CRYPTOGRAPHIC PROTOCOLS: PRACTICAL REVOCATION AND KEY ROTATION

Adam Shull
Recent Ph.D. Graduate
Indiana University
Access revocation on the cloud

\[
\begin{align*}
\text{Enc}_{pk}(k_1), \text{Enc}_{k_1}^\text{Sym}(f_1) \\
\text{Enc}_{pk}(k_2), \text{Enc}_{k_2}^\text{Sym}(f_2) \\
\text{Enc}_{pk}(k_3), \text{Enc}_{k_3}^\text{Sym}(f_3)
\end{align*}
\]
Access revocation on the cloud

\[
\begin{align*}
\text{Enc}_{pk}(k_1), \text{Enc}_{k_1}^{Sym}(f_1) \\
\text{Enc}_{pk}(k_2), \text{Enc}_{k_2}^{Sym}(f_2) \\
\text{Enc}_{pk}(k_3), \text{Enc}_{k_3}^{Sym}(f_3)
\end{align*}
\]
Revocation using proxy re-encryption

\[
\begin{align*}
&\left( \text{Enc}_{\text{pub}}(k_1), \text{Enc}_{k_1}(f_1) \right) \\
&\left( \text{Enc}_{\text{pub}}(k_2), \text{Enc}_{k_2}(f_2) \right) \\
&\left( \text{Enc}_{\text{pub}}(k_3), \text{Enc}_{k_3}(f_3) \right)
\end{align*}
\]

Proxy re-encrypt from \( pk \) to \( pk' \)

\[
\begin{align*}
&\left( \text{Enc}_{\text{pub}}'(k_1), \text{Enc}_{k_1}(f_1) \right) \\
&\left( \text{Enc}_{\text{pub}}'(k_2), \text{Enc}_{k_2}(f_2) \right) \\
&\left( \text{Enc}_{\text{pub}}'(k_3), \text{Enc}_{k_3}(f_3) \right)
\end{align*}
\]
Key-scraping attack

\[
\begin{align*}
(Enc_{pk}(k_1), Enc_{Sym}^k(f_1)) &\quad \text{The symmetric keys for the files are not changed!} \\
(Enc_{pk}(k_2), Enc_{Sym}^k(f_2)) &\quad \text{A revoked user may have stored } k_1, k_2, \text{ and } k_3 \\
(Enc_{pk}(k_3), Enc_{Sym}^k(f_3)) &
\end{align*}
\]
The symmetric key must be changed!

- Decrypt with old key, encrypt with new key
  - Requires trusted re-encryptor and takes two full passes for re-encryption
- Encrypt existing ciphertext with new key
  - Decryption takes one full pass for each previous re-encryption
- Key-homomorphic pseudorandom functions
  - Allow untrusted party to re-encrypt to fresh key
  - Existing key-homomorphic pseudorandom functions are extremely slow
Security model
Security model
All-or-nothing transform (AONT)

01111011001011110100100111001110001111010100010010010111

\[
T
\]

1010110000111111010010101101101100111101010001010010111

\[
T^{-1}
\]

01111011001011110100100111001110001111010100010010010111

RSA Conference 2018
All-or-nothing transform (AONT)

\[ 011110110010111101001001110011001110001111010100010010010111 \]

\[ T \]

\[ 1010110 \ 001111 \ 110100101010 \ 11 \ 10011101100010 \ 01 \ 0010101 \ 0 \ 11000 \]

\[ T^{-1} \]

\[ ? \]
Our approach using an AONT

\[ 011110110010111101001001110011001100111001111010100010010010111 \]

\[ T \]

\[ 101011000111111011001001101101010110100111000101100101011001 \]

\[ \text{XOR with a pseudorandom string} \]

\[ 101011010011110101100101010101101001110110001011001010100011100 \]
Security intuition

\[
\begin{align*}
1010110000111111010010101011101001110110001010100110111000 & \\
101011000111111010010101011101 & \\
1010110001111111010010101011101 & \\
10101100001111111010010101011101 & \\
10101100101111010010101011101 & \\
1010110100111101010010101011101 & \\
1010110100111101010010101011101 & \\
1010110100111101010010101011101 & \\
1010110100111101010010101011101 & \\
1010110100111101010010101011101 & \\
1010110100111101010010101011101 & \\
\end{align*}
\]
Proxy re-encryption construction

- Initial ciphertext: \( \left( \text{Enc}_{pk}(k_0), T \left( \text{Enc}_{k_0}^{\text{Sym}}(f) \right) \right) \)

- Once re-encrypted ciphertext:
  \[
  \left( \text{Enc}_{pk'}(k_0), \left[ \text{Enc}_{pk'}(s_1, k_1) \right], \left[ T \left( \text{Enc}_{k_0}^{\text{Sym}}(f) \right) \right]_{\text{Ind}(s_1),\text{Ctr}(k_1)} \right)
  \]

- Twice re-encrypted ciphertext:
  \[
  \left( \text{Enc}_{pk''}(k_0), \left[ \text{Enc}_{pk''}(s_1, k_1), \text{Enc}_{pk''}(s_2, k_2) \right], \left[ T \left( \text{Enc}_{k_0}^{\text{Sym}}(M) \right) \right]_{\text{Ind}(s_1),\text{Ctr}(k_1)} \right)_{\text{Ind}(s_2),\text{Ctr}(k_2)}
  \]
Result: Much faster re-encryption

Instances of AES blockcipher vs. File size (in KiB)

- Full re-encryption
- 50% downloaded
- 75% downloaded
- 90% downloaded
Result: Much faster decryption

Instances of AES block cipher vs. Number of re-encryptions

- Full 8 KiB
- Full 40 KiB
- Full 200 KiB
- Ours 8 KiB
- Ours 40 KiB
- Ours 200 KiB
Summary

- In scenarios such as access revocation and key rotation, symmetric-key ciphertexts may need to be re-encrypted.
- Existing solutions are either insecure or too slow to be used in practice.
- Using an all-or-nothing transform, we can re-encrypt efficiently while achieving provable security under a reasonable model.
- We provide constructions for updatable symmetric-key encryption, public-key and identity-based proxy re-encryption, and revocable-storage attribute-based encryption.
Realize the need to update the symmetric key
- Many papers on public-key revocation don’t consider hybrid encryption
- Realistic security models must address key-scraping attacks

Possible future work:
- Produce a general theorem encompassing all uses of symmetric-key encryption
- Assess tradeoffs between streaming efficiency and security
- Provide a full implementation of the construction
ASYNCHRONOUS PROVABLY-SECURE HIDDEN SERVICES

Fernando Krell    Philippe Camacho
Problem: how to hide the location of a server?

- Arbitrary network topology
- One node acts as a server
- Other nodes can be clients
Problem: how to hide the location of a server?

- Arbitrary network topology
- One node acts as a server
- Other nodes can be clients

- Avoid DoS
- Reduce attack surface
- Censorship resistance
- Traffic analysis
Naive Solution: Recursive Multicast

- If $C$ contacts $S_1$, the response will arrive after $\approx 2T$
- If $C$ contacts $S_2$, the response will arrive after $\approx 6T$
Anonymity: Synchronous Solutions

Mix-nets [Chaum ’81]

- Provably secure
Anonymity: Synchronous Solutions

Mix-nets [Chaum ’81]

- Provably secure

DC-nets [Chaum ’88]

- Provably secure
Anonymity: Asynchronous Alternatives

Crowds [Reiter & Rubin ’98]

- Asynchronous
- Several attacks
Anonymity: Asynchronous Alternatives

Crowds [Reiter & Rubin ’98]

- Asynchronous
- Several attacks

Tor [Dingledine & Mathewson & Syverson ’04]

- Asynchronous
- Several attacks
- Most popular
Intersection attack \[\Rightarrow\] lower bound on communication

Thus all the nodes must participate in order to hide the server’s location.
Can we get the best of both worlds (Provably Secure and Asynchronous)?

<table>
<thead>
<tr>
<th></th>
<th>Asynchronous</th>
<th>Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provably Secure</td>
<td><em>This work</em></td>
<td>DC-nets/ mix-nets, DO’00</td>
</tr>
<tr>
<td>Heuristic Security</td>
<td>Tor, Crowds</td>
<td>Herbivore [GRPS ’03]</td>
</tr>
</tbody>
</table>
Model

- Simulation based security definition.
- Communication restricted to use $\mathcal{F}_{\text{Network}}$
  - $P_i$ is allowed to send message to $P_j$, if they are directly connected.

$$G = \langle V, E \rangle$$

$\mathcal{F}_{\text{Network}}$

$$G = \langle V, E \rangle$$

$\mathcal{F}_{\text{Network}}(v_1, \text{msg}, v_5) = v_1 : \perp$

$\mathcal{F}_{\text{Network}}(v_1, \text{msg}, v_2) = v_2 : \{\text{msg}, v_1\}$
Overview of our solution

Participant’s behavior is indistinguishable from server’s.

1. **Client.** Broadcast the request

2. **Player** $P_i$. Upon seeing a request message, send a random value $s_i$ to the server (broadcast)

3. **Player** $P_i$. Upon seeing everybody’s values $\{s_j\}$:
   - If $P_i = \text{Server}$. Secret share response $r$ using $\{s_j\}$.
     Send share $r - \sum s_j$ to client.
   - Else. Submit $s_i$ to client.

4. **Client.** Upon receiving all shares, reconstruct the server’s response $r$. 

Naive implementation has $O(n/2)$ communication complexity.
Overview of our solution

Participant’s behavior is indistinguishable from server’s.

1. **Client.** Broadcast the request
2. **Player** $P_i$. Upon seeing a request message, send a random value $s_i$ to the server (broadcast)
3. **Player** $P_i$. Upon seeing everybody’s values $\{s_j\}$:
   - If $P_i = \text{Server}$. Secret share response $r$ using $\{s_j\}$.
     Send share $r - \sum s_j$ to client.
   - Else. Submit $s_i$ to client.
4. **Client.** Upon receiving all shares, reconstruct the server’s response $r$.

Naive implementation has $O(n^2)$ communication complexity.
Efficient Implementation

- Avoid recursive multicast on every message.
- Combine encrypted shares on intermediate nodes.
Efficient Implementation

- Avoid recursive multicast on every message.
- Combine encrypted shares on intermediate nodes.

Extra Tools:
- **Homomorphic Encryption**
  - $\text{Enc}_{pk}(m_1) + \text{Enc}_{pk}(m_2) = \text{Enc}_{pk}(m_1 + m_2)$
- **Spanning Tree**
Avoiding quadratic complexity:
Phase 1: Broadcast the Request
Phase 2.a): Shares UP to root

Shares are encrypted for the server, and sent up the tree.

All shares are added using homomorphic encryption:

$$\text{Enc}_{pk^S}(y) \cdot \text{Enc}_{pk^S}(z) = \text{Enc}_{pk^S}(y + z) = \text{Enc}_{pk^S}(y')$$
Phase 2.b): Shares’ sum DOWN to server

The encrypted sum $\sum_{N_i \neq C} N_i$ share$_{N_i}$ is sent down the tree so that the server $S$ can decrypt it.
Phase 3: Server change its share

- The response to req is computed by the server S:
  \[ \text{res} := F(\text{req}) \]

- The server recomputes its own share:
  \[ \text{share}_S^{\text{new}} := \text{res} - \left( \sum_{N_i \neq \text{C}} \text{share}_{N_i} - \text{share}_S^{\text{old}} \right) \]

- The new share of the server and the share of the other nodes add up to res:
Phase 4.a): Response shares sent to root

- All $N_i \neq C$ (including $S$) send their share share$_i$ to $C$.
- $S$ will send share$_S^{new}$ instead of share$_S^{old}$.
- All shares are added using homomorphic encryption (using $C$'s public key): $\text{Enc}_{pkC}(y) \cdot \text{Enc}_{pkC}(z) = \text{Enc}_{pkC}(y + z) = \text{Enc}_{pkC}(y')$
Phase 4.b) Encrypted response sent to client

The encrypted response \( \text{res} = \sum_{N_i \neq C} \text{share}_{N_i} \) is sent down the tree so that the client \( C \) can decrypt it.
Security based on simulation

- When client is not corrupted: just simulate protocol under fake messages.
- When client is corrupted:
  - Simulator $S$ gets response from ideal functionality.
  - $S$ changes honest parties shares so that they reconstruct the correct response.
Linear Complexity

- $O(1)$ messages per Spanning Tree Edge.
- $O(1)$ homomorphic encryption operations.

Although a node can have $O(n)$ worst case complexity.
Adversary’s strategies:
- Drop messages. DoS.
- Change shares. DoS.

New Protocol:
- Messages are signed.
- Use recursive multicast for all messages ($O(n^2)$ Comm. complexity).
- Append zero-knowledge proof that ciphertexts encrypt same share. Allow identification of malicious players.
Zero-knowledge proof

• Prove that two ciphertexts encrypt same message, except...

• The two ciphertexts encrypt same message OR
• The issuer is the server

• Do not reveal whether which of one.pnum, two.pnum is true.

• Reduces to simple Σ-protocol for relation:

\[ \{ (A, B; r) : A = g^{r/one.pnum} \land B = g^{r/two.pnum} \} \]

\[ \bigcup \{ (D; s) : D = g^{s} \} \]
Zero-knowledge proof

- Prove that two ciphertexts encrypt same message, except...
- Server actually **changes** the share.
- Proof needs to convince that
  - [1] The two ciphertexts encrypt same message **OR**
  - [2] The issuer is the server
Zero-knowledge proof

- Prove that two ciphertexts encrypt same message, except...
- Server actually **changes** the share.
- Proof needs to convince that
  - [1] The two ciphertexts encrypt same message OR
  - [2] The issuer is the server
- Do not reveal whether which of [1], [2] is true.
- Reduces to simple $\Sigma$-protocol for relation

$$R_{g_1,g_2} = \{(A, B; r) : A = g_1^r \land B = g_2^r\} \cup \{(D; s) : D = g^s\}$$
Future work

- Resilience. Protocol needs to succeed even if some players disappears
- Improve communication complexity of second protocol
- Empirical Study
- Find trade-offs to scale current solution
- Server anonymity

Questions?