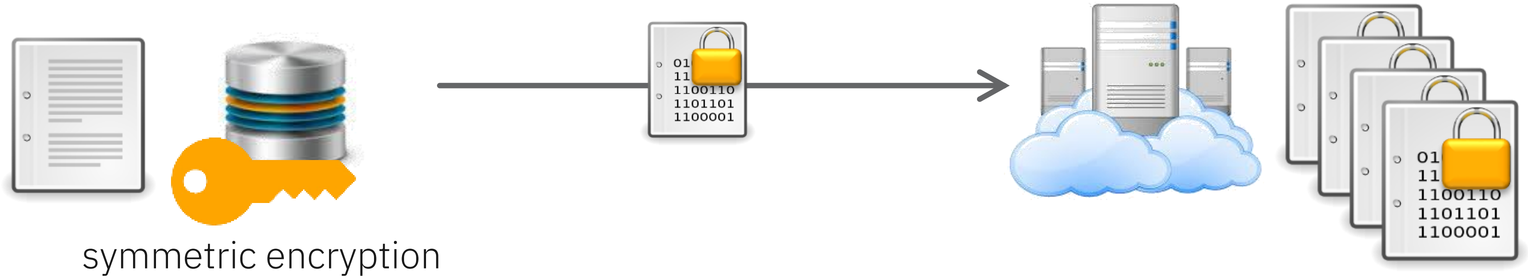
Updatable Encryption with Post-Compromise Security

Anja Lehmann & Björn Tackmann

IBM Research – Zurich

Motivation | Outsourced Storage

- Data owner stores encrypted data at (untrusted) data host

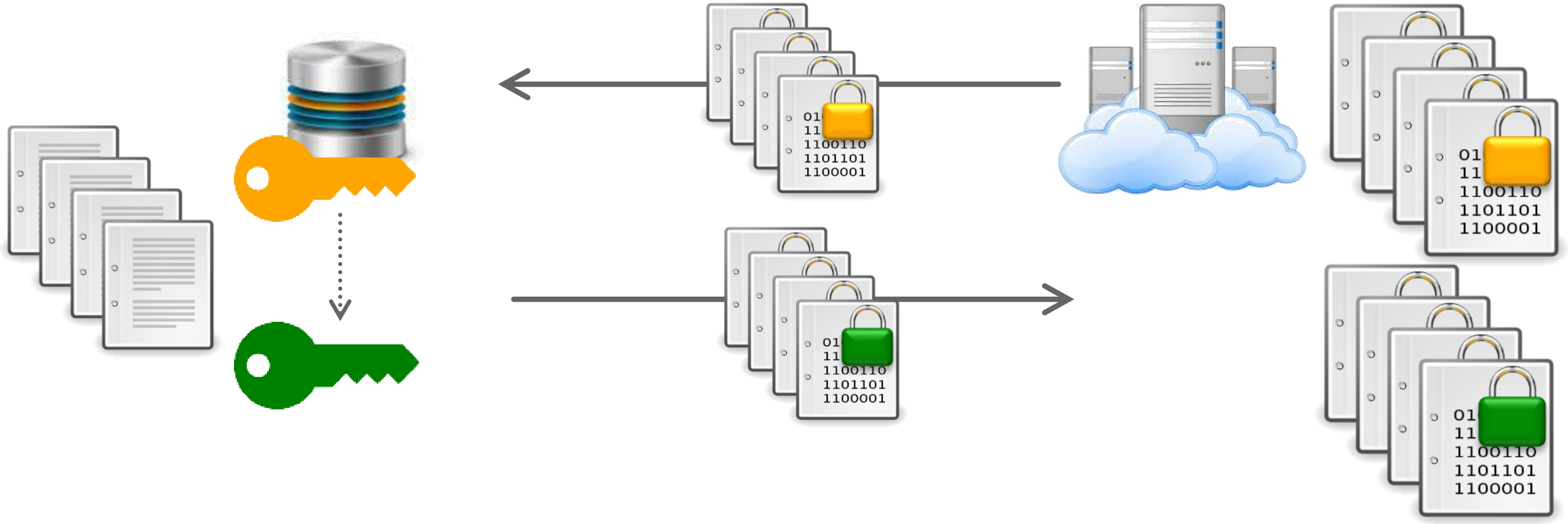


symmetric encryption

- Proactive security by periodically changing the secret key
  - Key rotation reduces risk & impact of key or data exposure
- Key rotation often mandated in high-security environments and by PCI DSS

Motivation | Key Rotation

- How to update exiting ciphertexts to the new key?

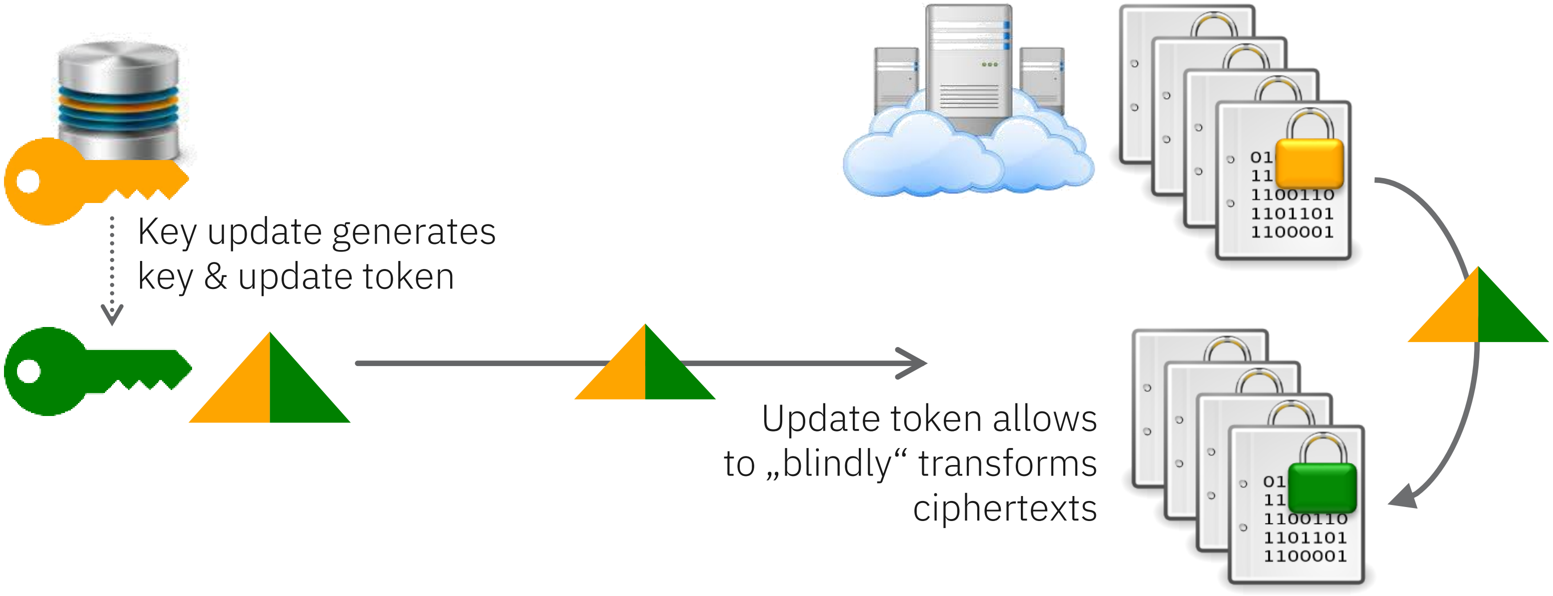


- Standard symmetric encryption → download all ciphertext & re-encrypt from scratch

- Inefficient: down&upload of all ciphertexts, symmetric key often protected by hardware

Motivation | Updatable Encryption

- Proposed by Boneh et al. [BLMR13]: ciphertexts can be updated w/o secret key



Key update generates key & update token

Update token allows to „blindly“ transforms ciphertexts

- Update operation of ciphertexts is shifted to (untrusted) data host w/o harming security

Updatable Encryption | State-of-the-Art

Ciphertext-Independent

$$\begin{aligned}&\mathrm{UE.setup}(\lambda) \to k_0\\ &\mathrm{UE.enc}(k_e, m) \to C_e\\ &\mathrm{UE.dec}(k_e, C_e) \to m\end{aligned}$$

$$\begin{aligned}&\mathrm{UE.next}(k_e) \to (k_{e+1}, \Delta_{e+1})\\ &\mathrm{UE.upd}\big(\Delta_{e+1,} C_e\big) \to C_{e+1}\end{aligned}$$

- BLMR13: high level idea & scheme, no security definitions
- EPRS17: partial definition & scheme

Ciphertext-Dependent

$$\begin{aligned}&\mathrm{UE.setup}(\lambda) \to k_0\\ &\mathrm{UE.enc}(k_e, m) \to C_e\\ &\mathrm{UE.dec}(k_e, C_e) \to m\end{aligned}$$

$$\begin{aligned}&\mathrm{UE.next}(k_e) \to k_{e+1}\\ &\mathrm{UE.token}(k_e, k_{e+1}, C_e) \to \Delta_{C,e+1}\\ &\mathrm{UE.upd}\big(\Delta_{C,e+1,} C_e\big) \to C_{e+1}\end{aligned}$$

- BLMR15: partial definitions & new scheme
- EPRS17: comprehensive treatment, improved definitions & schemes

Updatable Encryption | State-of-the-Art

Ciphertext-Independent

$\mathrm{UE.setup}(\lambda) \rightarrow k_0$

$\mathrm{UE.enc}(k_e, m) \rightarrow C_e$

$\mathrm{UE.dec}(k_e, C_e) \rightarrow m$

$\mathrm{UE.next}(k_e) \rightarrow (k_{e+1}, \Delta_{e+1})$

$\mathrm{UE.upd}\left(\Delta_{e+1,} C_e\right) \rightarrow C_{e+1}$

- BLMR13: high level idea & scheme, no security definitions
- EPRS17: partial definition & scheme
- **This work: formal definitions & secure schemes for ciphertext-independent setting**

Ciphertext-Dependent

- Less efficient: requires down&upload of (parts of) all ciphertexts & one token generation per ciphertext
- Less convenient: update requires coordination

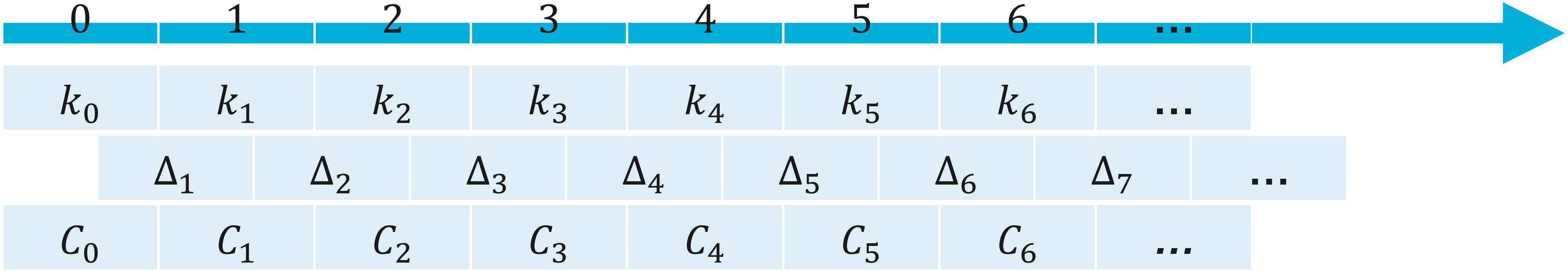
$\mathrm{UE.next}(k_e) \rightarrow k_{e+1}$

$\mathrm{UE.token}(k_e, k_{e+1}, C_e) \rightarrow \Delta_{C,e+1}$

$\mathrm{UE.upd}\left(\Delta_{C,e+1,} C_e\right) \rightarrow C_{e+1}$

- BLMR15: partial definitions & new scheme
- EPRS17: comprehensive treatment, improved definitions & schemes

Updatable Encryption | Sequential Setting



0 1 2 3 4 5 6 …

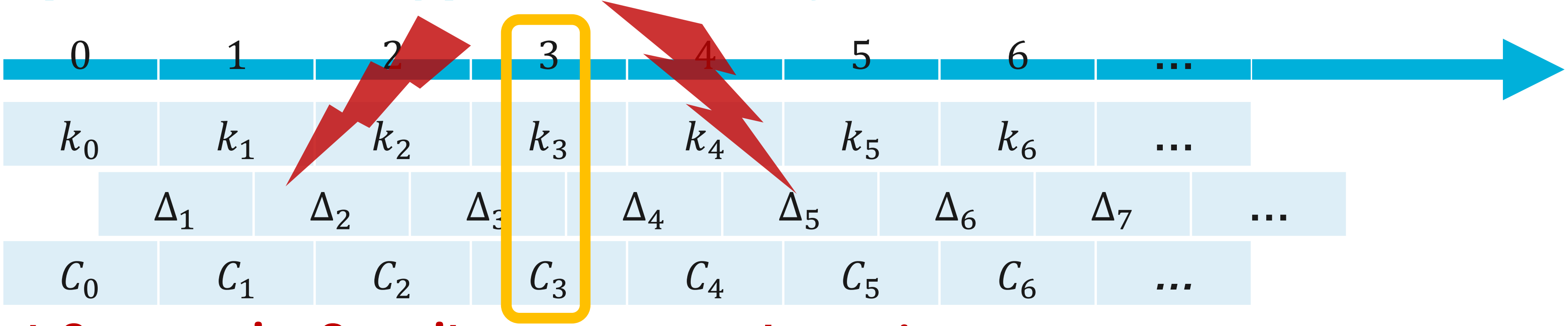
k_0 k_1 k_2 k_3 k_4 k_5 k_6 …

Δ_1 Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 Δ_7 …

C_0 C_1 C_2 C_3 C_4 C_5 C_6 …

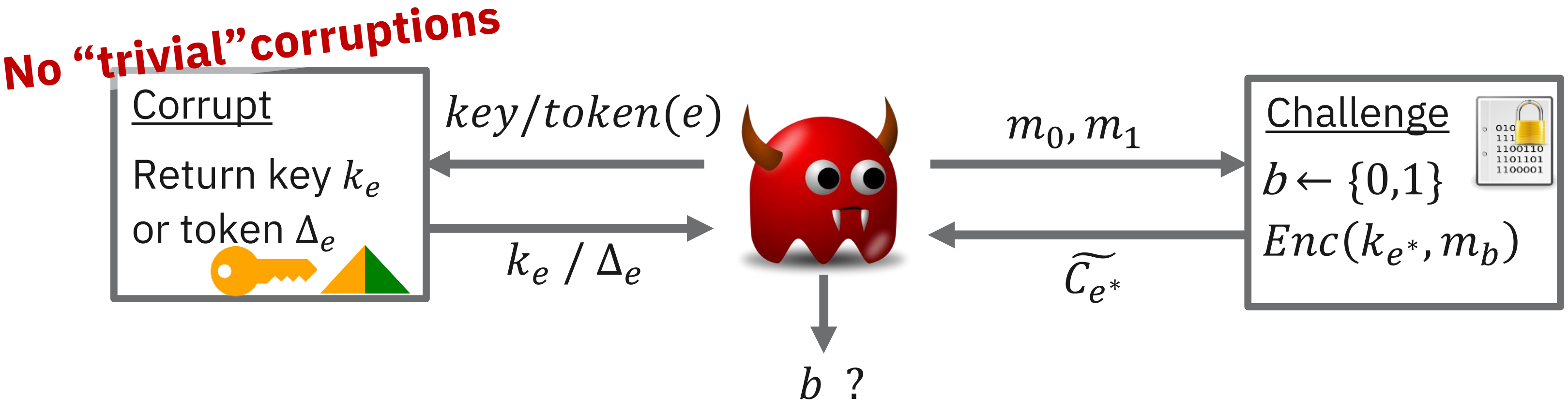
- This work: strictly sequential setting
- Previous works: adaptions of proxy re-encryption definition
  - Allows re-encryptions across arbitrary epochs (back & forward)
  - No notion of time → hard to grasp ***when*** key corruptions are allowed

Updatable Encryption | Security



0	1	2	3	4	5	6	…
k_0	k_1	k_2	k_3	k_4	k_5	k_6	…
Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	…
C_0	C_1	C_2	C_3	C_4	C_5	C_6	…

Post-Compromise Security + Forward Security = IND-ENC



No “trivial”corruptions

Corrupt

Return key k_e or token Δ_e

$key/token(e)$

k_e / Δ_e

m_0, m_1

Challenge

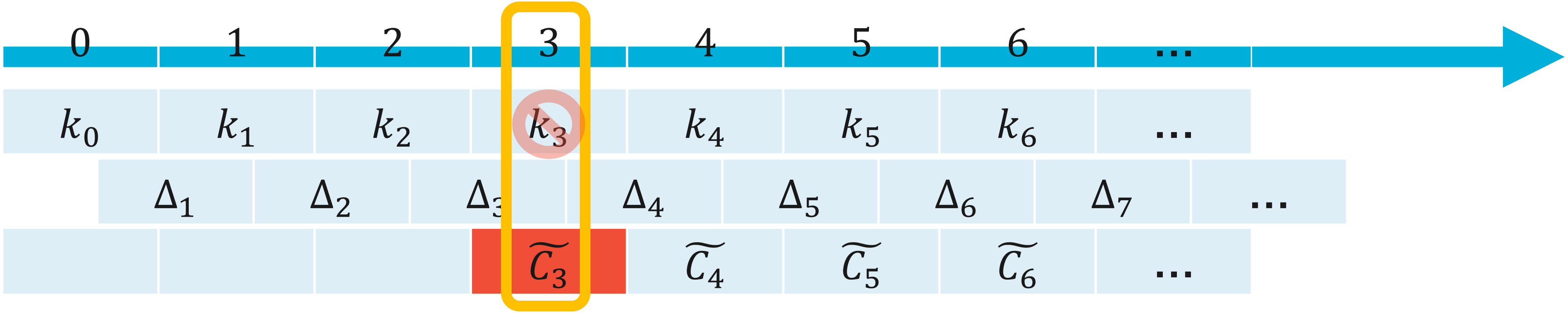
$b \leftarrow \{0,1\}$

$Enc(k_{e^*}, m_b)$

$\widetilde{C}_{e^*}$

b ?

Updatable Encryption | IND-ENC & Trivial Wins



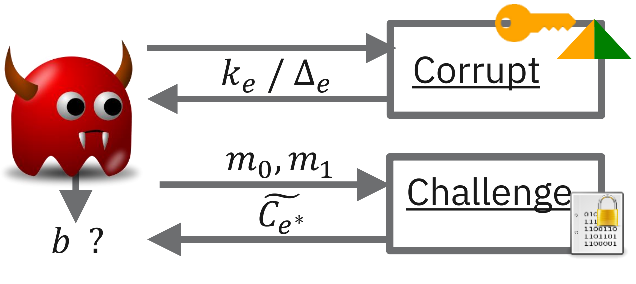
0 1 2 3 4 5 6 ...

k_0 k_1 k_2 k_3 k_4 k_5 k_6 ...

Δ_1 Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 Δ_7 ...

$\widetilde{C}_3$ $\widetilde{C}_4$ $\widetilde{C}_5$ $\widetilde{C}_6$ ...

- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates



k_e / Δ_e

Corrupt

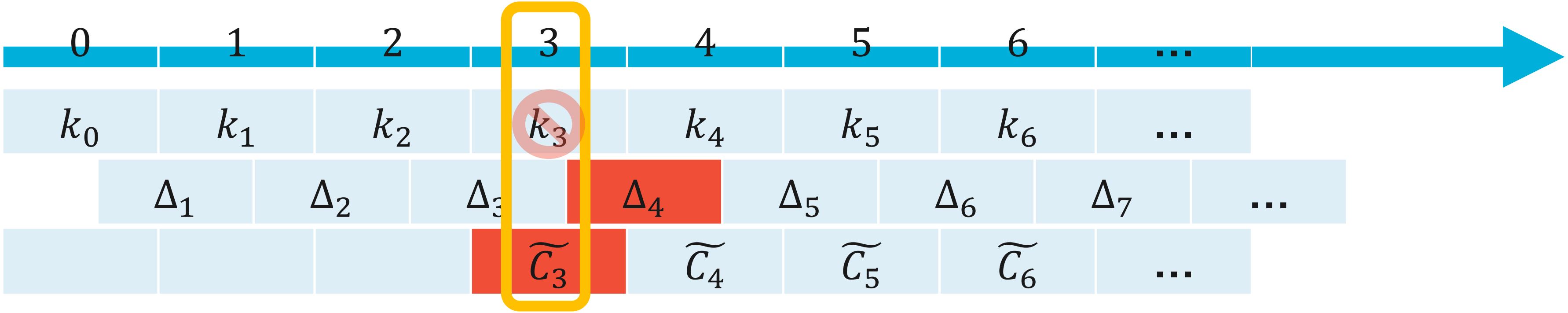
m_0, m_1

$\widetilde{C}_{e^*}$

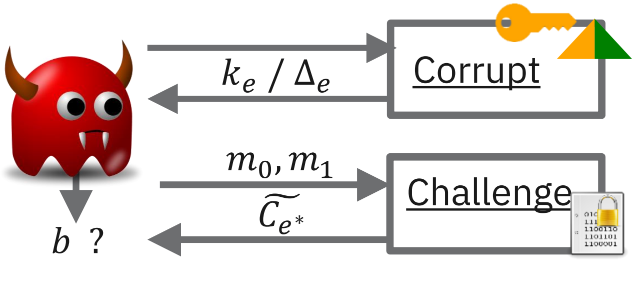
Challenge

b ?

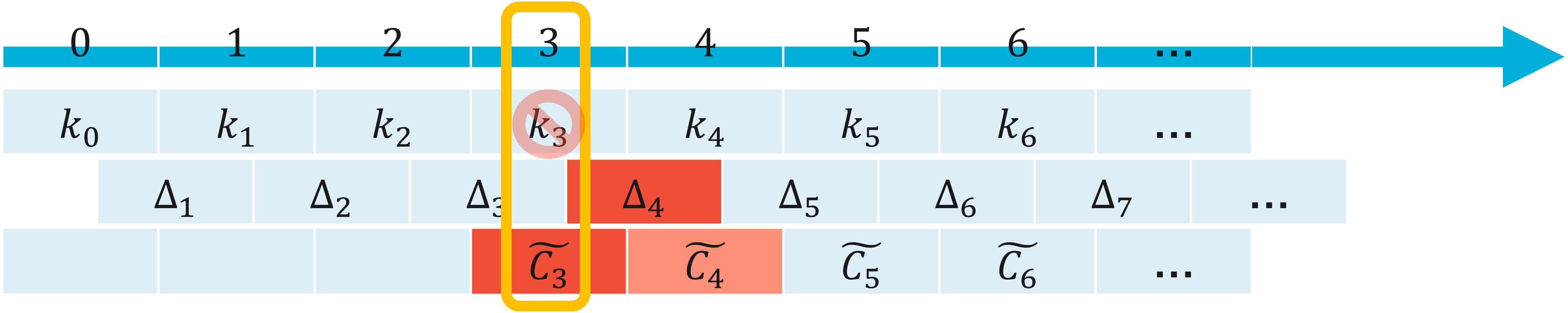
Updatable Encryption | IND-ENC & Trivial Wins



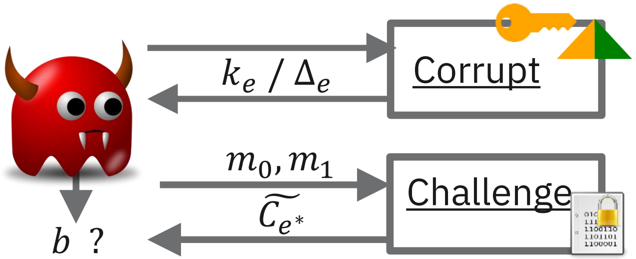
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates



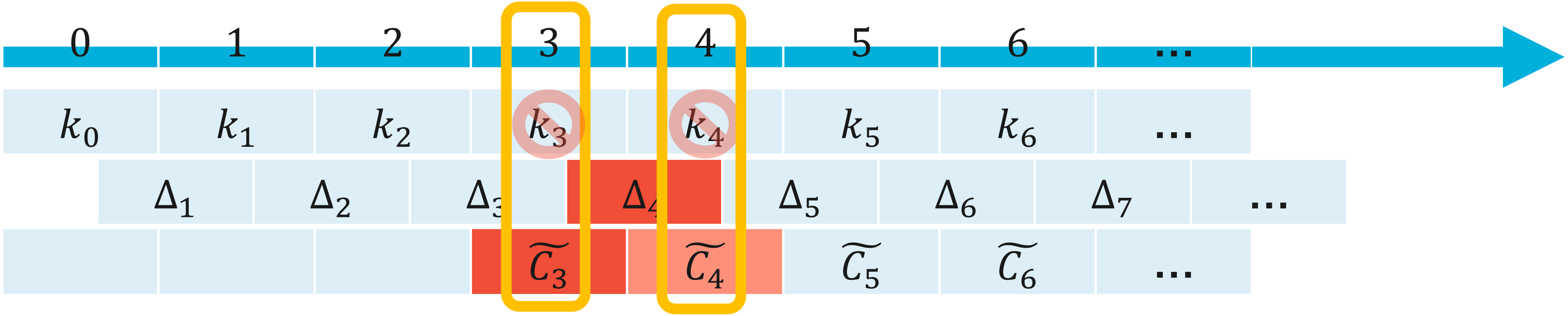
Updatable Encryption | IND-ENC & Trivial Wins


0 1 2 3 4 5 6 ...
k_0 k_1 k_2 k_3 k_4 k_5 k_6 ...
Δ_1 Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 Δ_7 ...
$\widetilde{C_3}$ $\widetilde{C_4}$ $\widetilde{C_5}$ $\widetilde{C_6}$ ...

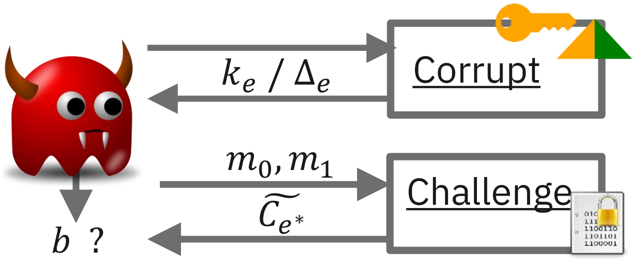
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates



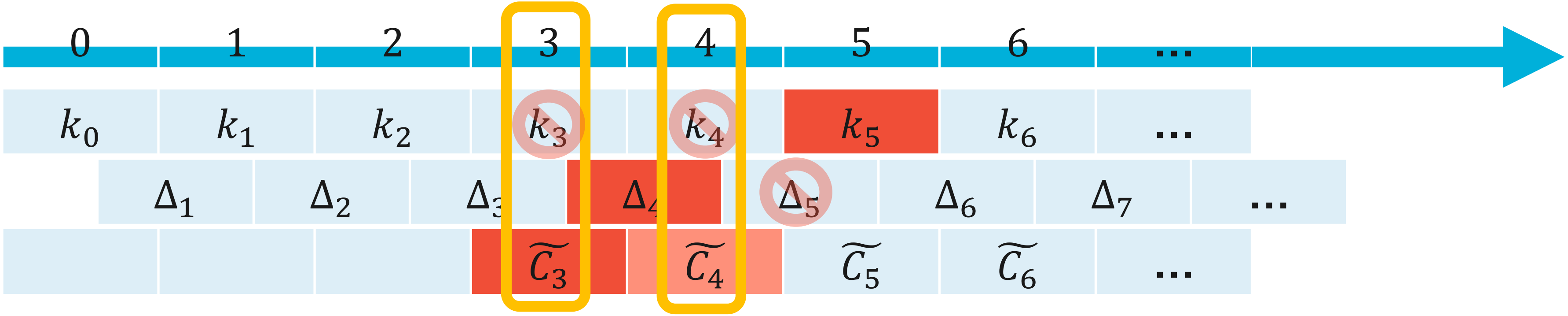
Updatable Encryption | IND-ENC & Trivial Wins



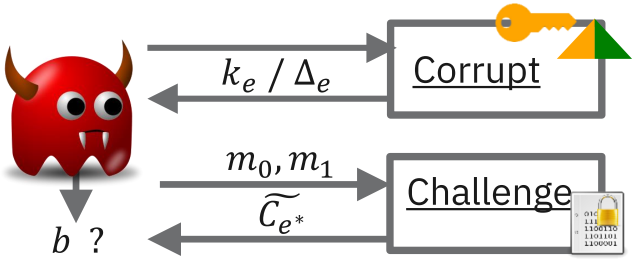
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
    - Ideal: **uni**directional ciphertext-updates



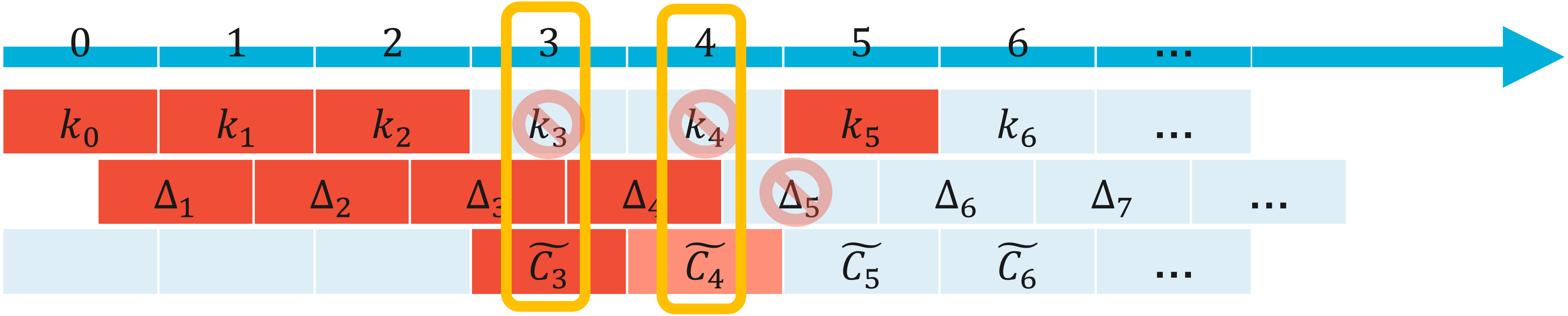
Updatable Encryption | IND-ENC & Trivial Wins



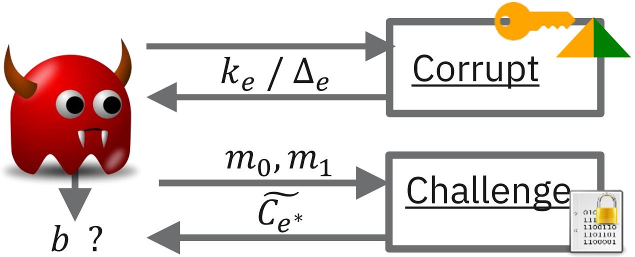
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
    - Ideal: **uni**directional ciphertext-updates



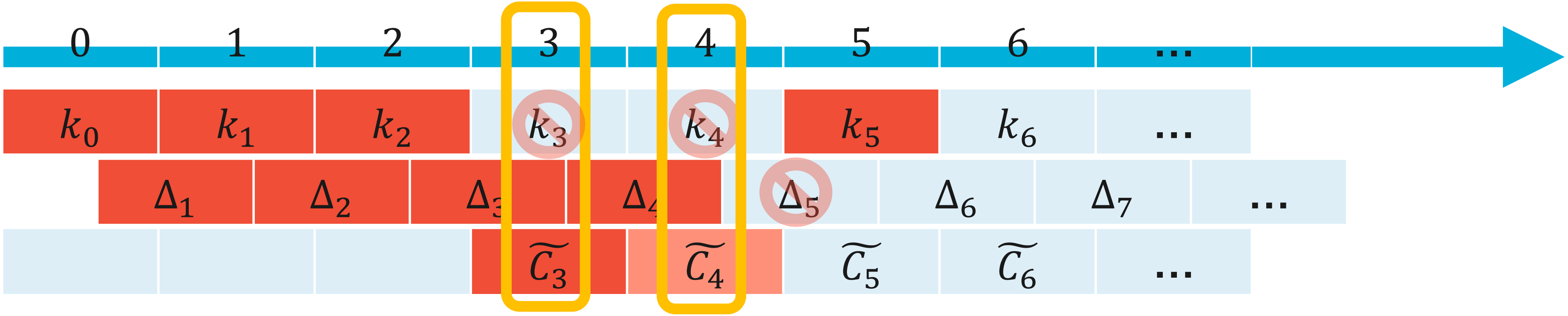
Updatable Encryption | IND-ENC & Trivial Wins



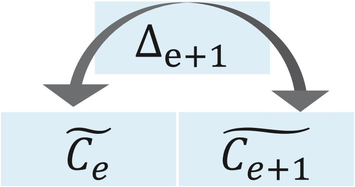
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates

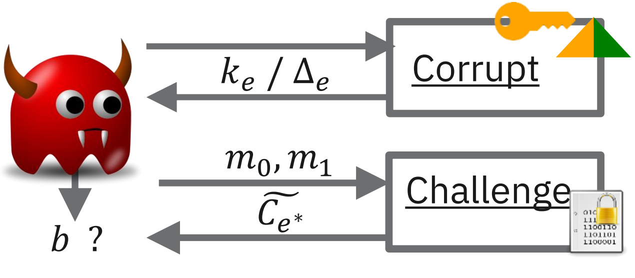


Updatable Encryption | IND-ENC & Trivial Wins

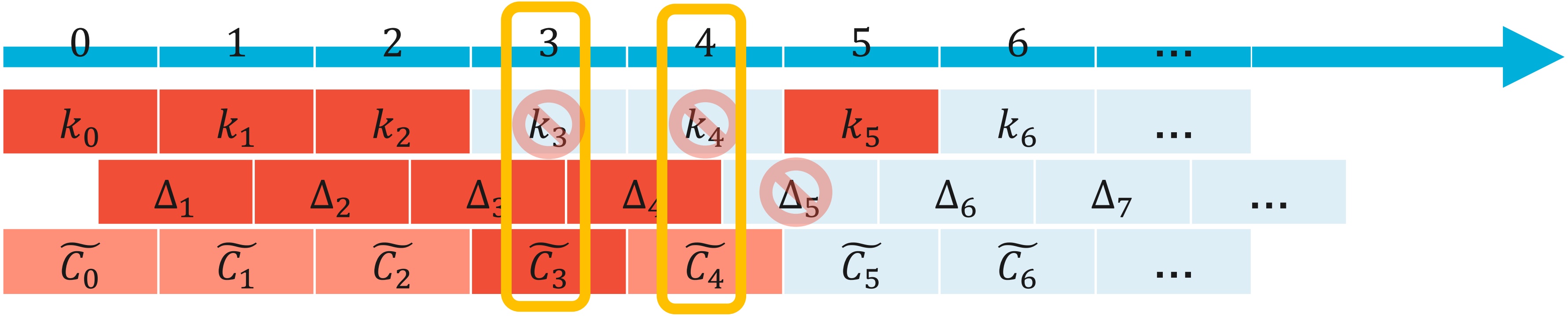


- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates
  - Real: **bi**directional ciphertext-updates

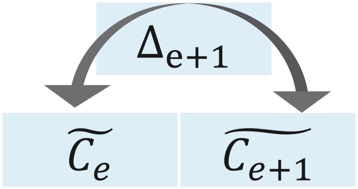


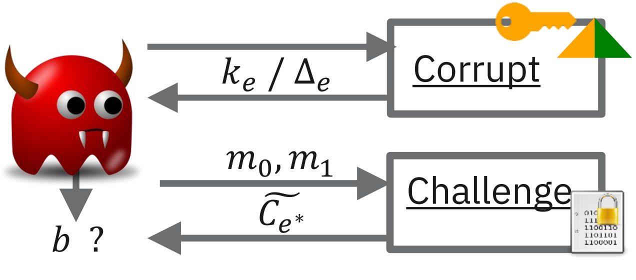


Updatable Encryption | IND-ENC & Trivial Wins

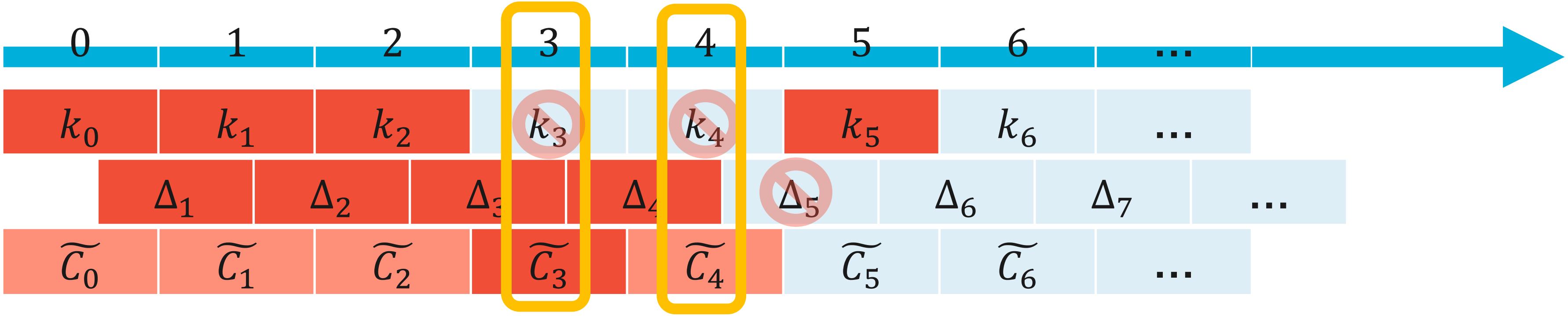


- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates
  - Real: **bi**directional ciphertext-updates

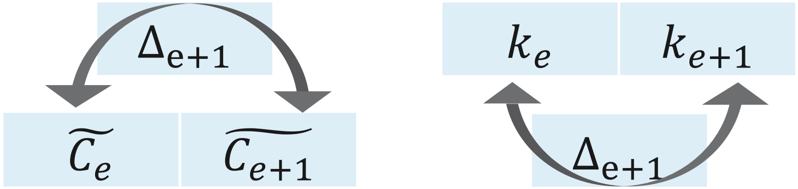


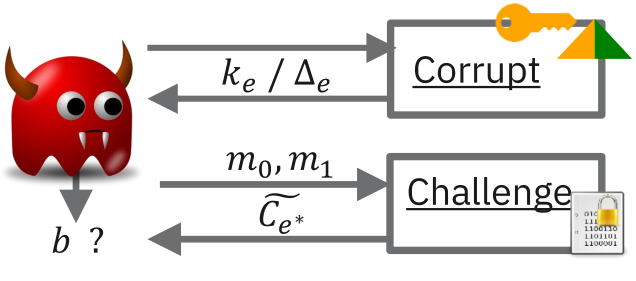


Updatable Encryption | IND-ENC & Trivial Wins

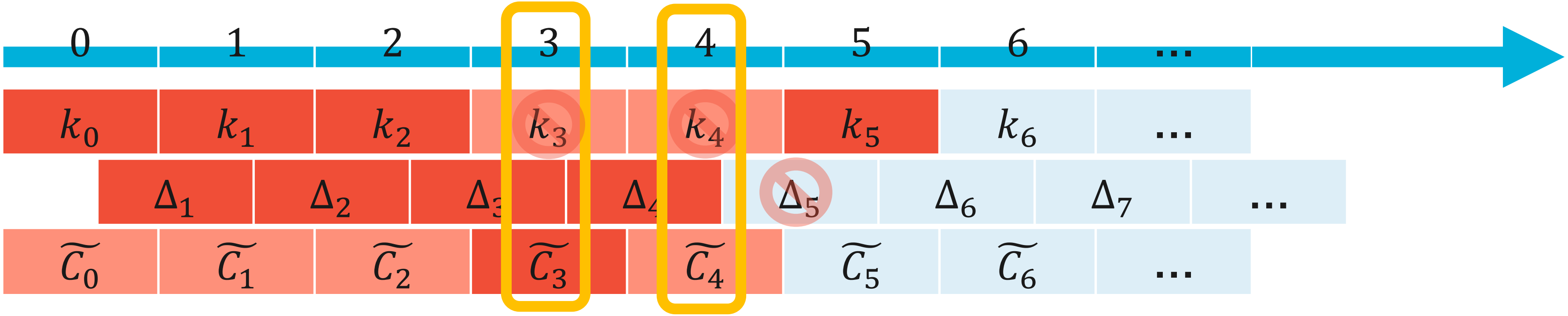


- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates
  - Real: **bi**directional ciphertext **& key-updates**

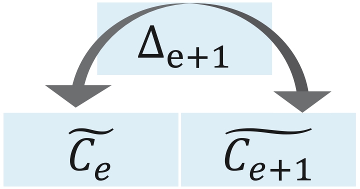


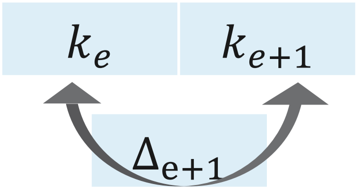


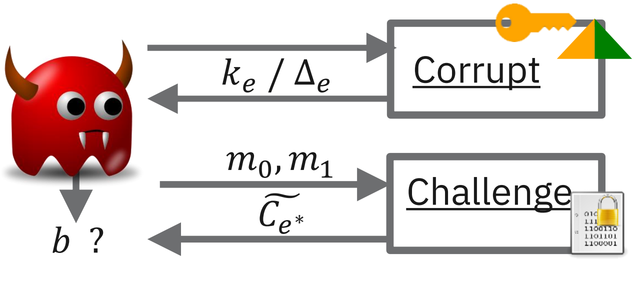
Updatable Encryption | IND-ENC & Trivial Wins



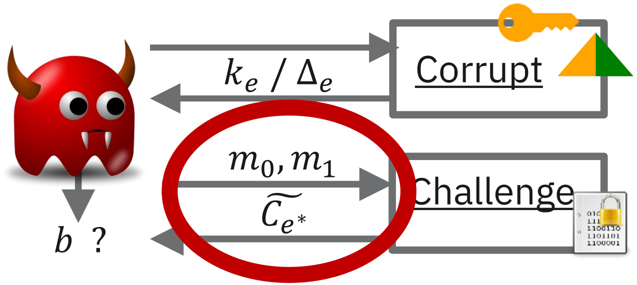
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **uni**directional ciphertext-updates
  - Real: **bi**directional ciphertext **& key-updates**







Updatable Encryption | IND-ENC



- IND-ENC definition
  - Adaptive and retroactive key & token corruptions
  - Formalizes indirect knowledge of keys & challenge cipherexts
  - Covers CPA, post-compromise and forward security for **fresh encryptions**

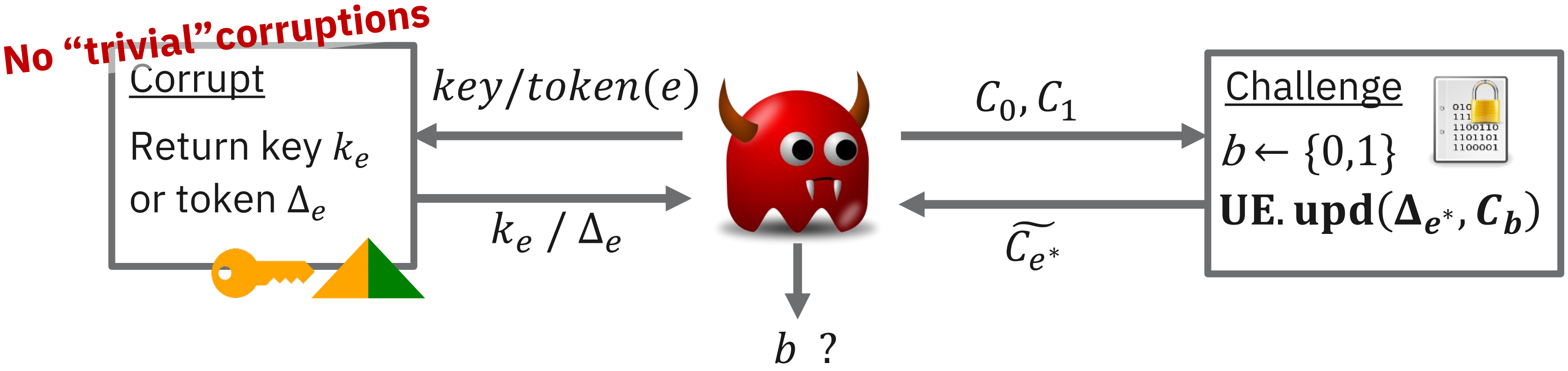
- IND-ENC is not sufficient: No guarantees about updated ciphertexts!
  - $\mathrm{UE.upd}\left(\Delta_{e+1,} C_e\right) \to C_{e+1}$

    could contain C_e, i.e., history of all old ciphertexts ($C'_3 = C_3, (C_2, (C_1,(C_0)))$)

    compromise of a single old key breaks security of updated ciphertexts

Updatable Encryption | IND-UPD

- IND-UPD definition = Update Indistinguishability
  - Adaptive and retroactive key & token corruptions
  - Formalizes indirect knowledge of keys & challenge cipherexts
  - Covers post-compromise and forward security for **updated ciphertexts**



IND-ENC + IND-UPD = Secure Updatable Encryption

Updatable Encryption | (In)Secure Schemes

Re-Randomizable Ciphertext-**I**ndependent **S**ymmetric **E**lGamal

	2ENC (folklore)	**XOR-KEM (EPRS17)**	**BLMR (BLMR13)**	**RISE**
Enc	$Enc(k_e^o, Enc(k^i, m))$	$(k_e \oplus x), Enc(x, m)$	$PRF(k_e, N) \otimes m, N$	see paper
Tok Δ_{e+1}	(k_e^o, k_{e+1}^o)	$k_e \oplus k_{e+1}$	$k_e \oplus k_{e+1}$	
IND-ENC	(with limitations)	🟥	Key-homomorph PRF	DDH
IND-UPD	(with limitations)	🟥	🟥	DDH

Key-homomorphic PRF: $PRF(k_1, N) \otimes PRF(k_2, N) = PRF(k_1 \oplus k_2, N)$

Also crucial building block in ReEnc [EPRS17] = ciphertext-*dependent* UE

Known instantiations either DL or lattice-based

Updatable Encryption | Efficiency & Summary

- RISE is more efficient than existing solutions

n = number of ciphertexts

Scheme		Encryption	TokenGen	Update
BLMR	Only IND-ENC secure	2 exp	2 exp	2n exp
RISE		**2 exp**	**1 exp**	**2n exp**
ReEnc [EPRS17]	Ciphertext Dependent	2 exp	2n exp	2n exp

- Summary
  - Security notions for Ciphertext-Independent Updatable Encryption
  - Existing schemes do not guarantee the desirable (post-compromise) security
  - RISE = fully secure scheme based on ElGamal encryption

Thanks! Questions?

anj@zurich.ibm.com

Updatable Encryption | Secure Construction (RISE)

RISE.setup(λ): $x \xleftarrow{\text{r}} \mathbb{Z}_q^*$, set $k_0 \leftarrow (x, g^x)$, return k_0

RISE.next(k_e): parse $k_e = (x, y)$, draw $x' \xleftarrow{\text{r}} \mathbb{Z}_q^*$,
$k_{e+1} \leftarrow (x', g^{x'})$, $\Delta_{e+1} \leftarrow (x'/x, g^{x'})$ return (k_{e+1}, Δ_{e+1})

RISE.enc(k_e, m): parse $k_e = (x, y)$, $r \xleftarrow{\text{r}} \mathbb{Z}_q$, return $C_e \leftarrow (y^r, g^r m)$

RISE.dec(k_e, C_e): parse $k_e = (x, y)$ and $C_e = (C_1, C_2)$, return $m' \leftarrow C_2 \cdot C_1^{-1/x}$

RISE.upd(Δ_{e+1}, C_e): parse $\Delta_{e+1} = (\Delta, y')$ and $C_e = (C_1, C_2)$,
$r' \xleftarrow{\text{r}} \mathbb{Z}_q$, $C_1' \leftarrow C_1^{\Delta} \cdot y'^{r'}$, $C_2' \leftarrow C_2 \cdot g^{r'}$, return $C_{e+1} \leftarrow (C_1', C_2')$