Improving Speed and Security in Updatable Encryption Systems

Dan Boneh  
Stanford University

Saba Eskandarian  
Stanford University

Sam Kim  
Stanford University

Maurice Shih  
Cisco Systems
Key Rotation
Key Rotation
Good Reasons to Rotate Keys

1. Recommended by NIST (Special Publication 800-57)
Good Reasons to Rotate Keys

1. **Recommended by NIST** (Special Publication 800-57)

2. **Recommended by Google** (cloud.google.com/kms/docs/key-rotation)
Good Reasons to Rotate Keys

1. **Recommended by NIST** *(Special Publication 800-57)*

2. **Recommended by Google** *(cloud.google.com/kms/docs/key-rotation)*

3. **Required by PCI DSS** *(PCI DSS 3.6.4)*
Good Reasons to Rotate Keys

1. Recommended by NIST (Special Publication 800-57)

2. Recommended by Google (cloud.google.com/kms/docs/key-rotation)

3. Required by PCI DSS (PCI DSS 3.6.4)

…But Why?
Good Reasons to Rotate Keys

Reasons to rotate keys for data stored in the cloud:

- Compromised keys need to be taken out of use
- Proactive refresh of keys
- Access control enforcement
How to Rotate Keys in the Cloud?

Idea 1: send keys to cloud
How to Rotate Keys in the Cloud?

Idea 1: send keys to cloud
How to Rotate Keys in the Cloud?

Idea 1: send keys to cloud
How to Rotate Keys in the Cloud?

Idea 1: send keys to cloud

No Security!!
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload

Note: cloud must be trusted not to keep old ciphertexts
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload

High communication and client computation cost!
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload

Can we do better?

High communication and client computation cost!
Updatable Encryption [BLMR13, EPRS17, LT18, KLR19, BDGJ19]

Client sends small *update token*

Server updates ciphertext *without* learning key or data
Our Contributions & Roadmap

Improvements over prior security definitions
- Additional requirements for security

Two new constructions of updatable encryption
- From Nested AES: very fast, only supports bounded updates
- From KH-PRF based on RLWE: ~500x faster than prior work

Performance evaluation and comparison to prior work

Recommendations for usage
Security and Functionality Goals

1. Adversary without access to any key does not learn data
Security and Functionality Goals

1. Adversary without access to any key does not learn data
2. Adversary with access to the current key/data cannot get more data than it has already exfiltrated after rekeying
Security and Functionality Goals

1. Adversary without access to any key does not learn data
2. Adversary with access to the current key/data cannot get more data than it has already exfiltrated after rekeying
3. Client-server communication small
Security and Functionality Goals

1. Adversary without access to any key does not learn data
2. Adversary with access to the current key/data cannot get more data than it has already exfiltrated after rekeying
3. Client-server communication small
4. Client computation small
Security and Functionality Goals

1. Adversary without access to any key does not learn data
2. Adversary with access to the current key/data cannot get more data than it has already exfiltrated after rekeying
3. Client-server communication small
4. Client computation small

Limitations
1. Server computation will be linear
Security and Functionality Goals

1. Adversary without access to any key does not learn data
2. Adversary with access to the current key/data cannot get more data than it has already exfiltrated after rekeying
3. Client-server communication small
4. Client computation small

Limitations

1. Server computation will be linear
2. Adversary with ongoing access to key updates will still get data
Defining Security [EPRS17]

Four properties to achieve:

- Correctness
- Compactness
- Confidentiality
- Integrity
Defining Security [EPRS17]

Four properties to achieve:

- Correctness
- Compactness
- Confidentiality
- Integrity
Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext.
Confidentiality

Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext.
Confidentiality

Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext
Confidentiality

Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext
Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext.
Our definitions additionally require hiding ciphertext age from attacker
Confidentiality

Key 1 -> Key 2 -> Key 3 -> Key 4

Update Token 1-2, Update Token 2-3, Update Token 3-4

Our definitions additionally require hiding ciphertext age from attacker
Building Updatable Encryption [BLMR13, EPRS17]
Building Updatable Encryption [BLMR13, EPRS17]
Building Updatable Encryption [BLMR13, EPRS17]
Building Updatable Encryption [BLMR13, EPRS17]
Building Updatable Encryption [BLMR13, EPRS17]
Building Updatable Encryption [BLMR13, EPRS17]
Building Updatable Encryption [BLMR13, EPRS17]

“Ciphertext-dependent” model
Updatable Encryption from Nested AES

Very fast, simple scheme

Only requires authenticated encryption (AES-GCM) and a PRG
Updatable Encryption from Nested AES

Very fast, simple scheme

Only requires authenticated encryption (AES-GCM) and a PRG

Caveats:
- Only works for a bounded number of re-encryptions, decided at encryption time
- Decryption time will be linear in the number of re-encryptions
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES

- **Header key**: Used to encrypt the ciphertext header.
- **Body key**: Used to encrypt the ciphertext body.

Diagram:
- Ciphertext header
- Ciphertext Body
- Lock symbol in ciphertext header (indicates a lock held in ciphertext header)
- Lock symbol in ciphertext body (indicates encryption)
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES

- Ciphertext header
- Ciphertext header
- Ciphertext header
- Ciphertext Body

Header key
Updatable Encryption from Nested AES

Re-Encryption: wrap previous layer

Decryption: unwrap all layers
Updatable Encryption from Nested AES

Re-Encryption: wrap previous layer

Decryption: unwrap all layers

Issue: leaks ciphertext age
Updatable Encryption from Nested AES

Re-Encryption: wrap previous layer

Decryption: unwrap all layers

Issue: leaks ciphertext age

Note: this satisfies prior definitions
Updatable Encryption from Nested AES

How to hide ciphertext age?
Updatable Encryption from Nested AES

How to hide ciphertext age?

Idea 1: pad up to fixed max size with random data
Updatable Encryption from Nested AES

How to hide ciphertext age?

Idea 1: pad up to fixed max size with random data

But this ruins integrity
Updatable Encryption from Nested AES

How to hide ciphertext age?

Idea 1: pad up to fixed max size with random data

But this ruins integrity

Idea 2: generate random data from PRG, include seed in header
Updatable Encryption from Nested AES

How to hide ciphertext age?

Idea 1: pad up to fixed max size with random data

But this ruins integrity

Idea 2: generate random data from PRG, include seed in header

See paper for full scheme
Updatable Encryption from KH-PRFs [BLMR13, EPRS17]

Supports as many re-encryptions as you want

Decryption time does not depend on number of re-encryptions

Still fast, but slower than nested scheme

New caveat: somewhat weaker integrity and age-hiding guarantee
Tool: Key-Homomorphic PRFs (KHPRFs) [NPR99]

Standard PRF (e.g. AES): $F(k, x)$ looks random if not given $k$
Tool: Key-Homomorphic PRFs (KHPRFs) [NPR99]

Standard PRF (e.g. AES): $F(k, x)$ looks random if not given $k$

Key-Homomorphic PRF: Same security property, new functionality
Tool: Key-Homomorphic PRFs (KHPRFs) [NPR99]

Standard PRF (e.g. AES): $F(k, x)$ looks random if not given $k$

Key-Homomorphic PRF: Same security property, new functionality

$$F(k_1, x) \oplus F(k_2, x) = F(k_1 + k_2, x)$$
Tool: Key-Homomorphic PRFs (KHPRFs) [NPR99]

Standard PRF (e.g. AES): $F(k, x)$ looks random if not given $k$

Key-Homomorphic PRF: Same security property, new functionality

\[ F(k_1, x) \boxplus F(k_2, x) = F(k_1 + k_2, x) \]

Example: $F(k, x) = H(x)^k$
Tool: Key-Homomorphic PRFs (KHPRFs) \cite{NPR99}

Standard PRF (e.g. AES): $F(k, x)$ looks random if not given $k$

Key-Homomorphic PRF: Same security property, new functionality

$$F(k_1, x) \oplus F(k_2, x) = F(k_1 + k_2, x)$$

Example: $F(k, x) = H(x)^k$

$$F(k_1, x) \ast F(k_2, x) = H(x)^{k_1} \ast H(x)^{k_2} = H(x)^{k_1+k_2} = F(k_1 + k_2, x)$$
Updatable Encryption from KH-PRFs [EPRS17]

Ciphertext header:
Authenticated Encryption of $H(msg)$ and KH-PRF key $k_1$
Updatable Encryption from KH-PRFs [EPRS17]

Ciphertext header:
Authenticated Encryption of $H(msg)$ and KH-PRF key $k_1$

Ciphertext body:
Encryption of msg in counter mode using KH-PRF
Updatable Encryption from KH-PRFs [EPRS17]

Ciphertext header:
   Authenticated Encryption of $H(msg)$ and KH-PRF key $k_1$

Ciphertext body:
   Encryption of msg in counter mode using KH-PRF

\[
c_0 = m_0 + F(k_1, 0) \\
c_1 = m_1 + F(k_1, 1) \\
\ldots \\
c_n = m_n + F(k_1, n)
\]
Updatable Encryption from KH-PRFs [EPRS17]

Ciphertext header:
Authenticated Encryption of $H(msg)$ and KH-PRF key $k_1$

Ciphertext body:
Encryption of msg in counter mode using KH-PRF

$c_0 = m_0 + F(k_1, 0)$
$c_1 = m_1 + F(k_1, 1)$
$\ldots$
$c_n = m_n + F(k_1, n)$

Update process:
1. Download/decrypt header
2. Pick key $k_2$
3. Upload new header and $k_{up} = k_2 - k_1$

Server updates body encryptions with $k_{up}$
Updatable Encryption from KH-PRFs [EPRS17]

Ciphertext header:
Authenticated Encryption of $H(msg)$ and KH-PRF key $k_1$

Ciphertext body:
Encryption of msg in counter mode using KH-PRF

$c_0' = c_0 + F(k_{up}, 0)$
$c_1' = c_1 + F(k_{up}, 1)$
\[ \vdots \]
$c_n' = c_n + F(k_{up}, n)$

Update process:
1. Download/decrypt header
2. Pick key $k_2$
3. Upload new header and $k_{up} = k_2 - k_1$

Server updates body encryptions with $k_{up}$
Updatable Encryption from KH-PRFs [EPRS17]

Ciphertext header:
Authenticated Encryption of $H(msg)$ and KH-PRF key $k_1$

Ciphertext body:
Encryption of msg in counter mode using KH-PRF

$c_0' = c_0 + F(k_{up}, 0) = m_0 + F(k_2, 0)$
$c_1' = c_1 + F(k_{up}, 1) = m_1 + F(k_2, 1)$
$\ldots$
$c_n' = c_n + F(k_{up}, n) = m_n + F(k_2, n)$

Update process:
1. Download/decrypt header
2. Pick key $k_2$
3. Upload new header and $k_{up} = k_2 - k_1$

Server updates body encryptions with $k_{up}$
Almost KH-PRFs [BLMR13]

EPRS17 uses a KH-PRF based on the DDH assumption*

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \]

*In Random Oracle model
Almost KH-PRFs [BLMR13]

EPRS17 uses a KH-PRF based on the DDH assumption* 

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \]

We use a new almost KH-PRF based on the Ring-LWE assumption* 

*In Random Oracle model
**Almost KH-PRFs** [BLMR13]

EPRS17 uses a KH-PRF based on the DDH assumption:

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \]

We use a new *almost KH-PRF* based on the Ring-LWE assumption:

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \] (where \( e \) is small in \( \mathbb{Z}_q^n \))

*In Random Oracle model*
Almost KH-PRFs [BLMR13]

EPRS17 uses a KH-PRF based on the DDH assumption*

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \]

We use a new almost KH-PRF based on the Ring-LWE assumption*

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \]  
(\text{where } e \text{ is small in } \mathbb{Z}_q^n)

See paper for construction

*In Random Oracle model
Almost KH-PRFs [BLMR13]

EPRS17 uses a KH-PRF based on the DDH assumption*

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \]

We use a new *almost KH-PRF* based on the Ring-LWE assumption*

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \] (where \( e \) is small in \( \mathbb{Z}_q^n \))

See paper for construction

Result: ~500x faster performance

*In Random Oracle model*
Almost KH-PRFs [BLMR13]

EPRS17 uses a KH-PRF based on the DDH assumption:

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \]

We use a new almost KH-PRF based on the Ring-LWE assumption:

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \]  

(where e is small in \( Z_q^n \))

See paper for construction

Result: \(~500x\) faster performance  …but how to handle the noise?

*In Random Oracle model
Updatable Encryption from *Almost* KH-PRFs

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \] (where e is small)

Issue: noisy KH-PRF corrupts message
Updatable Encryption from *Almost* KH-PRFs

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \]  
(\text{where } e \text{ is small})

Issue: noisy KH-PRF corrupts message

General solution: error correcting codes
Updatable Encryption from *Almost* KH-PRFs

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \quad \text{(where } e \text{ is small)} \]

**Issue:** noisy KH-PRF corrupts message

**General solution:** error correcting codes

**Observation:** noise is always on low-order bits
Updatable Encryption from *Almost* KH-PRFs

\[ F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) + e \quad \text{(where e is small)} \]

Issue: noisy KH-PRF corrupts message

General solution: error correcting codes

Observation: noise is always on low-order bits

Simple solution: pad low-order bits of each block with zeros
Evaluation
## Encryption and Re-encryption

Throughput for encrypting/re-encrypting 32KB messages (MB/sec)

<table>
<thead>
<tr>
<th></th>
<th>ReCrypt [EPRS17]</th>
<th>Almost KH-PRF</th>
<th>Nested (128 layers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>0.12</td>
<td>61.90</td>
<td>1836.9</td>
</tr>
<tr>
<td>Re-encrypt</td>
<td>0.15</td>
<td>83.06</td>
<td>2606.8</td>
</tr>
</tbody>
</table>
Encryption and Re-encryption

Throughput for encrypting/re-encrypting 32KB messages (MB/sec)

<table>
<thead>
<tr>
<th></th>
<th>ReCrypt [EPRS17]</th>
<th>Almost KH-PRF</th>
<th>Nested (128 layers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>0.12</td>
<td>61.90</td>
<td>1836.9</td>
</tr>
<tr>
<td>Re-encrypt</td>
<td>0.15</td>
<td>83.06</td>
<td>2606.8</td>
</tr>
</tbody>
</table>

Almost KH-PRF is ~500x faster than ReCrypt

Nested AES is ~30x faster than almost KH-PRF
Decryption

Decryption Time
32KB Messages

Time [μs]

Number of Re-encryptions

KH-PRF
Nested
Decryption

Decryption Time
32KB Messages

- KH-PRF
- Nested

Time [μs]

Number of Re-encryptions
Decryption

Nested construction faster for up to 50 re-encryptions

ReCrypt (not shown) 500x slower than KH-PRF construction
Decryption

Nested construction faster for up to 50 re-encryptions

ReCrypth (not shown) 500x slower than KH-PRF construction

Recommendations
Use nested AES construction for infrequent, routine re-keying
Use KH-PRF for frequent re-keying
Ciphertext Expansion

Nested AES and ReCrypt have smallest ciphertext expansion

<table>
<thead>
<tr>
<th>Ciphertext Expansion</th>
<th>32KB Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>KH-PRF UAE</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>q</td>
</tr>
<tr>
<td>$</td>
<td>q</td>
</tr>
<tr>
<td>$</td>
<td>q</td>
</tr>
<tr>
<td>$</td>
<td>q</td>
</tr>
<tr>
<td>Nested UAE</td>
<td></td>
</tr>
<tr>
<td>$t = 20$</td>
<td>3%</td>
</tr>
<tr>
<td>$t = 128$</td>
<td>19%</td>
</tr>
<tr>
<td>ReCrypt [EPRS17]</td>
<td>3%</td>
</tr>
</tbody>
</table>
Ciphertext Expansion

Nested AES and ReCrypt have smallest ciphertext expansion

Recommendations

Use nested AES construction for infrequent, routine re-keying

If space is costly and computation is cheap, use ReCrypt for frequent rekeying

<table>
<thead>
<tr>
<th></th>
<th>KH-PRF UAE</th>
<th>Nested UAE</th>
<th>ReCrypt [EPRS17]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>q</td>
<td>= 28$</td>
<td>133%</td>
</tr>
<tr>
<td>$</td>
<td>q</td>
<td>= 60$</td>
<td>36%</td>
</tr>
<tr>
<td>$</td>
<td>q</td>
<td>= 120$</td>
<td>20%</td>
</tr>
<tr>
<td>$</td>
<td>q</td>
<td>= 128$</td>
<td>19%</td>
</tr>
<tr>
<td>$t = 20$</td>
<td></td>
<td>3%</td>
<td></td>
</tr>
<tr>
<td>$t = 128$</td>
<td></td>
<td>19%</td>
<td></td>
</tr>
</tbody>
</table>
Can we do Better?

**Speed**: Not by much

- Nested scheme: already close to AES throughput
- Almost KH-PRF: KH-PRF implies key exchange \[\text{AMP19}\]
Can we do Better?

**Speed:** Not by much

- Nested scheme: already close to AES throughput
- Almost KH-PRF: KH-PRF implies key exchange \[\text{[AMP19]}\]

**Ciphertext expansion:** Good place for improvement

One potential approach: more elaborate error-correction to reduce bits wasted by padding
Improving Updatable Encryption

Improved security definitions for updatable encryption

Two new constructions -- from Nested AES and RLWE-based KH-PRF

Orders of magnitude performance improvement over prior work


Contact: saba@cs.stanford.edu
### Encryption and Re-Encryption

<table>
<thead>
<tr>
<th>Throughput (MB/sec)</th>
<th>KH-PRF UAE</th>
<th>ReCrypt [EPRS17]</th>
<th>Nested</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>q</td>
<td>= 28$</td>
<td>$</td>
</tr>
<tr>
<td><strong>Encrypt</strong></td>
<td>24.85</td>
<td>31.97</td>
<td>20.32</td>
</tr>
<tr>
<td><strong>ReEncrypt</strong></td>
<td>29.80</td>
<td>41.03</td>
<td>32.13</td>
</tr>
<tr>
<td><strong>4KB Messages</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Encrypt</strong></td>
<td>29.85</td>
<td>39.89</td>
<td><strong>61.90</strong></td>
</tr>
<tr>
<td><strong>ReEncrypt</strong></td>
<td>32.33</td>
<td>44.51</td>
<td><strong>83.06</strong></td>
</tr>
<tr>
<td><strong>32KB Messages</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Encrypt</strong></td>
<td>31.03</td>
<td>41.63</td>
<td><strong>65.11</strong></td>
</tr>
<tr>
<td><strong>ReEncrypt</strong></td>
<td>33.30</td>
<td>45.77</td>
<td><strong>79.63</strong></td>
</tr>
<tr>
<td><strong>100KB Messages</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• $H_0 : \{0, 1\}^\ell \to \mathcal{R}_q$,
• $H_1 : \mathcal{R}_q \times \{0, 1\}^\ell \to \{0, 1\}^r$.

We define our pseudorandom function $F : \mathcal{R}_q \times \{0, 1\}^\ell \to \mathcal{R}_q$ as follows:

$$F(s, x) :$$

1. Evaluate $a \leftarrow H_0(x)$, $\rho \leftarrow H_1(s, x)$.
2. Sample $e \leftarrow \text{Samp}_\chi(\rho)$.
3. Output $y \leftarrow a \cdot s + e$.

Where $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$