocks is $\rho_b = \frac{e}{n}$. Let t be the number of queried blocks for a challenge in the protocol proof. We have detection probability $P = 1 - (\frac{n-e}{n})^t = 1 - (1-\rho_b)^t$. Hence, the number of queried blocks is $t = \frac{\log(1-P)}{\log(1-\rho_b)} \approx \frac{P\cdot n}{e}$ for a sufficiently large n.[2] This means that the number of queried blocks t is directly proportional to the total number of file blocks n for the constant P and e. In Fig. 5, we show the results of the number of queried blocks under different detection probabilities (from 0.5 to 0.99), different number of file blocks (from 10 to 10,000), and constant number of disrupted blocks (100).

[2]In terms of $(1-\frac{e}{n})^t = 1-\frac{e\cdot t}{n}$, we have $P = 1-(1-\frac{e\cdot t}{n}) = \frac{e\cdot t}{n}$.

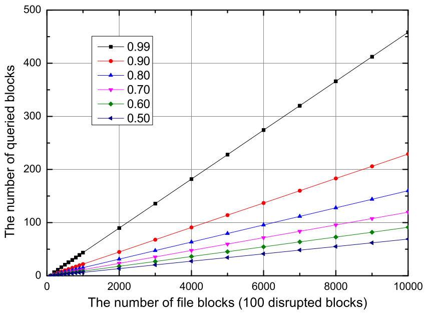


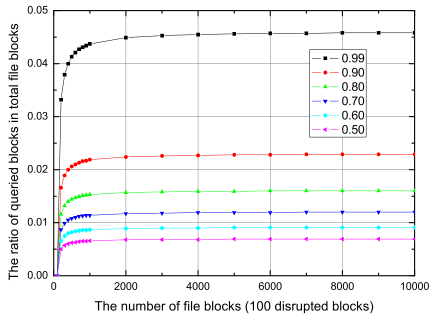
Figure 5: The number of queried blocks under different detection probabilities and different numbers of file blocks.

We observe the ratio of queried blocks in total file blocks $w = \frac{t}{n}$ under different detection probabilities. Based on the above analysis, it is easy to find that this ratio holds since the equation $w = \frac{t}{n} = \frac{\log(1-P)}{n\cdot\log(1-\rho_b)} \approx \frac{P}{e}$. However, the estimation of w is not an accurate measurement. To clearly represent this ratio, Fig. 6 plots w for different values of n, e and P. It is obvious that the ratio of queried blocks tends to be a constant value for a sufficiently large n. For instance, in Fig. 6 (Left) if there exist 100 disrupted blocks, the TPA asks for $w = 4.5\%$ and 2.3% of n ($n > 1,000$) in order to achieve P of at least 99% and 90%, respectively. However, this ratio w is also inversely proportional to the number of disrupted blocks e. For example, in Fig. 6 (Right) if there exist 10 disrupted blocks, the TPA needs to ask for $w = 45\%$ and 23% of n ($n > 1,000$) in order to achieve the same P, respectively. Hence, our audit scheme is very effective for higher probability of disrupted blocks.

5.2 Schedule of Periodic Verification

The sampling-based audit has the potential to significantly reduce the workload on the servers and increase the audit efficiency. Firstly, we assume that each audited file has a audit period T, which depends on how important it is for the owner. For example, common audit period may be assigned as one week or one month, and the audit period for important files may be set as one day. Of course, these audit activities should be carried out at night or on weekend.

Assume we make use of the audit frequency f to denote the number of occurrences of an audit event per unit time. This means that the number of TPA's queries is $T\cdot f$ times in an audit period T. According to the above analysis, we have the detection probability $P = 1-(1-\rho_b)^{n\cdot w}$ in each audit event. Let P_T denotes the detection probability in an audit period T. Hence, we have the equation $P_T = 1-(1-P)^{T\cdot f}$. In terms of $1 - P = (1-\rho_b)^{n\cdot w}$, the detection probability P_T can be denoted as $P_T = 1-(1-\rho_b)^{n\cdot w\cdot T\cdot f}$. In this equation, TPA can obtain the probability ρ_b depending on the transcendental knowledge for the cloud storage provider. Moreover, the audit period T can be appointed by a data owner in advance. Hence, the above equation can be used to analyze the parameter value w and f. It is obvious to obtain the equation $f = \frac{\log(1-P_T)}{w\cdot n\cdot T\cdot\log(1-\rho_b)}$. This means that the audit frequency f is inversely proportional to the ratio of queried blocks w. That is, with the increase of verification frequency, the number of queried blocks decreases at each verification process. In Fig. 7, we show the relationship between f and w under 10 disrupted blocks for 10,000 file blocks. we can



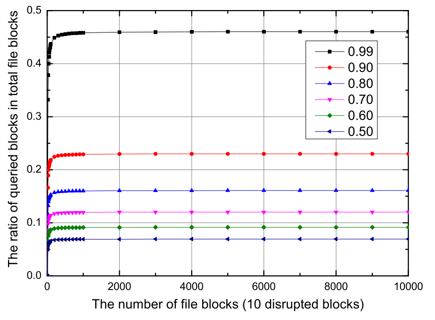


Figure 6: The ratio of queried blocks in total file blocks under different detection probabilities and different number of disrupted blocks (100 disrupted blocks for left-side and 10 disrupted blocks for right-side).

observe a marked drop of w along with the increasing of frequency.

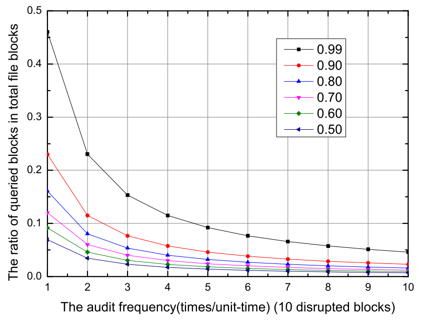


Figure 7: The ratio of queried blocks in total file blocks under different audit frequency for 10 disrupted blocks and 10,000 file blocks.

In fact, this kind of relationship between f and w is a comparatively stable value for certain P_T, ρ_b, and n due to $f \cdot w = \frac{\log(1-P_T)}{n \cdot T \cdot \log(1-\rho_b)}$. TPA should choose the appropriate frequency to balance the overhead according to the above equation. For example, if $e = 10$ blocks in 10,000 blocks ($\rho_b = 0.1\%$), then TPA asks for 658 blocks and 460 blocks for $f = 7$ and 10 in order to achieve P_T of at least 99%. Hence, appropriate audit frequency would greatly reduce sampling numbers, as well as computation and communication overheads of an audit service.

5.3 Implementation and Experimental Results

To validate our approaches, we have implemented a prototype of public audit service. Our prototype utilizes three existing services/applications: Amazon Simple Storage Service (S3) is an untrusted data storage server; local application server provides our audit service; and the prototype is built on top of an existing open source project called Pairing-Based Cryptography (PBC) library. We present some details about these three components as follows:

Storage service: Amazon Simple Storage Service (S3) is a scalable, pay-per-use online storage service. Clients can store a virtually unlimited amount of data, paying for only the storage space and bandwidth that they are using, without initial start-up fee. The basic data unit in S3 is an object, and the basic container for objects in S3 is called a bucket. In our example, objects contain both data and meta-data (tags). A single object has a size limit of 5 GB, but there is no limit on the number of objects per bucket. Moreover, a small script on Amazon Elastic Compute Cloud (EC2) is used to provide the support for verification protocol and dynamic data operations.

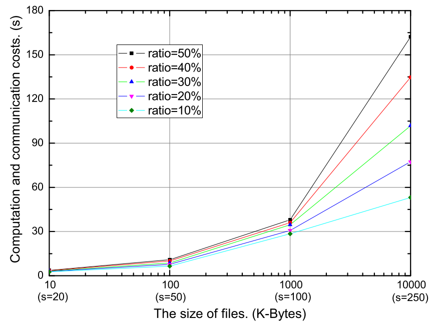
Audit service: We used a local IBM server with two Intel Core 2 processors at 2.16 GHz and 500M of RAM running Windows Server 2003. Our scheme was deployed in this server, and then it can implement the integrity checking in S3 storage according to the assigned schedule via 250 MB/sec of network bandwidth. A socket port was also opened to support the applications' accesses and queries for the audit service.

Prototype software: Using GMP and PBC libraries, we have implemented a cryptographic library upon which temporal attribute systems can be constructed. This C library contains approximately 5,200 lines of codes and has been tested on both Windows and Linux platforms. The elliptic curve utilized in our experiments is a MNT curve, with base field size of 159 bits and the embedding degree 6. The security level is chosen to be 80 bit, which means $|p| = 160$.

Firstly, we quantify the performance of our audit scheme under different parameters, such as file size sz, sampling ratio w, sector number per block s, and so on. Our analysis shows that the value of s should grow with the increase of sz in order to reduce computation and communication costs. Thus, experiments were carried out as follows: the stored files were chosen from 10KB to 10MB, the sector numbers were changed from 20 to 250 in terms of the file sizes, and the sampling ratios were also changed from 10% to 50%. The experimental results were showed in the left side of Fig. 8. These results indicate that computation and communication costs (including I/O costs) grow with increase of file size and sampling ratio.

Next, we compare the performance of each activity in our verification protocol. It is easy to derive theoretically that the overheads of "commitment" and "challenge" resemble one another, and the overheads of "response" and "verification" also resemble one another. To validate such theoretical results, we changed the sampling ratio w from 10% to 50% for a 10MB file and 250 sectors per block. In the right side of Fig. 8, we show the experiment results, in which the computation and communication costs of "commitment" and "challenge" are slightly changed for sampling ratio, but those for "response" and "verification" grow with the increase of sampling ratio.

Then, in the Amazon S3 service, we set that the size of block is 4K-Bytes and the value of s is 200. Our experiments



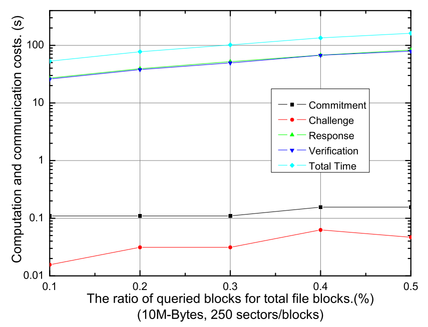


Figure 8: The experiment results under different file size, sampling ratio, and sector number.

showed that, in TagGen phase, the time overhead is directly proportional to the number of blocks. Fortunately, this process is only executed when the file is uploaded into a S3 service. The verification protocol can be run in approximate constant time. Similarly, three dynamic data operations can be performed in approximate constant time for any block.

Finally, reducing the communication overheads and average workloads are critical for an efficient audit schedule. In view of probabilistic algorithm, our scheme is able to realize the uniform distribution of verified sampling blocks according to the security requirements of clients, as well as the dependability of storage services and running environments. In our experiments, we make use of a simple schedule to periodically manage all audit tasks. The results showed that audit services based on our scheme can support a great deal of batch auditing tasks, and the performance of scheduled auditing can still be safely concluded as more preferable than the straightforward individual auditing.

6. CONCLUSIONS

In this paper, we presented a construction of dynamic audit services for untrusted and outsourced storage. We also presented an efficient method for periodic sampling audit to minimize the computation costs of third party auditors and storage service providers. Our experiments showed that our solution has a small, constant amount of overhead, which minimizes computation and communication costs.

7. ACKNOWLEDGMENTS

The work of Y. Zhu, H. Wang, and Z. Hu was partially supported by the grants from National Natural Science Foundation of China (No.61003216). This work of Gail-J. Ahn and Hongxin Hu was partially supported by the grants from National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360).

8. REFERENCES

[1] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, pages 598–609, 2007.

[2] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *In proceedings of CRYPTO'04*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, 2004.

[3] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology (CRYPTO'01)*, volume 2139 of LNCS, pages 213–229, 2001.

[4] H.-C. Hsiao, Y.-H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun, and B.-Y. Yang. A study of user-friendly hash comparison schemes. In *ACSAC*, pages 105–114, 2009.

[5] A. Juels and B. S. K. Jr. Pors: proofs of retrievability for large files. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, pages 584–597, 2007.

[6] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[7] H. Shacham and B. Waters. Compact proofs of retrievability. In *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107, 2008.

[8] 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*, pages 1 –9, 14-19 2010.

[9] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, editors, *VLDB*, pages 782–793. ACM, 2007.

[10] A. A. Yavuz and P. Ning. Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *ACSAC*, pages 219–228, 2009.

[11] A. R. Yumerefendi and J. S. Chase. Strong accountability for network storage. In *FAST*, pages 77–92. USENIX, 2007.

[12] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau. Cooperative provable data possession. Technical Report PKU-CSE-10-04, http://eprint.iacr.org/2010/234.pdf, Peking University and Arizona State University, April 2010.

[13] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau. Efficient provable data possession for hybrid clouds. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 756–758, 2010.

APPENDIX

A. CONSTRUCTION FOR OUR SCHEME

Let $\mathcal{H} = \{H_k\}$ be a *collision-resistance* hash family of functions $H_k : \{0,1\}^* \rightarrow \{0,1\}^n$ indexed by $k \in \mathcal{K}$. This hash function can be obtained from hash function of BLS signatures [2]. Further, we set up our systems using bilinear map group system $\mathbb{S} = \langle p, \mathbb{G}, \mathbb{G}_T, e\rangle$ proposed in [3].

A.1 Proposed Construction

We present our IPOR construction in Figure 9. In our scheme, each client holds a secret key sk, which can be used to generate the tags of many files. Each processed file will produce a public verification parameter $\psi = (u, \chi)$, where $u = (\xi^{(1)}, u_1, \cdots, u_s)$, $\chi = \{\chi_i\}_{i\in[1,n]}$ is the index-hash table. We define $\chi_i = (B_i||V_i||R_i)$, where B_i is the sequence number of block, V_i is the version number of updates for this block, and R_i is a random integer to avoid collision. The value $\xi^{(1)}$ can be considered as the signature of the secret $\tau_1, \cdots, \tau_s$. Note that, it must assure that ψ's is different for all processed files. Moreover, it is clear that our scheme admits short responses in verification protocol.

In our construction, the verification protocol has 3-move structure: commitment, challenge and response. This protocol is similar to Schnorr's Σ protocol [6], which is a zero-knowledge proof system (Due to the space limitation, the security analysis is omitted but can be found in [12]). By using this property, we ensure the verification process does not reveal anything.

KeyGen(1^κ): Given a bilinear map group system $\mathbb{S} = (p, \mathbb{G}, \mathbb{G}_T, e)$ and a collision-resistant hash function $H_k(\cdot)$, chooses a random $\alpha, \beta \in_R \mathbb{Z}_p$ and computes $H_1 = h^\alpha$ and $H_2 = h^\beta \in \mathbb{G}$. Thus, the secret key is $sk = (\alpha, \beta)$ and the public key is $pk = (g, h, H_1, H_2)$.

TagGen(sk, F): Splits the file F into $n \times s$ sectors $F = \{m_{i,j}\} \in \mathbb{Z}_p^{n\times s}$. Chooses s random $\tau_1, \cdots, \tau_s \in \mathbb{Z}_p$ as the secret of this file and computes $u_i = g^{\tau_i} \in \mathbb{G}$ for $i \in [1, s]$ and $\xi^{(1)} = H_\xi(\text{“}Fn\text{”})$, where $\xi = \sum_{i=1}^s \tau_i$ and Fn is the file name. Builds an index-hash table $\chi = \{\chi_i\}_{i=1}^n$ and fills out the item $\chi_i = (B_i = i, V_i = 1, R_i \in_R \{0,1\}^*)$ in χ for $i \in [1, n]$, then calculates its tag as $\sigma_i \leftarrow (\xi_i^{(2)})^\alpha \cdot g^{\sum_{j=1}^s \tau_j \cdot m_{i,j} \cdot \beta} \in \mathbb{G}$. where $\xi_i^{(2)} = H_{\xi^{(1)}}(\chi_i)$ and $i \in [1, n]$. Finally, sets $u = (\xi^{(1)}, u_1, \cdots, u_s)$ and outputs $\psi = (u, \chi)$ to TPA, and $\sigma = (\sigma_1, \cdots, \sigma_n)$ to CSP.

Proof(CSP, TPA): This is a 3-move protocol between Prover (CSP) and Verifier (TPA), as follows:

- **Commitment**($CSP \rightarrow TPA$): CSP chooses a random $\gamma \in \mathbb{Z}_p$ and s random $\lambda_j \in_R \mathbb{Z}_p$ for $j \in [1, s]$, and sends its commitment $C = (H_1', \pi)$ to TPA, where $H_1' = H_1^\gamma$ and $\pi \leftarrow e(\prod_{j=1}^s u_j^{\lambda_j}, H_2)$;
- **Challenge**($CSP \leftarrow TPA$): TPA chooses a random challenge set I of t indexes along with t random coefficients $v_i \in \mathbb{Z}_p$. Let Q be the set $\{(i, v_i)\}_{i \in I}$ of challenge index coefficient pairs. TPA sends Q to CSP;
- **Response**($CSP \rightarrow TPA$): CSP calculates the response θ, μ as $\sigma' \leftarrow \prod_{(i,v_i)\in Q} \sigma_i^{\gamma\cdot v_i}$, $\mu_j \leftarrow \lambda_j + \gamma \cdot \sum_{(i,v_i)\in Q} v_i \cdot m_{i,j}$, where $\mu = \{\mu_j\}_{j\in[1,s]}$. P sends $\theta = (\sigma', \mu)$ to TPA;

Check: The verifier TPA checks whether the response is correct by $\pi \cdot e(\sigma', h) \overset{?}{=} e(\prod_{(i,v_i)\in Q} (\xi_i^{(2)})^{v_i}, H_1') \cdot e(\prod_{j=1}^s u_j^{\mu_j}, H_2)$.

Figure 9: The proposed IPOR scheme.

A.2 Implementation of Dynamic Operations

To support dynamic data operations, it is necessary for TPA to employ an index-hash table χ to record the real-time status of the stored files. This kind of index structure can also be used to generate the value of Hash function $\xi_i^{(2)} = H_{\xi^{(1)}}(\chi_i)$ in our scheme. Some existing schemes in a dynamic scenario are insecure due to replay attack on the same Hash values. To solve this problem, a simple index-hash table $\chi = \{\chi_i\}$ used in the above-mentioned construction (see Figure 9) is described in Table 1, which includes four columns: No. denotes the real number i of data block m_i, B_i is the original number of block, V_i stores the version number of updates for this block, and R_i is a random integer to avoid collision.

Table 1: The index-hash table with random values.

No.	B_i	V_i	R_i	
0	0	0	0	← Used to head
1	1	2	r_1'	← Update
2	2	1	r_2	
3	4	1	r_3	← Delete
4	5	1	r_5	
5	5	2	r_5'	← Insert
⋮	⋮	⋮	⋮	
n	n	1	r_n	
n+1	n+1	1	r_{n+1}	← Append

In order to ensure the security, we require that each $\chi_i = \text{“}B_i||V_i||R_i\text{”}$ is unique in this table. Although the same values of “$B_i||V_i$” may be produced by repeating the insert and delete operations, the random R_i can avoid this collision. An alterative method is to generate an updated random value by $R_i' \leftarrow H_{R_i}(\sum_{j=1}^s m_{i,j}')$, where the initial value is $R_i \leftarrow H_{\xi^{(1)}}(\sum_{j=1}^s m_{i,j})$ and $m_i = \{m_{i,j}\}$ denotes the i-th data block. We show a simple example to describe the change of index-hash table for the different operations in Table 1, where an empty record ($i = 0$) is used to support the operations on the first record. The “Insert” operation on the last record is replaced with “Append” operation. It is easy to prove the each χ_i is unique in χ in the above algorithms, that is. In an index table $\chi = \{\chi_i\}$ and $\chi_i = \text{“}B_i||V_i||R_i\text{”}$, there exists no two same records for dynamic data operations, if $R_i \neq R_j'$ for any indexes $i, j \in \mathbb{N}$.

Update(sk, ψ, m_i'): modifies the version number by $V_i \leftarrow \max_{B_i=B_j}\{V_j\} + 1$ and chooses a new R_i in $\chi_i \in \chi$ to get a new ψ'; computes the new hash $\xi_i^{(2)} = H_{\xi^{(1)}}(\text{“}B_i||V_i||R_i\text{”})$; by using sk, computes $\sigma_i' = (\xi_i^{(2)})^\alpha \cdot (\prod_{j=1}^s u_j^{m_{i,j}'})^\beta$, where $u = \{u_j\} \in \psi$, finally outputs (ψ', σ_i', m_i').

Delete(sk, ψ, m_i): computes the original σ_i by m_i and computes the new hash $\xi_i^{(2)} = H_{\xi^{(1)}}(\text{“}B_i||0||R_i\text{”})$ and $\sigma_i' = (\xi_i^{(2)})^\alpha$ by sk; deletes i-th record to get a new ψ'; finally outputs $(\psi', \sigma_i, \sigma_i')$.

Insert(sk, ψ, m_i'): inserts a new record in i-th position of the index-hash table $\chi \in \psi$, and the other records move backward in order; modifies $B_i \leftarrow B_{i-1}$, $V_i \leftarrow \max_{B_i=B_j}\{V_j\} + 1$, and a random R_i in $\chi_i \in \chi$ to get a new ψ'; computes the new hash $\xi_i^{(2)} = H_{\xi^{(1)}}(\text{“}B_i||V_i||R_i\text{”})$ and $\sigma_i' = (\xi_i^{(2)})^\alpha \cdot (\prod_{j=1}^s u_j^{m_{i,j}'})^\beta$, where $u = \{u_j\} \in \psi$, finally outputs (ψ', σ_i', m_i').

Check: The application sends the above result to cloud store provider P via secret channel. For Update or Insert operations, P must check the following equation for (ψ', σ_i', m_i') in terms of $e(\sigma_i', h) \overset{?}{=} e(\xi_i^{(2)}, H_1) \cdot e(\prod_{j=1}^s u_j^{m_{i,j}'}, H_2)$. For Delete operation, P must check whether σ_i is equal to the stored σ_i and $e(\sigma_i', h) \overset{?}{=} e(H_{\xi^{(1)}}(\text{“}B_i||0||R_i\text{”}), H_1)$. Further, TPA must replace ψ by the new ψ' and check the completeness of $\chi \in \psi$.

Figure 10: The algorithms for dynamic operations.

According to the construction of index-hash tables, we propose a simple method to provide dynamic data modification in Figure 10. All tags and the index-hash table should be renewed and reorganized periodically to improve the performance. Of course, we can replace the sequent lists by the dynamically linked lists to improve the efficiency of updating index-hash table. Further, we omit the discuss of the head and tail index items in χ, and they are easy to implement.