DupLESS

Server-Aided Encryption for Deduplicated Storage

Mihir Bellare\textsuperscript{1} \hspace{1cm} Sriram Keelveedhi\textsuperscript{1} \hspace{1cm} Thomas Ristenpart\textsuperscript{2}

\textsuperscript{1}University of California, San Diego \hspace{2cm} \textsuperscript{2}University of Wisconsin-Madison
Deduplication

Avoid storing multiple copies of the same data

Deduplication happens here

Enterprise network

Used in outsourced storage services

Dropbox  Google Drive

Savings of 50% in enterprise networks [MB11]

Savings after $n$ uploads

<table>
<thead>
<tr>
<th>No dedup</th>
<th>$n \cdot \text{size of } f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedup</td>
<td>$1 \cdot \text{size of } f$</td>
</tr>
</tbody>
</table>

Store $f$ iff new
Our goals

1. Secure deduplication: Dedup + Strong security against untrusted storage

2. Compromise resilience: Meaningful security under client compromise
Overview

**DupLESS** (DuplicateLess Encryption for simple storage)

**First solution** to achieve secure deduplication with compromise resilience

- Can be deployed *transparently* over existing systems
  - Implementations over Dropbox, Google Drive
- **Modest performance overhead** over plaintext dedup
- Storage savings match plaintext dedup
Current approaches
Attempt 1: Client specific keys

\[ C_A \leftarrow E(K_A, f) \]

\[ C_B \leftarrow E(K_B, f) \]

Deduplication cannot work

[GSMB03], [KRS*03], [KPR11]
Attempt 2: Network-wide key

Untrusted Storage service

No compromise resilience
All data is insecure even if one client is compromised

[CBO07], [RS06]
Attempt 3: Convergent Encryption

To encrypt message $m$:

1. Compute $k = H(m)$.
2. Compute $c^1 = E_k(m)$.
3. Compute $c^2 = E_{K_A}(m)$.

To decrypt ciphertext $c^1, c^2$:

1. Compute $k = D_{K_A}(c^1)$.
2. Compute $m = D_k(c^2)$.

H: Hash fn. $\rightarrow$ SHA256
$\mathcal{E} = (E, D)$: Enc. scheme $\rightarrow$ CTR[AES128]
Attempt 3: Convergent encryption

\[ C^1 \leftarrow E(H(f), f) \]
\[ C_A^2 \leftarrow E(K_A, H(f)) \]
\[ C^1, C_A^2 \]

✓ **Deduplication**: Everyone encrypting \( f \) gets \( C^1 \)

✓ **Compromise resilience**: No system-wide secret

[ABC*02], [SGLM08],...
Attempt 3: Convergent encryption

Brute force attacks: The dirty secret of convergent encryption

If $m$ comes from $S = \{m_1, m_2, ..., m_n\}$
attacker can recover $m$ from $c \leftarrow E(H(m), m)$

<table>
<thead>
<tr>
<th>BruteForce$_S(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $m_i \in S$ do</td>
</tr>
<tr>
<td>$m' \leftarrow D(H(m_i), c)$</td>
</tr>
<tr>
<td>If $m_i = m'$ then return $m_i$</td>
</tr>
</tbody>
</table>

Attack runs in time proportional to $|S|$

Security only when $|S|$ too large to exhaust

Real files are often predictable!

Message-Locked encryption [BKR13]

• Generalizes convergent encryption
• Captures properties needed for secure deduplication

Thm: Brute-force attacks exist for all message-locked encryption schemes
<table>
<thead>
<tr>
<th>Property</th>
<th>Client specific keys</th>
<th>Network wide key</th>
<th>Convergent encryption</th>
<th>DupLESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deduplication</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Compromise resilience</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Brute-force attack resilience</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

**DupLESS: First to achieve all three properties!**
Server-aided encryption
Our key insight: Server-aided encryption

\[ F : \text{A pseudorandom function (PRF)} \]
Examples: AES128, HMAC[SHA256]

**Deduplication**: Any client encrypting \( f \) produces same \( C^1 \)
\( C^2 \) ciphertexts cannot be dedup’ed, but they are tiny
# Dealing with attacks

<table>
<thead>
<tr>
<th>Attack type</th>
<th>Reason for security</th>
<th>Best attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>External attacks</td>
<td>Authenticating clients</td>
<td>Break encryption <em>(very hard)</em></td>
</tr>
<tr>
<td>Client compromise</td>
<td>KS interaction overhead</td>
<td>Online brute-force <em>(slow)</em></td>
</tr>
<tr>
<td>KeyServer compromise</td>
<td>Obliviously evaluating $F$</td>
<td>Brute-force attacks</td>
</tr>
</tbody>
</table>

## File server

- Alice
  - $K_A$ ← $F(K_S, H_f)$
  - $C_A^1 ← E(K, f)$
  - $C_A^2 ← E(K_A, K)$
  - $C_A^1, C_A^2$ ← $K$

- Bob
  - $C_B^1 ← E(K_B, K)$
  - $C_B^2 ← E(K_A, K)$
  - $C_B^1, C_B^2$ ← $K$

- Insecure

## KeyServer

- $K ← F(K_S, H_f)$

## Authenticate clients

- Online brute-force

## Online brute-force

- $K_1 ← F(K_S, H_f)$

## Attack type

- External attacks
- Client compromise
- KeyServer compromise

## Reason for security

- Authenticating clients
- KS interaction overhead
- Obliviously evaluating $F$
Oblivious PRF (OPRF) protocol

\( F \): A Pseudorandom function (PRF)

Security, informally:

1. \( F \) is a PRF (when not given \( VK \))
2. Server learns nothing, client learns only \( K \)
3. Client can detect when server does not return \( K \)

Verifiable OPRF: Client can verify \( K = F(K_S, H(f)) \)
Oblivious PRF protocol

Securely evaluate AES circuit? **Too slow!**

Oblivious PRFs from **unique blind signatures** [CNS07, DeCSTW12]

Blind Signatures from RSA-FDH [C82, BNPS09]

**Main idea**

- Server signs messages with **RSA-FDH signatures**
- Obliviousness through blinding

- Verifiable
- Single round
- **KeyServer**: 1 RSA exponentiation
- **Client**: 2 RSA exponentiations + 1 inverse
Client-KS protocol

Assume PKI with trusted CA

**Standard protocol**

- TLS 2way auth handshake + OPRF query & response over secure channel
- 4 rounds for each query

**Optimized protocol**

- **Session initialization**
  - TLS 2way auth handshake
  - Session key sent over secure channel

- **Making a query**
  - Client sends OPRF input
  - KeyServer performs checks, returns OPRF output

**Preventing query forgery**

- Per session keys + sequence numbers + MAC
- 1 round for each query
KS performance

<table>
<thead>
<tr>
<th></th>
<th>Naïve HTTPS based</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>384ms</td>
<td>278 ms</td>
</tr>
<tr>
<td>Query response (Low load)</td>
<td>83 ms</td>
<td>118 ms</td>
</tr>
<tr>
<td>Query response (Heavy load)</td>
<td>118 ms</td>
<td></td>
</tr>
<tr>
<td>Ping times</td>
<td>78 ms</td>
<td></td>
</tr>
</tbody>
</table>

KS located on an Amazon EC2 m1Large instance.

Heavy load \( \approx 3k \) queries per second
Rate limiting

**Goal**: Slow down online brute-force trials from attacker controlled clients

**Strategy**: Limit clients to $q$ queries per epoch
- One epoch lasts $\tau$ units of time

Setting bound $q$
- **Too low**: Normal usage affected
- **Too high**: Attacks not slowed down

Setting epoch duration $\tau$
- Must handle bursty workloads
- Systems exhibit periodic patterns, Eg: 1 week

Randomized encryption when KS unavailable
- Availability not affected by bad parameter choices

Rate limiting can slow down brute-force attacks by 4000x
DupLESS system design
DupLESS (DuplicateLess Encryption for simple storage)

Implement API over encrypted data

**Encrypt and decrypt files**

**Handle file names and paths**

**Run Transparently:**
- Low overhead
- Works when KS is down
- No client-side state
A put query in DupLESS

\[ (p, f, m) : \]

1. Derive key \( K \) for \( m \) from KeyServer
2. \( C_p \leftarrow \text{DAE}(K_A, p); C_f \leftarrow \text{DAE}(K_A, f) \)
3. If not \text{shouldDedup}(p, f, m) then pick \( K \) at random
4. \( C^1 \leftarrow E(K, m); C^2 \leftarrow E(K_A, K) \)
5. \text{PutSS} \((C_p, C_f0, C^1)\), \text{PutSS} \((C_p, C_f1, C^2)\)

Put (\( p, f, m \)):

\[ p: \text{Path} \]
\[ f: \text{File name} \]
\[ m: \text{Contents} \]
Performance: Latency

**DupLESS client**

- Written in Python, command-line interface
- Dropbox and Google Drive can work as storage service

### Put

<table>
<thead>
<tr>
<th>File size</th>
<th>16KB</th>
<th>16MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead*</td>
<td>16%</td>
<td>14%</td>
</tr>
</tbody>
</table>

### Get

<table>
<thead>
<tr>
<th>File size</th>
<th>16KB</th>
<th>16MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead*</td>
<td>10%</td>
<td>5%</td>
</tr>
</tbody>
</table>

* Overhead of DupLESS over Dropbox

---

X-axis: File size (KB)  
Y-axis: Time (ms)
Bandwidth overhead

<table>
<thead>
<tr>
<th>File size</th>
<th>16KB</th>
<th>16MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DupLESS bandwidth overhead compared to plain Dropbox</td>
<td>16%</td>
<td>&lt;1%</td>
</tr>
</tbody>
</table>

Storage overhead

DupLESS storage overhead compared to dedup over plaintexts: 4.4%

Amazon AMI dataset, total size: 2035 GB
Conclusion

Encrypted deduplication with the aid of a KeyServer

• **First solution** to provide secure deduplication + compromise resilience
• Can be deployed **transparently** over existing systems
  • Implementations over Dropbox, Google Drive
• **Nominal performance overhead** over plaintext dedup
• Storage savings match plaintext dedup
Future work

• Supporting keyword search

• Defense in depth at the KeyServer
  • Combine DoS prevention and rate-limiting

• Support complex file-systems
  • NFS, CIFS, etc.

• Exploring dedup heuristics
  • Rules on which files to select for dedup
DupLESS

Server-Aided Encryption for Deduplicated Storage

Mihir Bellare\textsuperscript{1} \hspace{1cm} Sriram Keelveedhi\textsuperscript{1} \hspace{1cm} Thomas Ristenpart\textsuperscript{2}

Thank you!

Paper available at 
\texttt{eprint.iacr.org/2013/429.pdf}

Code available at 
\texttt{cseweb.ucsd.edu/users/skeelvee/dupless}

\textsuperscript{1}University of California, San Diego \hspace{2cm} \textsuperscript{2}University of Wisconsin-Madison