Improving Speed and Security in Updatable Encryption Schemes

Dan Boneh
Stanford University

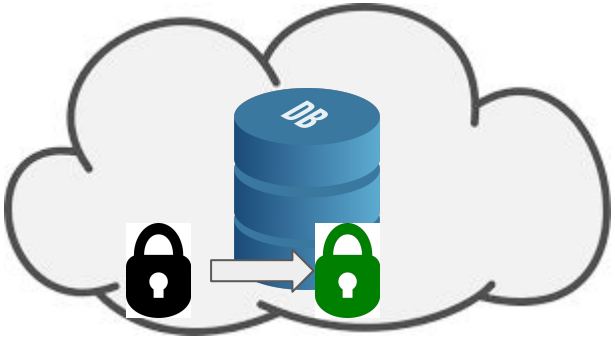
Saba Eskandarian
Stanford University

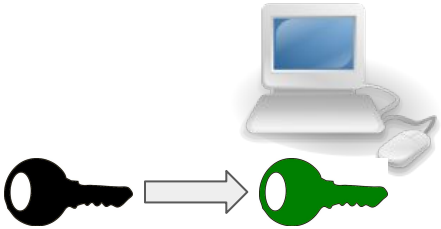
Sam Kim
Stanford University

Maurice Shih
Cisco Systems

Key Rotation

Key Rotation



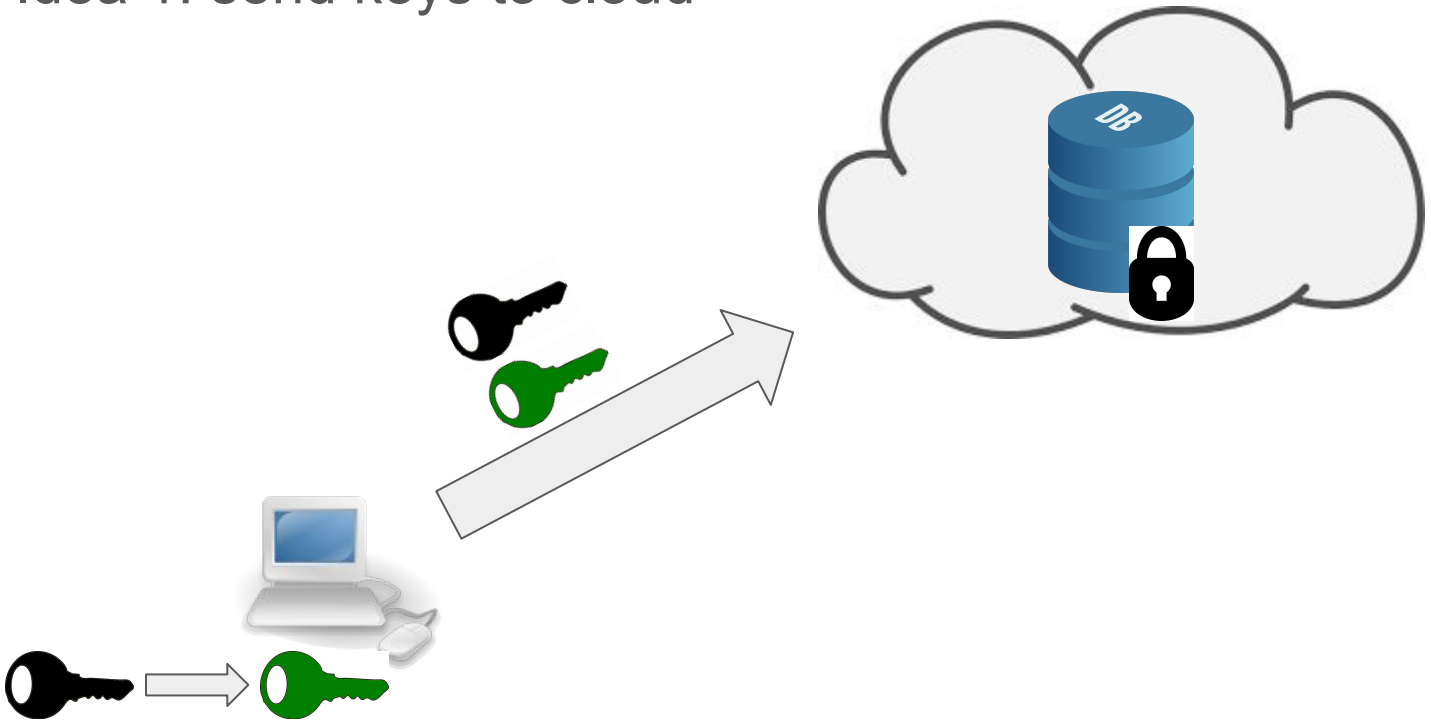


Many Good Reasons to Rotate Keys

1. Retire compromised keys
2. Proactively refresh keys
3. Access control enforcement
4. Recommended by NIST (Special Publication 800-57)
5. Recommended by Google (cloud.google.com/kms/docs/key-rotation)
6. Required by PCI DSS (PCI DSS 3.6.4)

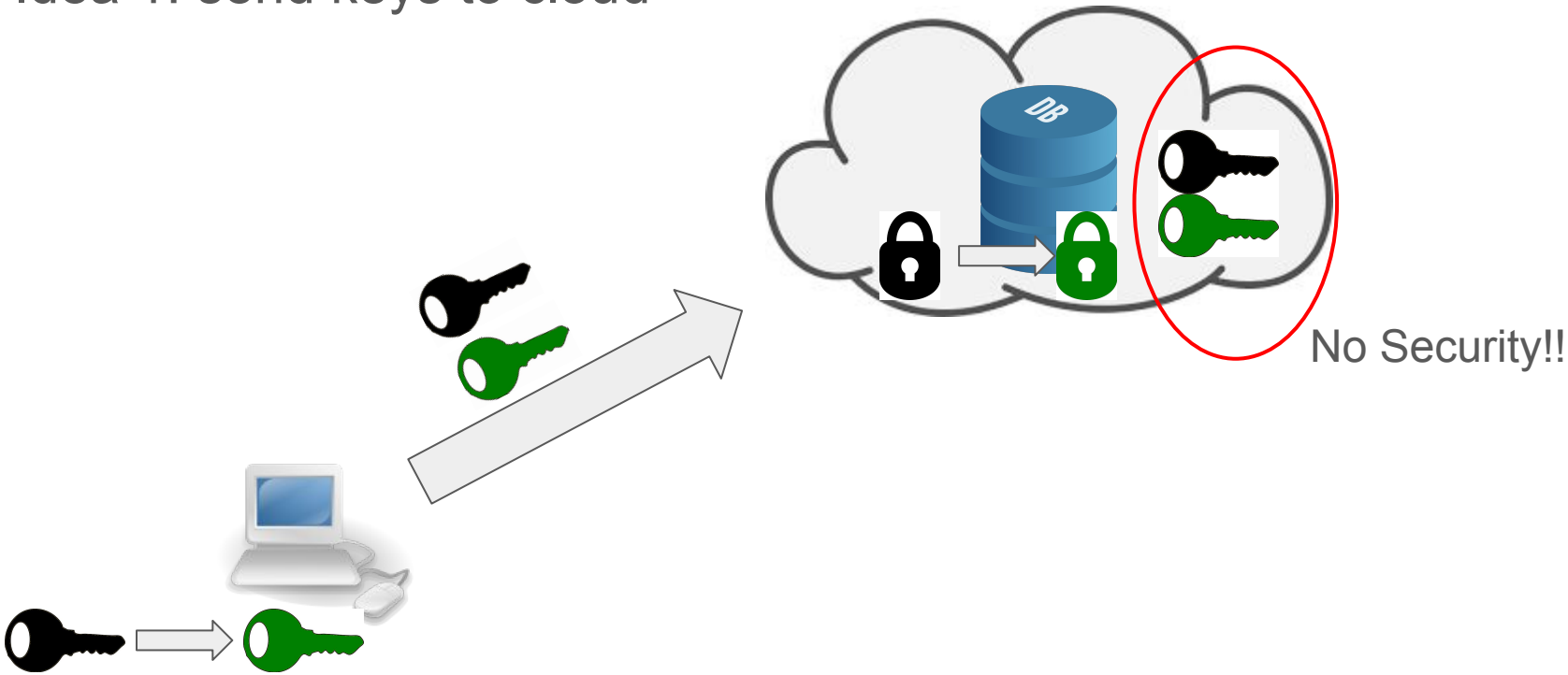
How to Rotate Keys in the Cloud?

Idea 1: send keys to cloud



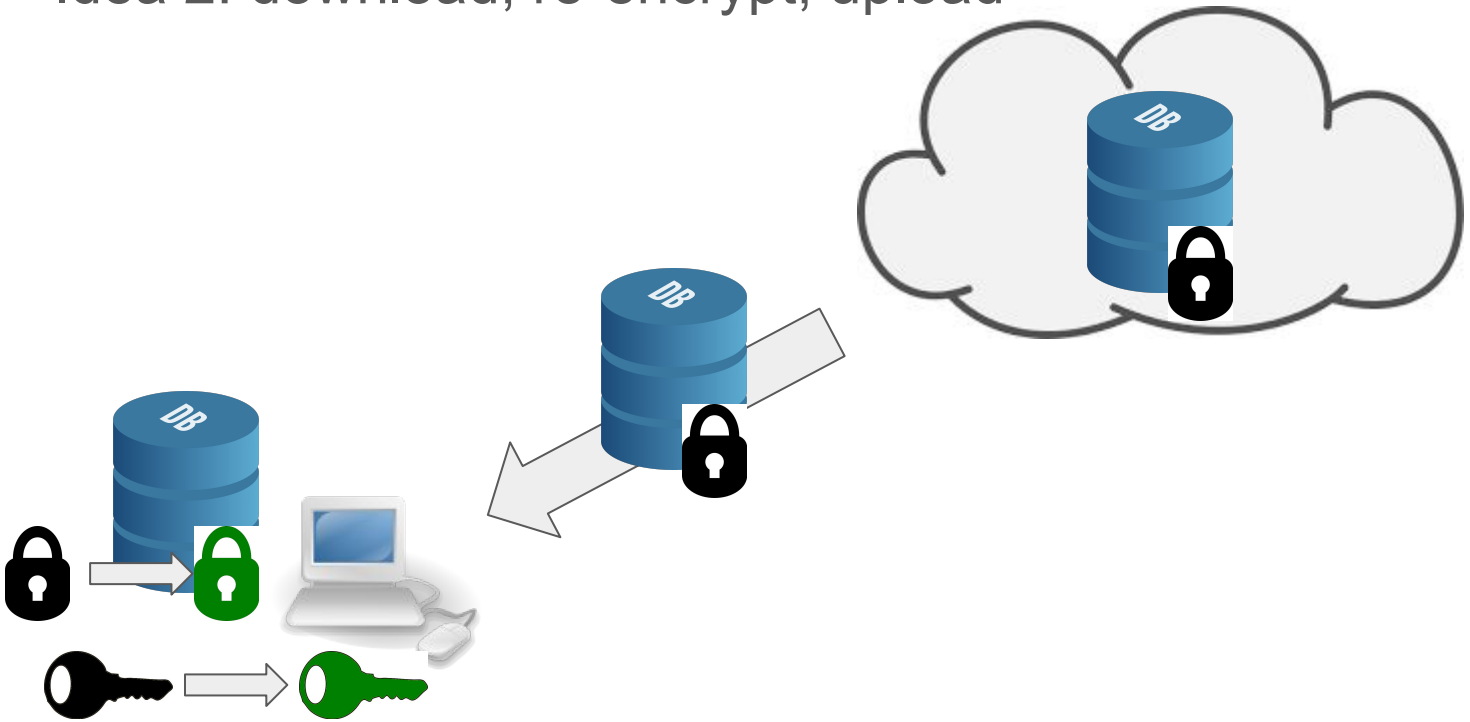
How to Rotate Keys in the Cloud?

Idea 1: send keys to cloud



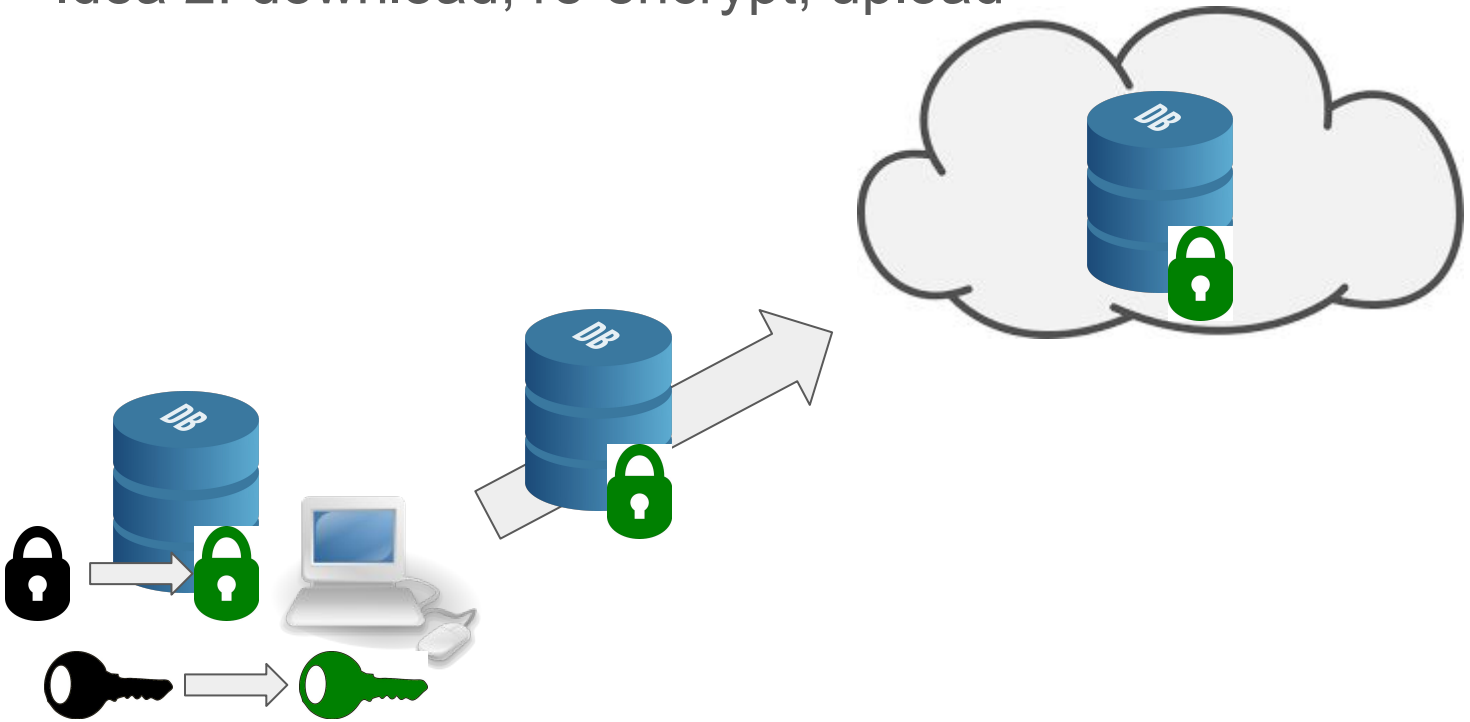
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload



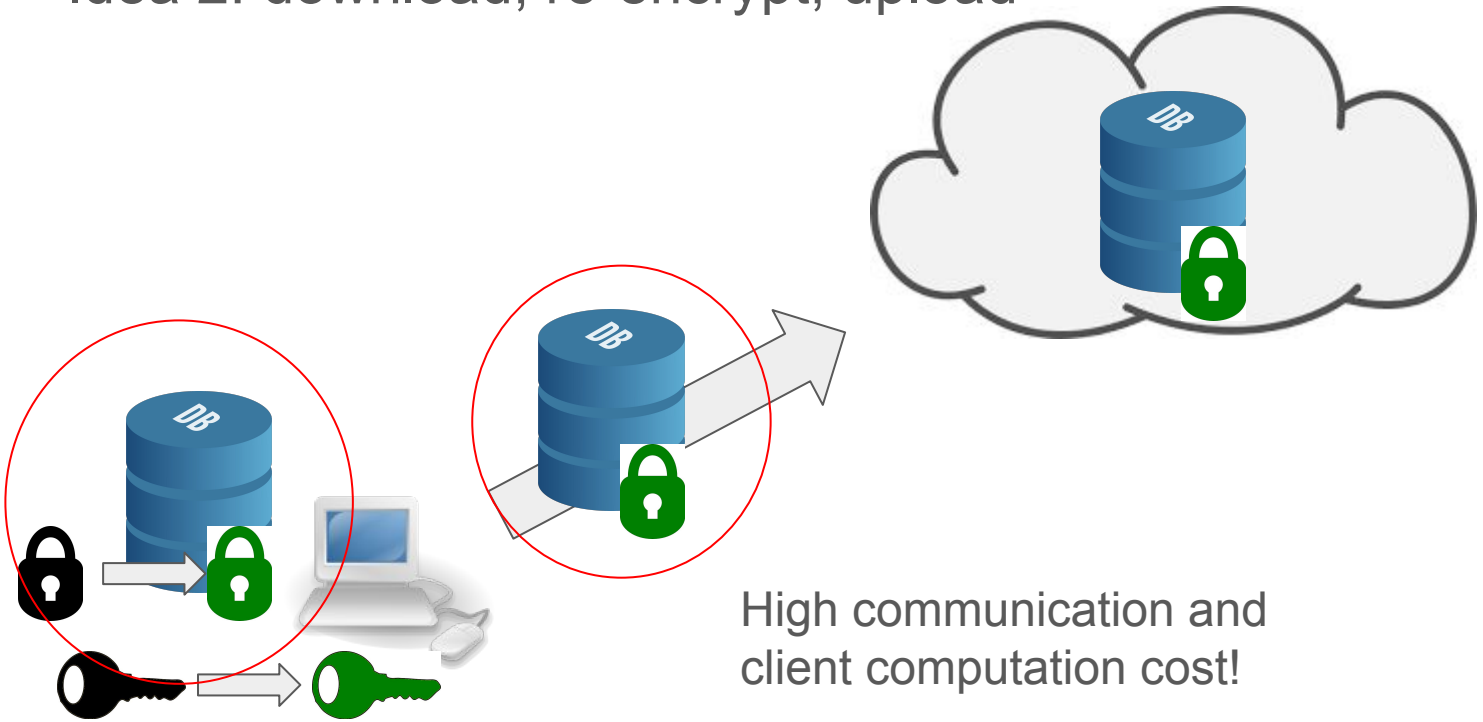
How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload



How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload

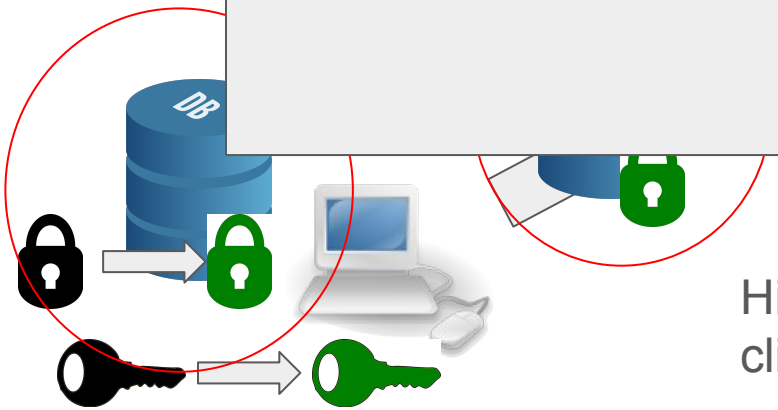


High communication and client computation cost!

How to Rotate Keys in the Cloud?

Idea 2: download, re-encrypt, upload

Can we do better?

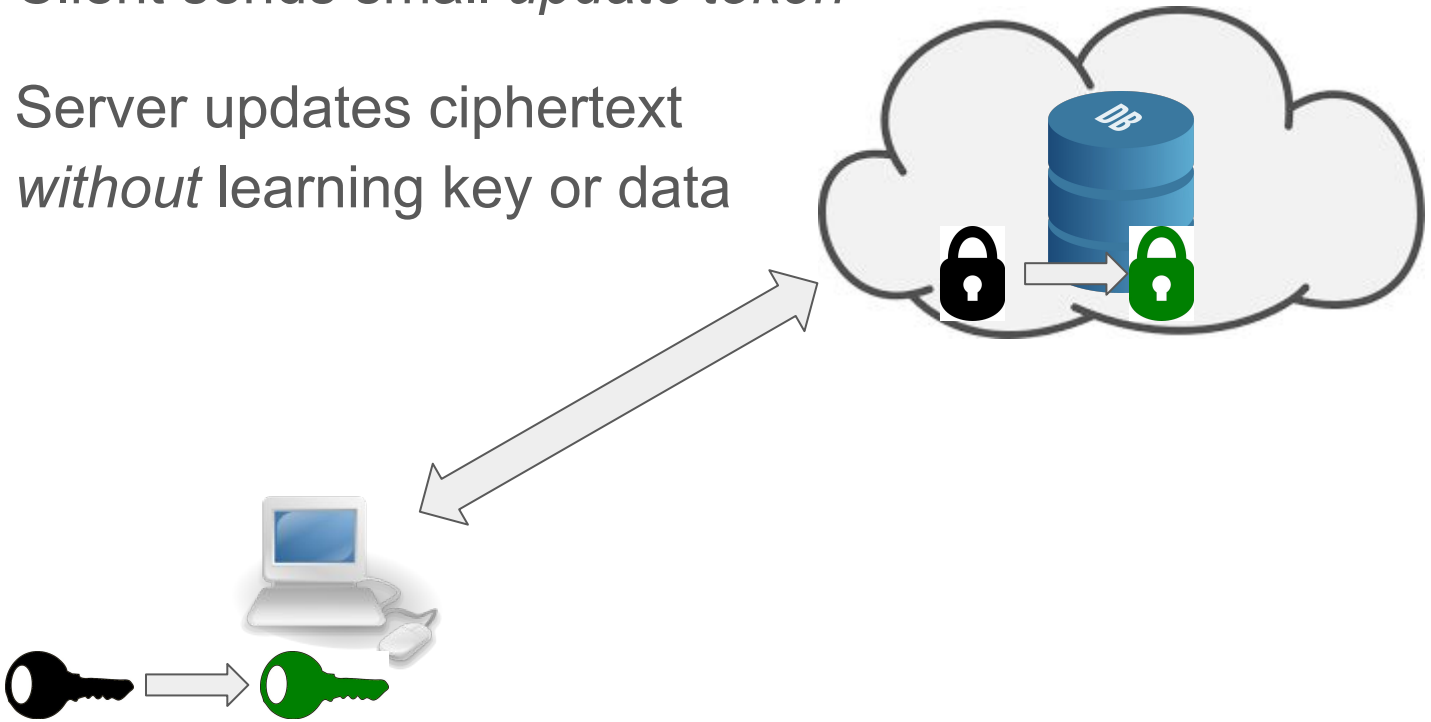


High communication and client computation cost!

Updatable Encryption [BLMR13, EPRS17, LT18, KLR19, BDGJ19]

Client sends small *update token*

Server updates ciphertext *without* learning key or data



Our Contributions

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

Defining Security [EPRS17]

Four properties to achieve:

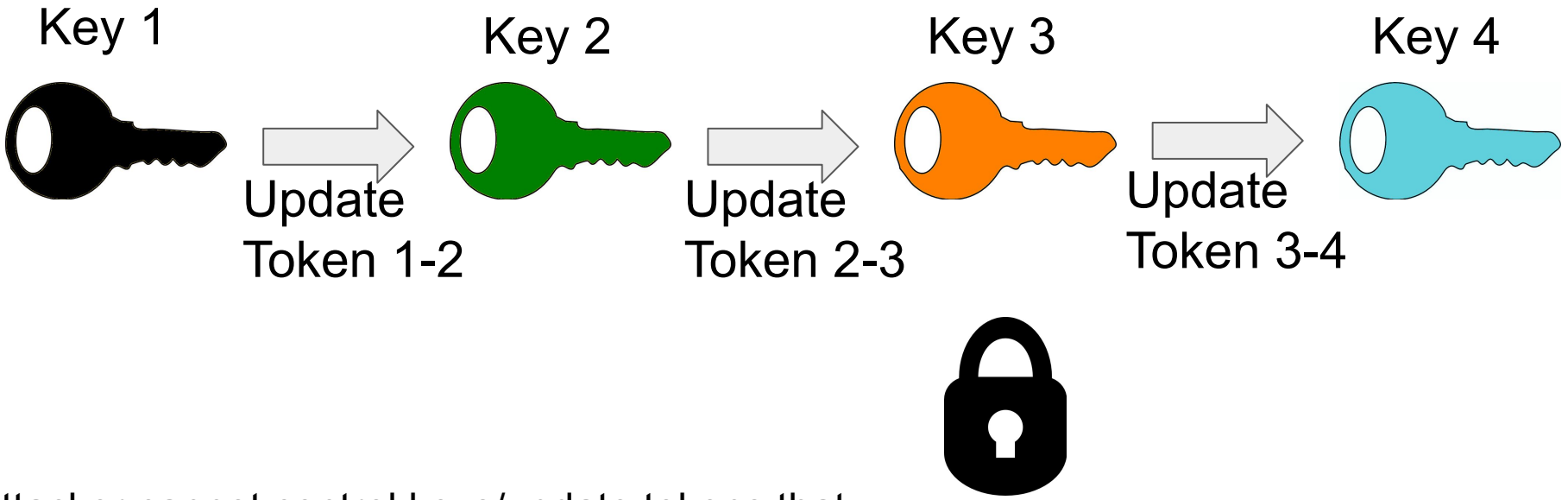
- Correctness
- Compactness
- Confidentiality
- Integrity

Defining Security [EPRS17]

Four properties to achieve:

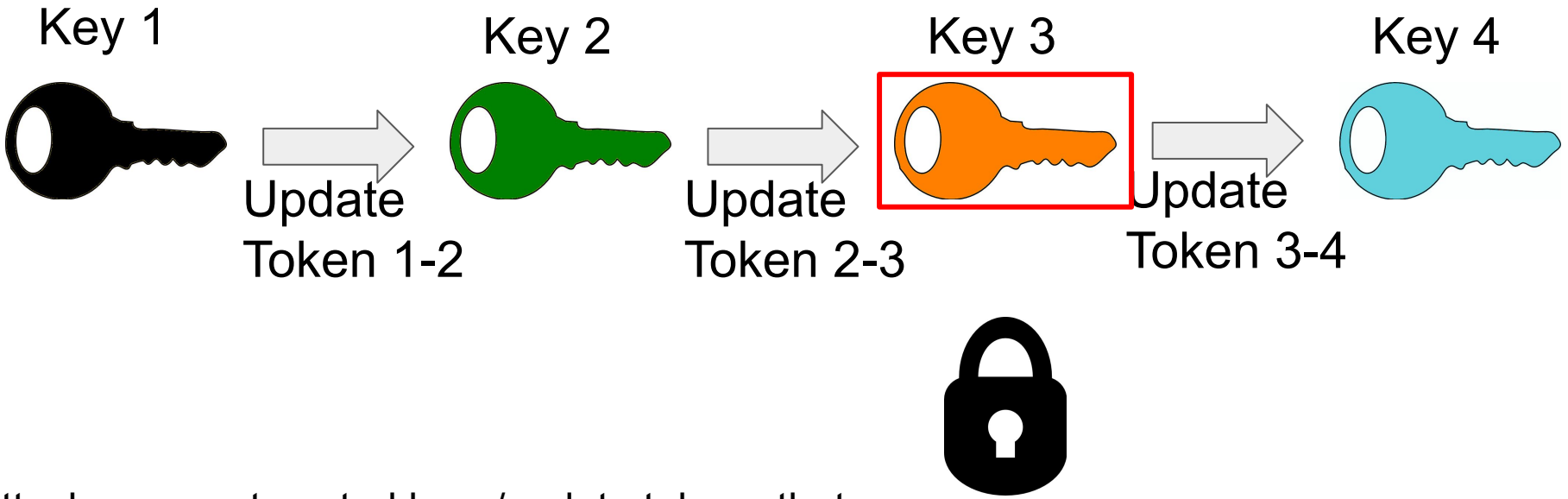
- Correctness
- Compactness
- **Confidentiality**
- Integrity

Confidentiality



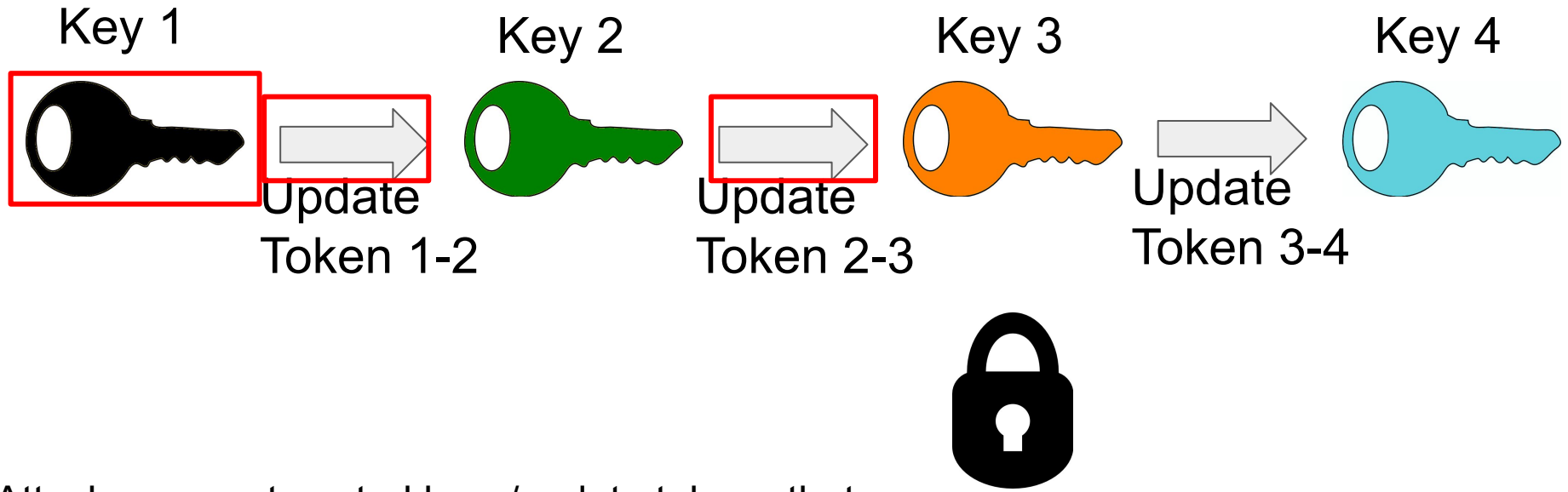
Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext

Confidentiality



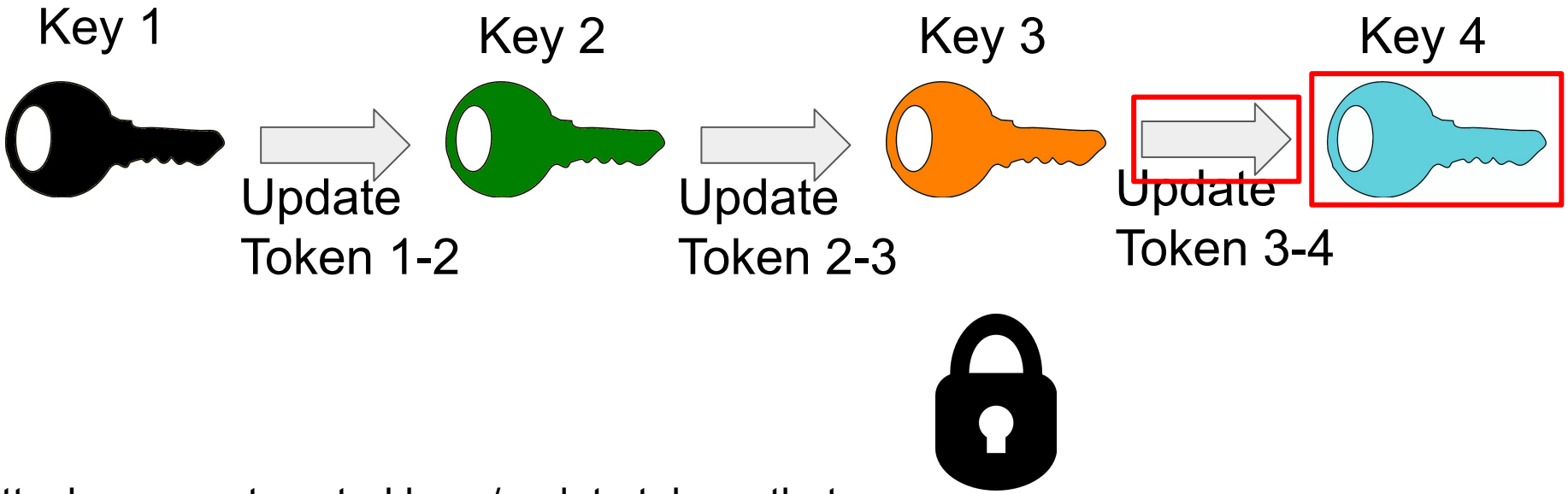
Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext

Confidentiality



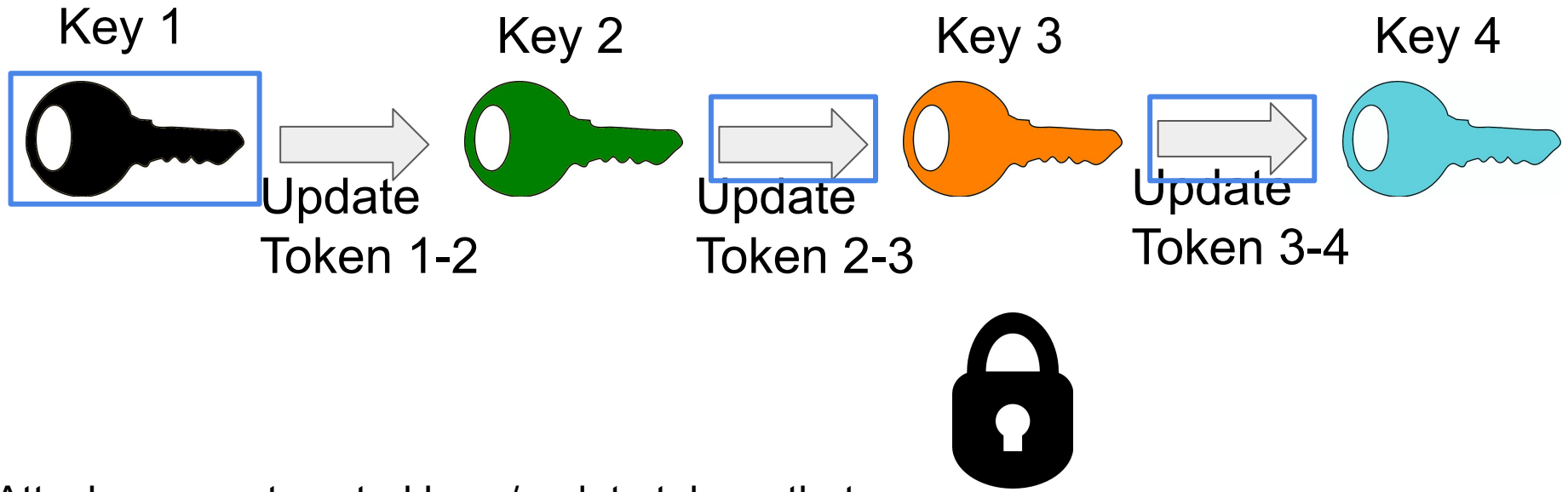
Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext

Confidentiality



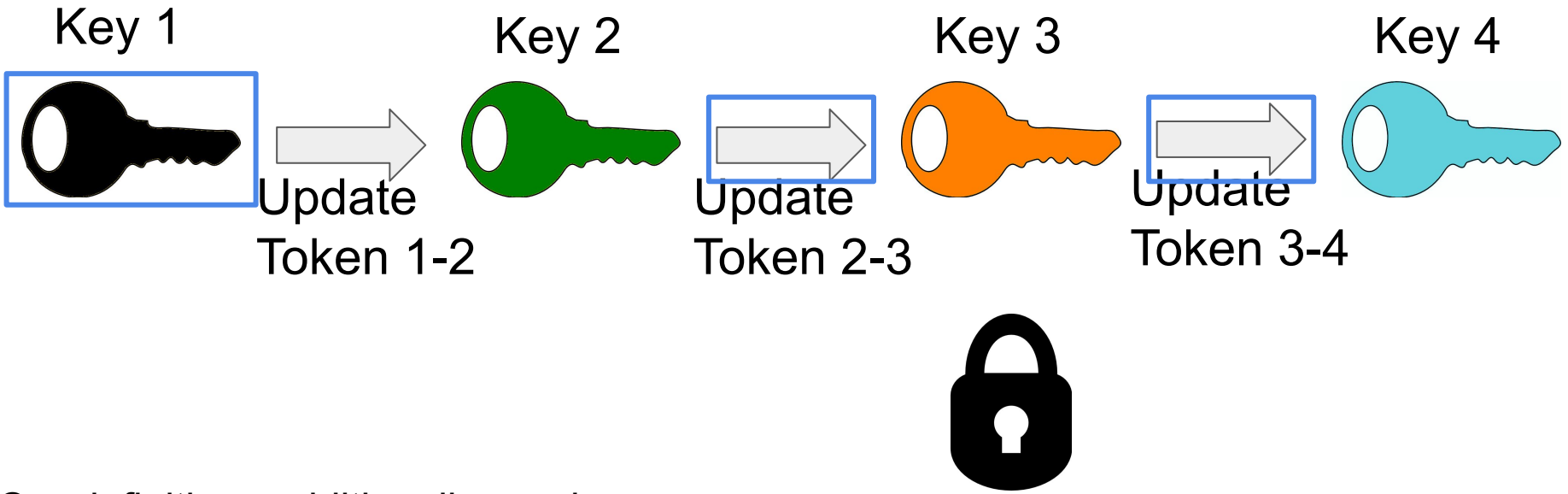
Attacker cannot control keys/update tokens that give a path to key used to encrypt a ciphertext

Confidentiality



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

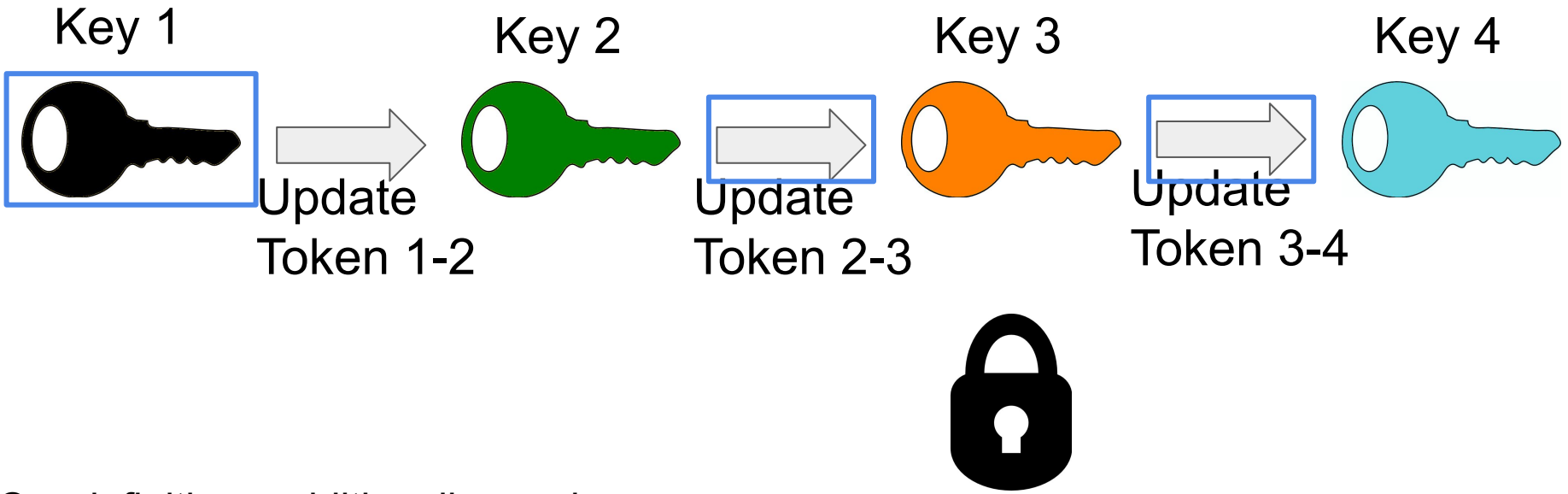
Confidentiality



Our definitions additionally require hiding ciphertext age from attacker

Confidentiality

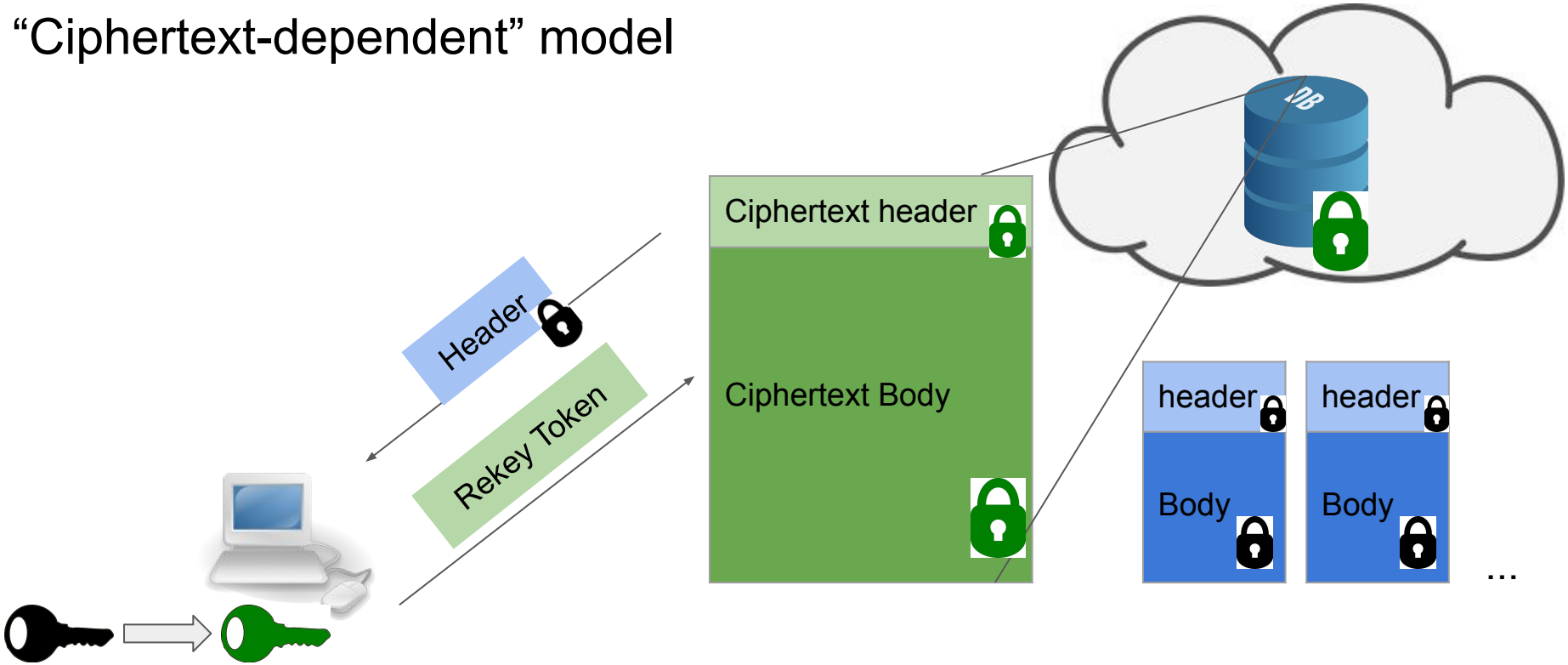
Note: cloud must always be trusted not to keep old ciphertexts



Our definitions additionally require hiding ciphertext age from attacker

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

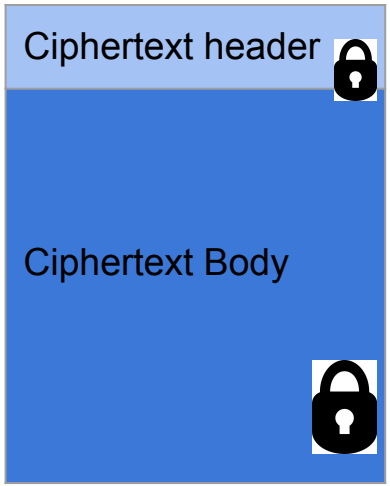
Very fast, simple scheme

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

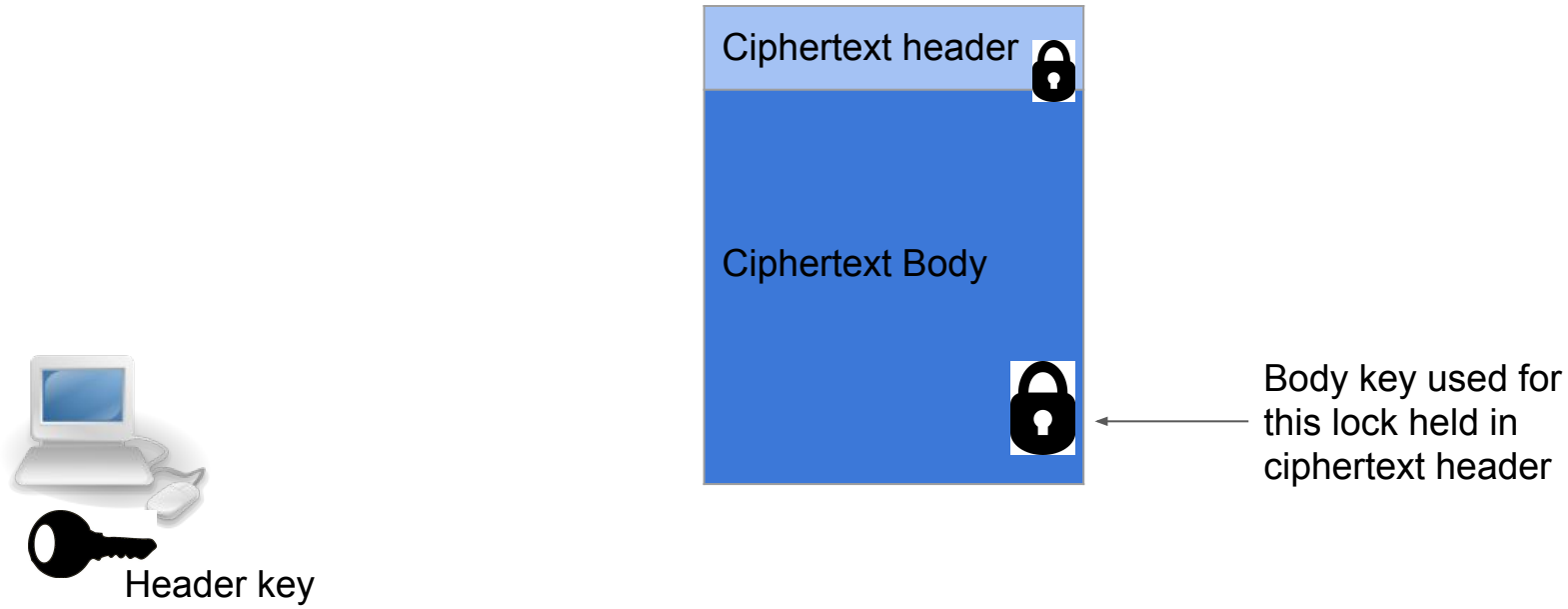
Caveats:

- Only works for a *bounded* number of re-encryptions, decided at encryption time
- Decryption time will be linear in the number of re-encryptions

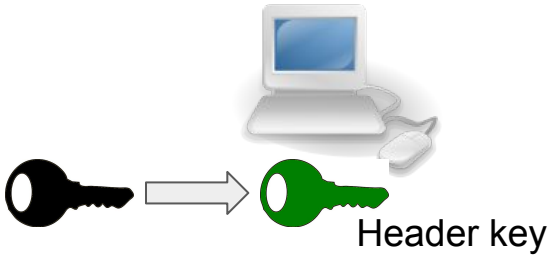
Updatable Encryption from Nested AES

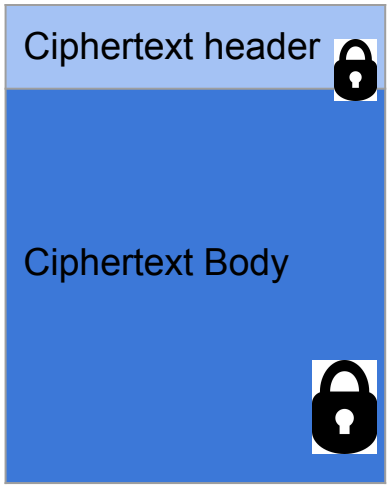


Updatable Encryption from Nested AES

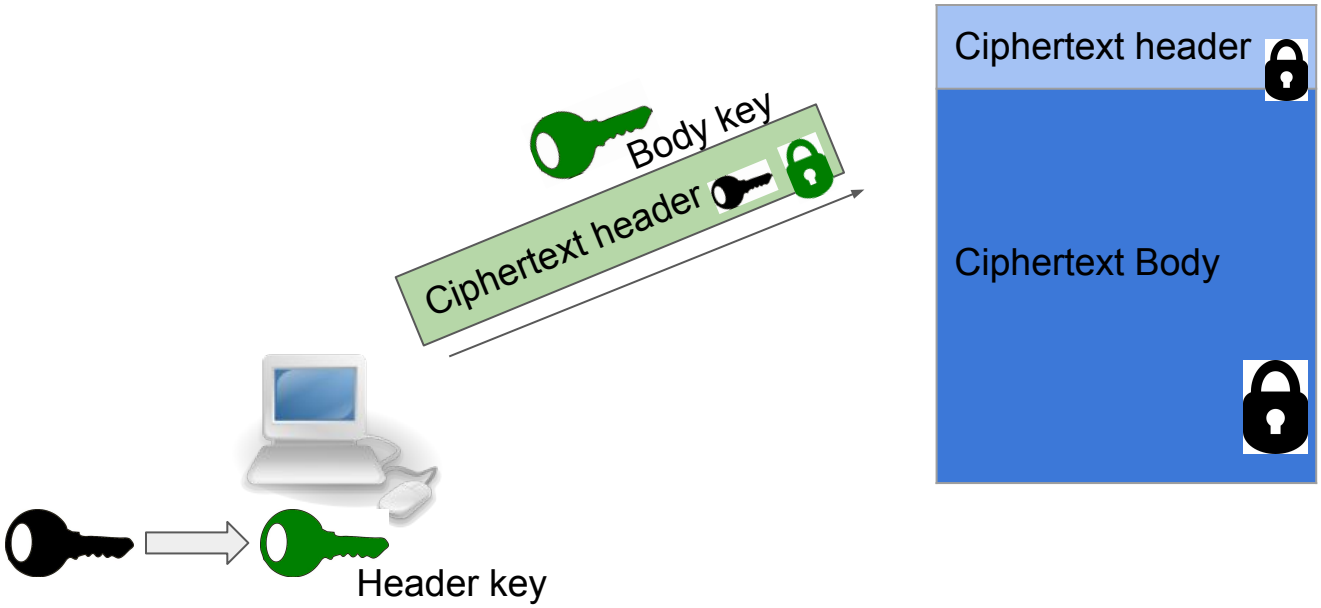


Updatable Encryption from Nested AES

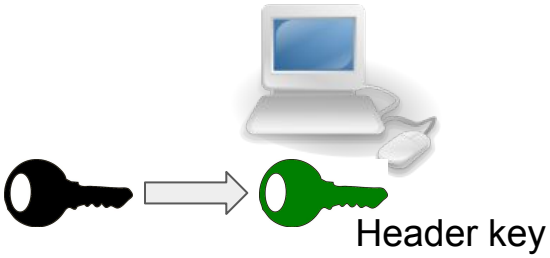


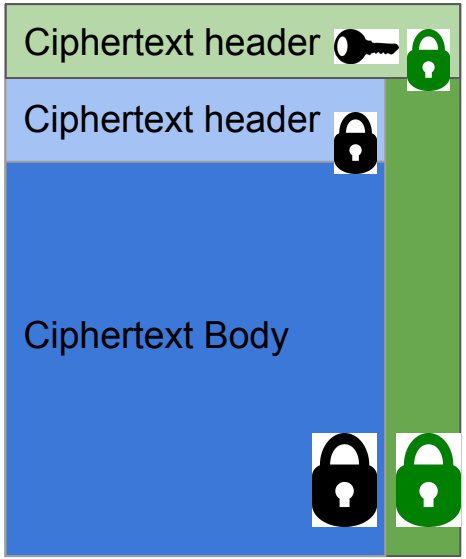


Updatable Encryption from Nested AES

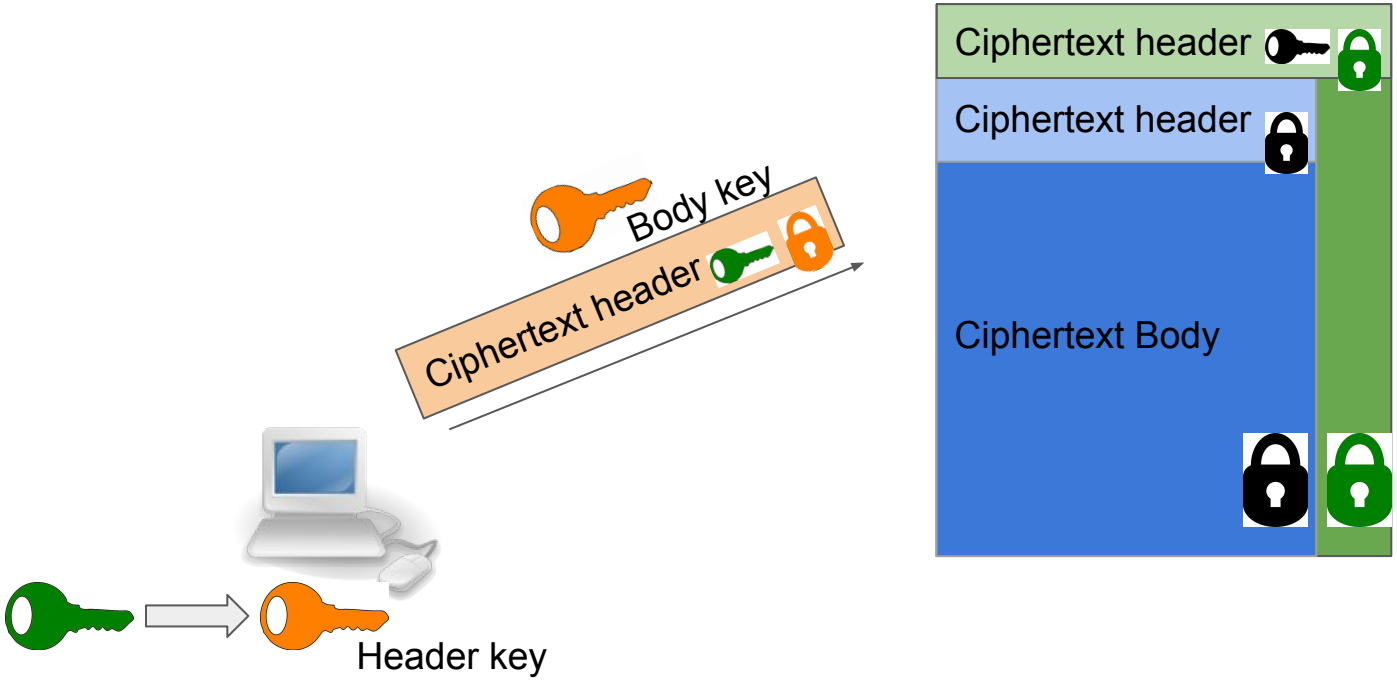


Updatable Encryption from Nested AES

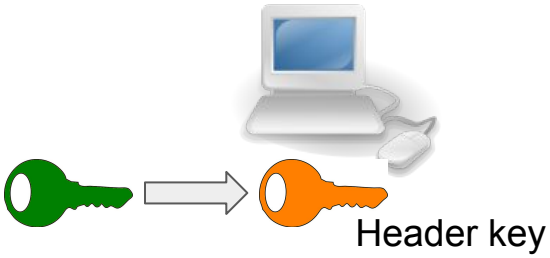


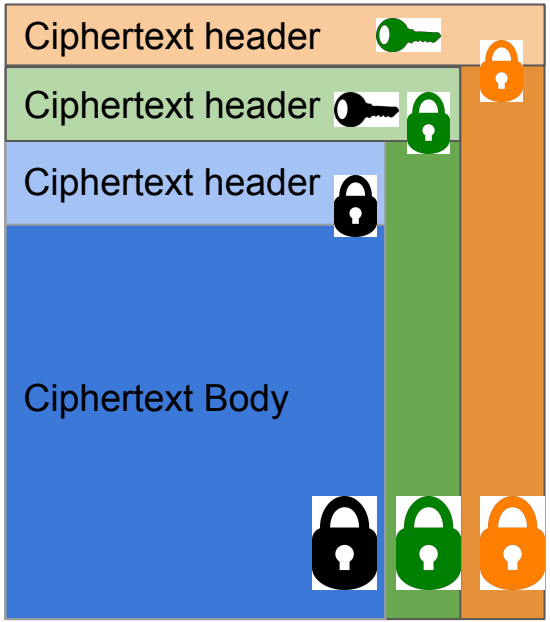


Updatable Encryption from Nested AES



Updatable Encryption from Nested AES

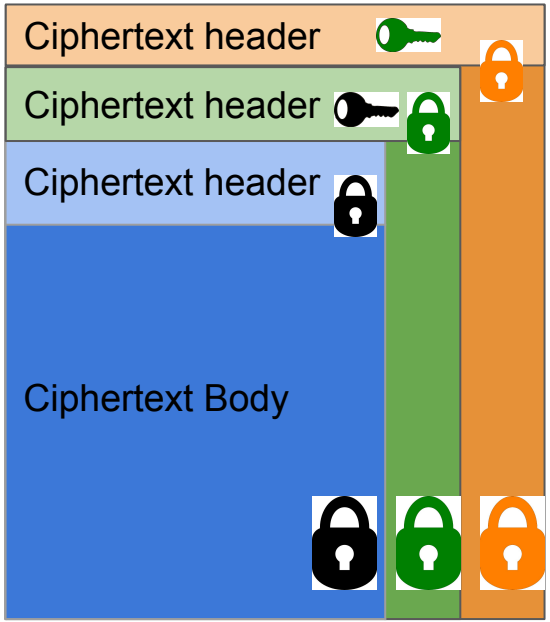




Updatable Encryption from Nested AES

Re-Encryption: wrap previous layer

Decryption: unwrap all layers

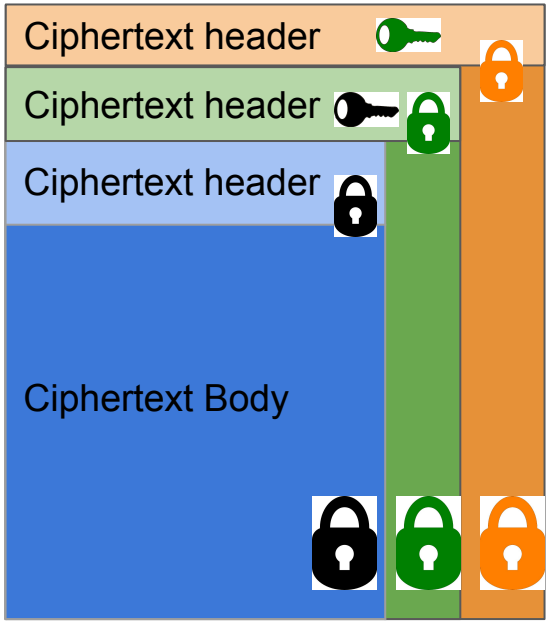


Updatable Encryption from Nested AES

Re-Encryption: wrap previous layer

Decryption: unwrap all layers

Issue: leaks ciphertext age



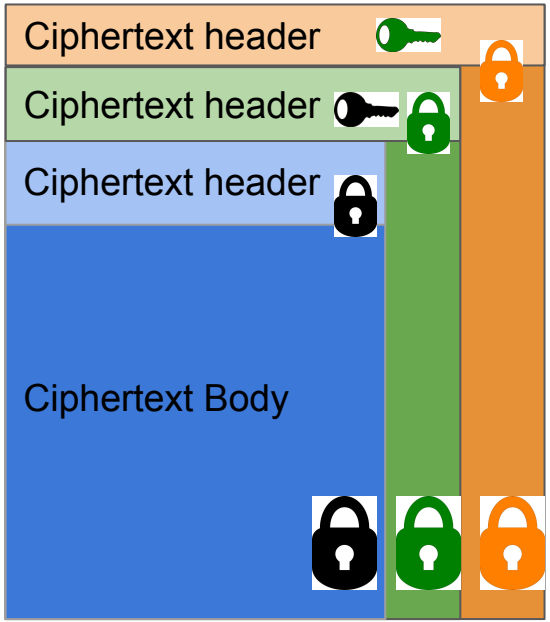
Updatable Encryption from Nested AES

Re-Encryption: wrap previous layer

Decryption: unwrap all layers

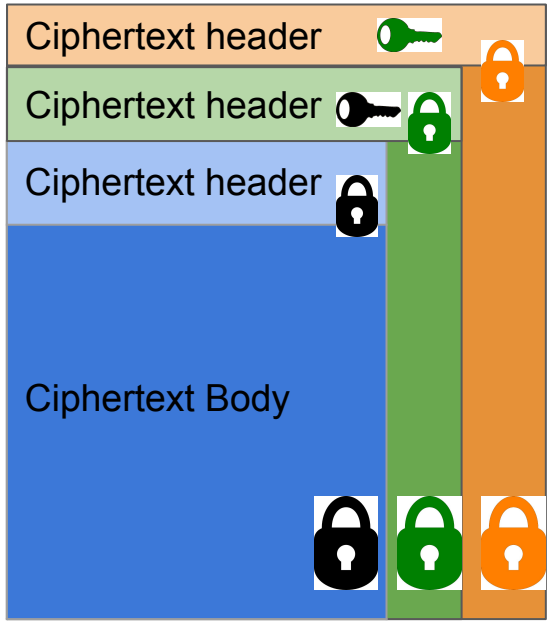
Issue: leaks ciphertext age

Note: this satisfies prior definitions



Updatable Encryption from Nested AES

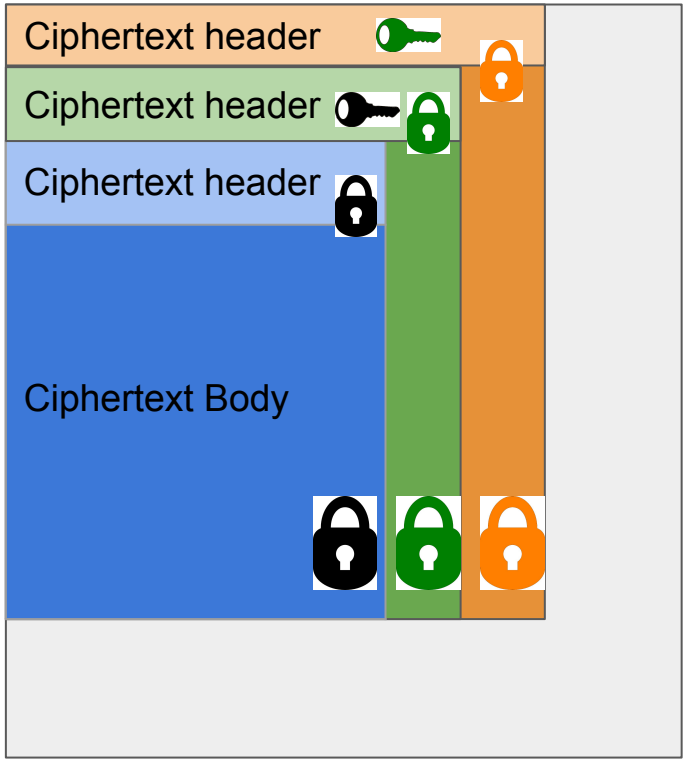
How to hide ciphertext age?



Updatable Encryption from Nested AES

How to hide ciphertext age?

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

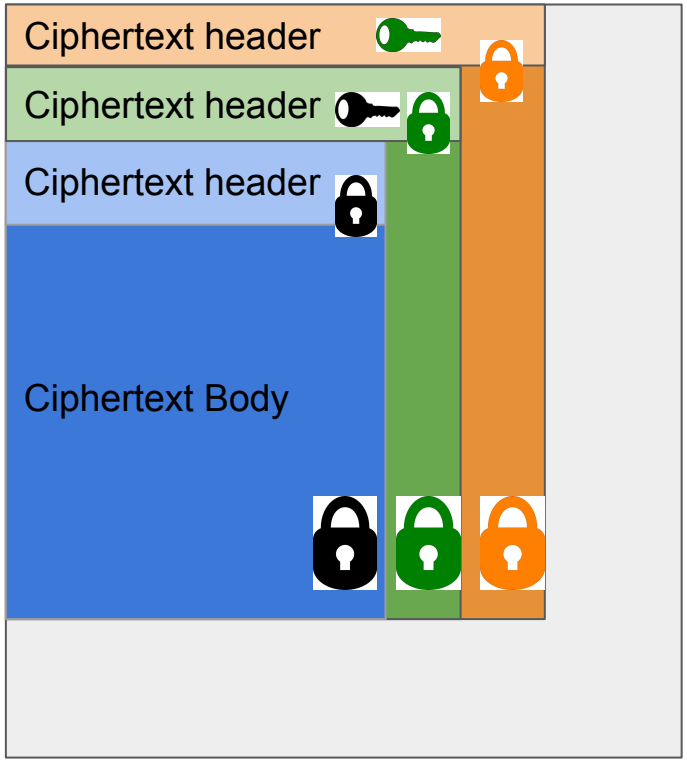


Updatable Encryption from Nested AES

How to hide ciphertext age?

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

But this ruins integrity



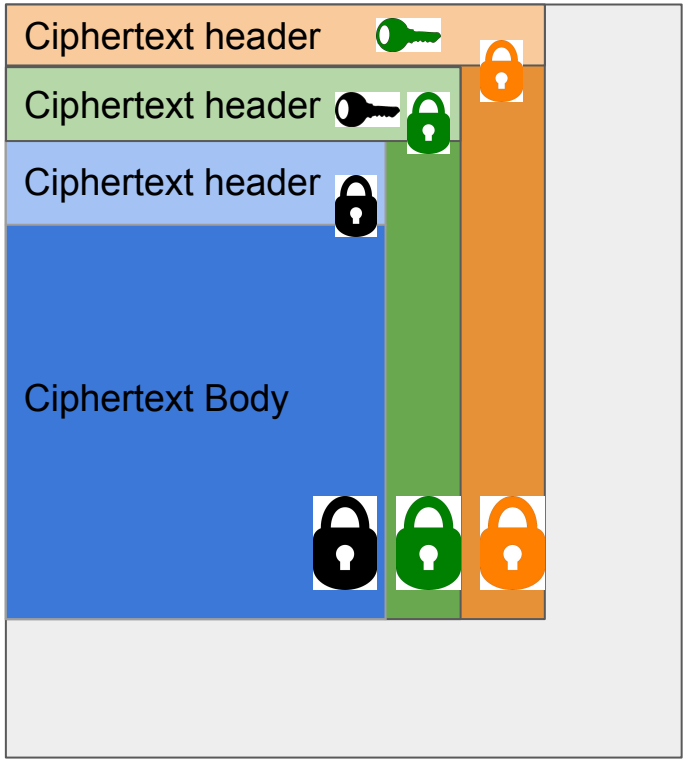
Updatable Encryption from Nested AES

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

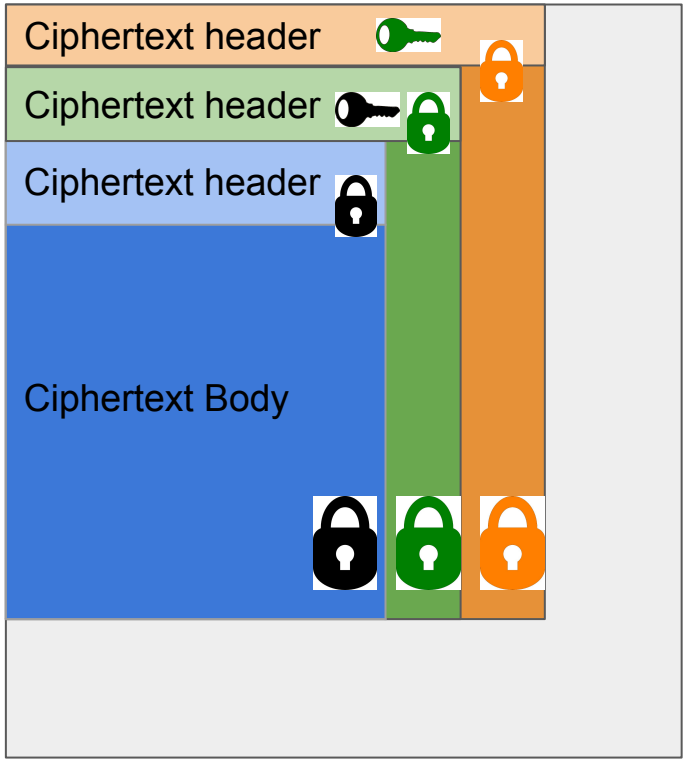
How to hide ciphertext age?

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

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

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

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

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

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

$c_0' = c_0 + F(k_{up}, 0)$

$c_1' = c_1 + F(k_{up}, 1)$

…

$c_n' = c_n + F(k_{up}, n)$

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

$c_0' = c_0 + F(k_{up}, 0) = m_0 + F(k_2, 0)$

$c_1' = c_1 + F(k_{up}, 1) = m_1 + F(k_2, 1)$

…

$c_n' = c_n + F(k_{up}, n) = m_n + F(k_2, n)$

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*

$$F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \underline{+ e} \qquad \text{(where } e \text{ is small in } Z_q^n\text{)}$$

*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*

$$F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) \underline{+ e} \qquad \text{(where } e \text{ is small in } Z_q^n\text{)}$$

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(

We us nption*

F(k1 n Z_q^n)

See paper for construction

Result: ~500x faster performance

*In Random Oracle model

Why lattice crypto?

Evaluation

Encryption and Re-encryption

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

	ReCrypt [EPRS17]	Almost KH-PRF	Nested (128 layers)
Encrypt	0.12	61.90	1836.9
Re-encrypt	0.15	83.06	2606.8

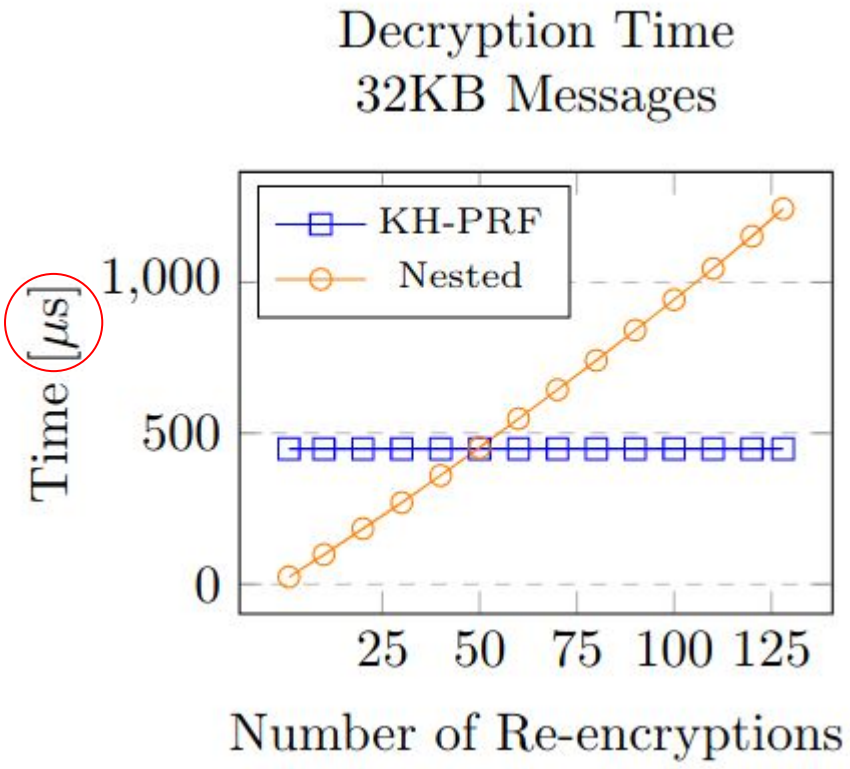
Almost KH-PRF is ~500x faster than ReCrypt

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

Decryption

Nested construction faster for up to 50 re-encryptions

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



Ciphertext Expansion

Nested AES and ReCrypt have smallest ciphertext expansion

KH-PRF or Nested AES for many re-encryptions have larger ciphertext expansion

Ciphertext Expansion 32KB Messages	
KH-PRF UAE	
$\|q\| = 28$	133%
$\|q\| = 60$	36%
$\|q\| = 120$	20%
$\|q\| = 128$	19%
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

Paper: `eprint.iacr.org/2020/222.pdf`

Source Code: `https://github.com/moshih/UpdateableEncryption_Code`

Contact: `saba@cs.stanford.edu`