Title
Cloud Data Storage Security

From
2. Toward Publicly Auditable Secure Cloud Data Storage Services
4. Ensuring Data Storage Security in Cloud Computing

Speaker
Jyun-Yao Huang, Piotr Tatarczuk (5th group)
Outline

- Cisco Cloud Computing Overview
- Publicly Auditable Secure Cloud Data Storage Service
- Ensuring Data Storage Security in Cloud Computing
- Summary
Reference Materials


resources can be provisioned immediately when needed, released when no longer required, and billed only when used
Cisco Definition of Cloud Computing

service provides the illusion of infinite resource availability in order to meet whatever demands are made of it at scale
resources are provided to many consumers from a single implementation, saving the provider significant costs.

multitenant environment
Benefits

reduced cost
flexibility
improved automation
focus on core competency
sustainability
Cisco architecture framework
Trust in a cloud data center

Cisco believes that gaining the advantages of cloud computing in the enterprise begins with establishing a trusted approach to the cloud.

Three major concepts:

security
control
compliance and
service-level management
Cisco Secure Cloud Data Center Framework

Threat Profile
- Service Disruption
- Intrusions and Takeover
- Data Leakage
- Data Disclosure
- Data Modification
- Identify Theft and Fraud

Cloud Data Center Visibility
- Identification
- Monitoring
- Correlation

Cloud Data Center Protection
- Harden
- Isolate
- Enforce

Cloud Data Center Building Blocks
- Unified Compute
- Unified Fabric
- Virtualization

Cloud Data Center Control

Cloud Data Center Compliance and SLA
Cisco Secure Cloud Data Center
Control data provision, access management systems, data encryption policies, and Identity management.
Challenges

outages and security breaches
Amazon S3 downtime, Gmail email deletion

unfaithful cloud service providers
discarding data, hiding data loss

insufficient traditional cryptographic primitives
downloading for integrity verification
Publicly Auditable Cloud Storage Services

• data owners resort to an external third party auditor (TPA)

• transparent and cost-effective method for establishing trust between data owner and cloud server

• TPA reports help users to evaluate the risk of cloud data services and help cloud service provider to improve their cloud based service platform
System Architecture
Problem statement

Data owner represents either the individual or the enterprise customer which has large data files to be stored in the cloud repository.

Cloud services provide on-demand sharing among a group of trusted users or employees in the enterprise organization.

We assume single writer/many readers scenario – only data owner can dynamically interact with CS to update the data.
Problem statement

**data owner**

**user**

**cloud servers (CS)**

**TPA**

**Cloud Server** is an entity managed by **Cloud Service Provider (CSP)**. CS has significant storage space and computation resource to maintain the clients’ data.

The CS is semi-trusted in the sense that most of the time it behaves properly and does not deviate from the prescribed protocol execution.
Problem statement

data owner
user
cloud servers (CS)
TPA

TPA is a reliable and independent institution. It should be able to efficiently audit the cloud data storage **without local copy of data** and **without any additional online burden** for data owners.
Desirable Properties

• Minimize Auditing Overhead
• Protect Data Privacy
• Support Data Dynamics
• Support Batch Auditing
Desirable Properties

• Minimize Auditing Overhead

Overhead imposed on the CS by the auditing process must not outweigh its benefits. Overhead includes I/O cost for data access and bandwidth cost for data transfer.
Desirable Properties

• Protect Data Privacy

Important aspect of service level agreement. It should not violate the owner’s data privacy. TPA should be able to audit the data without demanding the local copy of data or learning the data content.
Desirable Properties

• Support Data Dynamics

Owners should be able to modify the data dynamically whenever they have to. Thus auditing protocol should incorporate the feature of data dynamics in cloud computing.
Desirable Properties

• Support Batch Auditing

When receiving multiple auditing tasks from different owners’ delegations, a TPA should be able to handle them in a fast yet cost-effective fashion. It enables the scalability of the auditing process.
Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li
“Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing,”
Refer to Cisco Cloud Computing Architecture Framework
Introduction

- In the cloud, the clients themselves are unreliable or may not be able to afford the overhead of performing frequent integrity checks.
- Another major concern among previous designs is that of supporting dynamic data operation for cloud data storage applications.
- Moreover, as will be shown later, the direct extension of the current provable data possession (PDP) or proof of retrievability (PoR) schemes to support data dynamics may lead to security loopholes.
Problem Statement

- It focuses on **how to ensure publicly auditable secure cloud data storage services**
- It assumes the TPA, who is in the business of auditing, is reliable and independent, and thus has no incentive to collude with either the CS(Cloud Server) or the owners during the auditing process.
- The TPA should be able to efficiently audit the cloud data storage **without local copy** of data and **without any additional online burden for data owners**.
Motivate the public auditing system of data storage security in Cloud Computing, and propose a protocol supporting for fully dynamic data operations, especially to support block insertion, which is missing in most existing schemes.

Extend our scheme to support scalable and efficient public auditing in Cloud Computing. In particular, this scheme achieves batch auditing where multiple delegated auditing tasks from different users can be performed simultaneously by the TPA.

Prove the security of our proposed construction and justify the performance of our scheme through concrete implementation and comparisons with the state of the art.
System Model

- It only considers verification schemes with public auditability:
  - Any TPA in possession of the public key can act as a verifier.
  - Assume that TPA is unbiased while the server is untrusted.
  - The client may frequently perform block-level operations on the data files.
  - The most general forms of these operations we consider in this paper are modification, insertion, and deletion.
The checking scheme is secure if
1) there exists no polynomial time algorithm that can cheat the verifier with non-negligible probability.

2) there exists a polynomial time extractor that can recover the original data files by carrying out multiple challenges-responses.
Design Goals

- **Public auditability for storage correctness assurance:** to allow anyone, not just the clients who originally stored the file on cloud servers, to have the capability to verify the correctness of the stored data on demand.

- **Dynamic data operation support:** to allow the clients to perform block-level operations on the data files while maintaining the same level of data correctness assurance. The design should be as efficient as possible so as to ensure the seamless integration of public auditability and dynamic data operation support.

- **Blockless verification:** no challenged file blocks should be retrieved by the verifier (e.g., TPA) during verification process for efficiency concern.
Construction

- To effectively support public auditability without having to retrieve the data blocks themselves, we resort to the **homomorphic authenticator technique**.

- **Homomorphic authenticators** are unforgeable metadata generated from individual data blocks, which can be securely aggregated in such a way to assure a verifier that a linear combination of data blocks is correctly computed by verifying only the aggregated authenticator.

- It proposes to use PKC (Public Key Cryptography) based homomorphic authenticator (e.g., BLS signature).
Dynamic Data Operation with Integrity Assurance
Merkle Hash Tree (MHT)

Fig. 2. Merkle hash tree authentication of data elements. We treat the leaf nodes $h(x_1), \ldots, h(x_n)$ as the left-to-right sequence.
The verifier with the authentic hr requests for \{ x_2, x_7 \} and requires the authentication of the received blocks. The prover provides the verifier with the auxiliary authentication information (AAI) \( \Omega_2 = < h(x_1), h_d > \) and \( \Omega_7 = < h(x_8), h_e > \). The verifier can then verify \( x_2 \) and \( x_7 \) by first computing \( h(x_2), h(x_7) \); 
\( h_c = h(h(x_1) \parallel h(x_2)) \) 
\( h_f = h(h(x_7) \parallel h(x_8)) \); 
\( h_a = h(h_c \parallel h_d) \); 
\( h_b = h(h_e \parallel h_f) \) 
\( h_r = h(h_a \parallel h_b) \), and then checking if the calculated \( h_r \) is the same as the authentic one.
Protocol for Provable Data Update (Modification and Insertion)

1. Generate \( \sigma'_i = (H(m'_i) \cdot u'^{m'_i})^\alpha_i \);

\[
\frac{(\mathcal{M}(\mathcal{T}), i, m'_i, \sigma'_i)}{\text{update request update}}
\]

2. Update \( F \) and compute \( R' \).

\[
\frac{(\Omega_i, H(m_i), \text{sig}_k(H(R)), R')}{{\text{update proof} \ P_{\text{update}}}}
\]

3. Compute \( R \) using
   \( \{H(m_i), \Omega_i\} \);

4. Verify \( \text{sig}_k(H(R)) \).
   Output \( \text{FALSE} \) if fail.

5. Compute \( R_{\text{new}} \) using
   \( \{\Omega_i, H(m'_i)\} \).
   Verify
   \( \text{update} \) by checking
   \( R_{\text{new}} \not= R' \).
   Sign \( R' \) if succeed.

\[
\frac{\text{sig}_k(H(R'))}{{\text{update R's signature}}}
\]

6. Update R's signature.
Dynamic Data Operation with Integrity Assurance

- Data Modification:

Fig. 3. Example of MHT update under block modification operation. Here, $n_i$ and $n'_i$ are used to denote $H(m_i)$ and $H(m'_i)$, respectively.
Data Insertion

Fig. 4. Example of MHT update under block insertion operation. Here, $n_i$ and $n^*$ are used to denote $H(m_i)$ and $H(m^*)$, respectively.
Data Deletion

Fig. 5. Example of MHT update under block deletion operation.
Performance Analysis

- It implemented both our BLS and RSA-based instantiations as well as the extending original PDP scheme in Linux.
  - C++ with Pairing-Based Cryptography (PBC) library version 0.4.18 and the crypto library of OpenSSL version 0.9.8h.
  - Intel Core 2 processor 2.4GHz.
  - Memory: 768 MB
  - 7200 RPM Western Digital 250 GB
Tolerance rate limit: 99%

It uses batch auditing technique to reduce the computation time, which technique aggregates much verifications into one.
Introduction

- Traditional cryptographic primitives for the purpose of data security protection can not be directly adopted due to the users’ loss control of data under Cloud Computing.
- To ensure storage correctness under dynamic data update is hence of paramount importance.
- Some researches can not address all the security threats in cloud data storage, since they are all focusing on single server scenario and most of them do not consider dynamic data operations.
- In this paper, we propose an effective and flexible distributed scheme with explicit dynamic data support to ensure the correctness of users’ data in the cloud.
Contribution

- Use challenge-response protocol in our work further provides the localization of data error.
- The new scheme supports secure and efficient dynamic operations on data blocks, including: update, delete and append.
- Extensive security and performance analysis shows that the proposed scheme is highly efficient and resilient against Byzantine failure, malicious data modification attack, and even server colluding attacks.
Problem Statement

Assumption

- Data redundancy can be employed.
- For application purposes, the user interacts with the cloud servers via CSP to access or retrieve his data.
- The user may need to perform block level operations on his data. (Block update, delete, insert, append)
- Maintained. That is, users should be equipped with security means so that they can make continuous correctness assurance of their stored data even without the existence of local copies.
Problem Statement (cont.)

- We consider two types of adversary with different levels of capability in this paper:
- Weak Adversary: The adversary is interested in corrupting the user’s data files stored on individual servers.
- Strong Adversary: This is the worst case scenario, in which we assume that the adversary can compromise all the storage servers so that he can intentionally modify the data files as long as they are internally consistent.
Design Goals

- Storage correctness
- Fast localization of data error
- Dynamic data support
- Dependability: to enhance data availability against Byzantine failures, malicious data modification and server colluding attacks, i.e. minimizing the effect brought by data errors or server failures.
- Lightweight: to enable users to perform storage correctness checks with minimum overhead.
Ensuring Cloud Data Storage

- File Distribution Preparation
- Challenge Token Precomputation
- Correctness Verification and Error Localization
- File Retrieval and Error Recovery
A \((m + k, k)\) Reed-Solomon erasure-correcting code is used to create \(k\) redundancy parity vectors from \(m\) data vectors in such a way that the original \(m\) data vectors can be reconstructed from any \(m\) out of the \(m + k\) data and parity vectors.

For support of efficient sequential I/O to the original file, our file layout is systematic, i.e., the unmodified \(m\) data file vectors together with \(k\) parity vectors is distributed across \(m + k\) different servers.
The systematic layout with parity vectors is achieved with the information dispersal matrix $A$, derived from an $m \times (m + k)$ Vandermonde matrix with elementary row transformations:

$$A = (I|P) = \begin{pmatrix}
1 & 1 & \cdots & 1 & 1 & \cdots & 1 \\
\beta_1 & \beta_2 & \cdots & \beta_m & \beta_{m+1} & \cdots & \beta_n \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
\beta_1^{m-1} & \beta_2^{m-1} & \cdots & \beta_m^{m-1} & \beta_{m+1}^{m-1} & \cdots & \beta_n^{m-1}
\end{pmatrix}$$

where $I$ is a $m \times m$ identity matrix and $P$ is the secret parity generation matrix with size $m \times k$. 
The user obtains the encoded file $G$

$$G = F \cdot A = (G^{(1)}, G^{(2)}, \ldots, G^{(m)}, G^{(m+1)}, \ldots, G^{(n)})$$

$$= (F_1, F_2, \ldots, F_m, G^{(m+1)}, \ldots, G^{(n)}),$$

where $G^{(j)} = (g_1^{(j)}, g_2^{(j)}, \ldots, g_l^{(j)})^T \; (j \in \{1, \ldots, n\})$. 
In order to achieve assurance of data storage correctness and data error localization simultaneously, our scheme entirely relies on the pre-computed verification tokens.

Main idea:
- Before file distribution the user pre-computes a certain number of short verification tokens on individual vector $G^{(j)} (j \in \{1, \ldots, n\})$, each token covering a random subset of data blocks.
- When the user wants to make sure the storage correctness for the data in the cloud, he challenges the cloud servers with a set of randomly generated block indices.
- Upon receiving challenge, each cloud server computes a short “signature” over the specified blocks and returns them to the user. The values of these signatures should match the corresponding tokens pre-computed by the user.
Correctness Verification and Error Localization

- The response values from servers for each challenge not only determine the correctness of the distributed storage, but also contain information to locate potential data error(s).

1) The user reveals the $\alpha_i$ as well as the $i$-th permutation key $k_{p rp}^{(i)}$ to each server.
2) The server storing vector $G^{(j)}$ aggregates those $r$ rows specified by index $k_{p rp}^{(i)}$ into a linear combination

$$R_{i}^{(j)} = \sum_{q=1}^{r} \alpha_i^q \times G^{(j)}[\phi_{k_{p rp}^{(i)}}(q)].$$
Once the inconsistency among the storage has been successfully detected, we can rely on the pre-computed verification tokens to further determine where the potential data error(s) lies in.
The detection probability $P$ against data modification. We show $P$ as a function of $l$ (the number of blocks on each cloud storage server) and $r$ (the number of rows queried by the user, shown as a percentage of $l$) for three values of $z$ (the number of rows modified by the adversary).
As $m$ increases, the length $l$ of data vectors on each server will decrease, which results in fewer calls to the Reed-Solomon encoder. Thus the cost in the top table decreases when more data vectors are involved.

### TABLE I: The cost of parity generation in seconds for an 8GB data file. For set I, the number of parity servers $k$ is fixed; for set II, the number of data servers $m$ is constant.
Summary

1. Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing
2. Ensuring Data Storage Security in Cloud Computing
Thank you!
Note

- Homomorphic Authenticator

---

Kotla, Alvisi, Dahlin, Usenix 2007:

\[ (h = h(M)) \lor (M) \]

\[ P \]
**Question**: How is the internal node value of a hash tree recalculated in case of an error?

**Ans**: As the tree node values are the result of the hash function calculation, in case if some nodes are failed, the tree just needs to recalculate the values of the nodes where the error has occurred by using its children node values.