On Integral Data Sharing Between Untrusted Clients with Accurate and Authentic Proofs

Jun Li

University of Oregon
Network Security Research Laboratory
http://www.cs.uoregon.edu/~lijun
Problem
Problem

Server

I’m heavily loaded

client

client

client

client

client

client

client
Problem

Server

client

client

client

client

client

client

client

client
Problem

Server

I can easily serve thousands of clients

Hmm … it is not secure to share data among clients
SSL is limited

- SSL cannot address the problem designed to protect client-server channel
- thus only protects those clients directly fetching data from a server
- Both client-server and client-client communications must be protected
- this is a hybrid peer-to-peer paradigm
- Security issues often involve three parties: server, provider clients, recipient clients
Outline

- mSSL overview
- mSSL's data integrity solution
- mSSL's proof of service design
- Evaluation results
- Conclusion
mSSL Overview

- mSSL is a protocol to support various applications, which can have different security needs.

- While allowing clients to share data traditionally downloaded directly from their server, the security should not be weakened compared to the traditional client-server model.
mSSL Security Functionalities

- **Client authentication**: to decide whether a client is authenticated to obtain data
- **Data confidentiality**: to encrypt data so that only authenticated clients can decrypt it
- **Data integrity**: to verify the content and origin of data whether it is directly received from a server or other clients
- **Proof of service**: to allow a client to prove that it provided certain data to other clients
Data Integrity

- Not at file level, but instead block level
- Not a public-key-based signature per block, but an authenticatable hash per block
- No pre-loading of all hash values, but downloading them on-demand
Merkle Hash Tree
Merkle Hash Tree

File X
Merkle Hash Tree
Merkle Hash Tree
If $H_{18}$ is authentic, all its descendants also are!
Block Integrity via Authentication Path
Block Integrity via Authentication Path

$H_{18}$

$H_{34}$

$H_2$

$b_1$

$H_{58}$
Block Integrity via Authentication Path
Block Integrity via Authentication Path
Block Integrity via Authentication Path
Block Integrity via Authentication Path
Block Integrity via Authentication Path
Block Integrity via Authentication Path
Block Integrity via Authentication Path
A block’s integrity path only consists of those hash values from its authentication path that are not locally available.

Theorem 1: Assume a block b’s authentication path $A(b) = <H^m, H^{m-1}, ..., H^1>$, if $H^{x-1}$ is available but $H^x$ is not, then b’s integrity path $mip(b) = <H^m, H^{m-1}, ..., H^x>$. 
mSSL Integrity Path

A block’s integrity path only consists of those hash values from its authentication path that are not locally available.

Theorem 1: Assume a block b’s authentication path $A(b) = <H^m, H^{m-1}, ..., H^1>$, if $H^{x-1}$ is available but $H^x$ is not, then b’s integrity path $mip(b) = <H^m, H^{m-1}, ..., H^x>$. 

leaf
mSSL Integrity Path

A block’s integrity path only consists of those hash values from its authentication path that are not locally available.

Theorem 1: Assume a block b’s authentication path $A(b) = \langle H^m, H^{m-1}, \ldots, H^1 \rangle$, if $H^{x-1}$ is available but $H^x$ is not, then b’s integrity path $mip(b) = \langle H^m, H^{m-1}, \ldots, H^x \rangle$. 
mSSL Integrity Path

- A block’s integrity path only consists of those hash values from its authentication path that are not locally available

- Theorem 1: Assume a block b’s authentication path \( A(b) = \langle H^m, H^{m-1}, ..., H^1 \rangle \), if \( H^{x-1} \) is available but \( H^x \) is not, then b’s integrity path \( mip(b) = \langle H^m, H^{m-1}, ..., H^x \rangle \).
mSSL Integrity Path

A block’s integrity path only consists of those hash values from its authentication path that are not locally available.

Theorem 1: Assume a block b’s authentication path \( A(b) = <H^m, H^{m-1}, ..., H^1> \), if \( H^{x-1} \) is available but \( H^x \) is not, then b’s integrity path \( \text{mip}(b) = <H^m, H^{m-1}, ..., H^x> \).
mSSL Integrity Path (cont’d)

- **Theorem 2**: To request \( \text{mip}(b) \), a client can just indicate \( |\text{mip}(b)| \).

- **Theorem 3**: To verify \( b \)'s integrity using \( \text{mip}(b) = \langle H^m, H^{m-1}, ..., H^x \rangle \), instead of the root on the tree, only \( \text{sibling}(H^{x-1}) \) needs to be calculated and verified.
Proof of Service

- Incentives are needed for peer clients to share data, such as:
  - crediting a provider client for providing data to other clients
  - offering discount to a recipient client for downloading data from a provider instead of the server

- A provider client needs to report its service

- The report must be trustworthy
Basic Solution

- Also block-based
  - Every file is divided into multiple blocks
  - An interlocking block-by-block verification between every pair of provider and recipient
Basic Solution

- Also block-based
  - Every file is divided into multiple blocks
  - An interlocking block-by-block verification between every pair of provider and recipient
Basic Solution

- Also block-based
  - Every file is divided into multiple blocks
- An interlocking block-by-block verification between every pair of provider and recipient
Basic Solution

- Also block-based
  - Every file is divided into multiple blocks
  - An interlocking block-by-block verification between every pair of provider and recipient
Basic Solution

Also block-based

Every file is divided into multiple blocks

An interlocking block-by-block verification between every pair of provider and recipient
Basic Solution

- Also block-based
  - Every file is divided into multiple blocks
- An interlocking block-by-block verification between every pair of provider and recipient

verify block integrity
verify ack
use this ack as prf of svc
Basic Solution

- Also block-based
  - Every file is divided into multiple blocks
- An interlocking block-by-block verification between every pair of provider and recipient

verify block integrity

verify ack
use this ack as prf of svc

next block
Issues to Consider

- Proof explosion?
  - A separate proof per block?
- Precise?
  - Runaway without acknowledging the last block received
- Server overloaded?
  - Does it have to verify each acknowledgment for a provider?
Proof of Service Protocol

A recipient will receive an encrypted block first, and receive the decrypting block key after acknowledging the receipt of the block.

Acknowledgments are cumulative, thus the most recent one can replace previous ones as the proof of service.

Every acknowledgment is signed and can be verified using the public key of the recipient.
exchange auth info and public keys
p and r exchange auth info and public keys. Then p requests block b from r.
exchange auth info and public keys

request block b

encrypted block
\[ k = f(p, r, \text{file, block id}, k_p) \]
\( k = f(p, r, \text{file}, \text{block id}, k_p) \)
\[ k = f(p, r, \text{file}, \text{block id}, k_p) \]

**Exchange Phase:**
- **p** and **r** exchange auth info and public keys.

**Request Phase:**
- **p** requests block b.
- **r** sends an encrypted block.

**Acknowledgment Phase:**
- **p** sends an acknowledgment (ack).

**Signature Phase:**
- **r** sends an acknowledgment signature (acksig = \( \text{PRV}_r \{p, r, \text{file}, \text{sack}, \text{timestamp}, \text{dig(encrypted block)} \} \))
$k = f(p, r, \text{file, block id, } k_p)$

verify ack
Use it as prf of svc

acksig = PRV_r \{p, r, \text{file, sack, timestamp, dig(encrypted block)}\}
$k = f(p, r, \text{file, block id, } k_p)$

verify ack
Use it as prf of svc

$\text{acksig} = \text{PRV}_r \{p, r, \text{file, sack, timestamp, dig(encrypted block)}\}$
$k = f(p, r, \text{file, block id, } k_p)$

verify ack
Use it as prf of svc

$\text{PRV}_p\{p, r, \text{file, block id, PUB}_r\{k\}\}$

request block $b$

encrypted block

ack

$\text{acksig} = \text{PRV}_r\{p, r, \text{file, sack, timestamp, dig(encrypted block)}\}$

block key


\[ k = f(p, r, \text{file, block id, } k_p) \]

verify ack
Use it as prf of svc

\[ \text{PRV}_p\{p, r, \text{file, block id, PUB}_r\{k\}\} \]

decrypt the block and verify its integrity
\( k = f(p, r, \text{file}, \text{block id}, k_p) \)

Verify \( \text{ack} \)

Use it as prf of svc

\( \text{PRV}_p\{p, r, \text{file}, \text{block id}, \text{PUB}_r\{k\}\} \)

\( \text{acksig} = \text{PRV}_r\{p, r, \text{file}, \text{sack}, \text{timestamp}, \text{dig(encrypted block)}\} \)

Decrypt the block and verify its integrity
\( k = f(p, r, \text{file}, \text{block id}, k_p) \)

**verify ack**
Use it as prf of svc

\( \text{PRV}_p \{p, r, \text{file}, \text{block id}, \text{PUB}_r\{k\} \} \)

\( \text{PRV}_r \{p, r, \text{file}, \text{sack}, \text{timestamp}, \text{dig(encrypted block)} \} \)

decrypt the block and verify its integrity
Pipelining the Process

The recipient must verify the integrity of block(i) before acknowledging block(i+1)

- otherwise the provider can use ack(i+1) as a proof of sending blocks up to block(i+1)
- even if block(i) is corrupted

This is then a stop-and-go process. Slow!

Every ack (thus every proof) to include digests of last $m$ encrypted blocks

- the recipient can promptly acknowledge last $(m-1)$
Colluded Cheating

server

p

ack

r
Colluded Cheating

server

$a_n$, $a_2$, $a_1$
Colluded Cheating

server

\[ a_n \quad a_2 \quad a_1 \]
Colluded Cheating

Diagram:
- Server
- Nodes: $a_n$, $a_2$, $a_1$
- Connections:
  - From server to $a_n$ and $a_2$ and $a_1$
  - From $a_n$ to $p$
  - From $a_1$ to $r$
  - Acknowledgment from $r$ to $a_1$
Colluded Cheating

server

\(a_n\) \(a_2\) \(a_1\)

\(p \leftrightarrow r\)

ack
Colluded Cheating

```
server

p<->r  p<->r  p<->r

a_n    a_2    a_1

ack

p<->r

r
```
Colluded Cheating

ACTUALLY NOT
p<->r !!
Colluded Cheating

server
Colluded Cheating

server
Colluded Cheating

server
Colluded Cheating

A <-> A

server
Colluded Cheating

server

THIS IS A SELF-SERVICE!
Colluded Cheating

server

no data!

ack
Evaluating mSSL

- The overhead of applying mSSL should be acceptable
- Experiments performed in our lab
- Under five different scenarios, depending on what mSSL functionalities are employed
- A file-sharing application is implemented to measure mSSL
Server Capacity

- # of client requests a server can handle per time unit

![Bar graph showing number of requests per minute across different scenarios.


- The graph indicates varying capacities across different scenarios, with 'P (AIP)' showing the lowest number of requests per minute.]
File Downloading Time

- The time from initiating a connection with a server to receiving the whole file

- Startup latency

![Graph showing startup latency for different scenarios](image)
**File Downloading Time**

- The time from initiating a connection with a server to receiving the whole file

- **Data transferring time**

![Graph showing data transferring time vs. file size](image)
Storage Overhead

- **Integrity:** the Merkle hash tree of a file being downloaded needs to be stored by both the provider and the recipient.

  - if a hash value is 16 bytes, and the file has $2^n$ blocks, this will be about $2^{n+5}$ bytes.

- **Proof of service:** typically a couple hundred bytes.
Control Traffic Volume

- Integrity: For n blocks, n hash values to download
- Proof of service: corresponding to each block, 1 acknowledgment, 1 delivery of a block key. Also generally 1 proof per file.
Conclusions
Conclusions

- Security concerns often arise in the hybrid peer-to-peer environment
- Higher risk compared to directly receiving data from a server
Conclusions

- Security concerns often arise in the hybrid peer-to-peer environment
  - Higher risk compared to directly receiving data from a server
- mSSL offers a set of security functionalities with low overhead
Conclusions

- Security concerns often arise in the hybrid peer-to-peer environment
  - Higher risk compared to directly receiving data from a server
- mSSL offers a set of security functionalities with low overhead
- Integrity solution: block-level, integrity-path-based, on-demand
Conclusions

- Security concerns often arise in the hybrid peer-to-peer environment
  - Higher risk compared to directly receiving data from a server
- mSSL offers a set of security functionalities with low overhead
- Integrity solution: block-level, integrity-path-based, on-demand
- Proof of service solution: minimal overhead on server, small-sized proofs, and precise!