

# Updatable Encryption & Key Rotation

Anja Lehmann

IBM Research – Zurich

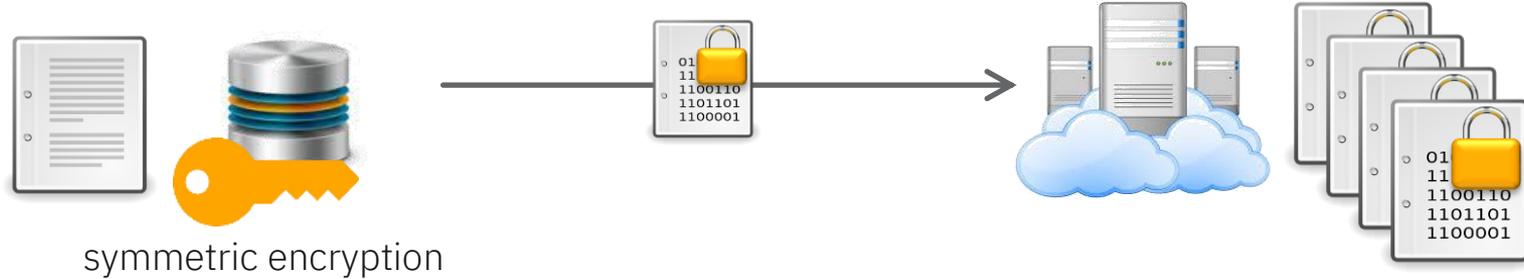
(R)CCA Secure Updatable Encryption with Integrity Protection. EUROCRYPT 2019  
M Kloos, A Lehmann, A Rupp

Updatable Encryption with Post-Compromise Security. EUROCRYPT 2018  
A Lehmann, B Tackmann



# Motivation | Outsourced Storage

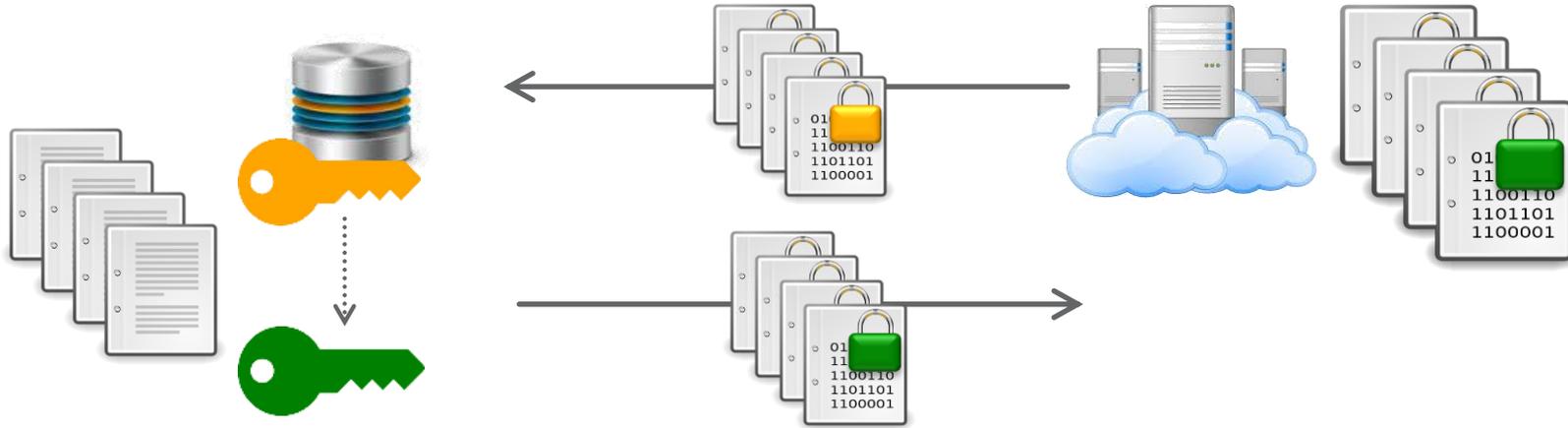
- Data owner stores encrypted data at (untrusted) data host



- 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

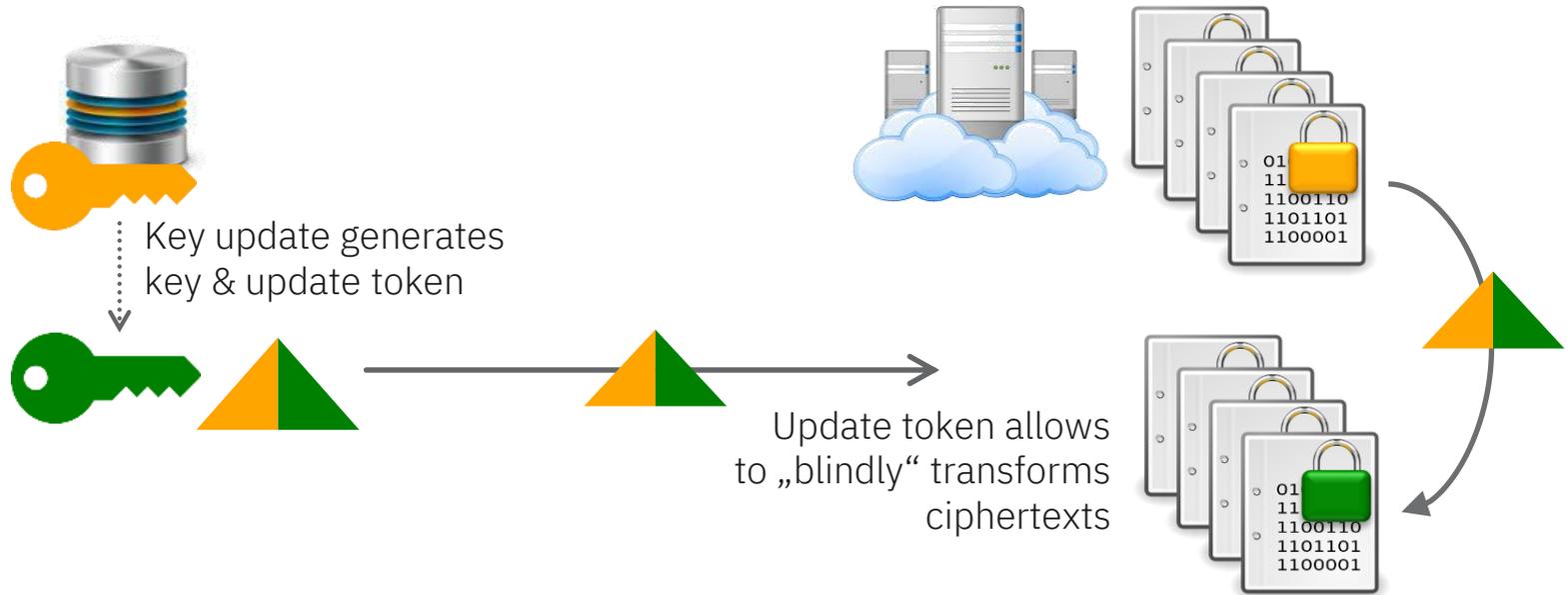
- How to update exiting ciphertexts to the new key?



- 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



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

# Updatable Encryption | State-of-the-Art

## Ciphertext-Independent

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

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

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

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

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

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

## Ciphertext-Dependent

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

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

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

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

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

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

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

# Updatable Encryption | State-of-the-Art

Ciphertext-Independent

UE. setup( $\lambda$ )  $\rightarrow k_0$

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

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

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

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

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

Ciphertext-Dependent

- Fine-grained control of updates

- Less efficient: requires download & upload of (parts of) all ciphertexts & one token generation per ciphertext

- Less convenient: update requires coordination

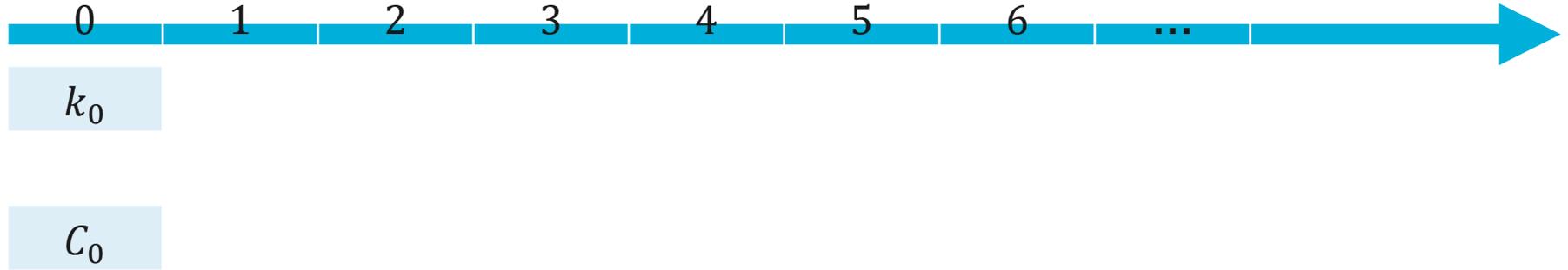
UE. next( $k_e$ )  $\rightarrow k_{e+1}$

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

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

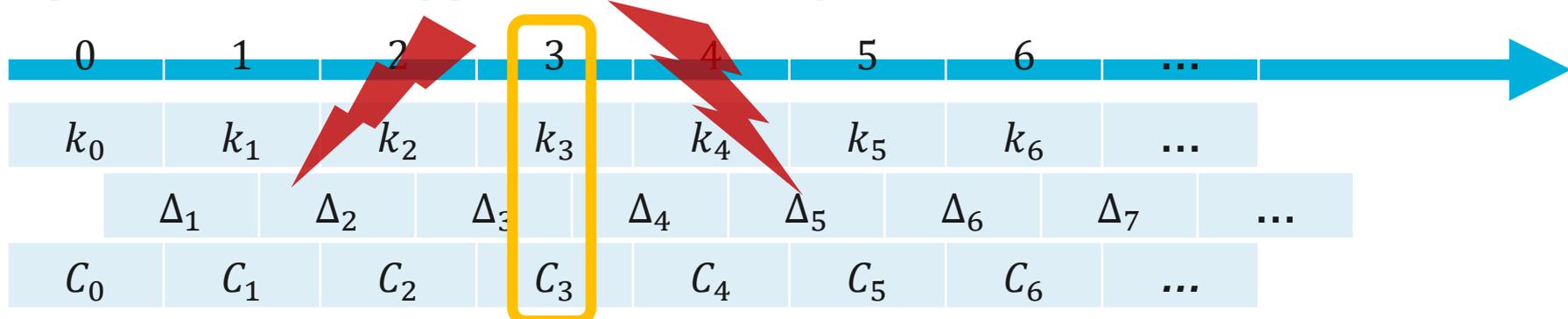
- BLMR15: partial definitions & new scheme
- EPRS17: comprehensive treatment, improved definitions & schemes

# Updatable Encryption | Sequential Setting



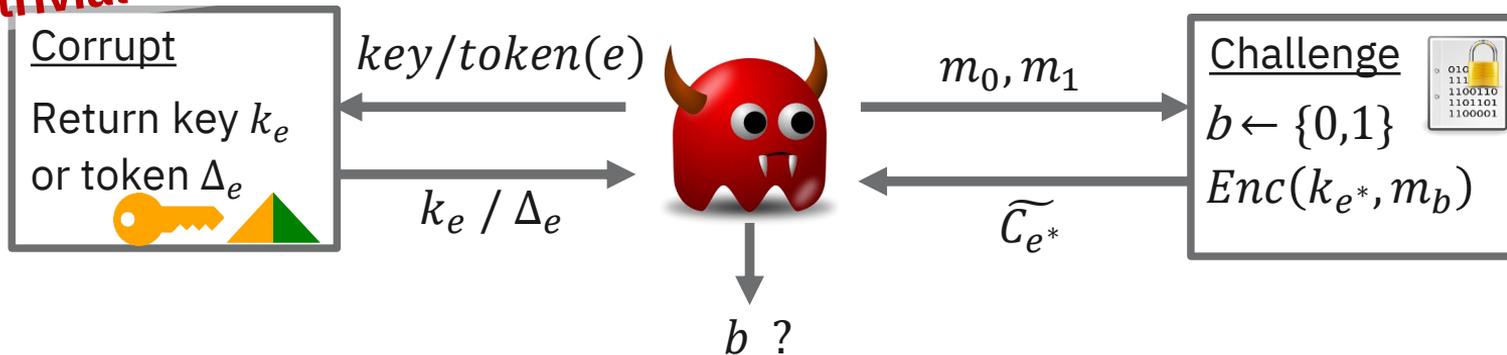
- Our work: strictly sequential setting
- Previous works: adaptations of proxy re-encryption definition
  - Allows re-encryptions across arbitrary epochs (back & forward)
  - No notion of time → hard to grasp *when* key corruptions are allowed

# Updatable Encryption | Security

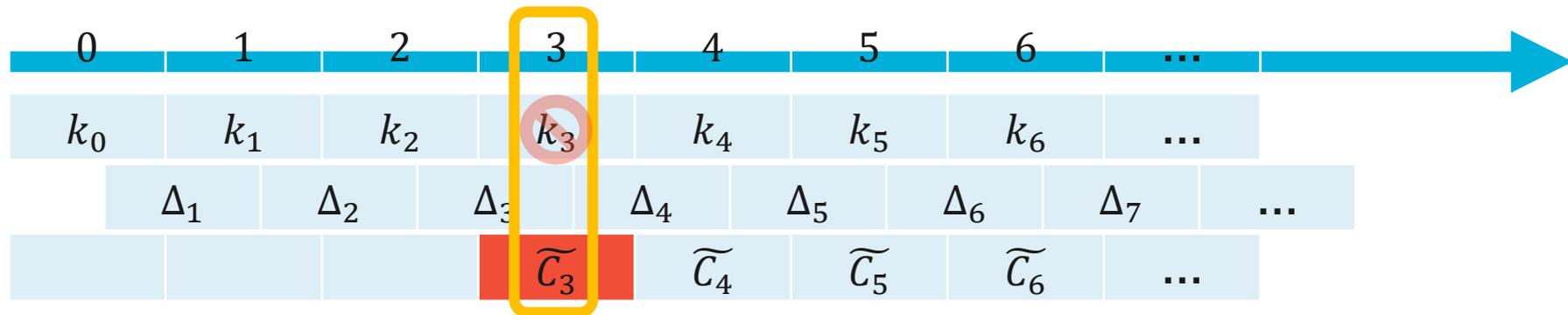


**Post-Compromise Security + Forward Security = IND-ENC**

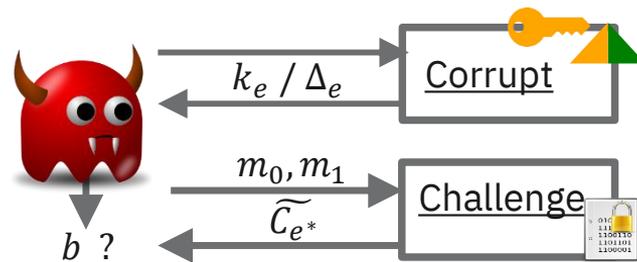
**No "trivial" corruptions**



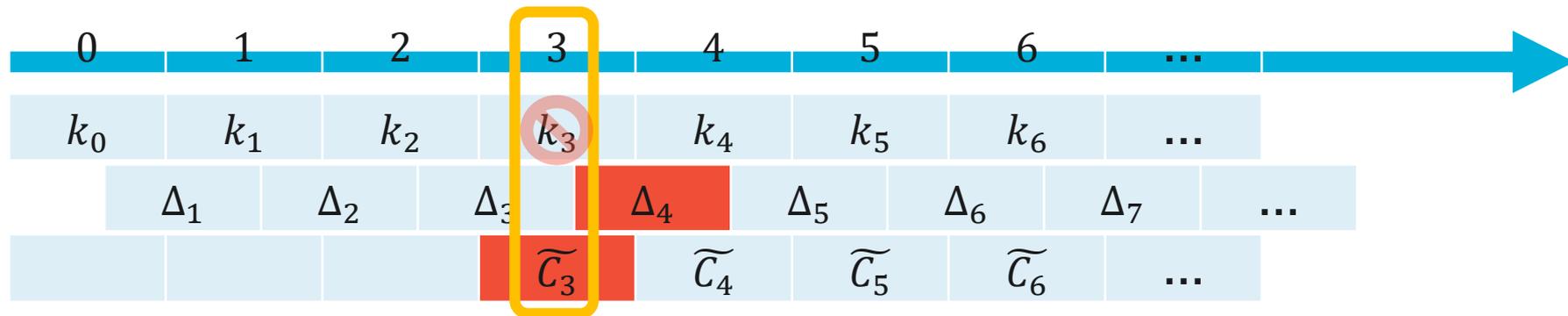
# Updatable Encryption | Capturing Trivial Wins



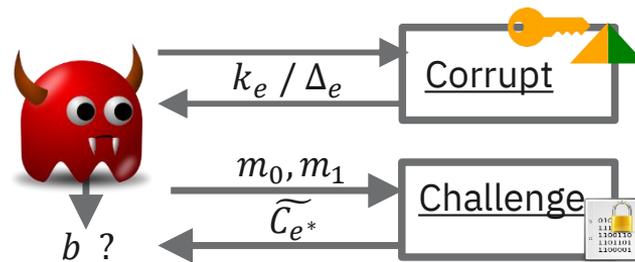
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates



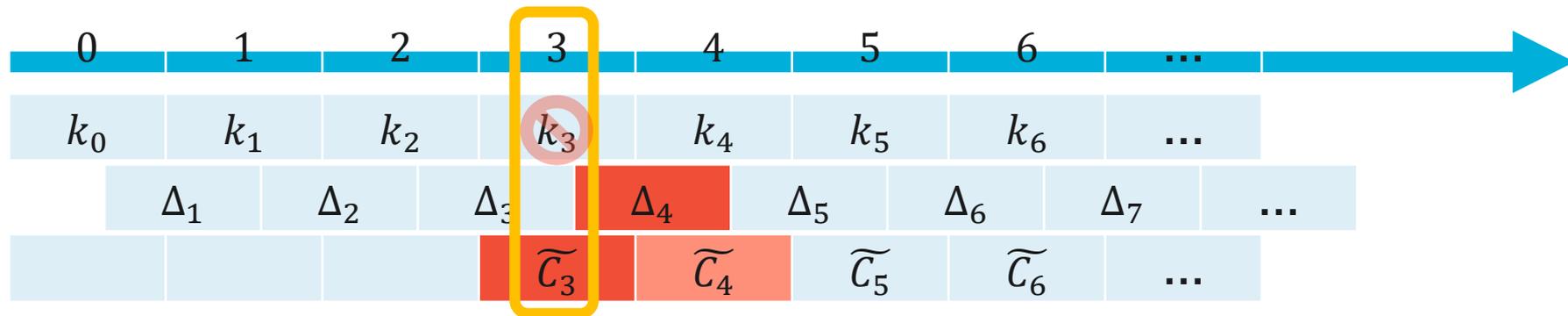
# Updatable Encryption | Capturing Trivial Wins



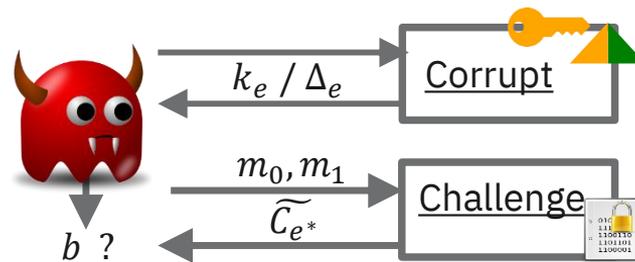
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates



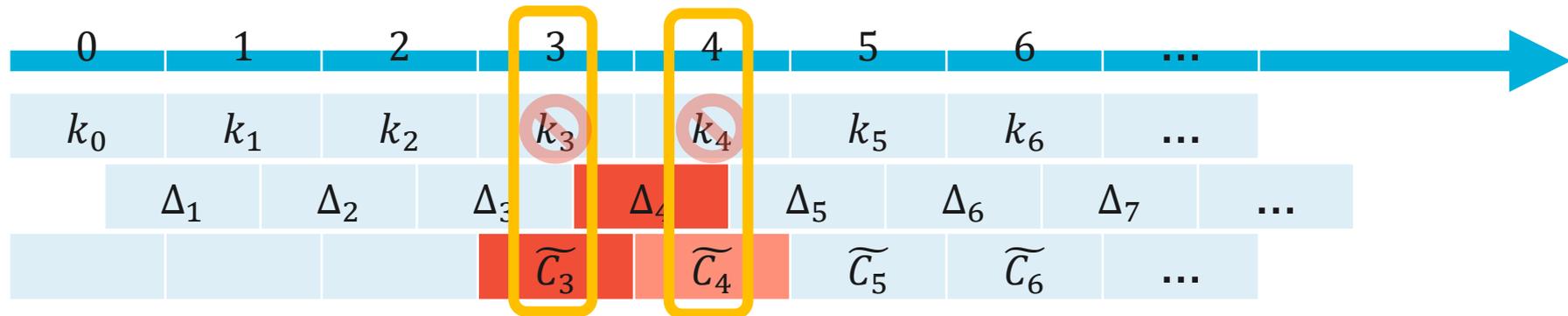
# Updatable Encryption | Capturing Trivial Wins



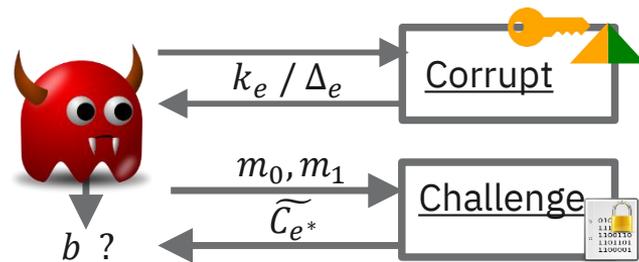
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates



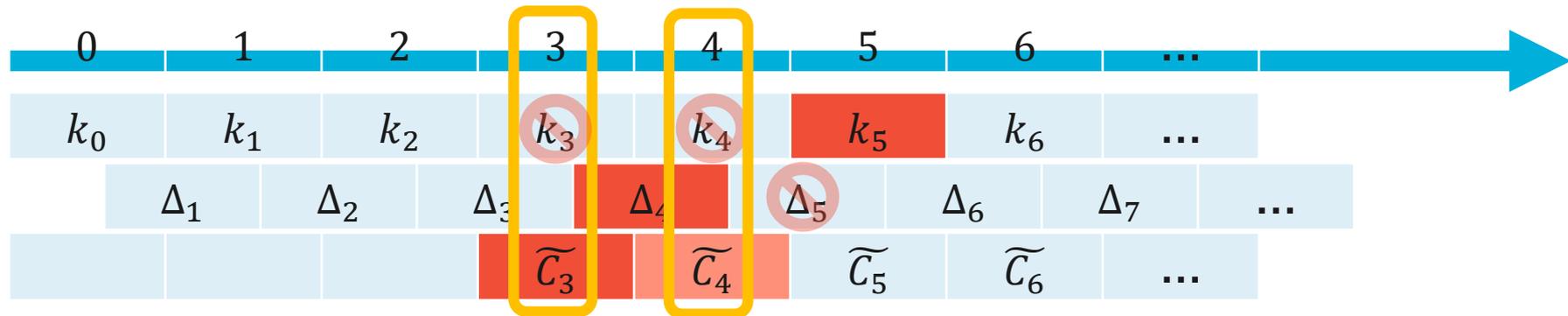
# Updatable Encryption | Capturing Trivial Wins



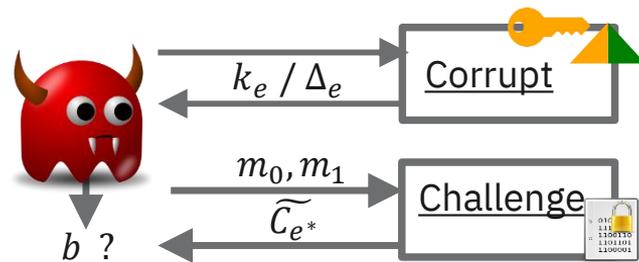
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates



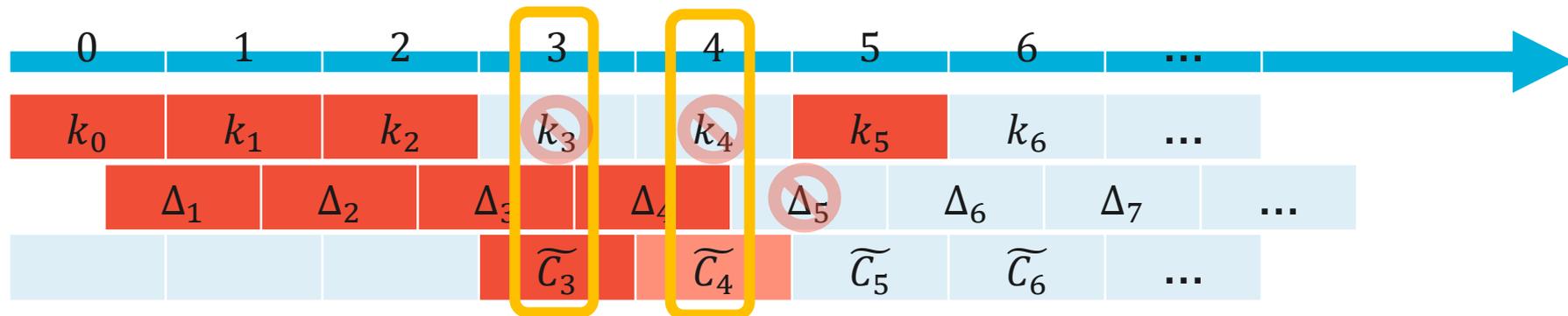
# Updatable Encryption | Capturing Trivial Wins



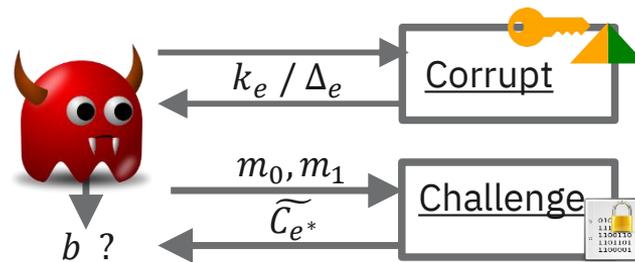
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates



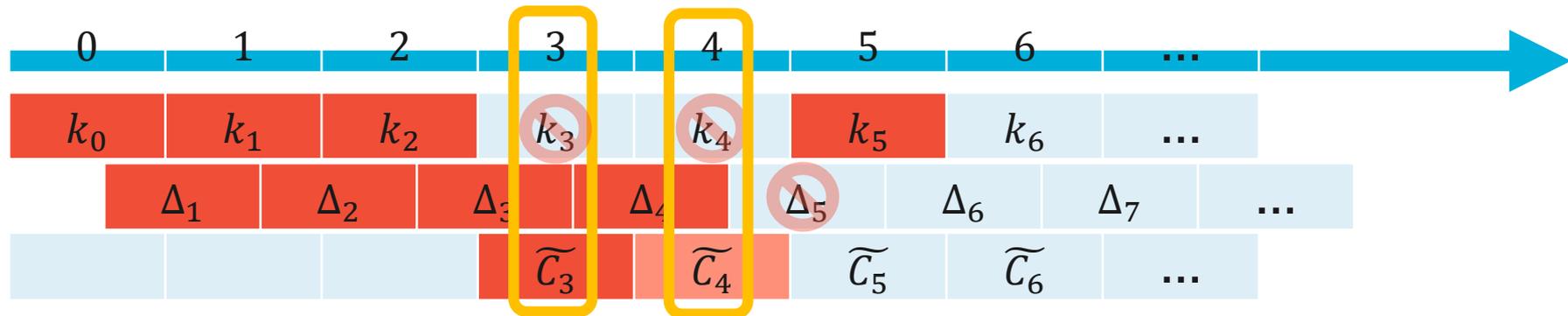
# Updatable Encryption | Capturing Trivial Wins



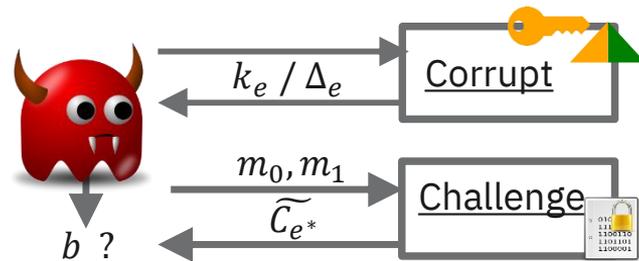
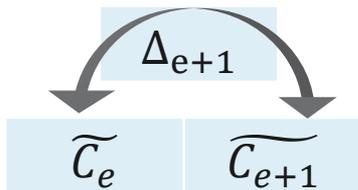
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates



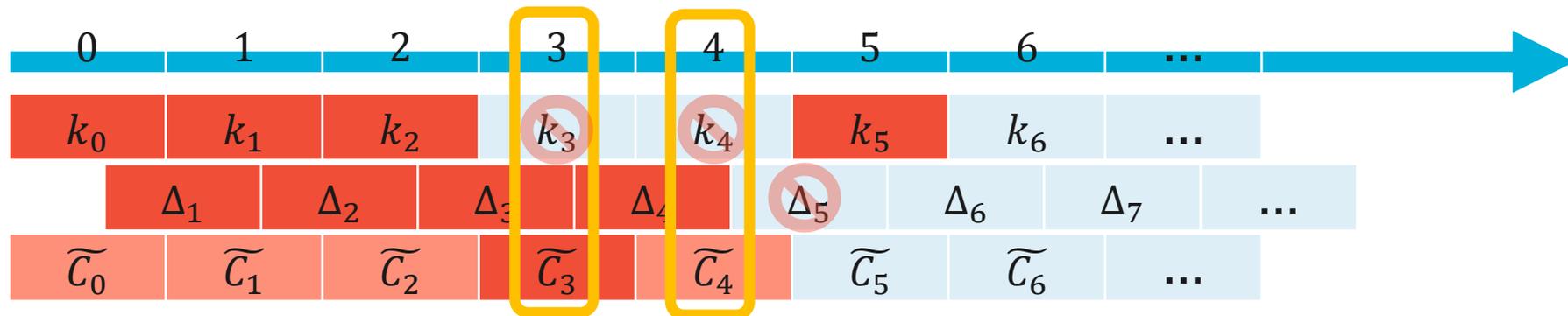
# Updatable Encryption | Capturing Trivial Wins



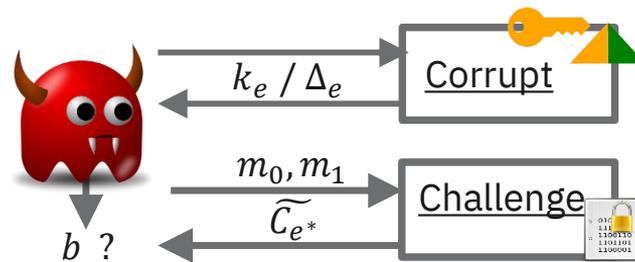
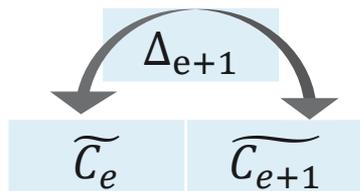
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates
  - Real: **bidirectional** ciphertext-updates



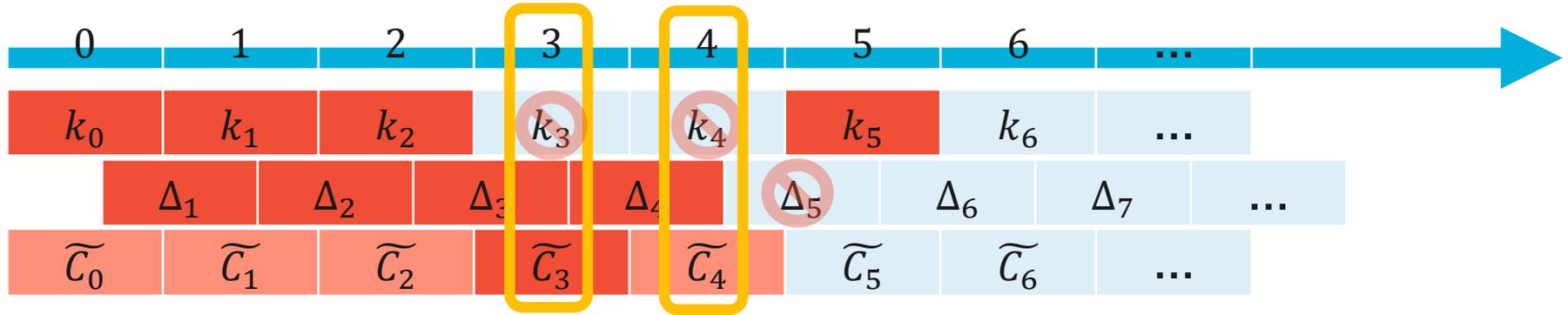
# Updatable Encryption | Capturing Trivial Wins



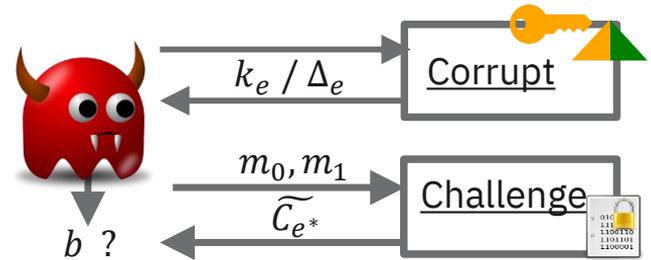
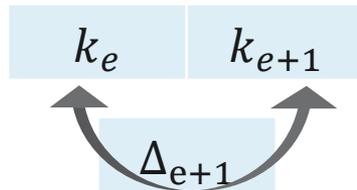
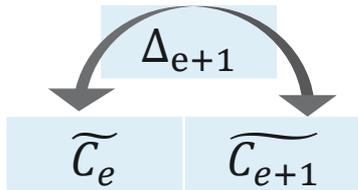
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates
  - Real: **bidirectional** ciphertext-updates



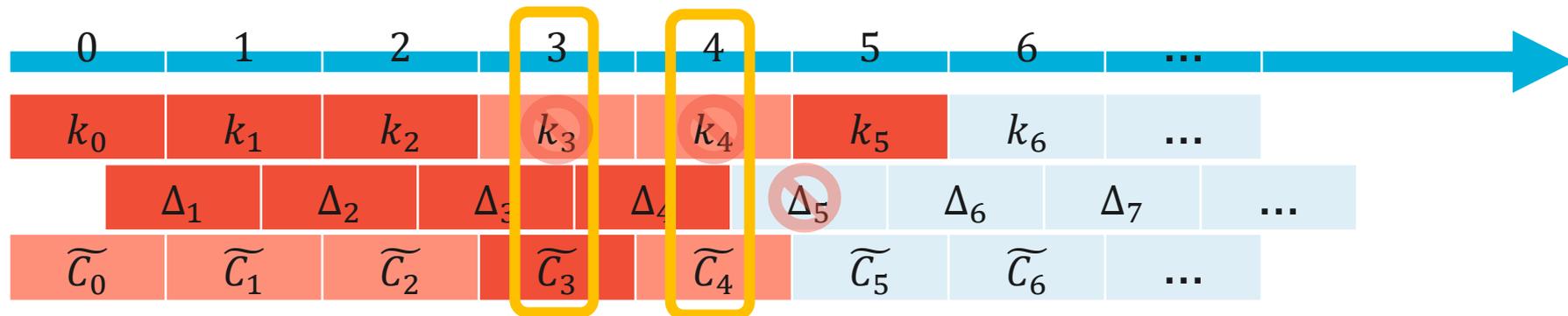
# Updatable Encryption | Capturing Trivial Wins



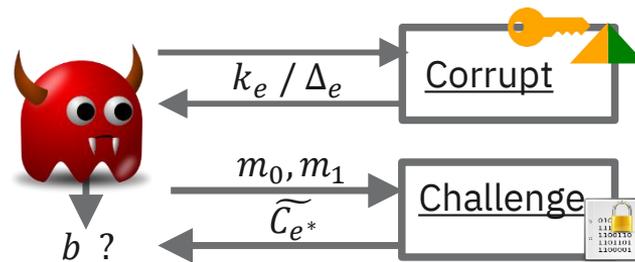
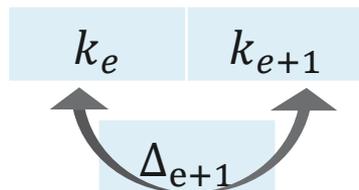
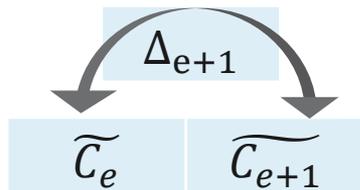
- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates
  - Real: **bidirectional** ciphertext & key-updates



# Updatable Encryption | Capturing Trivial Wins

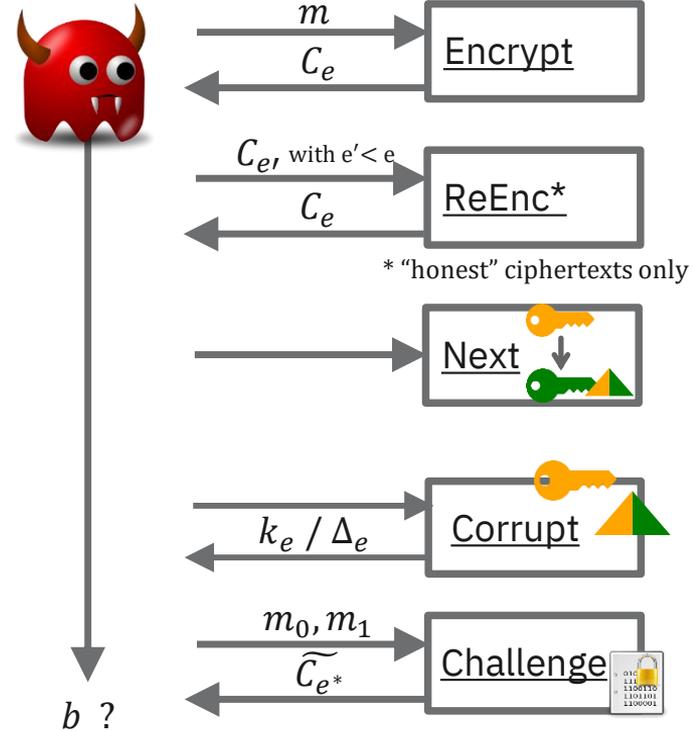


- Trivial win: secret key corruption in a challenge-equal epoch
- Capturing inferable information:
  - Ideal: **unidirectional** ciphertext-updates
  - Real: **bidirectional** ciphertext & key-updates

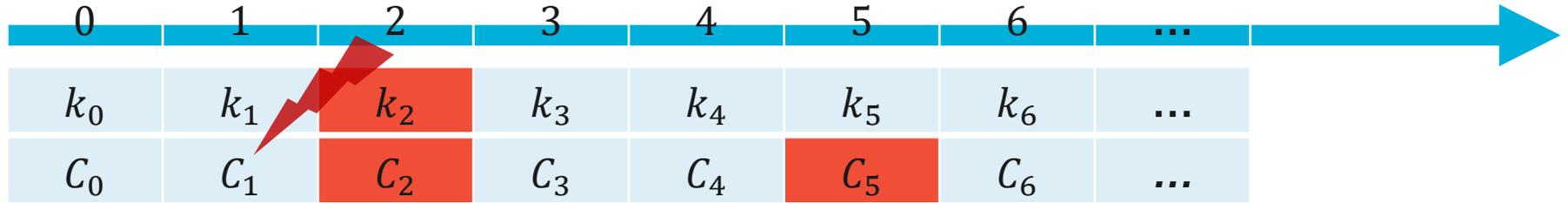


# Updatable Encryption | IND-ENC

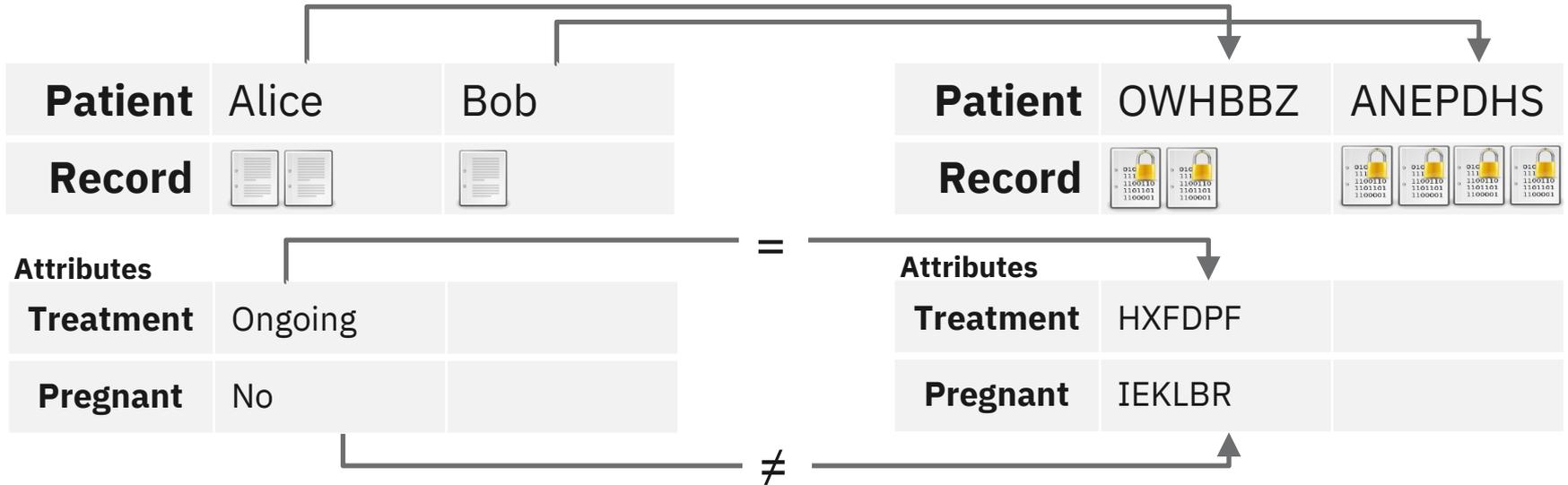
- IND-ENC definition
  - Adaptive and retroactive key & token corruptions
  - Formalizes inferable information of keys & challenge ciphertexts → exclude trivial wins
  - Covers CPA, post-compromise and forward security for **fresh encryptions & updated ciphertexts**
- Wrong claim in EC'18 paper:  
IND-ENC is not sufficient.  
No guarantees about updated ciphertexts!



# Updatable Encryption | What IND-ENC is not guaranteeing

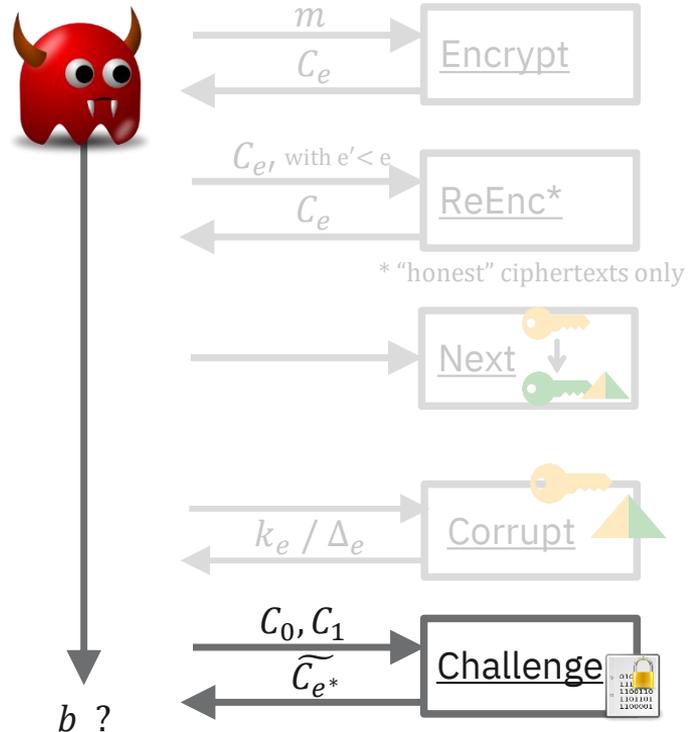
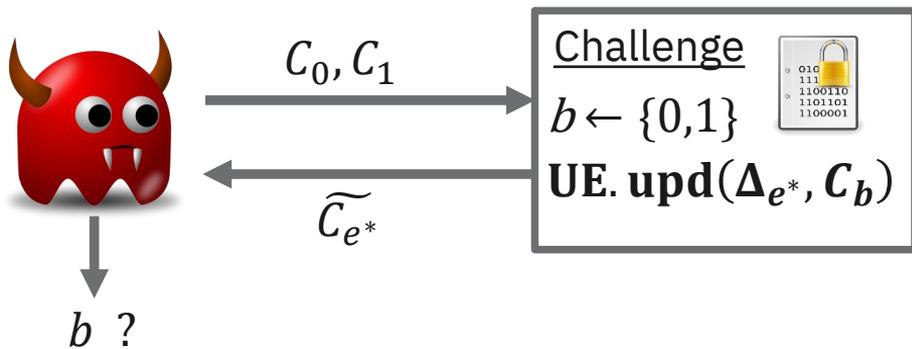


- No security after full breach – inference attacks through linkability



# Updatable Encryption | IND-UPD

- IND-UPD definition = Update Indistinguishability
  - Unlinkability of updated ciphertexts – no leakage through correlation attacks



IND-ENC = Secure Updatable Encryption

IND-ENC + IND-UPD = Strongly Secure Updatable Encryption

But much more expensive (for large ciphertexts)

# Updatable Encryption | (In)Secure Schemes

Re-Randomizable Ciphertext-Independent Symmetric ElGamal

	2ENC (folklore)	XOR-KEM (EPRS17)	BLMR (BLMR13)	RISE (LT18)
Enc	$Enc(k_e^o, Enc(k^i, m))$	$(k_e \oplus x), Enc(x, m)$	$PRF(k_e, N) \otimes m, N$	
Tok $\Delta_{e+1}$	$(k_e^o, k_{e+1}^o)$	$k_e \oplus k_{e+1}$	$k_e \oplus k_{e+1}$	
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 ReCrypt [EPRS17] = ciphertext-*dependent* UE

Known instantiations either DL or lattice-based

# Updatable Encryption | Secure Construction (RISE)

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

RISE.enc( $k_e, m$ ): parse  $k_e = (x, y)$ ,  $r \xleftarrow{r} \mathbb{Z}_q$ , 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  $m' \leftarrow C_2 \cdot C_1^{-1/x}$

# Updatable Encryption | Secure Construction (RISE)

Token doesn't leak info about secret key  
(but allows bidirectional key & ciphertext updates)

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

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

RISE.enc( $k_e, m$ ): parse  $k_e = (x, y)$ ,  $r \xleftarrow{r} \mathbb{Z}_q$ , 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( $\Delta_{e+1}, C_e$ ): parse  $\Delta_{e+1} = (\Delta, y')$  and  $C_e = (C_1, C_2)$ ,  
 $C'_1 \leftarrow C_1^\Delta$  return  $C_{e+1} \leftarrow (C'_1, C_2)$

Re-randomization  $\rightarrow$  updated ciphertexts are unlinkable  
(fresh & updated ones are indistinguishable)

# Updatable Encryption | Efficiency & Comparison

Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	Enc&MAC (KLR19)	NY&GS (KLR19)
IND-ENC	Green	Green	Grey	Grey
IND-UPD	Red	Green	Grey	Grey

Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
Green
Green

Encrypt	2 exp	2 exp	Grey	Grey
Token	2 exp	1 exp	Grey	Grey
ReEnc	2n exp	2.5n exp	Grey	Grey

2 exp
2n exp
2n exp

# Updatable Encryption | Efficiency & Comparison

## Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	Enc&MAC (KLR19)	NY&GS (KLR19)
IND-ENC	CPA	CPA		
IND-UPD		CPA		

## Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
CPA
CPA

Encrypt	2 exp	2 exp		
Token	2 exp	1 exp		
ReEnc	2n exp	2.5n exp		

2 exp
2n exp
2n exp

# Updatable Encryption | Efficiency & Comparison

Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	<b>Enc&amp;MAC (KLR19)</b>	<b>NY&amp;GS (KLR19)</b>
IND-ENC	CPA	CPA	CCA	
IND-UPD		CPA	CCA	
Integrity			CTXT	

Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
CPA
CPA
CTXT

Encrypt	2 exp	2 exp		
Token	2 exp	1 exp		
ReEnc	2n exp	2.5n exp		

2 exp
2n exp
2n exp

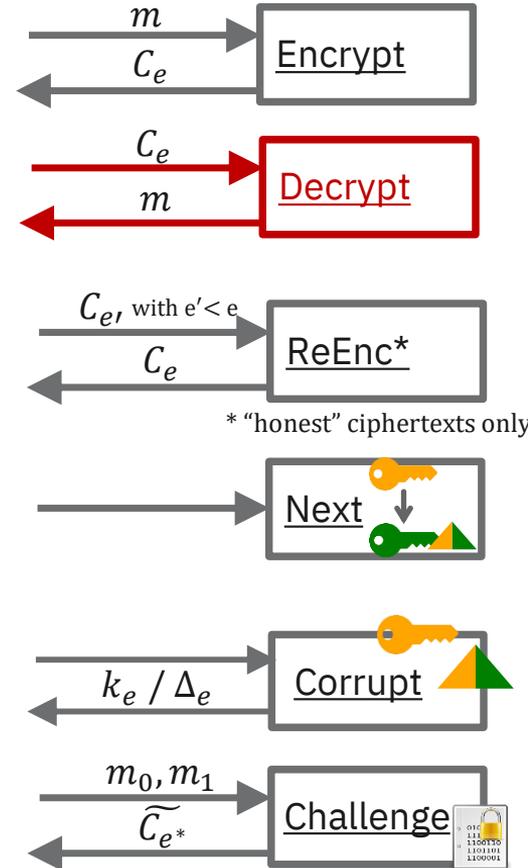
# CCA Security | How to add Dec Oracle?

## IND-ENC-CCA

- Decryptions of challenge ciphertext must be prevented
  - e.g., RISE has re-randomizable ciphertexts
- Same trick as in ReEnc? → makes Dec oracle obsolete
- Use idea of EPRS17:
  - Deterministic re-encryption  $\text{UE.upd}(\Delta_{e+1}, C_e) \rightarrow C_{e+1}$
  - Unique challenge ciphertext  $\widetilde{C}_e$  in each epoch
  - Dec takes all ciphertexts except  $\widetilde{C}_e$



$b ?$



# Ciphertext-Integrity

## INT-CTXT

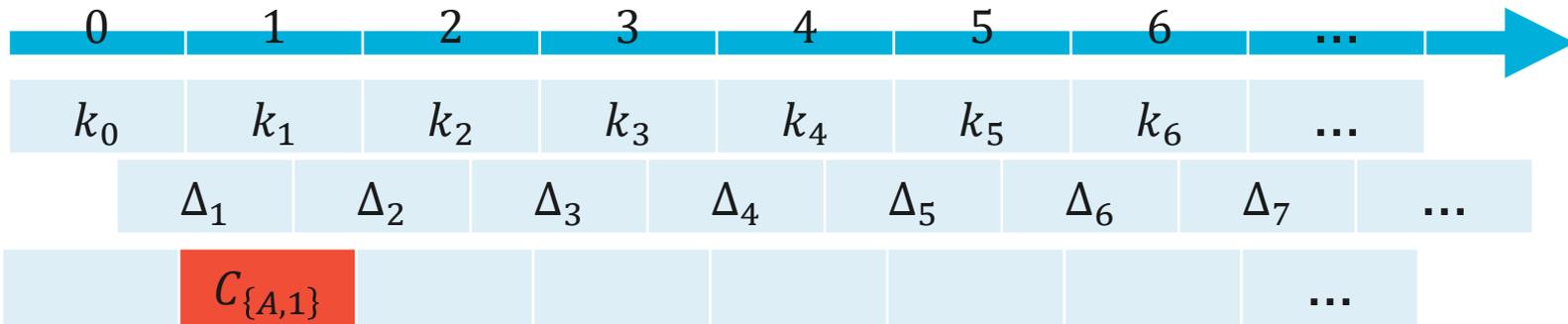
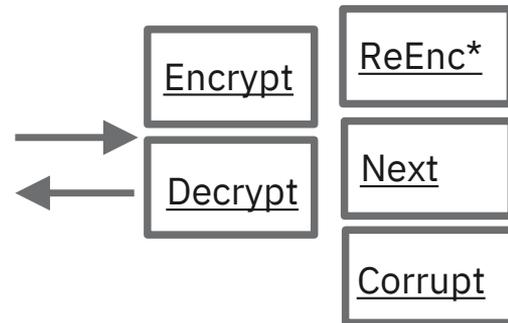
- Adversary must produce valid & non-trivial ciphertext

$$C_e^* \text{ s.t. } \text{UE.dec}(C_e^*, k_e) \neq \perp$$

- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



$C_e^*$



# Ciphertext-Integrity

## INT-CTXT

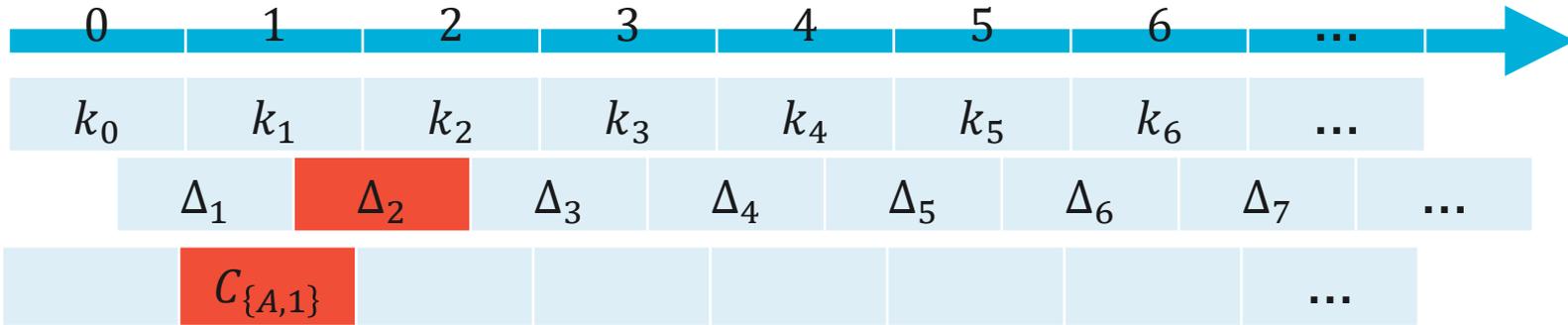
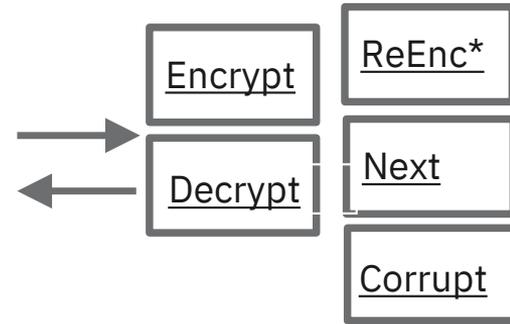
- Adversary must produce valid & non-trivial ciphertext

$$C_e^* \text{ s.t. } \text{UE.dec}(C_e^*, k_e) \neq \perp$$

- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



$C_e^*$



# Ciphertext-Integrity

## INT-CTXT

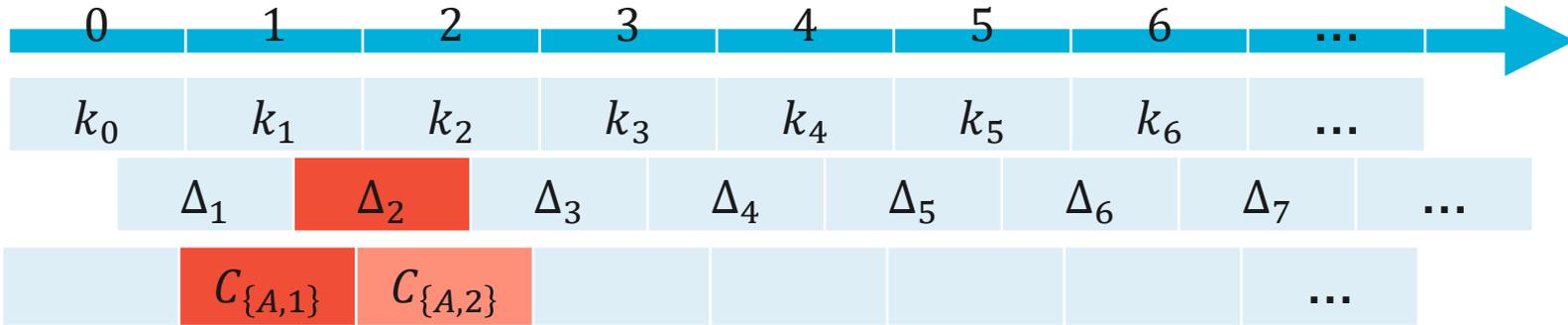
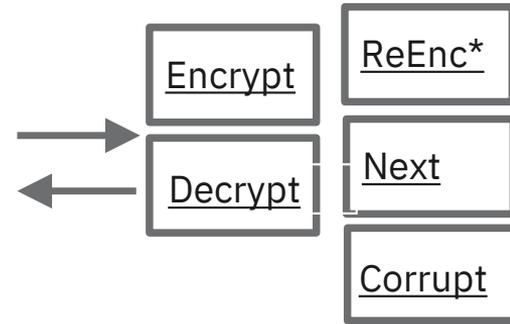
- Adversary must produce valid & non-trivial ciphertext

$$C_e^* \text{ s.t. } \text{UE.dec}(C_e^*, k_e) \neq \perp$$

- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



$C_e^*$



# Ciphertext-Integrity

## INT-CTXT

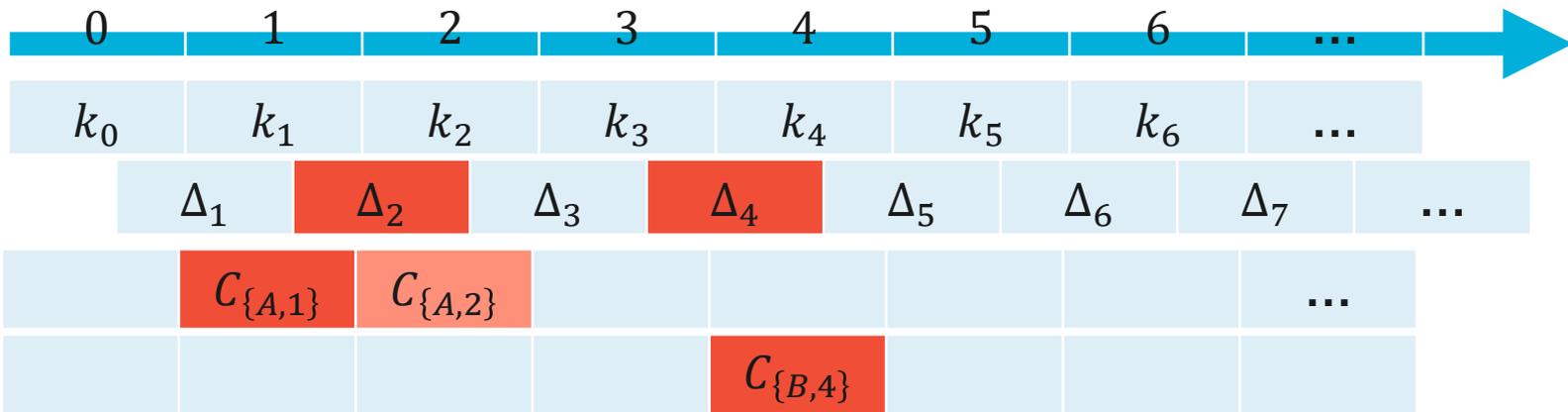
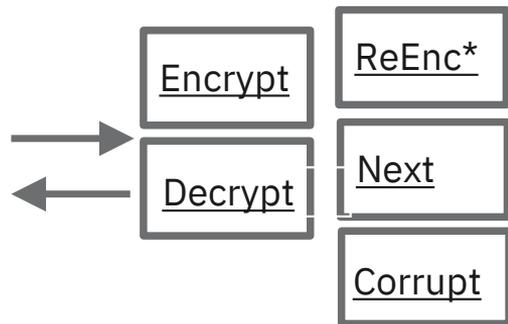
- Adversary must produce valid & non-trivial ciphertext

$$C_e^* \text{ s.t. } \text{UE.dec}(C_e^*, k_e) \neq \perp$$

- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



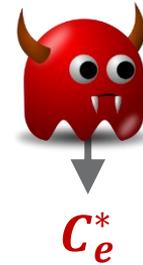
$C_e^*$



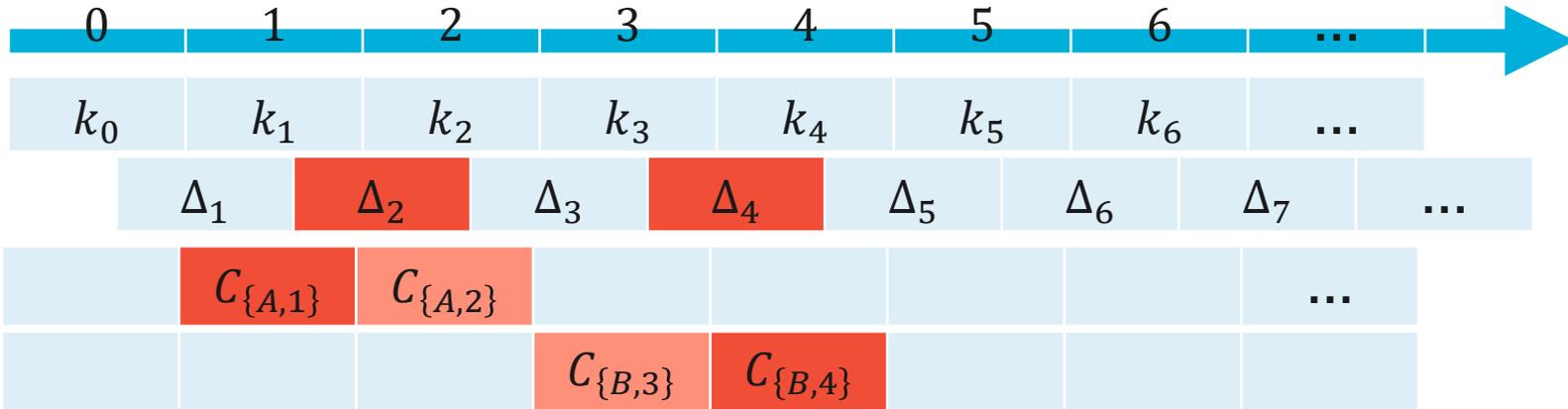
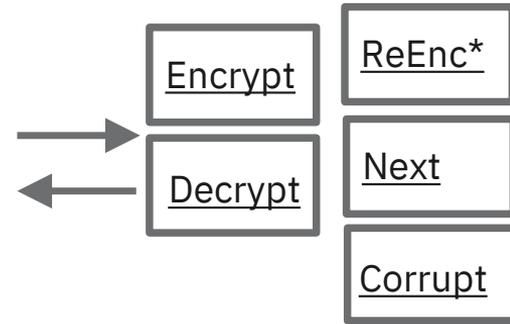
# Ciphertext-Integrity

## INT-CTXT

- Adversary must produce valid & non-trivial ciphertext  $C_e^*$  s.t.  $\text{UE.dec}(C_e^*, k_e) \neq \perp$
- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



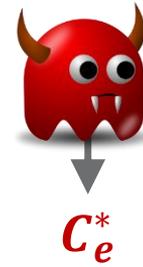
$C_e^*$



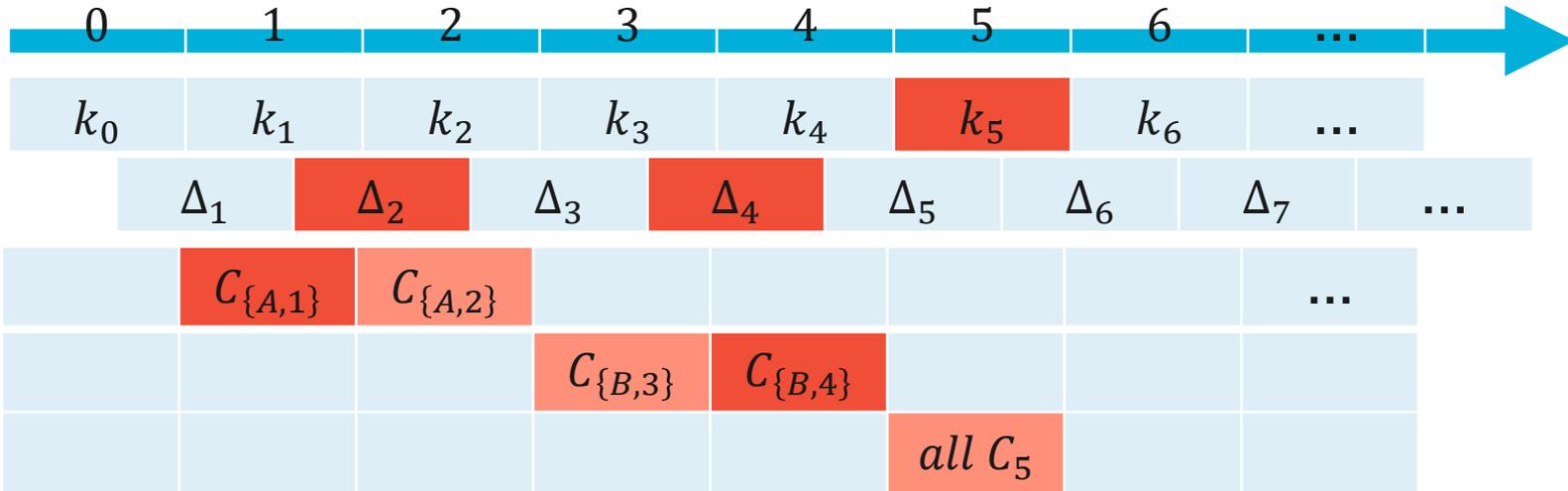
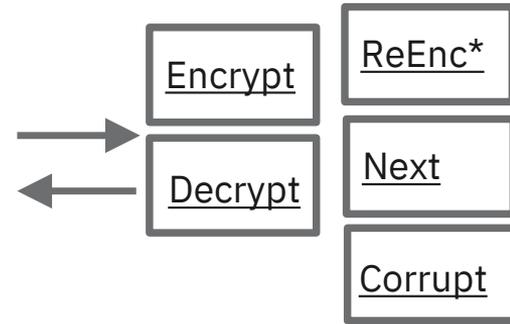
# Ciphertext-Integrity

## INT-CTXT

- Adversary must produce valid & non-trivial ciphertext  $C_e^*$  s.t.  $\text{UE.dec}(C_e^*, k_e) \neq \perp$
- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



$C_e^*$



# Ciphertext-Integrity

## INT-CTXT

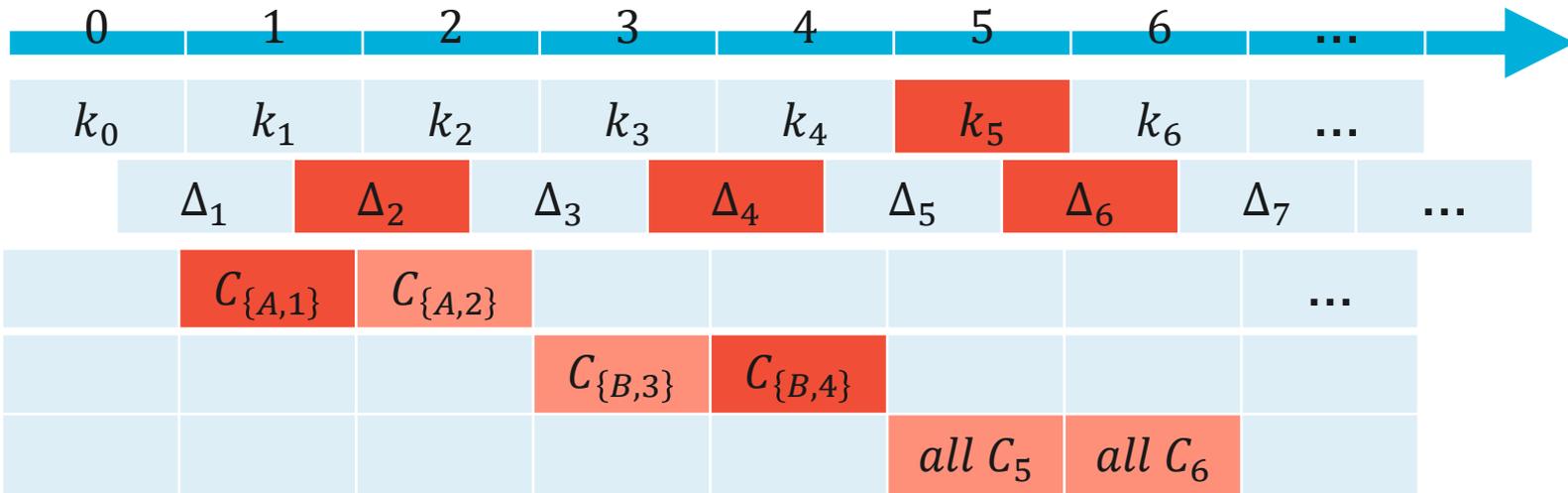
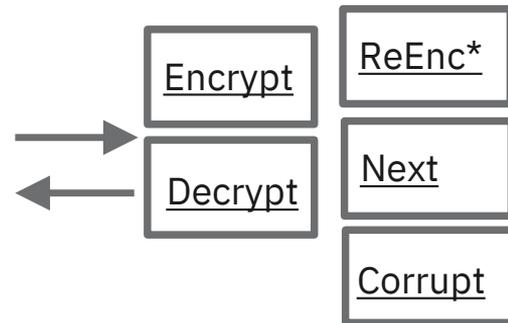
- Adversary must produce valid & non-trivial ciphertext

$$C_e^* \text{ s.t. } \text{UE.dec}(C_e^*, k_e) \neq \perp$$

- $C_e^*$  must not be response from Enc/ReEnc or trivial re-encryption  
→ Deterministic ReEnc allows to keep track of trivial ciphertexts



$C_e^*$



# CCA & CTXT | High-Level Idea (Enc & MAC)

- (Somewhat) generic transformation of CPA-secure encryption and PRF
- SE needs some special properties: tidy, randomness recoverable, ...
  - $Enc(k_{SE}, m; r) \rightarrow C$
  - $Dec(k_{SE}, C) \rightarrow m, r$
- Encrypt  $m$ :  $Enc(k_{SE}, m; r) \rightarrow C$  and  $PRF(k_{PRF}, (m, r)) \rightarrow t$ . Return  $(C, t)$
- Decrypt  $(C, t)$ :  $Dec(k_{SE}, C) \rightarrow (m', r')$  and  $PRF(k_{PRF}, (m', r')) \rightarrow t'$ . Return  $m'$  if  $t = t'$
- Update: update  $(C, t)$  using key-rotatable building blocks

# CCA & CTXT | Building Blocks: Updatable PRF

- Updatable PRF – based on DDH-PRF by NPR99
- Group  $(G, g, q)$  in which DDH assumption holds, hash function  $H: \{0,1\}^* \rightarrow G$ 
  - KeyGen:  $k \in Z_q^*$
  - Eval:  $t = H(m)^k$
- TokenGen for old key  $k$  new key  $k'$ :  $\Delta = k' / k$
- Update:  $t' = t^\Delta = H(m)^{k'}$

**Standard PRF security** (under DDH & RO)

**Simulatable Token Generation**

# CCA & CTXT | Building Blocks: Updatable SE

- Updatable (CPA) Encryption Scheme – ElGamal based
- Group  $(G, g, q)$  in which DDH assumption holds

– KeyGen:  $x_1, x_2 \in Z_q^*$ .  $k_{SE} = (x_1, x_2)$

**Re-encryption = decrypt-then-encrypt:**  
 $UE.upd(\Delta, C) = UE.enc(k', UE.dec(k, C))$

– Encrypt:  $g^r \in G$ .  $C_1 = g^{rx_1}$  and  $C_2 = g^{rx_2} \cdot m$

– Decrypt:  $g^r = C_1^{-x_1}$  and  $m = C_2 / g^{rx_2}$

– TokenGen for old key  $(x_1, x_2)$ , new key  $(x'_1, x'_2)$ :  $\Delta_1 = \frac{x'_1}{x_1}$  and  $\Delta_2 = \frac{x'_2 - x_2}{x_1}$

– Update:  $C'_1 = C_1^{\Delta_1} = g^{rx'_1}$  and  $C'_2 = C_1^{\Delta_2} \cdot C_2 = g^{rx'_2} \cdot m$

**Simulatable Token**

**Deterministic Re-encryption & entire ciphertext is updated**

# CCA & CTXT | Efficiency & Comparison

Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	<b>Enc&amp;MAC (KLR19)</b>	NY&GS (KLR19)
IND-ENC	CPA	CPA	CCA	
IND-UPD		CPA	CCA	
Integrity			CTXT	

Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
CPA
CPA
CTXT

- Encrypt  $m$ :  $Enc(k_{SE}, m; r) \rightarrow C$  and  $PRF(k_{PRF}, (m, r)) \rightarrow t$ . Return  $(C, t)$
- Decrypt  $(C, t)$ :  $Dec(k_{SE}, C) \rightarrow (m', r')$  and  $PRF(k_{PRF}, (m', r')) \rightarrow t'$ . Return  $m'$  if  $t = t'$
- TokenGen: get SE.  $TokGen(k_{SE}, k'_{SE}) \rightarrow \Delta_{SE}$  and PRF.  $TokGen(k_{PRF}, k'_{PRF}) \rightarrow \Delta_{PRF}$
- Update: update  $(C, t)$  using SE.  $Upd(\Delta_{SE}, C) \rightarrow C'$  and PRF.  $Upd(\Delta_{PRF}, t) \rightarrow t'$

# CCA & CTXT | Efficiency & Comparison

Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	<b>Enc&amp;MAC (KLR19)</b>	<b>NY&amp;GS (KLR19)</b>
IND-ENC	CPA	CPA	CCA	
IND-UPD		CPA	CCA	
Integrity			CTXT	

Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
CPA
CPA
CTXT

Encrypt	2 exp	2 exp	3 exp	
Token	2 exp	1 exp	3 exp	
ReEnc	2n exp	2.5n exp	3n exp	

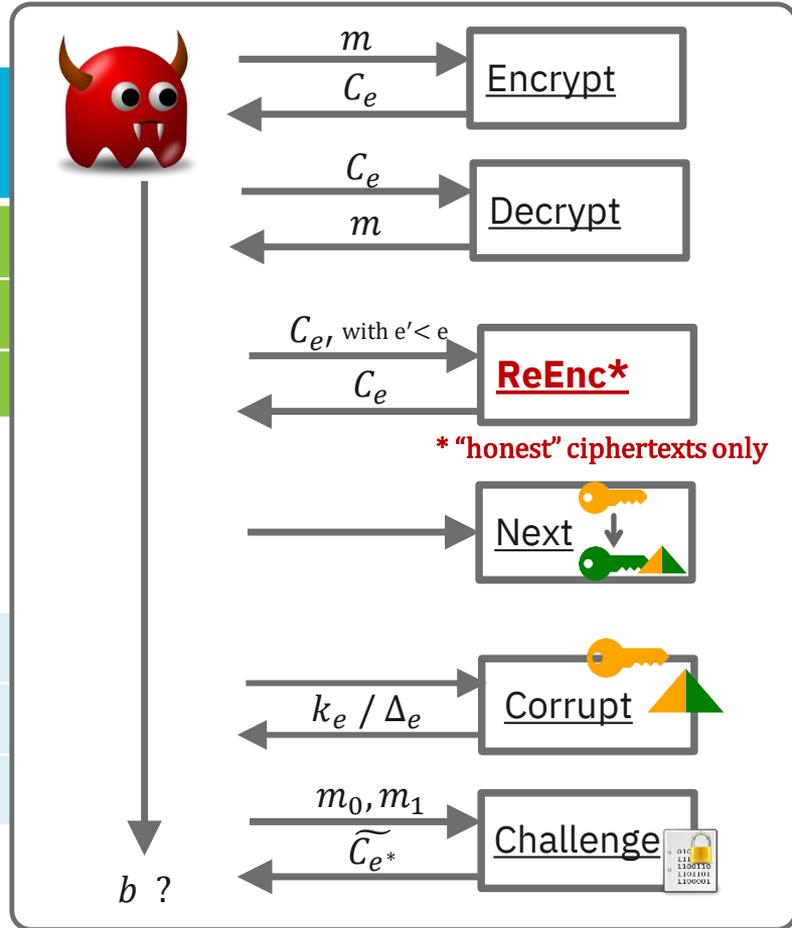
2 exp
2n exp
2n exp

# CCA & CTXT | Efficiency & Comparison

Ciphertext-Independent

	BLMR (BLMR13)	RISE (LT18)	Enc&MAC (KLR19)
IND-ENC	CPA	CPA	CCA
IND-UPD		CPA	CCA
Integrity			CTXT

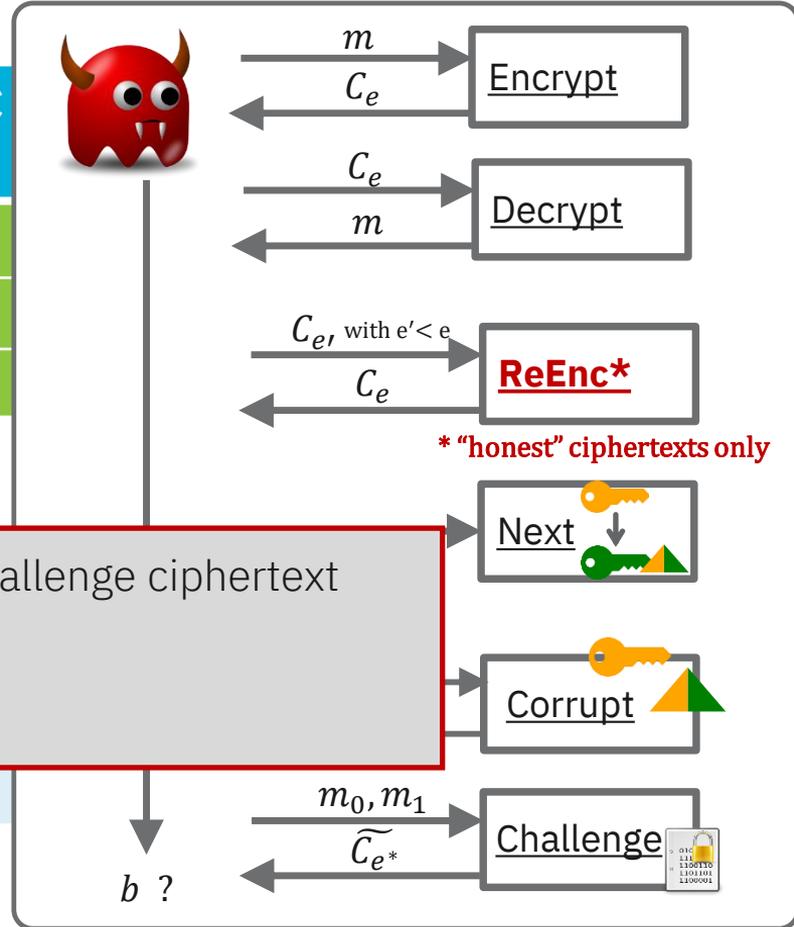
Encrypt	2 exp	2 exp	3 exp
Token	2 exp	1 exp	3 exp
ReEnc	2n exp	2.5n exp	3n exp



# CCA & CTXT | Efficiency & Comparison

Ciphertext-Independent

	BLMR (BLMR13)	RISE (LT18)	Enc&MAC (KLR19)
IND-ENC	CPA	CPA	CCA
IND-UPD		CPA	CCA
Integrity			CTXT



- ReEnc marks epochs as challenge-equal when  $C$  is challenge ciphertext
- RISE and Enc&MAC can “blind” query to ReEnc oracle
  - Submit invalid ciphertext  $\rightarrow$  unblind later

ReEnc	2n exp	2.5n exp	3n exp
-------	--------	----------	--------

# CCA & CTXT | Efficiency & Comparison

Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	<b>Enc&amp;MAC (KLR19)</b>	<b>NY&amp;GS (KLR19)</b>
IND-ENC	CPA	CPA	CCA	
IND-UPD		CPA	CCA	
Integrity			CTXT	
Arb. ReEnc				

Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
CPA
CPA
CTXT
partially

Encrypt	2 exp	2 exp	3 exp	
Token	2 exp	1 exp	3 exp	
ReEnc	2n exp	2.5n exp	3n exp	

2 exp
2n exp
2n exp

# Arb. ReEnc Security | High-Level Idea

- Make ciphertext validity publicly verifiable  $\rightarrow$  Naor-Yung transform (CPA  $\rightarrow$  CCA)
    - $Enc(k_1, m) \rightarrow C_1$  and  $Enc(k_2, m) \rightarrow C_2$
    - $NIZK\{C_1 = Enc(k_1, m) \wedge Enc(k_2, m)\} \rightarrow \pi$
    - *Ciphertext*:  $(C_1, C_2, \pi)$
  - Use CPA-secure Updatable Encryption: RISE
  - Use Malleable NIZK: Groth-Sahai (GS) proofs
  - Both building blocks create re-randomizable  $C / \pi \rightarrow$  CCA and CTXT is impossible
  - Achieve RCCA & PTXT instead ReEnc is not deterministic anymore
- /
- Dec oracle rejects if  $Dec(k, C) \rightarrow m_0$  or  $m_1$  (challenge plaintext)

# Updatable Encryption | Efficiency & Comparison

Ciphertext-Independent

	<b>BLMR (BLMR13)</b>	<b>RISE (LT18)</b>	<b>Enc&amp;MAC (KLR19)</b>	<b>NY&amp;GS (KLR19)</b>
IND-ENC	CPA	CPA	CCA	RCCA
IND-UPD		CPA	CCA	RCCA
Integrity			CTXT	PTXT
Arb. ReEnc				

deterministic ReEnc

Ciphertext-Dependent

<b>ReCrypt (EPRS17)</b>
CPA
CPA
CTXT
partially

deterministic ReEnc

Encrypt	2 exp	2 exp	3 exp	
Token	2 exp	1 exp	3 exp	<i>a lot...</i>
ReEnc	2n exp	2.5n exp	3n exp	

2 exp
2n exp
2n exp

# Summary

- Key rotation mandatory in many high-security environments
- Updatable Encryptions allow convenient updates of ciphertexts in untrusted domain
  - Ciphertext-dependent vs. Ciphertext-independent updates
- Lots of open problems:
  - CCA/CTXT with Arbitrary ReEnc Security ....w/o deterministic ReEnc
  - All schemes are bi-directional → are there uni-directional ones?
  - IND-UPD Definition implies PKE → weaker notions?

Thanks! Questions?

anj@zurich.ibm.com