Improving Speed and Security in Updatable Encryption Systems

Dan Boneh  Saba Eskandarian  Sam Kim  Maurice Shih
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

Quiz! Which of the following is the best reason to rotate keys for data you store in the cloud:
Good Reasons to Rotate Keys

Quiz! Which of the following is the *best* reason to rotate keys for data you store in the cloud:

A) Old keys will be cryptanalyzed
Good Reasons to Rotate Keys

Quiz! Which of the following is the *best* reason to rotate keys for data you store in the cloud:

A) Old keys will be cryptanalyzed

B) Compromised keys need to be taken out of use
Good Reasons to Rotate Keys

Quiz! Which of the following is the best reason to rotate keys for data you store in the cloud:

A) Old keys will be cryptanalyzed

B) Compromised keys need to be taken out of use

C) Not changing keys regularly makes it harder when needed
Good Reasons to Rotate Keys

Quiz! Which of the following is the best reason to rotate keys for data you store in the cloud:

A) Old keys will be cryptanalyzed

B) Compromised keys need to be taken out of use

C) Not changing keys regularly makes it harder when needed
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

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

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
Confidentiality

Key 1
Update Token 1-2

Key 2
Update Token 2-3

Key 3
Update Token 3-4

Key 4
Confidentiality

Key 1
Update Token 1-2

Key 2
Update Token 2-3

Key 3
Update Token 3-4

Key 4
Confidentiality

Key 1

Update Token 1-2

Key 2

Update Token 2-3

Key 3

Update Token 3-4

Key 4
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

Ciphertext header

Ciphertext Body

Body key used for this lock held in ciphertext header

Header key
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES

Ciphertext header

Body key

Ciphertext Body

Header key
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES
Updatable Encryption from Nested AES
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]

Tool: Key-Homomorphic PRFs (KHPRFs) [NPR99]
Updatable Encryption from KH-PRFs [BLMR13, EPRS17]

Tool: Key-Homomorphic PRFs (KHPRFs) [NPR99]

Standard PRF (e.g. AES): $F(k, x)$ looks random if not given $k$
Updatable Encryption from KH-PRFs [BLMR13, EPRS17]

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
Updatable Encryption from KH-PRFs [BLMR13, EPRS17]

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

$$F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x)$$
Updatable Encryption from KH-PRFs \cite{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

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

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

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

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 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)

See paper for full construction and UE scheme

*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)

See paper for full construction and UE scheme

Result: ~500x faster performance

*In Random Oracle model
Evaluation
## Encryption and Re-encryption

| Encryption and ReEncrytp Throughput (MB/sec) | KH-PRF UAE | | | | | ReCrypt [EPRS17] | Nested |
|---------------------------------------------|-----------|-----------|-----------|-----------|----------------|----------------|
| | \(|q| = 28\) | \(|q| = 28\) (AVX) | \(|q| = 60\) | \(|q| = 120\) | \(|q| = 128\) | \(t = 128\) |
| 4KB Messages | Encrypt | 24.85 | **31.97** | 20.32 | 0.76 | 0.70 | 0.12 | 406.69 |
| | ReEncrypt | 29.80 | **41.03** | 32.13 | 0.82 | 0.74 | 0.14 | 706.37 |
| 32KB Messages | Encrypt | 29.85 | 39.89 | **61.90** | 5.94 | 5.50 | **0.12** | **1836.9** |
| | ReEncrypt | 32.33 | 44.51 | **83.06** | 6.43 | 5.85 | **0.15** | **2606.8** |
| 100KB Messages | Encrypt | 31.03 | 41.63 | **65.11** | 9.42 | 9.12 | **0.12** | **3029.5** |
| | ReEncrypt | 33.30 | 45.77 | **79.63** | 9.92 | 8.70 | **0.14** | **3766.2** |
Decryption

Decryption Time
32KB Messages

Time [μs]

Number of Re-encryptions

KH-PRF
Nested
Decryption

![Decryption Time Graph]

- **Decryption Time**
- **32KB Messages**

- 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

ReCryp (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>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>
</tbody>
</table>

| Nested UAE            |                |
| $t = 20$              | 3%             |
| $t = 128$             | 19%            |

| ReCrypt [EPRS17]      | 3%             |
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
• $H_0 : \{0, 1\}^\ell \to \mathcal{R}_q$,
• $H_1 : \mathcal{R}_q \times \{0, 1\}^\ell \to \{0, 1\}^r$.

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

\[
F(s, x) :
1. \text{Evaluate } a \leftarrow H_0(x), \; \rho \leftarrow H_1(s, x).
2. \text{Sample } e \leftarrow \text{Samp}_X(\rho).
3. \text{Output } y \leftarrow a \cdot s + e.
\]