Enabling Efficient Batch Verification on Data Integrity for Cloud

Chin-Laung Lei
Department of Electrical Engineering
National Taiwan University
Outline

- Introduction
- System model
- Protocol
- Experiments and performance analysis
- Conclusion
Motivations

- Online sharing and co-working become a trend. ex: Dropbox, Google Docs
- We want to know everyone’s modification are intact and to make modifications undeniable.
- We wish our protocol can be applied to most of existing PDP schemes based on public-key cryptography.
Security for Cloud (Remote) Storage

- Confidentiality
  - Various encryption systems
- Integrity
  - Integrity verification protocols
- Availability
  - Redundancy
  - Error correcting code
Integrity Verification

- **Message digest**
  - Naïve approach
  - No authenticated data integrity; Bandwidth wasting
  - Deterministic

- **Provable data possession (PDP)**
  - Authenticated data integrity
  - Probabilistic

- **Proof of retrievability (PoR)**
  - Authenticated data integrity & improved availability
  - Probabilistic
Lifecycle

- Repository (data) deployment
  - Generate tags
- Integrity verification
  - Challenge data integrity
  - Generate proof of storage
- (Optional) Repository evolution
  - Generate tags for modified part
Scenario for Integrity Verification

1. Deploying Data

2. Verifying Data Integrity

Data Owner

Cloud Storage Provider

Third Party Verifier
Issues

- Replay attack
  - The status of repository is not clear
- Performance
  - Slow verification
  - Even on personal computer
- Batch verification
  - Single user
  - Multiple users
Approaches

- Replay attack
  - Revision number as timestamp
- Performance
  - Multiplication instead of exponential operations
- Batch verification
  - Repository as an single file
Scenario for Single User

Repositories on the Cloud

Data Owner

Third Party Verifier

Batch Integrity Verification

Deployment, Modification, ...

Integrity Verification
Bilinear Map

- **Bilinear:** \( \forall h_1, h_2 \in G, \text{ and } \forall a, b \in \mathbb{Z}_p, \)
  \[
e(h_1^a, h_2^b) = e(h_1, h_2)^{ab}.
\]

- **Non-degenerate:** \( e(g, g) \neq 1, \) for \( g \) is a generator of \( G. \)

- **From above properties, we can get another:**
  \[
  \forall u, v_1, v_2 \in G, \quad e(u, v_1 v_2) = e(u, v_1) \cdot e(u, v_2).
  \]
BLS Signatures

• A signer with the secret key: $\alpha$, the public key $g^\alpha$.
• Signing: $\sigma_i = H(m_i)^\alpha$.
• Verification: $e(\sigma_i, g) = e(H(m_i), g^\alpha)$. 
Tokens

• **Wang et al.** [1]:
  - **Token Generation**: \( \sigma_i = (H(m_i^*) \cdot u_{m_i}^*)^{SK} \).
  - **Verification**: \( e(\sigma, g) = e(u^\mu \prod H(m_i^*)^{v_i} , g^{SK}) \).

• **Ours method**:
  - **Token Generation**: \( \sigma_i = (h^{H(m_i^*)} \cdot u_{m_i}^*)^{SK} \).
  - **Verification**: \( e(\sigma, g) = e(u^\mu h^{\sum v_i H(m_i^*)} , g^{SK}) \).
Security Concern

- **Exponential operations → multiplication**
  - Security flaw?
    - Ex: \((H(m))^\alpha \rightarrow (g^{h(m)})^\alpha\)
    - \((g^{h(m)})^\alpha h(m)^{-1} h(m') = (g^{h(m')})^\alpha\)
Security Concern

Exponential operations $\rightarrow$ multiplication

- Security relies upon the CDH assumption
  - CDH assumption:
    - $\text{Prob}[g, g^\alpha, g^\beta \rightarrow g^{\alpha\beta}]$ is negligible
  - $\sigma \leftarrow \left( (g^\beta)^{h(m)} (g^\gamma)^x \right)^\alpha$
  - $A_1(g, g^\alpha, g^\beta, g^\gamma, h(m), x) \rightarrow \sigma$
  - $A_2(\sigma, g^\alpha, \gamma, h(m), x) \rightarrow g^{\alpha\beta}$
  - $g^{\alpha\beta} = \left( \sigma \cdot \left( (g^\alpha)^{xy} \right)^{-1} \right)^{(h(m))^{-1}}$
Repository Deployment
Repository Deployment

- **Parameter:**
  - $H_1 : \{0,1\}^* \rightarrow G$; $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$; $H_3 : \{0,1\}^* \rightarrow \{0,1\}^l$
  - $PK \leftarrow (V, \tilde{V}, h) = (g^\alpha, g^\beta, g^\gamma); SK \leftarrow (\alpha, \beta, \gamma)$
  - $u \leftarrow \tilde{V}^{H_2(ID||fn)}$

- **Tag generation**
  - For each $m_i \in F$
    - $\sigma_i \leftarrow (h^{H_2(H_2(m_i)||i)} \cdot u^m_i)^\alpha$
  - $\Phi \leftarrow \{\sigma_i\}_{1 \leq i \leq n}$
  - Generate MHT from $\{H_3(H_2(H_2(m_i)||i))\}_{1 \leq i \leq n}$
  - Sign on the root of the MHT:
    - $sig(R, ver) \leftarrow (H_1(R||ver))^\alpha$
Integrity Verification

Legend:
- VER
- SRV
- Public Bulletin
Integrity Verification

Proof:
- $I$ is the set of challenged blocks
- $\mu \leftarrow \sum_{i \in I} v_i m_i \land \sigma \leftarrow \prod_{i \in I} \sigma_i$
- $\Omega_I$ is the set of nodes to rebuild the root

Verify:
- Rebuild root $R'$
- Check if $e(sig(R, ver), g) = e(H_1(R' || ver), V)$
- $\rho \leftarrow \sum_{i \in I} H_2(H_2(m_i) || i) \cdot v_i$
- Check if $e(\sigma, g) = e(h^\rho u^\mu, V)$
Repository Evolution
Batch Verification for Single User

\begin{itemize}
  \item \texttt{TokenGen(*)} of $F_m$ in B-VCPDP
  \item Insert $F_m$ into workspace $W$
  \item $\texttt{sigw}$
  \item $\texttt{sig}_{F_m}$
  \item $\texttt{F_m}$
  \item $\texttt{F_m}$
  \item Create $\texttt{F_m}$
  \item $\texttt{F_m}$
  \item $\texttt{F_m}$
  \item $\texttt{F_m}$
  \item $\texttt{F_m}$
  \item $\texttt{F_m}$
\end{itemize}
Probabilistic Detection

Number of blocks needed to fulfill certain detection rate under various data corruption rate.
## Probabilistic Detection

### Check points

<table>
<thead>
<tr>
<th># of Challenged Blocks</th>
<th>Detection Rate</th>
<th>Data Corruption Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>95%</td>
<td>1%</td>
</tr>
<tr>
<td>460</td>
<td>99%</td>
<td>1%</td>
</tr>
<tr>
<td>4610</td>
<td>99%</td>
<td>0.1%</td>
</tr>
<tr>
<td>6910</td>
<td>99.9%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>
## Benchmarks

<table>
<thead>
<tr>
<th>Operation</th>
<th>Personal Computer</th>
<th>Smart Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication $Z_p (ab)$</td>
<td>$5 \times 10^{-6} \text{sec}$</td>
<td>$6 \times 10^{-5} \text{sec}$</td>
</tr>
<tr>
<td>Multiplication $G (gh)$</td>
<td>$1.45 \times 10^{-4} \text{sec}$</td>
<td>$1.26 \times 10^{-3} \text{sec}$</td>
</tr>
<tr>
<td>Exponential $G (g^a)$</td>
<td>$3.29 \times 10^{-2} \text{sec}$</td>
<td>$2.64 \times 10^{-1} \text{sec}$</td>
</tr>
<tr>
<td>Pairing $G \times G \rightarrow G_T (e(g^a, h^b))$</td>
<td>$3.84 \times 10^{-2} \text{sec}$</td>
<td>$5.41 \times 10^{-1} \text{sec}$</td>
</tr>
</tbody>
</table>
Verification Time

Client-side verification time with 6910-block challenge
Verification Time

Client-side verification time with 512-megabyte file
Scenario for Multiple Users

Repository with Three Privilege Domains

Batch Integrity Verification

Integrity Verification

Third Party Verifier

Data Owner

Deployment, Modification, ...
From Single User to Multiple Users

- **Access control**
  - Who can commit modifications of a certain part?
- **Batch verification**
  - Verify integrity across different users’ data
- **Race condition**
  - Concurrent write of the same project?
Approaches

- Access control
  - Multiple authority
    - Hierarchical
- Batch verification
  - Repository as a single file
- Race condition
  - Branching-and-merging
Repository Deployment
Key Delegation

1) $\eta_{CL}, \theta_{CL} \overset{R}{\leftarrow} Z^*_p$
2) $\pi_{CL,i} \leftarrow \alpha \pi_i + \eta_{CL}, \forall i \in P_{CL}$, where $P_{CL}$ is the set of privilege domains which are granted to $CL$
3) $\tilde{W}_{CL,i} \leftarrow g^{-\eta_{CL} \pi_i^{-1}}, \forall i \in P_{CL}$
4) $SK_{CL} \leftarrow \{\theta_{CL}, \pi_{CL,i}\}_{i \in P_{CL}}$
5) $PK_{CL} \leftarrow (W_{CL}, \tilde{W}_{CL}, \overline{W}_{CL}) = (g^{\theta_{CL}}, \{\tilde{W}_{CL,i}\}_{i \in P_{CL}}, H_1(W_{CL} \parallel \tilde{W}_{CL})^\alpha)$
Tag Generation

1) \( \sigma_i \leftarrow (h^{H_2(H_2(m_i)||i)} \cdot u^{m_i})^{\pi_{cL}, x} \)

2) \( \Phi \leftarrow \{\sigma_i\}_{1 \leq i \leq n} \)

3) Generate an MHT, \( \mathcal{T}_B \), from
\( \{H_3(H_2(H_2(m_i)||i))\}_{1 \leq i \leq n} \)

4) \( \text{sig}_{cL}(R, \text{ver}) \leftarrow (H_1(R||\text{ver}))^{\theta_{cL}}, \text{where } R \text{ is the} \) root of \( \mathcal{T}_B \)

5) \( \mathcal{CL} \rightarrow \mathcal{SRV}: \{fn, u, \text{ver, F, } \Phi, \text{sig}_{cL}(R, \text{ver})\}, \text{where } fn \text{ is the file name} \)
Integrity Verification
Repository Evolution
Batch Verification

\[ \text{TokenGen}(\cdot) \text{ of } F_m \]
Branching-and-Merging

- Before modify shared data
  - Copy to one’s own privilege domain (branching)
- After finish the modification
  - Coordinate with other collaborators
  - Write the modifications to the trunk (merging)
Branching-and-Merging (Example)
Conclusion

- Efficient integrity verification
  - Can even run on smart phone!
- Batch verification
  - Convenient for verifiers
  - Suitable for online co-working
Thank You
Appendix

Step 1. Choose a File to Verify!

A.txt  ✔
B.txt  ❌
C.txt  ✔
D.txt  ❌

OK

Step 1. Choose a File to Verify!

458  ✔
687  ❌
6904  ❌
6904(pre)  ❌

PDP

Finishing challenge generating.

[22:13:45]
Send challenge to server.

[22:13:46]
Waiting server reply the corresponding proof...

[22:13:57]
Receive a proof.

[22:13:57]
Calculating the result...

[22:14:16]
File: A.txt  C.txt  458 challenge blocks
The Result: TRUE