The Triple Ratchet Protocol:

A Bandwidth Efficient Hybrid-Secure Signal Protocol

Yevgeniy Dodis

Shuichi Katsumata

Daniel Jost

Thomas Prest

Rolfe Schmidt

PQ SHIELD

NYU

AIST

Signal

The Signal Protocol

- ✓ Gold standard for **two-user** secure messaging.
- ✓ **Forward secrecy** and **post-compromise security**.
- ✓ Mainly relies on hardness of **Diffie-Hellman**.

Signal = X3DH + Double Ratchet

Signal Protocol

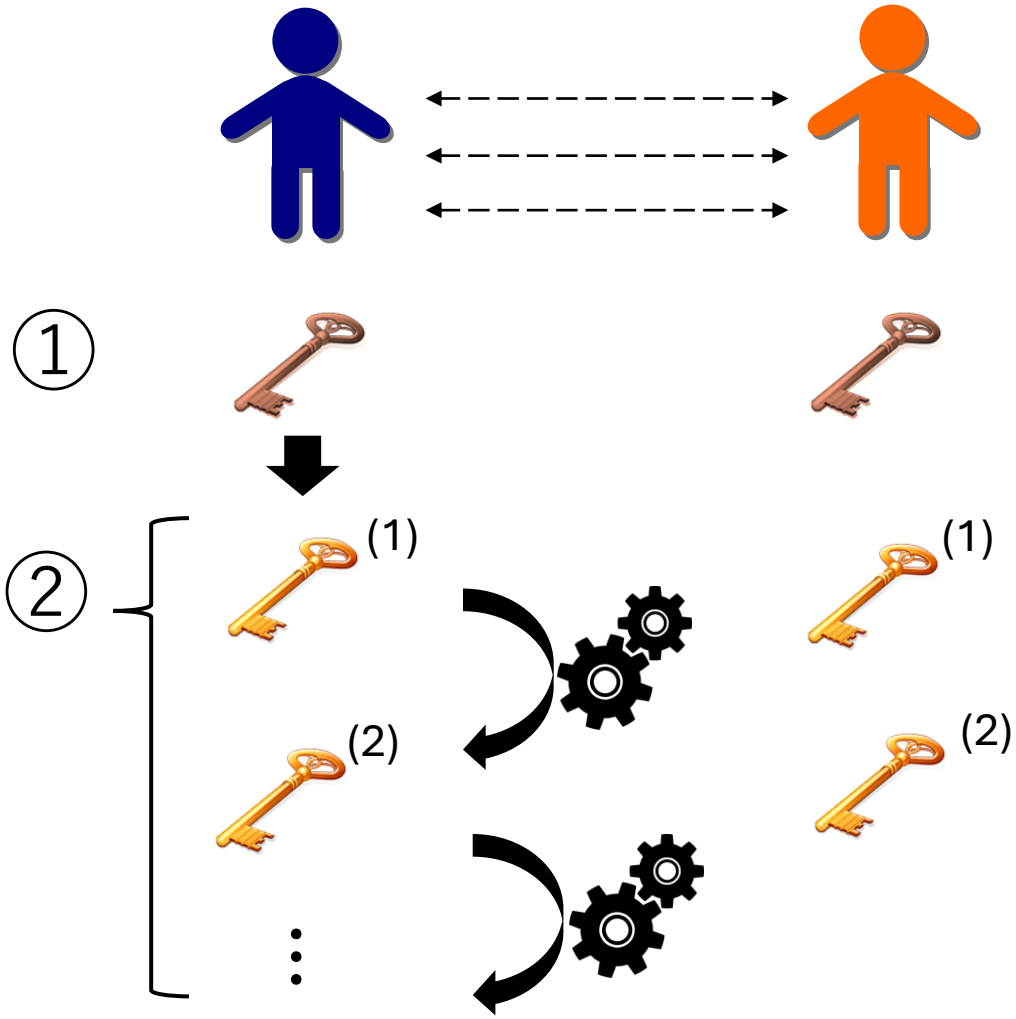
① X3DH

"Establish shared secret key"

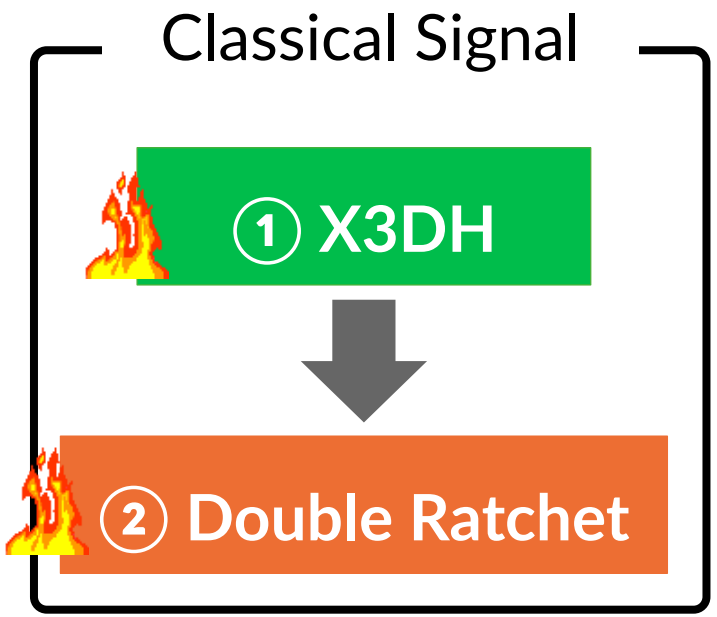
② Double Ratchet

"Encrypted communication
w/ continuous key updates"

➔ Strong PCS guarantees

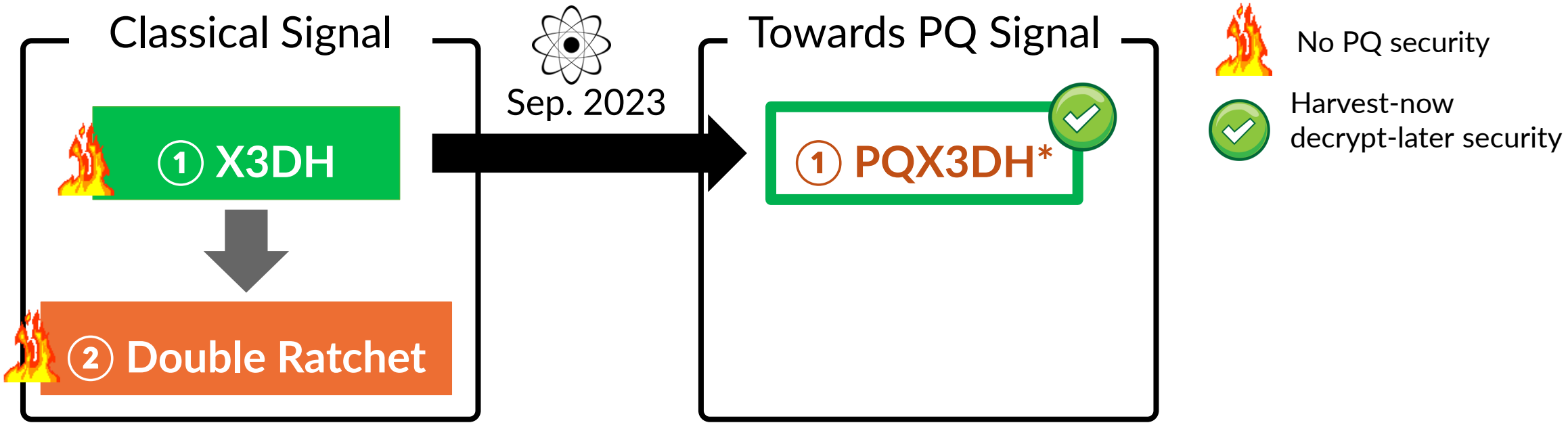


Transitioning to PQ Security

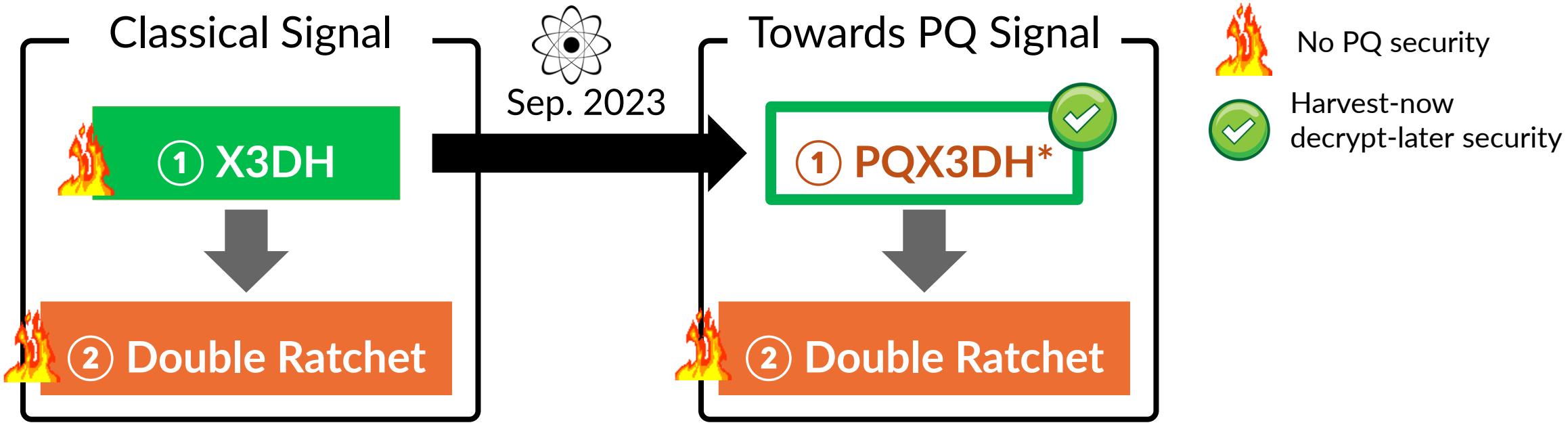



No PQ security

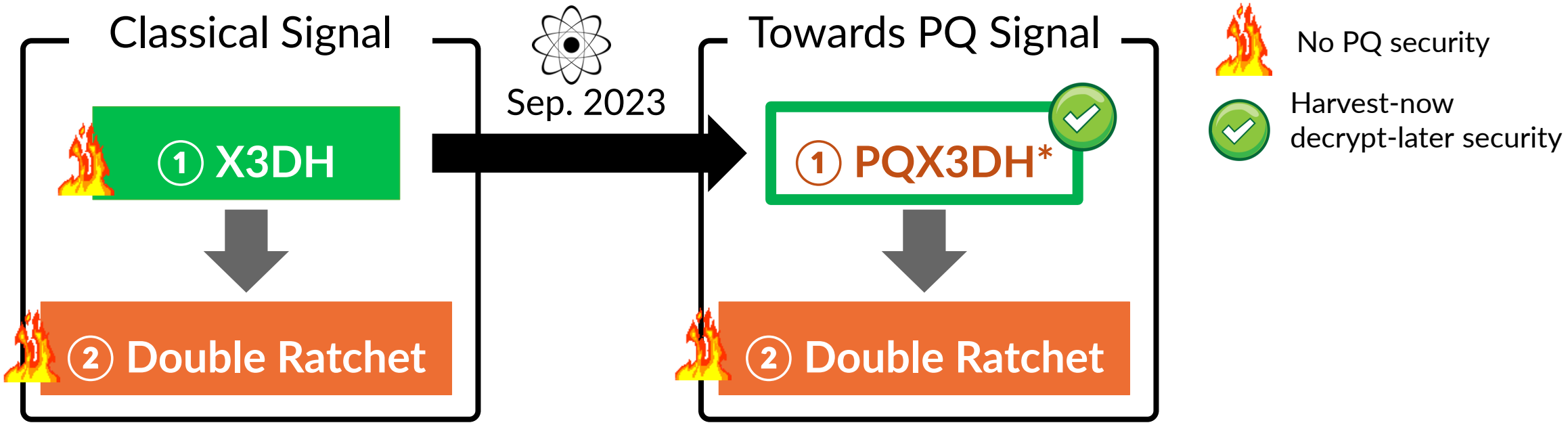
Transitioning to PQ Security



Transitioning to PQ Security



Transitioning to PQ Security



This Talk

How can we make an ***efficient*** PQ Double Ratchet?

The Double Ratchet Protocol

Signal's
Double Ratchet

=

Diffie-Hellman
(Public) Ratchet

*Guarantees PCS
⇨ *heals* state after compromise.

+

Symmetric Ratchet

*Guarantees FS
⇨ past messages remain secure when compromised.

The Double Ratchet Protocol

Signal's
Double Ratchet

=

Diffie-Hellman
(Public) Ratchet

*Guarantees PCS
⇨ *heals* state after compromise.

+

Symmetric Ratchet

*Guarantees FS
⇨ past messages remain secure when compromised.

The Diffie-Hellman Ratchet

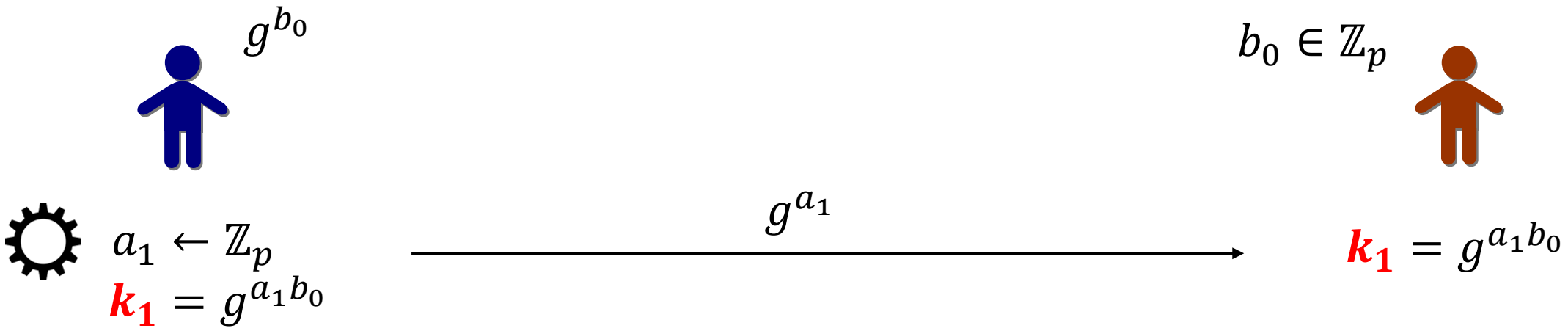
DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]

g^{b_0}

$b_0 \in \mathbb{Z}_p$

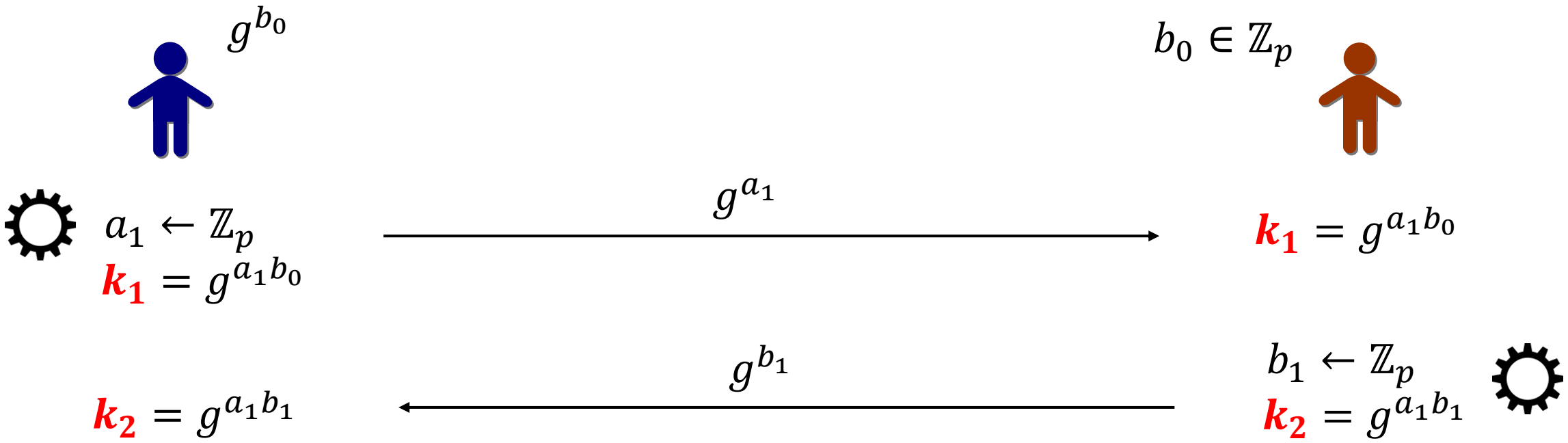
The Diffie-Hellman Ratchet

DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]



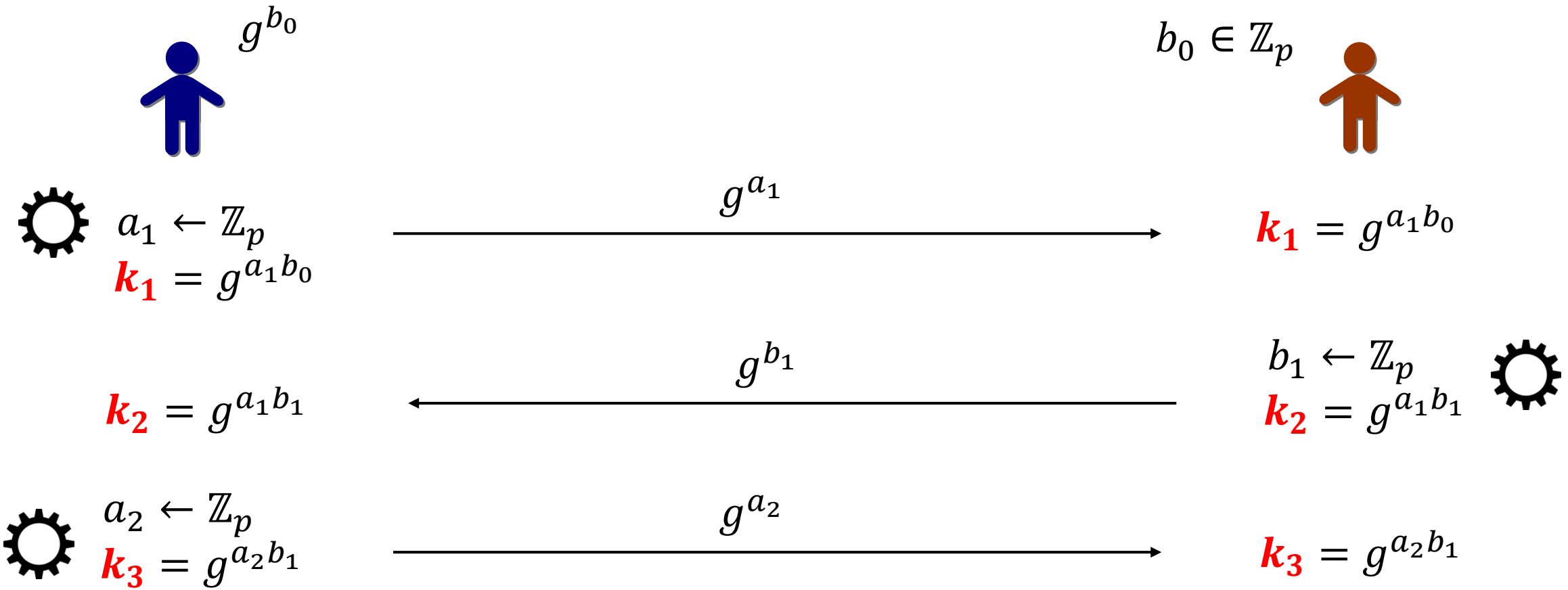
The Diffie-Hellman Ratchet

DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]



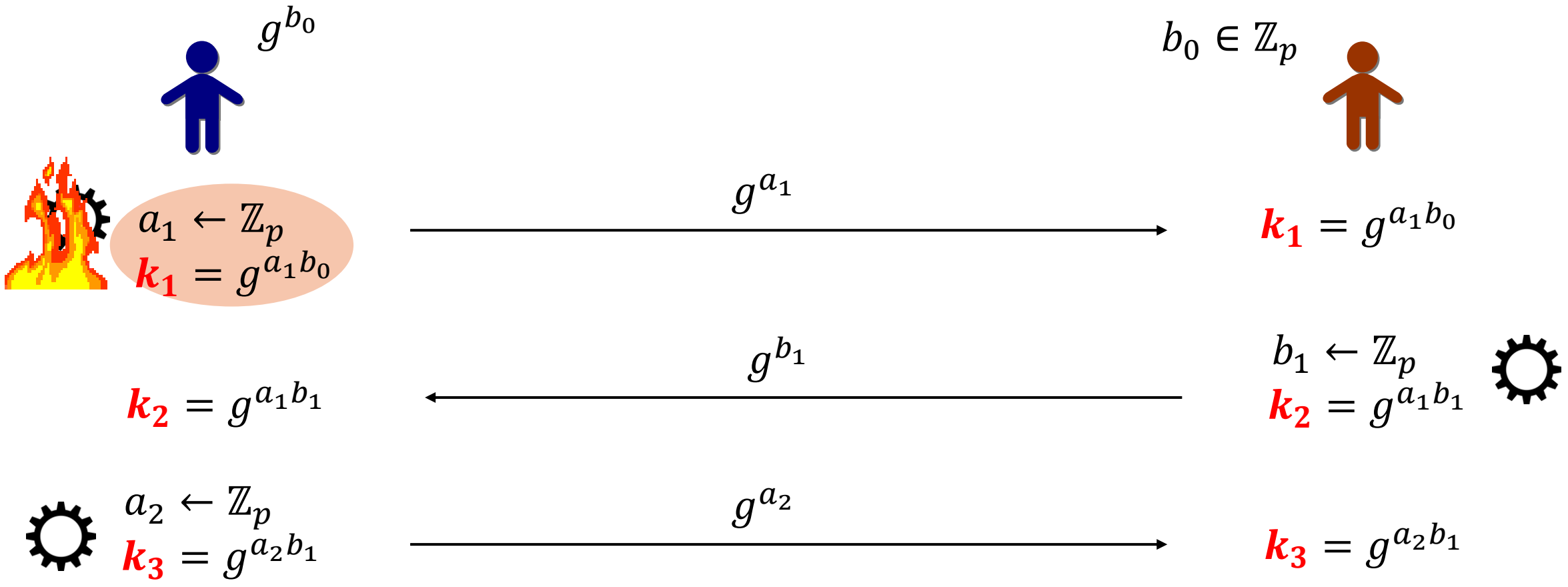
The Diffie-Hellman Ratchet

DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]



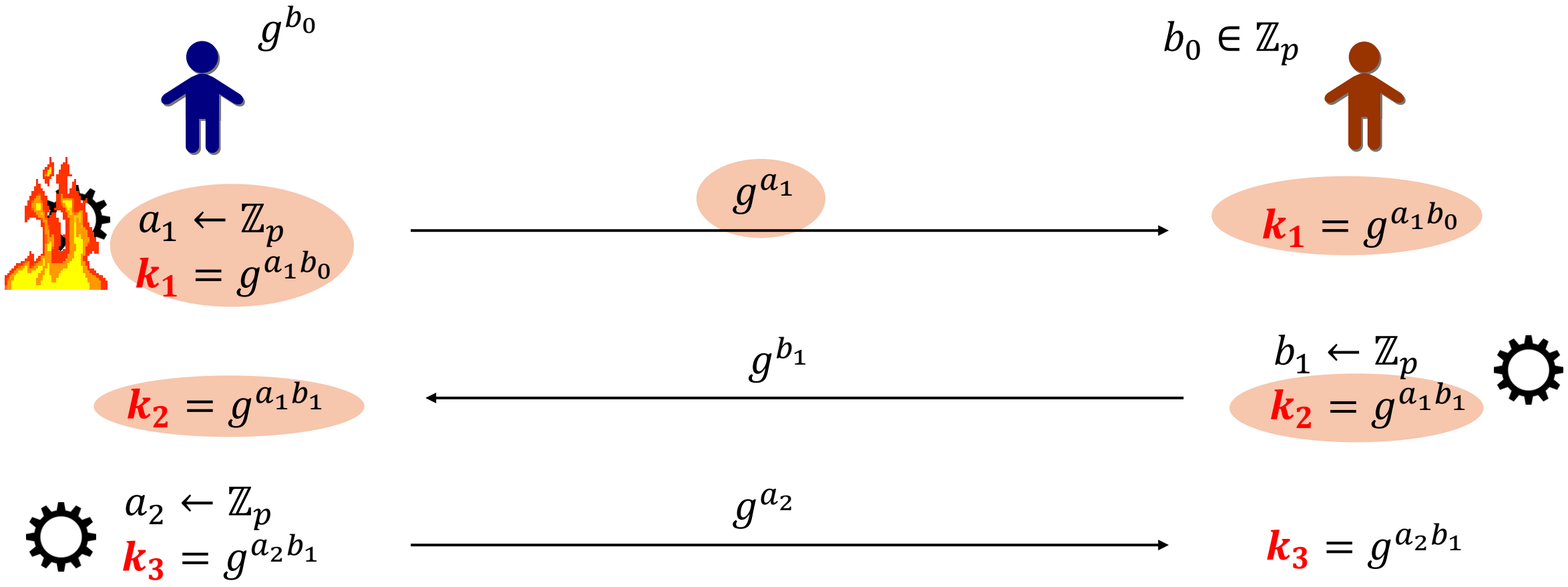
The Diffie-Hellman Ratchet

DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]



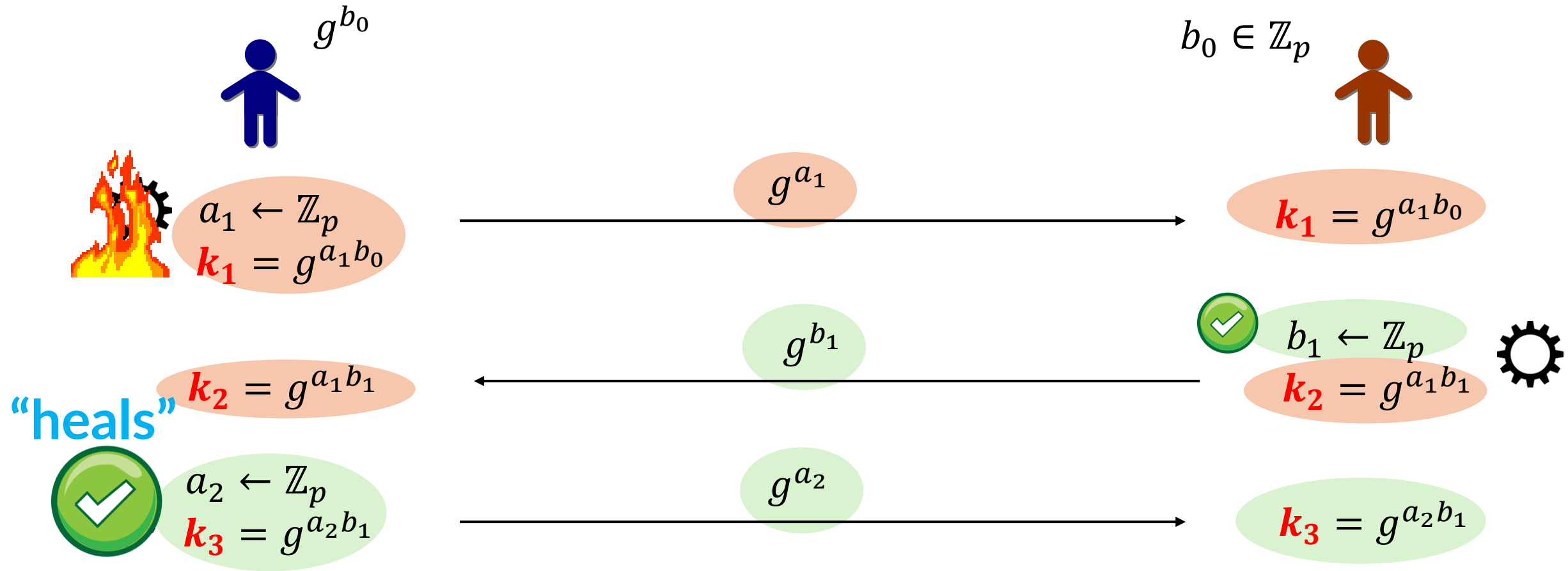
The Diffie-Hellman Ratchet

DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]



The Diffie-Hellman Ratchet

DH Ratchet is a *Continuous Key Agreement (CKA)* [EC:ACD19]

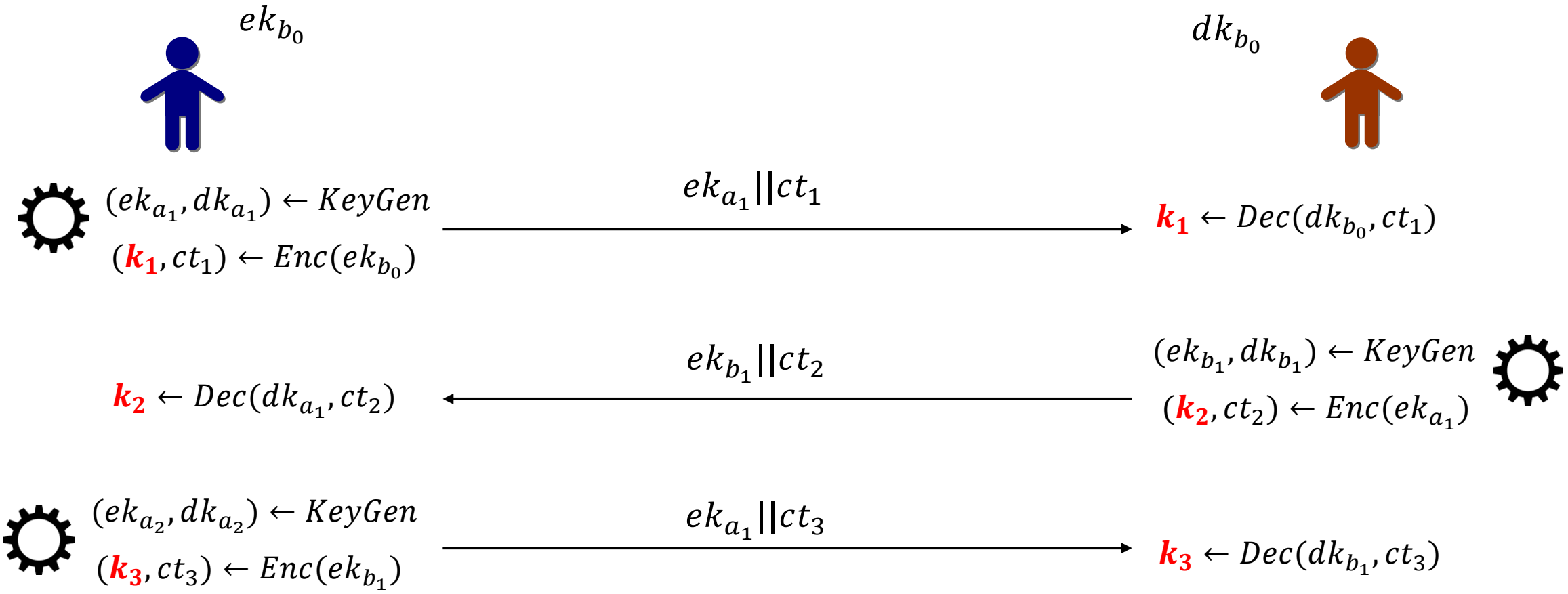


A Generic Ratchet

PQ Ratchet based on **any KEM** [EC:ACD19]

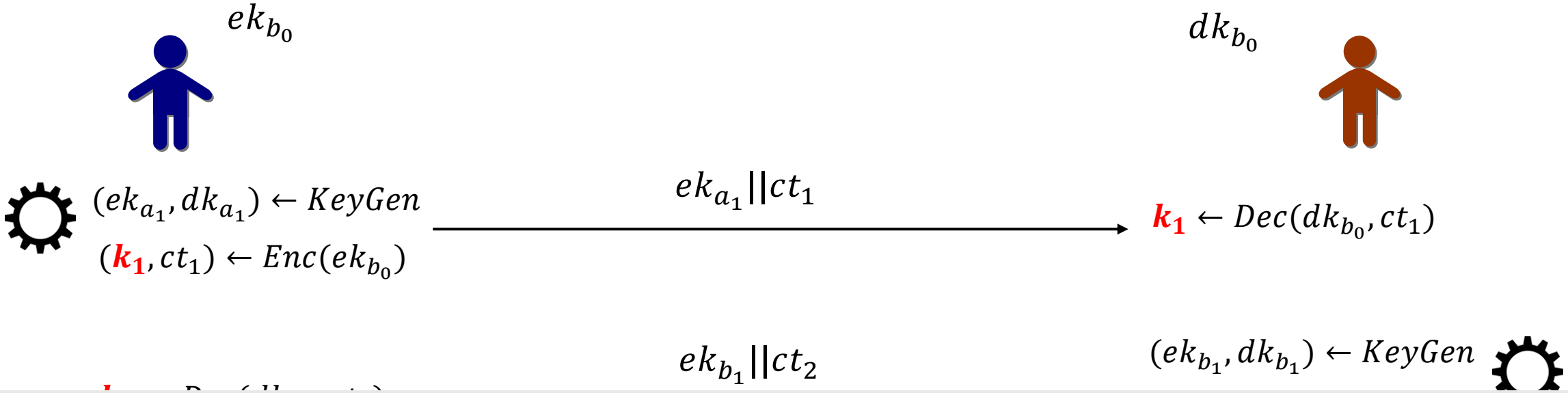
A Generic Ratchet

PQ Ratchet based on **any KEM** [EC:ACD19]



A Generic Ratchet

PQ Ratchet based on **any KEM** [EC:ACD19]



- One **PQ Ratchet is ≈ 2KB**, bigger than x70 DH Ratchet.
- To keep the impact on performance and bandwidth consumption minimal, **Signal aims to limit overhead to 40B per message**.

Recipe for Triple Ratchet

Step 1: Ratcheting KEM: **Katana** ⚔️

Better PQ Ratchet than running two KEMs in parallel.

Step 2: **Chunking** with error correcting codes

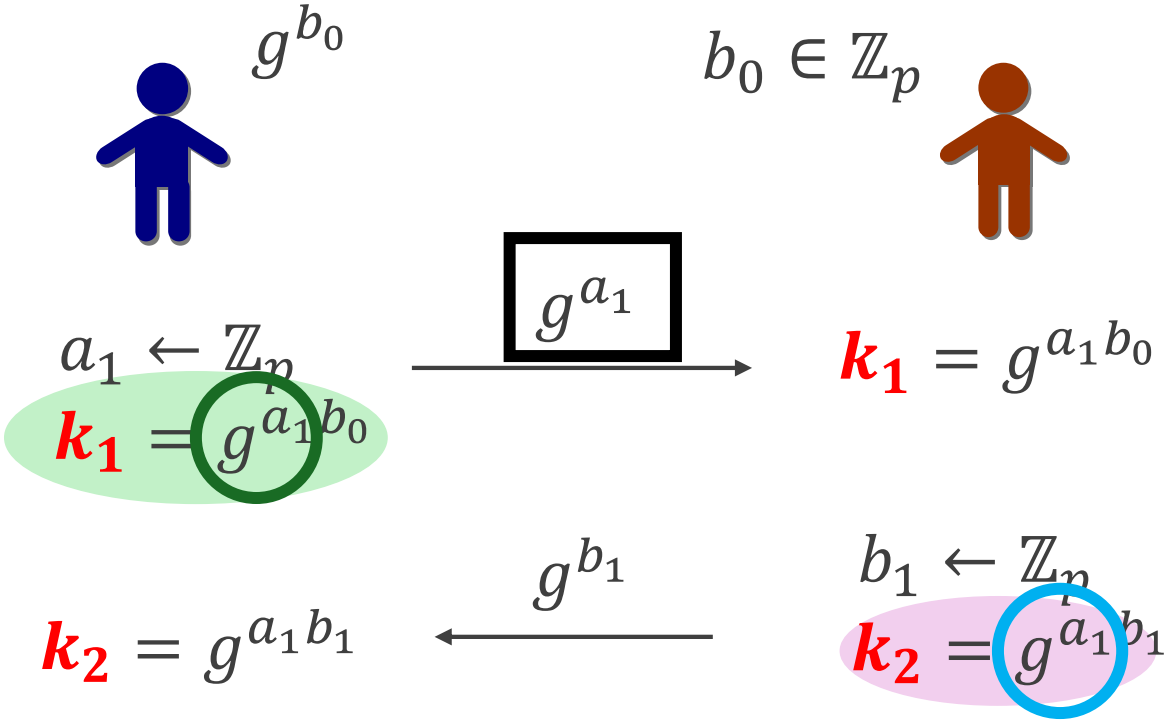
Forcing smaller bandwidth but with worse PCS security.

Step 3: **Hybrid Messaging**

Combining with DH-based ratchet for optimal security.

2. Our Efficient PQ Ratchet

Signal's DH Ratchet



g^{a_1} is **reused** for two purposes

- *Receiver* message of a DH KEX.
- *Sender* message of a DH KEX.

Goal: have a similar optimization for PQ-ratchet

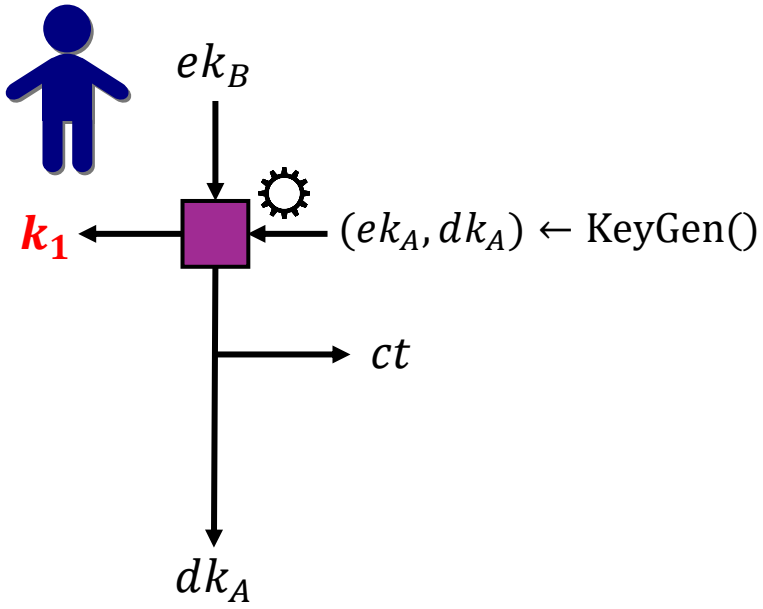
Ratcheting KEM

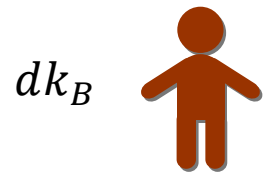
ek_B

dk_B

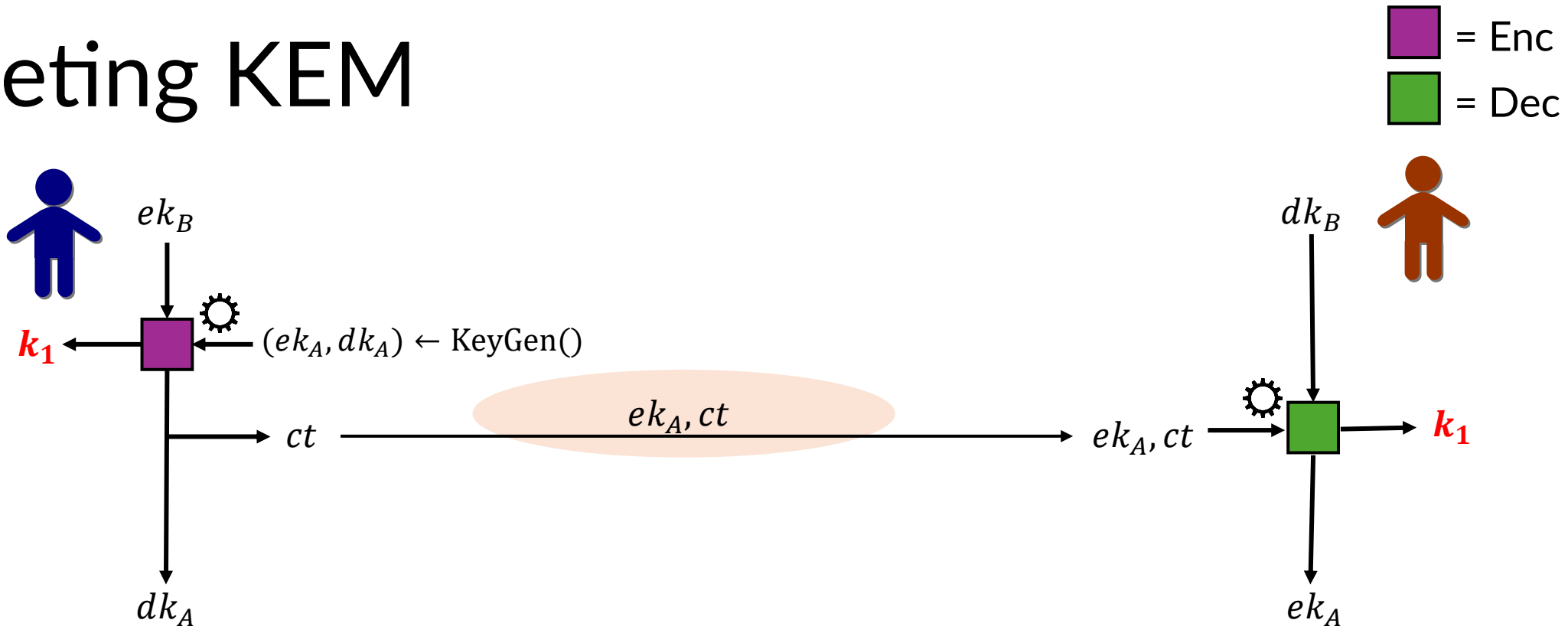
Ratcheting KEM

= Enc

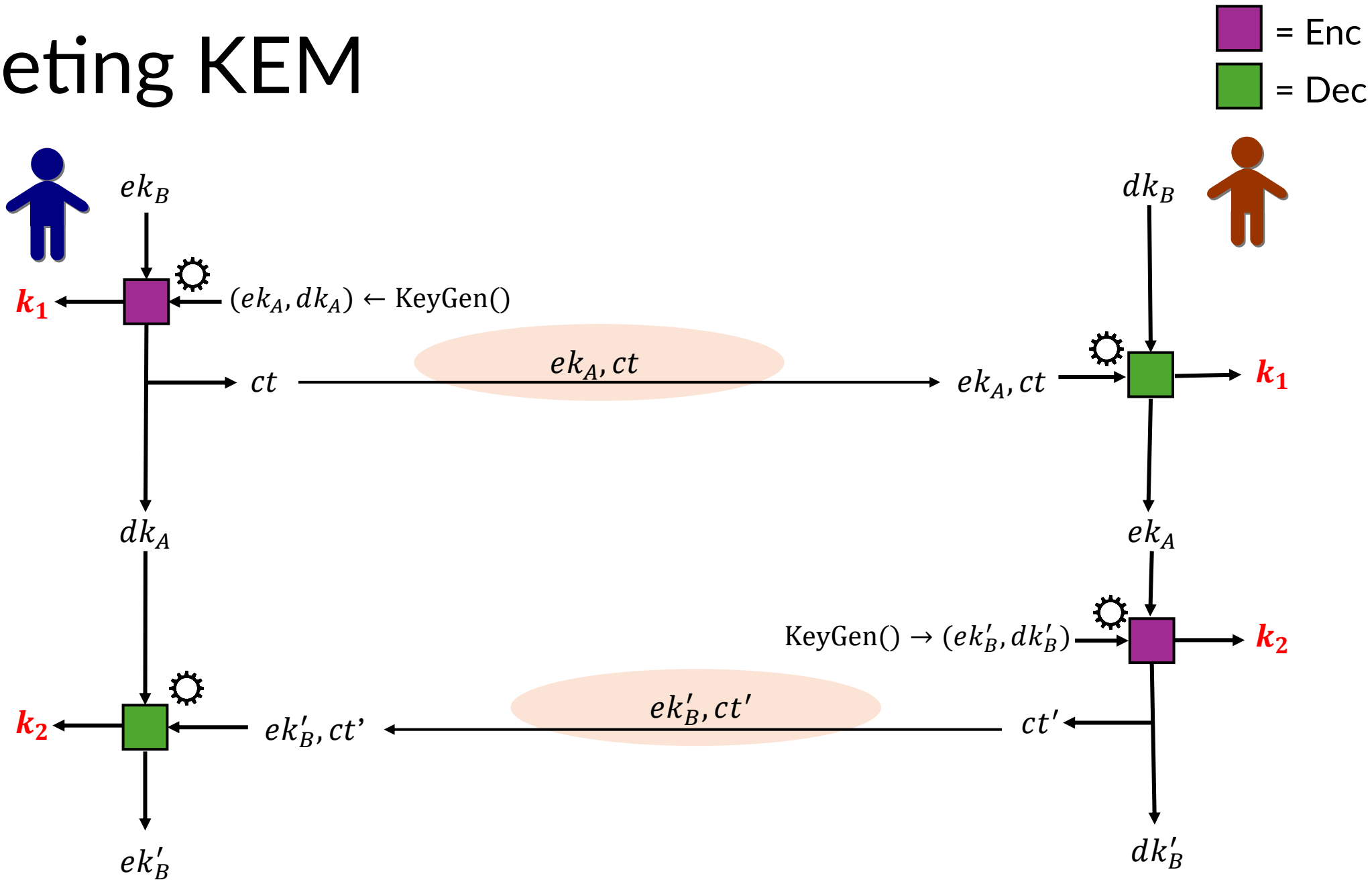




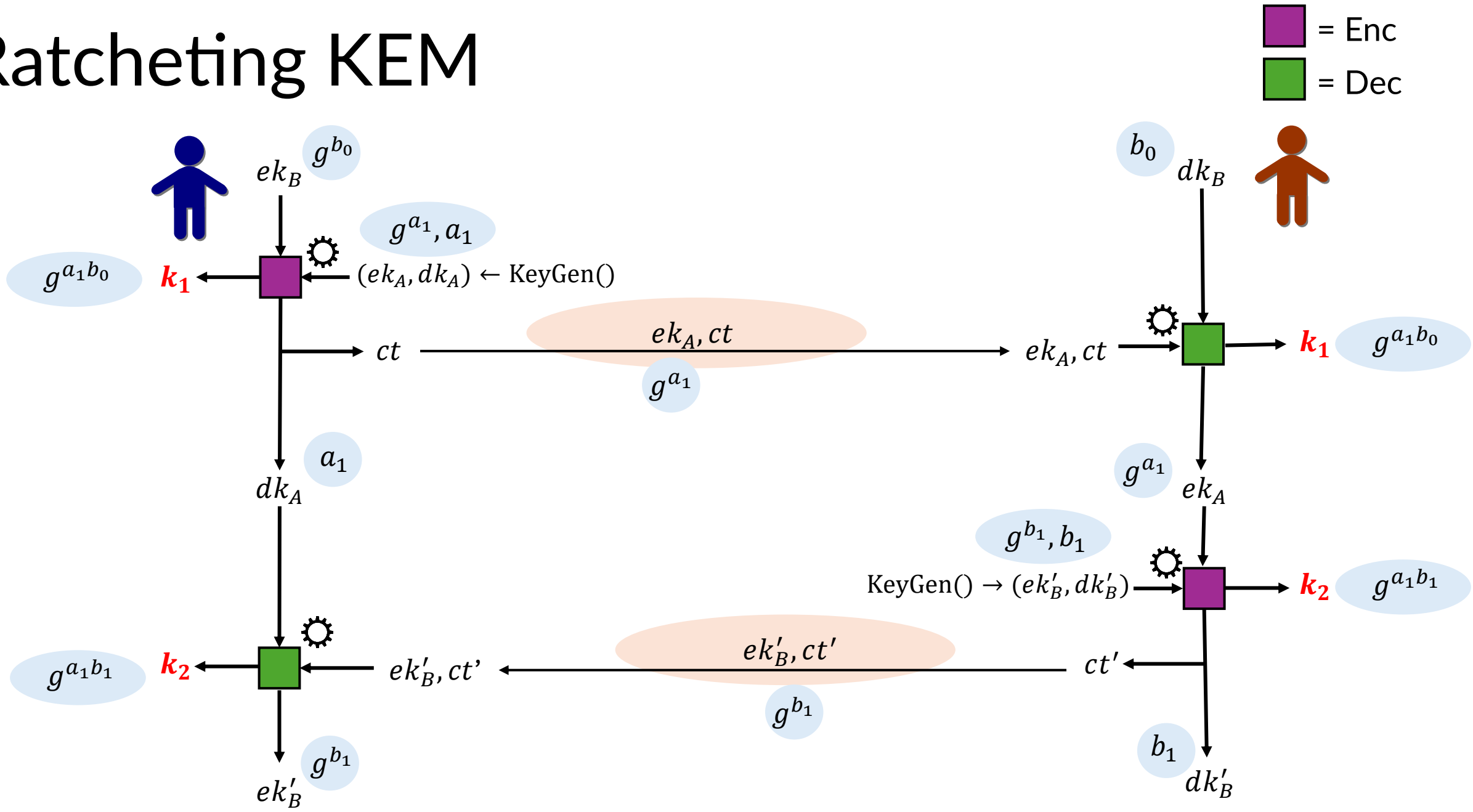
Ratcheting KEM



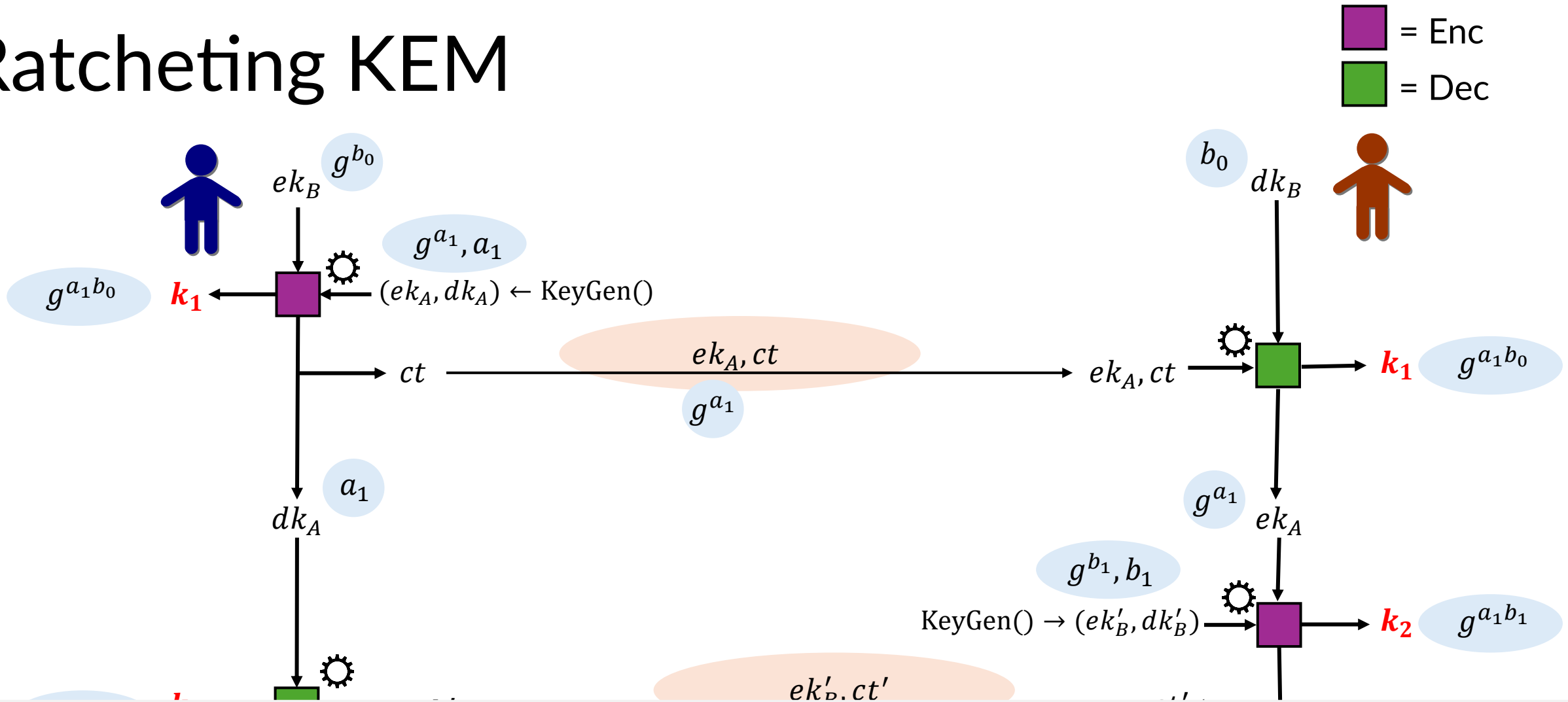
Ratcheting KEM



Ratcheting KEM



Ratcheting KEM



The **Double Ratchet** is a natural instantiation of Ratcheting-KEM with an **<u>empty ciphertext</u>**

Katana ⚔️

A Ratcheting KEM where ek can be reused as part of ct.

A common lattice-based KEM

$$ek = Ds + x$$
$$ct = (ct_0, ct_1) = (D^T r + z, \quad ek^T r + z + K[\frac{q}{2}])$$

Katana ⚔️

A Ratcheting KEM where ek can be reused as part of ct.

A common lattice-based KEM

$$ek = Ds + x$$

$$ct = (ct_0, ct_1) = (D^T r + z, \quad ek^T r + z + K[\frac{q}{2}])$$

Reconciliation
(→ Eliminate noise)

Noisy key exchange

Katana ⚔️

A Ratcheting KEM where ek can be reused as part of ct.

A common lattice-based KEM

$$ek = Ds + x$$

$$ct = (ct_0, ct_1) = (D^T r + z, \quad ek^T r + z + K[\frac{q}{2}])$$

Katana ⚔️

A Ratcheting KEM where ek can be reused as part of ct.

A common lattice-based KEM

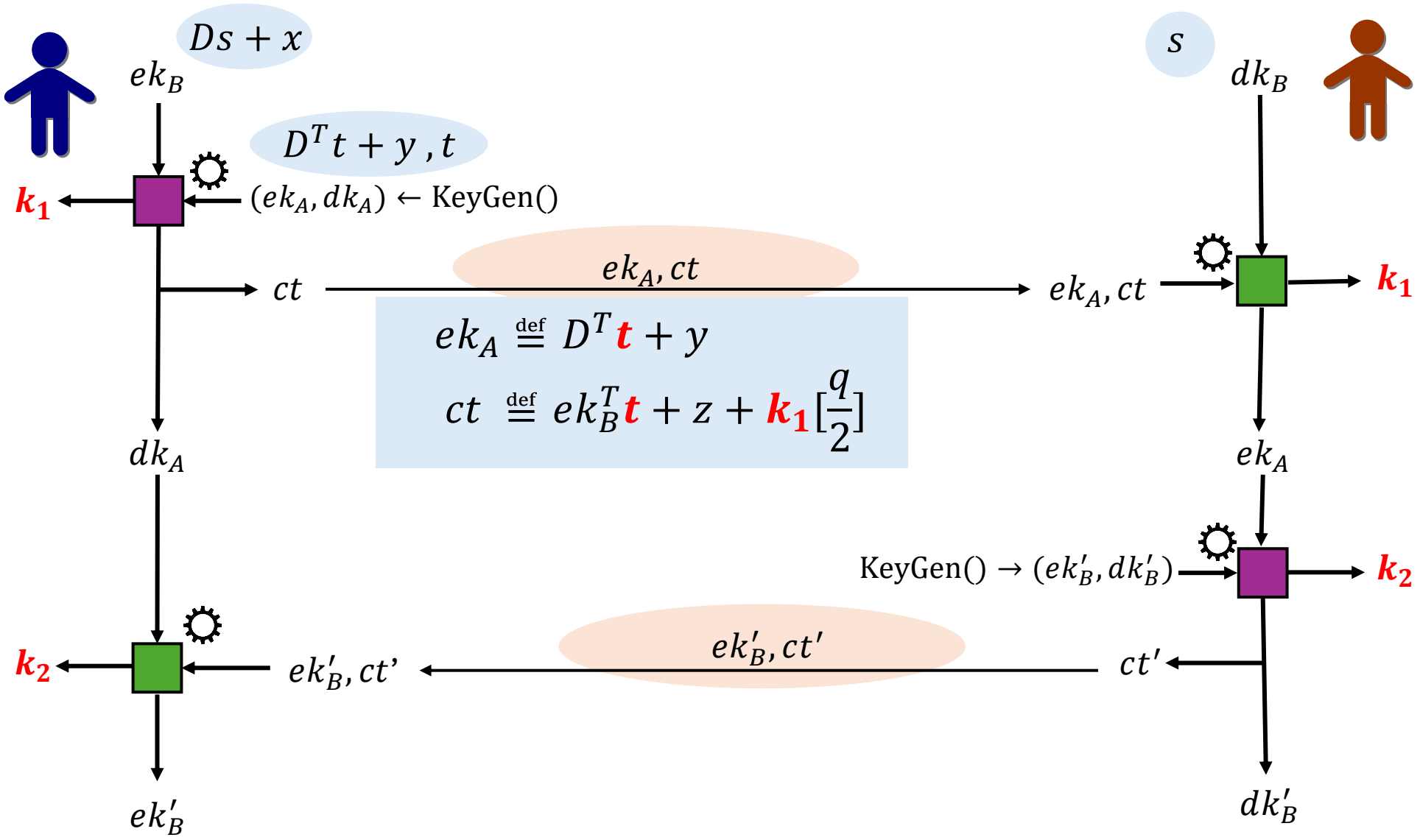
$$ek = Ds + x$$

$$ct = (ct_0, ct_1) = (D^T r + z, \quad ek^T r + z + K[\frac{q}{2}])$$

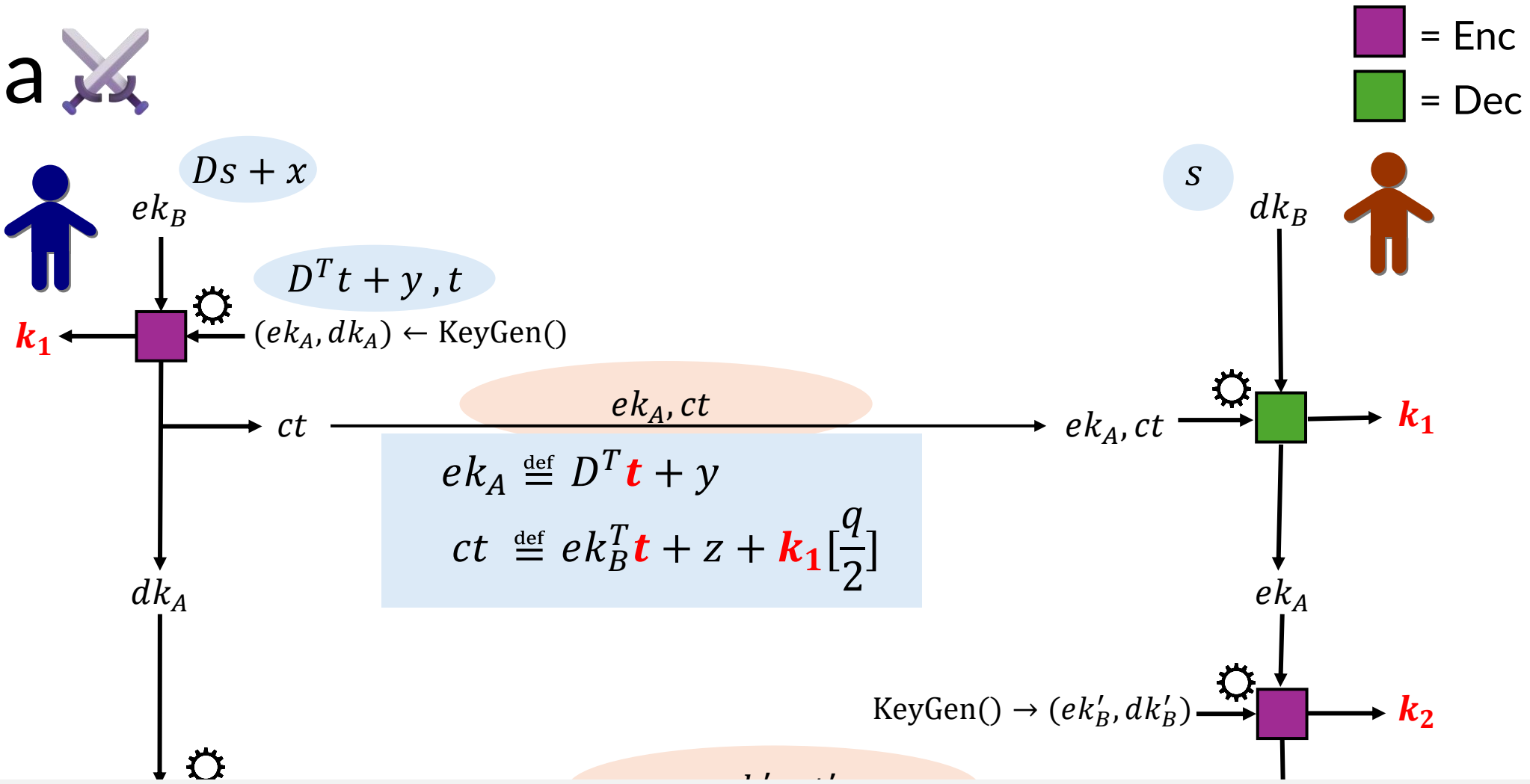
➡
- Reuse ek as ct_0 and only send ct_1 as the ciphertext ct.
- Since $|ct_0| \gg |ct_1|$, significant size reduction.

* Such an optimization was attempted in [EC:ACD19,ACCESS:LKS23] for special PQ-KEMs, but the security proofs are incorrect and the scheme becomes insecure for certain parameters.

Katana

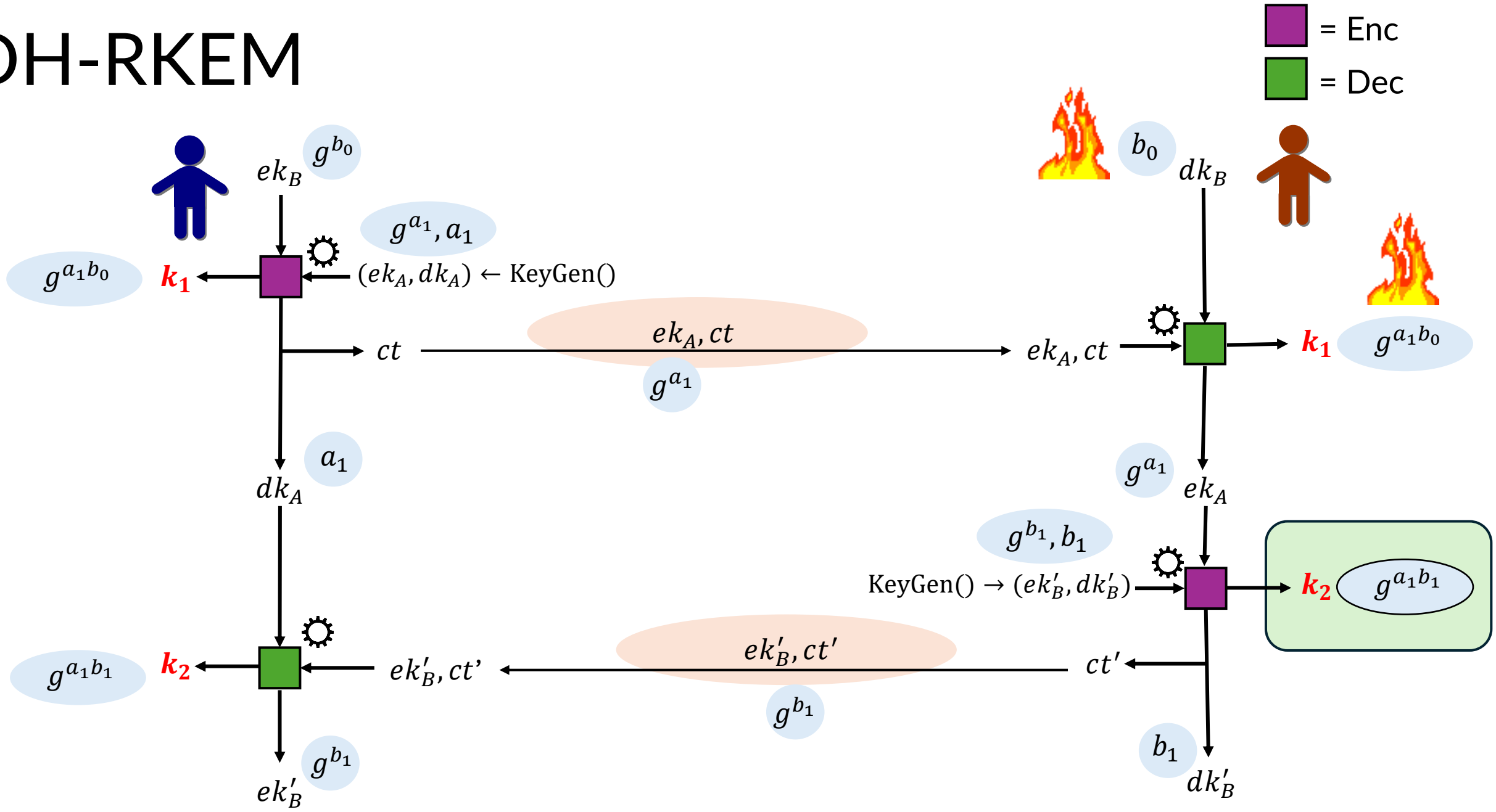

= Enc
= Dec
$Ds + x$
ek_B
$D^T t + y\ , t$
k_1
$(ek_A, dk_A) \leftarrow \text{KeyGen}()$
ct
ek_A, ct
$ek_A \stackrel{\text{def}}{=} D^T t + y$
$ct \stackrel{\text{def}}{=} ek_B^T t + z + k_1[\frac{q}{2}]$
dk_A
k_2
ek'_B, ct'
ek'_B
s
dk_B
ek_A, ct
k_1
ek_A
$\text{KeyGen}() \rightarrow (ek'_B, dk'_B)$
k_2
ct'
dk'_B

Katana ⚔️

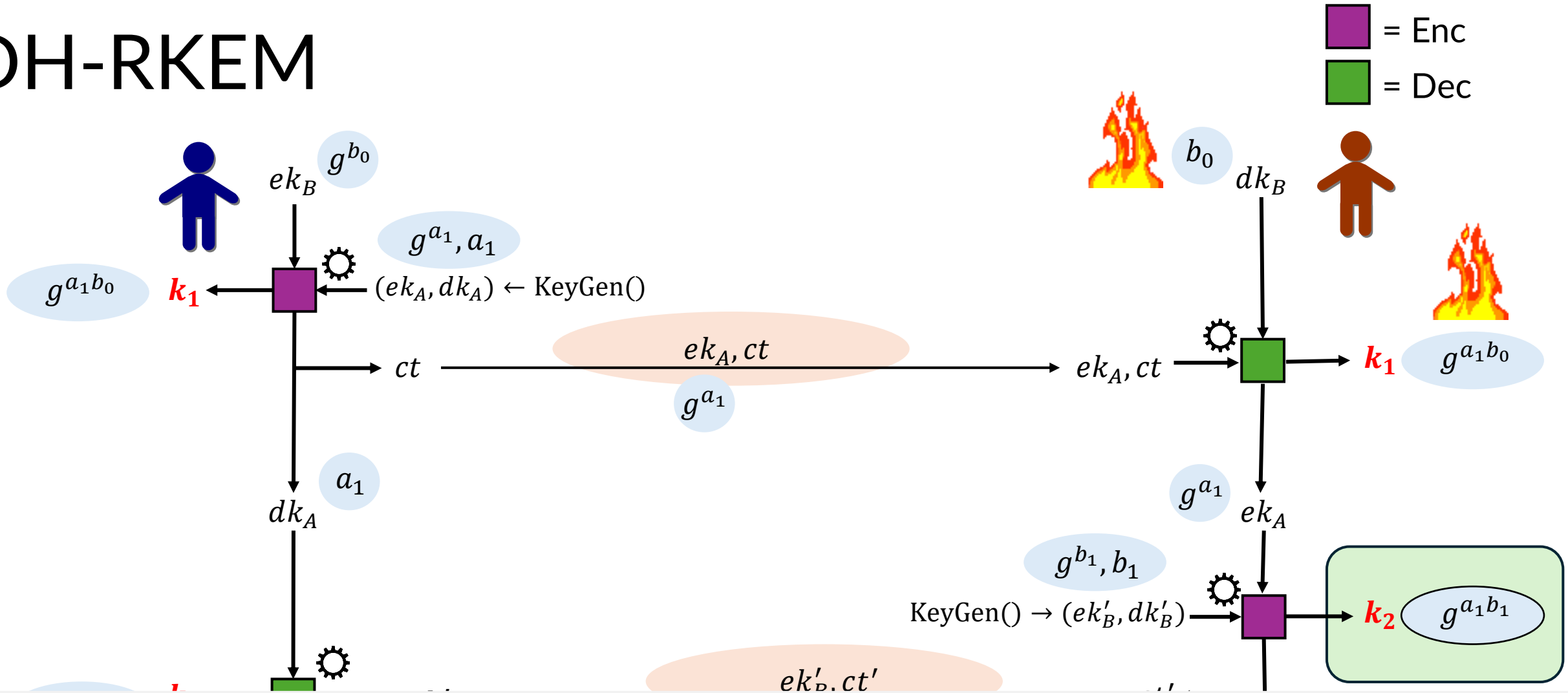


➢ **But is it secure?**

DH-RKEM

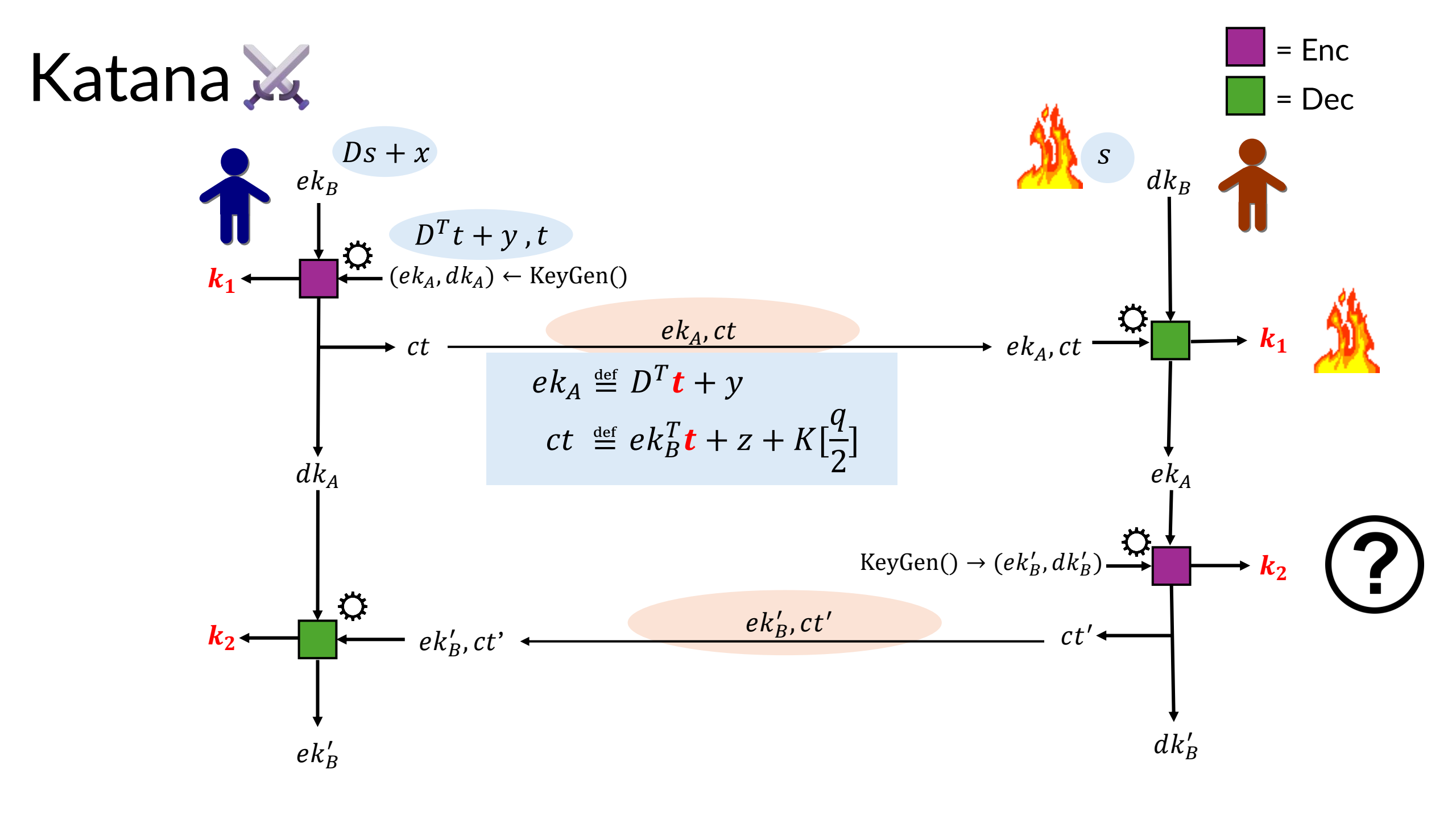

= Enc
= Dec
ek_B
g^{b_0}
g^{a_1}, a_1
$g^{a_1 b_0}$
k_1
$(ek_A, dk_A) \leftarrow \text{KeyGen}()$
ct
ek_A, ct
g^{a_1}
a_1
dk_A
b_0
dk_B
ek_A, ct
k_1
$g^{a_1 b_0}$
g^{a_1}
ek_A
g^{b_1}, b_1
$\text{KeyGen}() \rightarrow (ek'_B, dk'_B)$
k_2
$g^{a_1 b_1}$
$g^{a_1 b_1}$
k_2
ek'_B, ct'
ct'
ek'_B, ct'
g^{b_1}
g^{b_1}
ek'_B
b_1
dk'_B

DH-RKEM

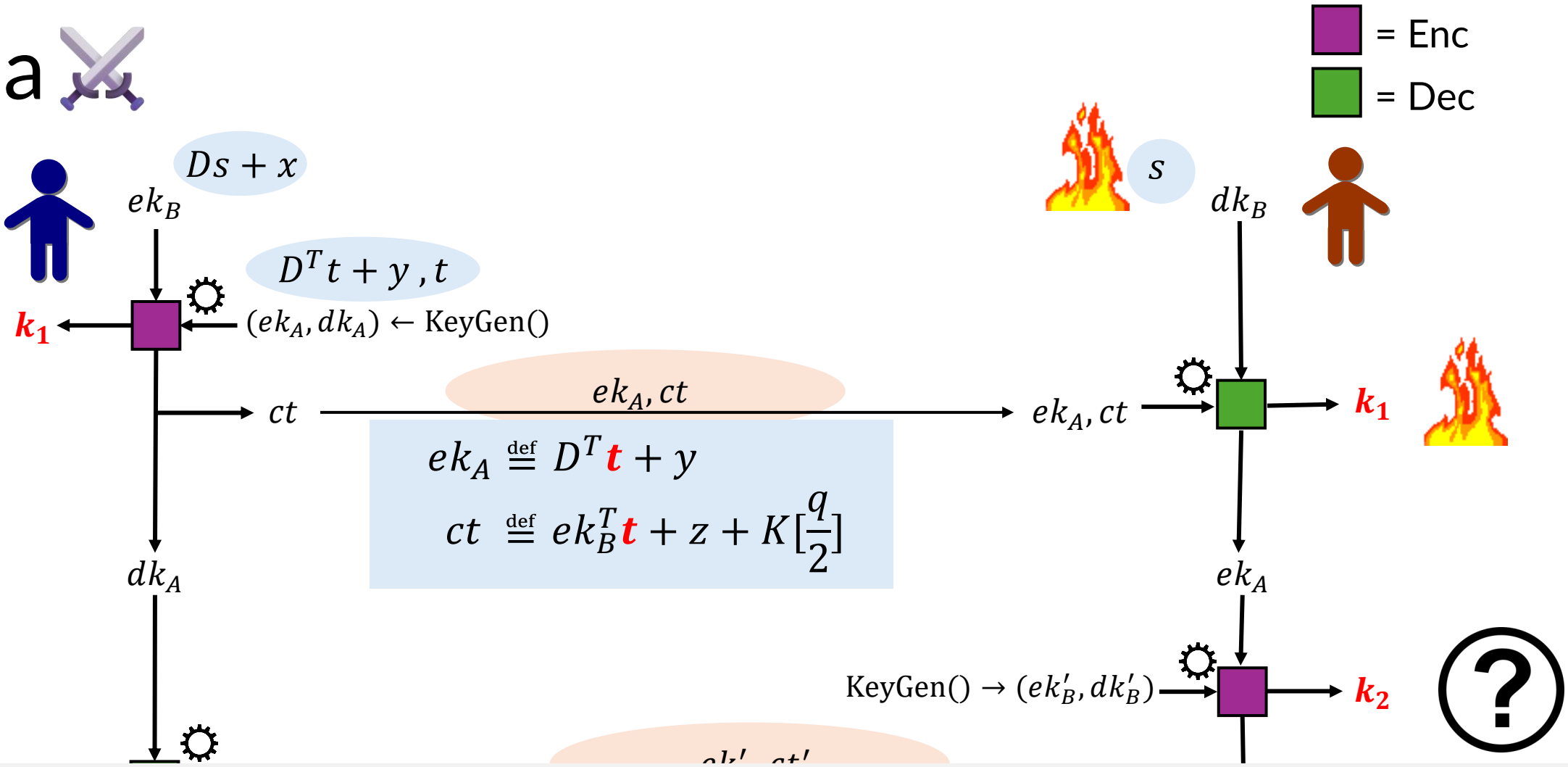


- $k_1 = g^{a_1 b_0}$ can be simulated based on (b_0, g^{a_1}, g^{b_1}) only
- **Security of k_2 follows from standard DDH**

Katana


= Enc
= Dec
$Ds + x$
ek_B
$D^T t + y\ , t$
$(ek_A, dk_A) \leftarrow \text{KeyGen}()$
k_1
ct
ek_A, ct
$ek_A \stackrel{\text{def}}{=} D^T t + y$
$ct \stackrel{\text{def}}{=} ek_B^T t + z + K[\frac{q}{2}]$
dk_A
k_2
ek'_B, ct'
ek'_B
s
dk_B
k_1
ek_A
$\text{KeyGen}() \rightarrow (ek'_B, dk'_B)$
k_2
ct'
dk'_B
?

Katana



- Noisy key exchange ct and key k_2 cannot be (jointly) simulated based on (s, ek_A, ek'_B)
- **Security of K' does not follow from standard LWE**

Katana ⚔️

Module learning with errors (MLWE)

KATANA cannot be based on MLWE ☹

$$|\Pr[\mathcal{A}(D, D \cdot s + e) = 1] - \Pr[\mathcal{A}(D, b) = 1]|$$

Katana ⚔️

Module learning with errors (MLWE)

$$|\Pr[\mathcal{A}(D, D \cdot s + e) = 1] - \Pr[\mathcal{A}(D, b) = 1]|$$

KATANA cannot be based on MLWE ☹

Hint-MLWE [KLSS'23, EEN'24]

$$|\Pr[\mathcal{A}(D, D \cdot s + e, M, h) = 1] - \Pr[\mathcal{A}(D, b, M, h) = 1]|$$

$$h = M \begin{bmatrix} s \\ e \end{bmatrix} + z$$

Katana ⚔️

Module learning with errors (MLWE)

$$|\Pr[\mathcal{A}(D, D \cdot s + e) = 1] - \Pr[\mathcal{A}(D, b) = 1]|$$

KATANA cannot be based on MLWE ☹

Hint-MLWE [KLSS'23, EEN'24]

$$|\Pr[\mathcal{A}(D, D \cdot s + e, M, h) = 1] - \Pr[\mathcal{A}(D, b, M, h) = 1]|$$

$$h = M \begin{bmatrix} s \\ e \end{bmatrix} + z$$

- Stronger assumption
- Need slightly different parameters

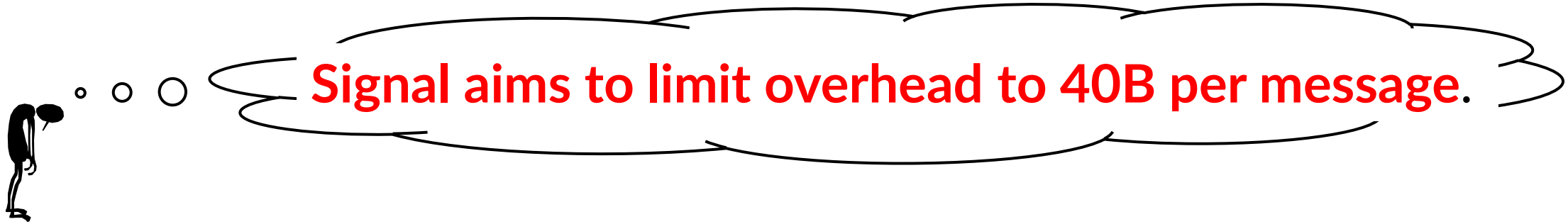
Size Reduction

	Target security	ek	ct	Total (bytes) |ek| + |ct|
ML-KEM	128	800	768	1568
Katana ⚔️		832	48	880
ML-KEM	192	1184	1088	2272
Katana ⚔️		1344	72	1416

2. Bandwith-constrained Ratcheting

Bandwith Constraint

- Katana ⚔️ saves ≈2x bandwidth ☺
- **BUT… Katana ⚔️ isn’t enough!!**



Bandwith Constraint

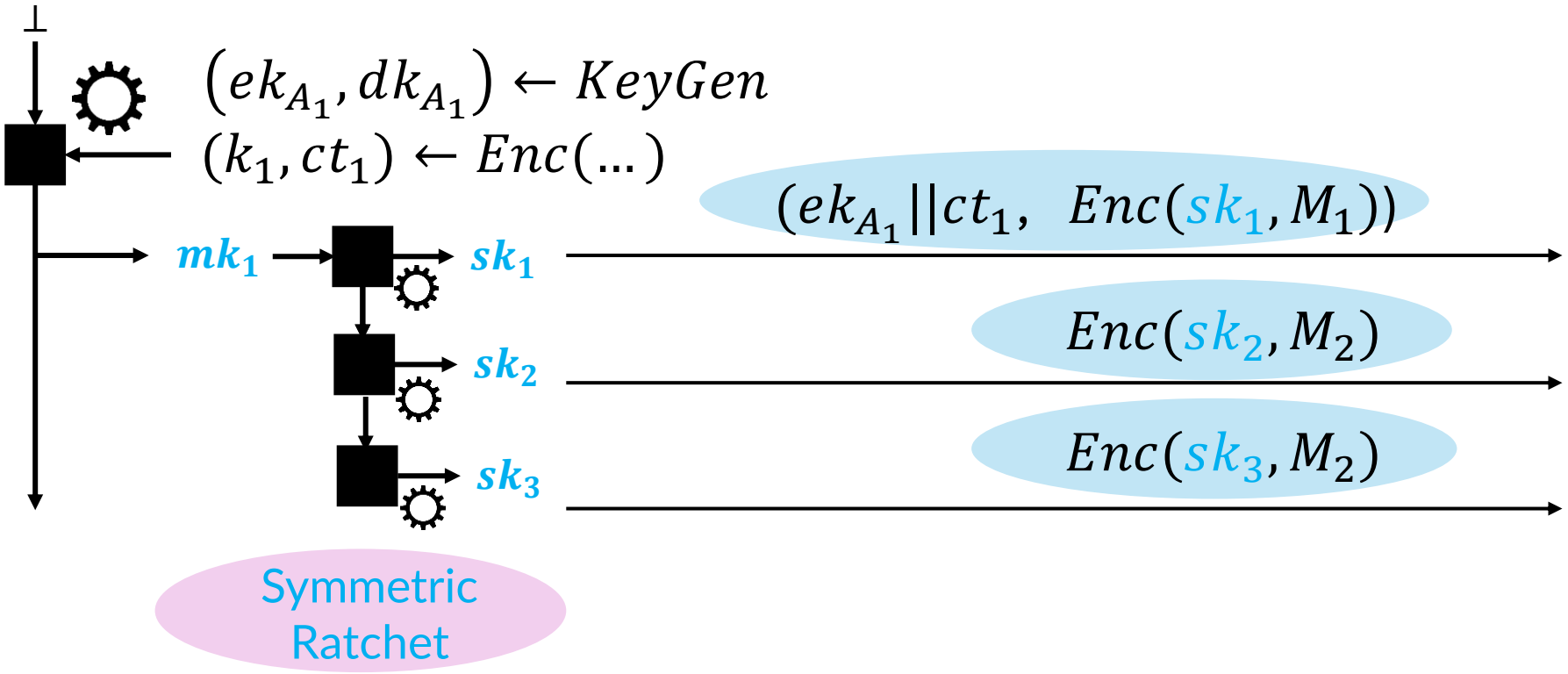
Approach 1: Only send RKEM ciphertext every X messages

- Amortize communication cost

Bandwith Constraint

Approach 1: Only send RKEM ciphertext every X messages

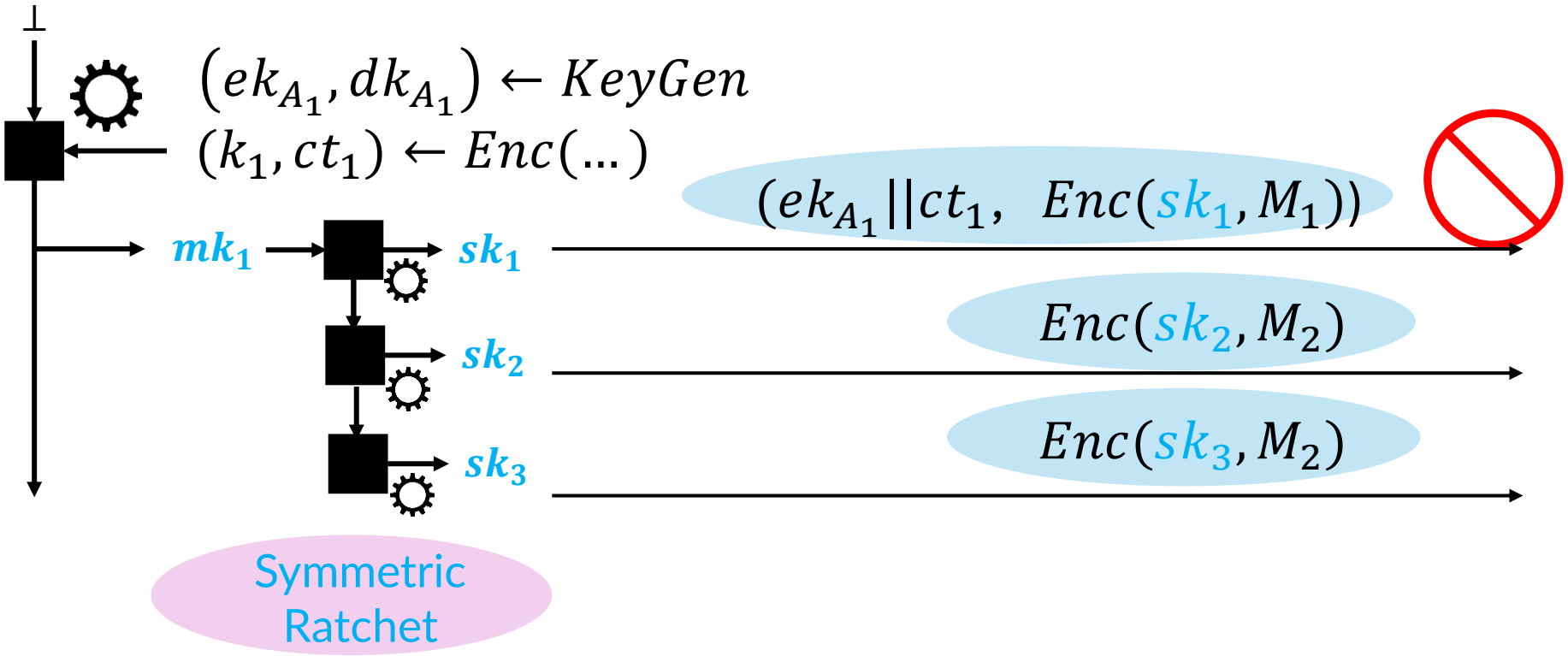
- Amortize communication cost



Bandwith Constraint

Approach 1: Only send RKEM ciphertext every X messages

- Amortize communication cost

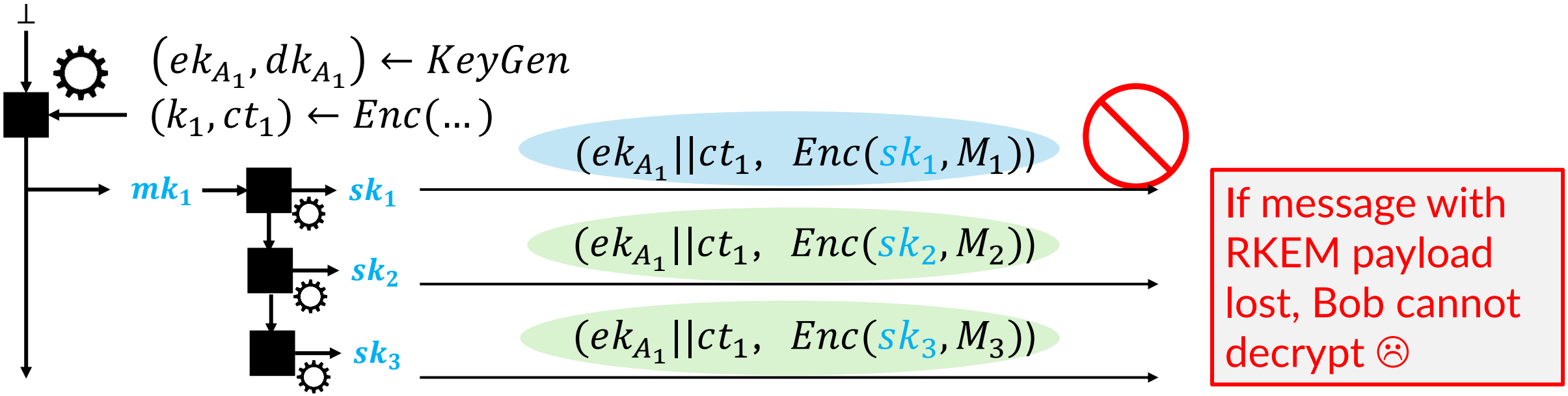


If message with RKEM payload lost, Bob cannot decrypt ☹

Bandwith Constraint

Approach 1: Only send RKEM ciphertext every X messages (*)

- Amortize communication cost



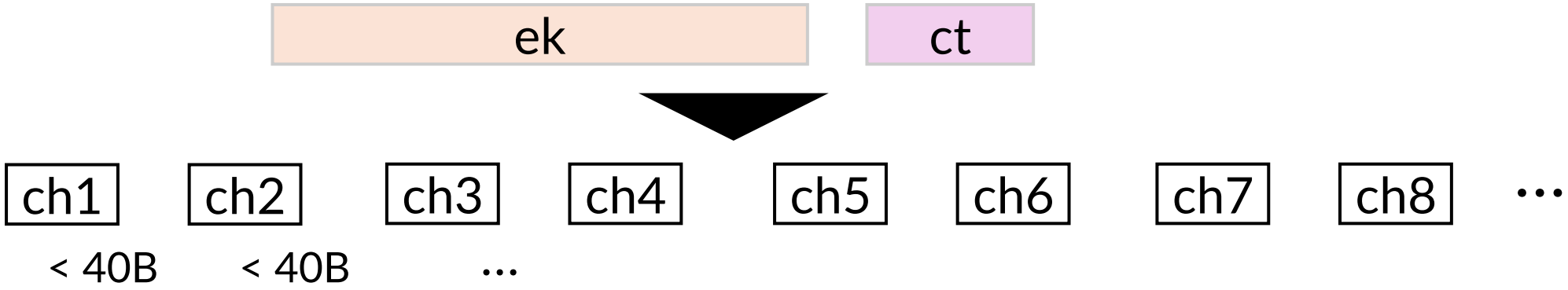
Immediate decryption forces Alice to repeat long payload until acknowledged. Worst case: no amortization!

(*) Apple's PQ3 Protocol ratchets keys *every 50 messages, but repeats until the key is confirmed*

Bandwith Constraint

Approach 2: Split (ek, ct) into smaller pieces

- Amortize communication cost
- If single message lost, only need to resend chunk
- However still wastes bandwidth (not knowing which chunk to repeat)

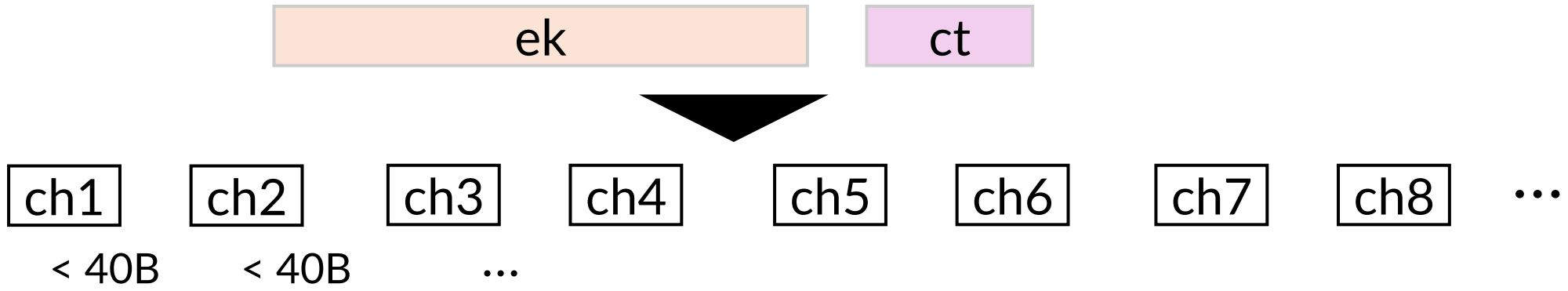


Bandwith Constraint

Approach 2: Split (ek, ct) into smaller pieces

- Amortize communication cost
- If single message lost, only need to resend chunk
- However still wastes bandwidth (not knowing which chunk to repeat)

Approach 3: Chunk using **erasure code**



Bandwith Constraint

Approach 2: Split (ek, ct) into smaller pieces

- Amortize communication cost
- If single message lost, only need to resend chunk
- However still wastes bandwidth (not knowing which chunk to repeat)

Approach 3: Chunk using **erasure code**

- Overhead per message is ***small and fixed***.
- Alice does not waste bandwidth when Bob is offline for a while.

3. Hybrid Secure Messaging

Idea 3: Hybrid Messaging

- We have seen how to **make PQ-SM practical** using KATANA ⚔️ and **chunking**

- But what if DH turns out to be more secure than (Hint-)LWE?

Idea 3: Hybrid Messaging

- We have seen how to **make PQ-SM practical** using KATANA ⚔️ and **chunking**

- But what if DH turns out to be more secure than (Hint-)LWE?

- We want **hybrid** SM (best of both worlds)

Idea 3: Hybrid Messaging

Approach 1: Build a **hybrid RKEM**

- Chunk the output of the hybrid RKEM
- Use the standard symmetric ratchet (from Signal)

Idea 3: Hybrid Messaging

Approach 1: Build a **hybrid RKEM**

- Chunk the output of the hybrid RKEM
- Use the standard symmetric ratchet (from Signal)

- Slower PCS against classical adversaries

Idea 3: Hybrid Messaging

Approach 1: Build a **hybrid RKEM**

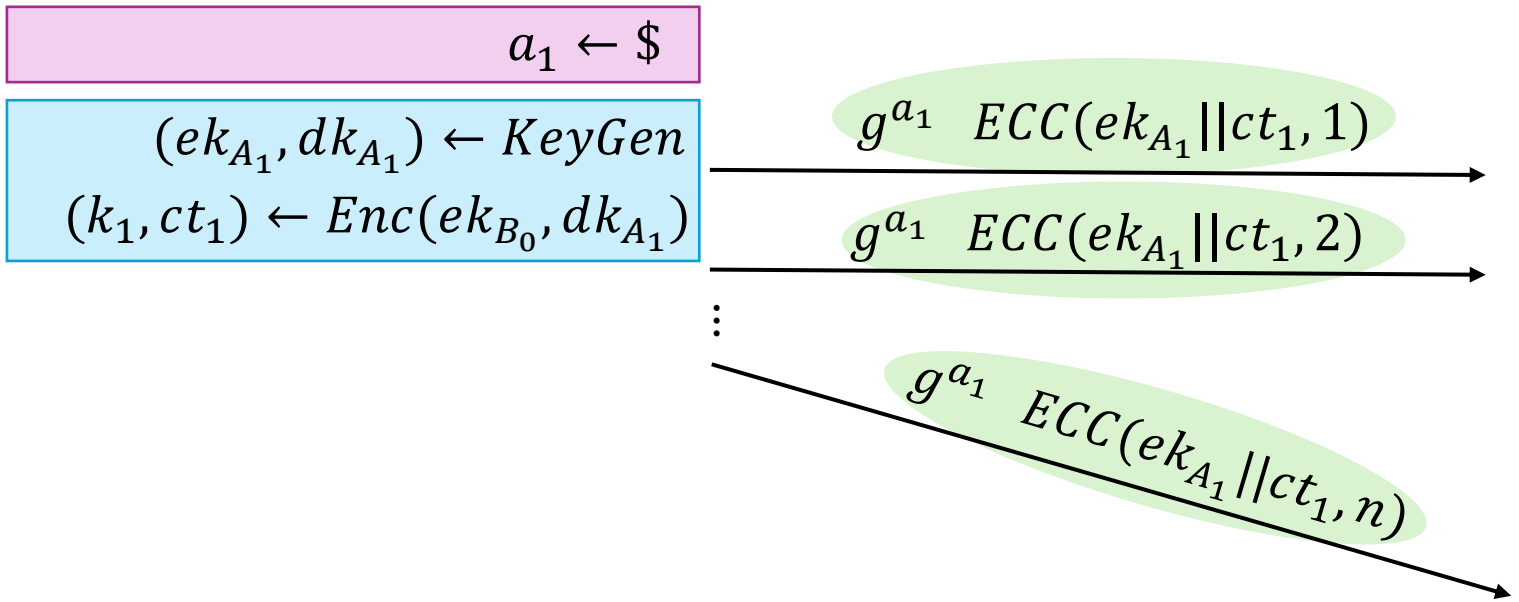
- Chunk the output of the hybrid RKEM
- Use the standard symmetric ratchet (from Signal)

- Slower PCS against classical adversaries

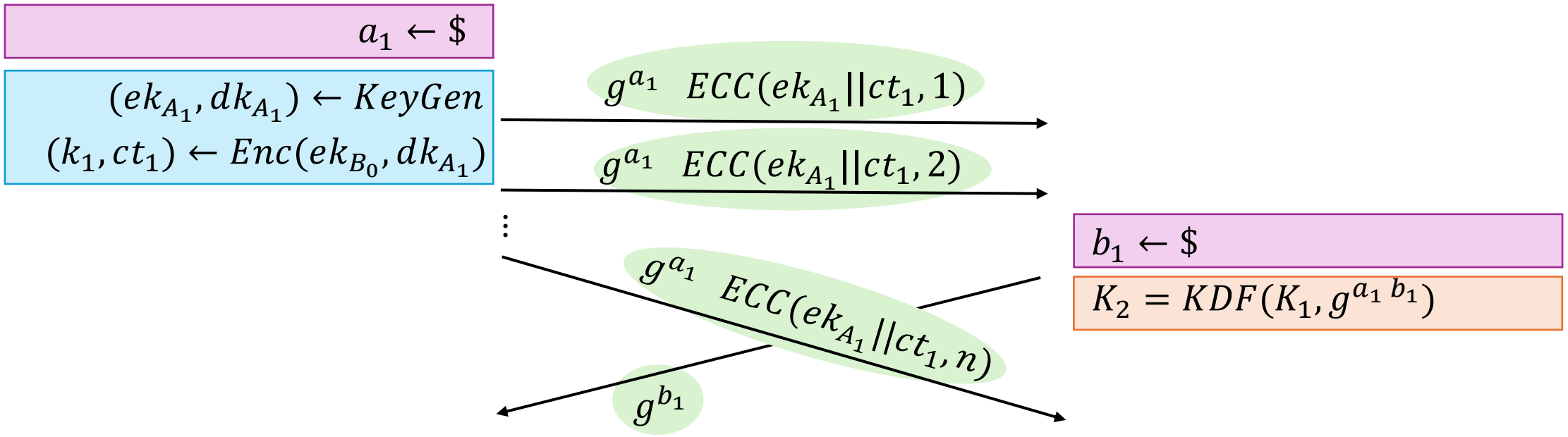
Approach 2: Run two ratcheted KEMs in parallel

- How to mix in the keys into the root key chain?

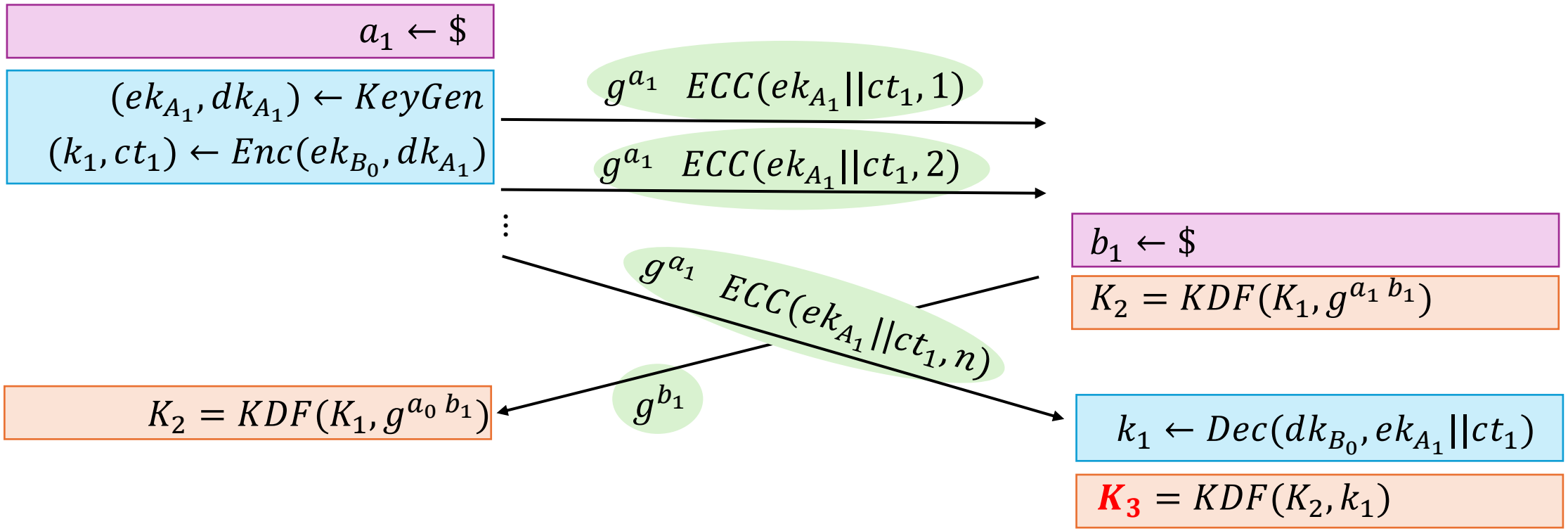
Idea 3: Hybrid Messaging



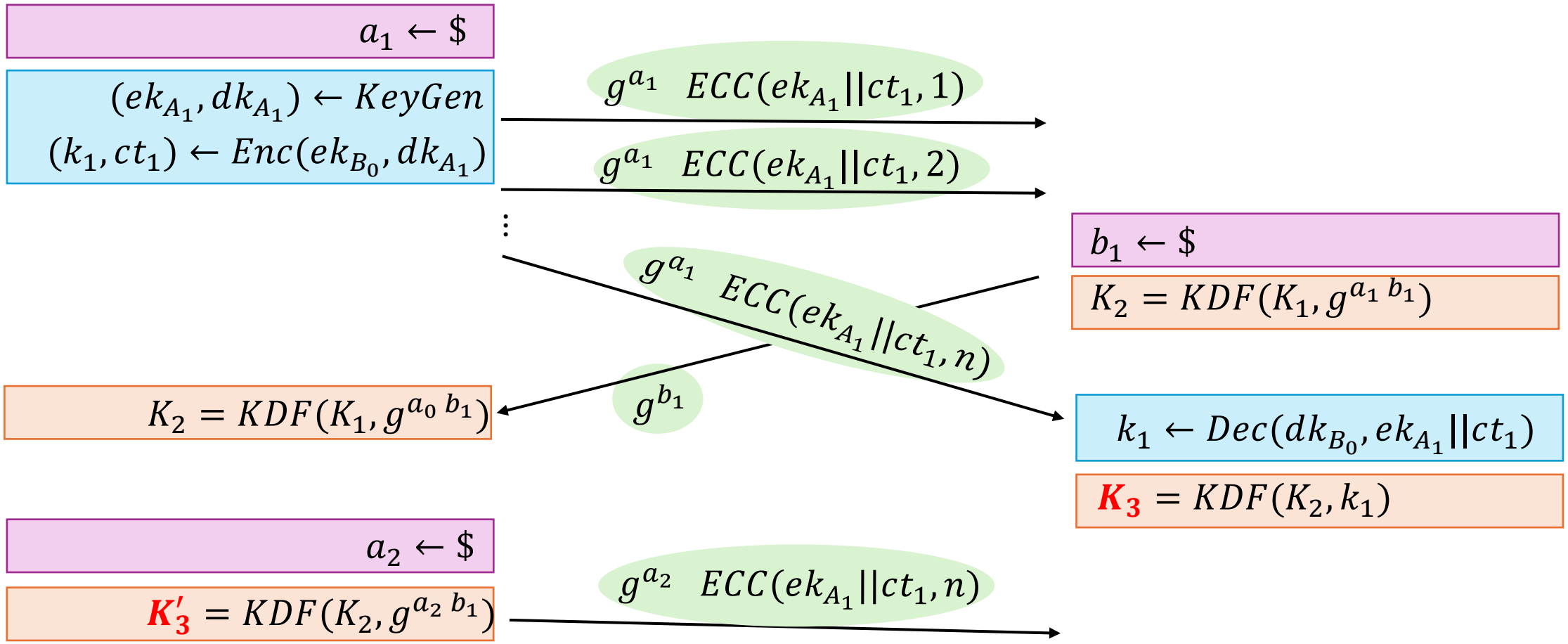
Idea 3: Hybrid Messaging



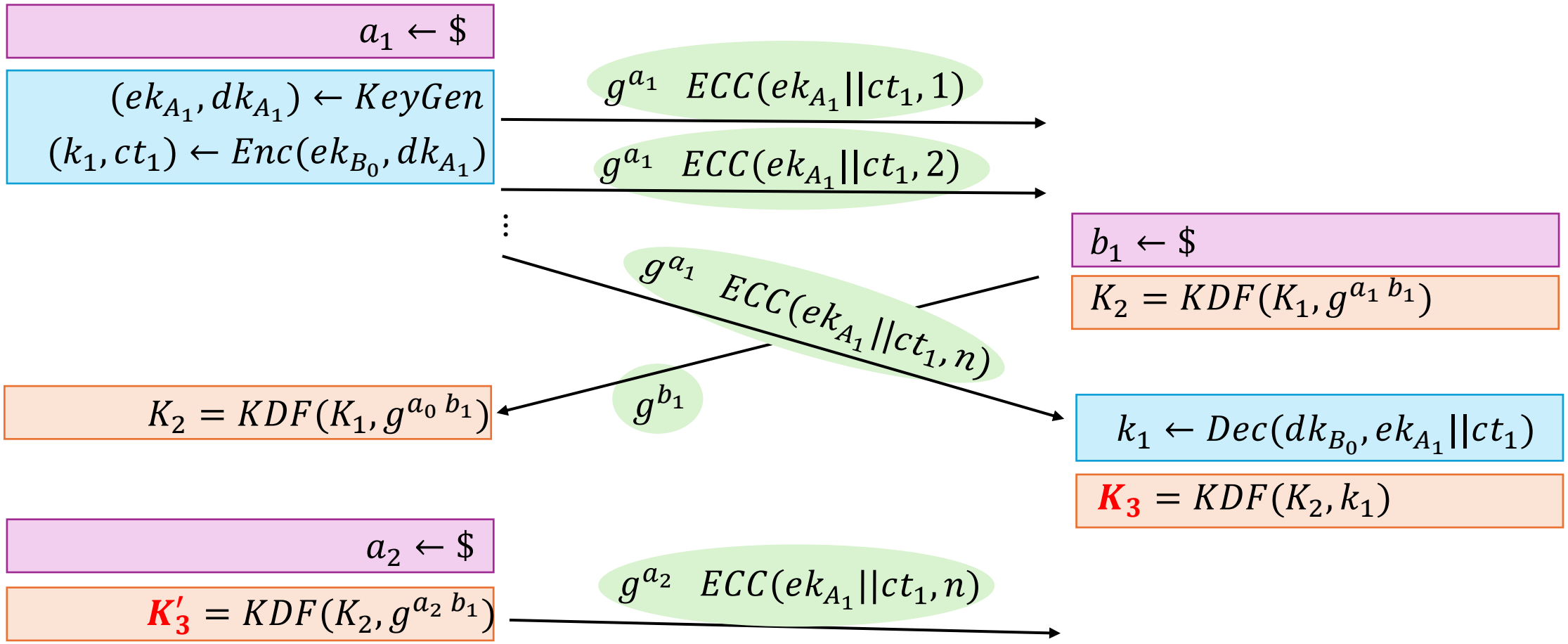
Idea 3: Hybrid Messaging



Idea 3: Hybrid Messaging



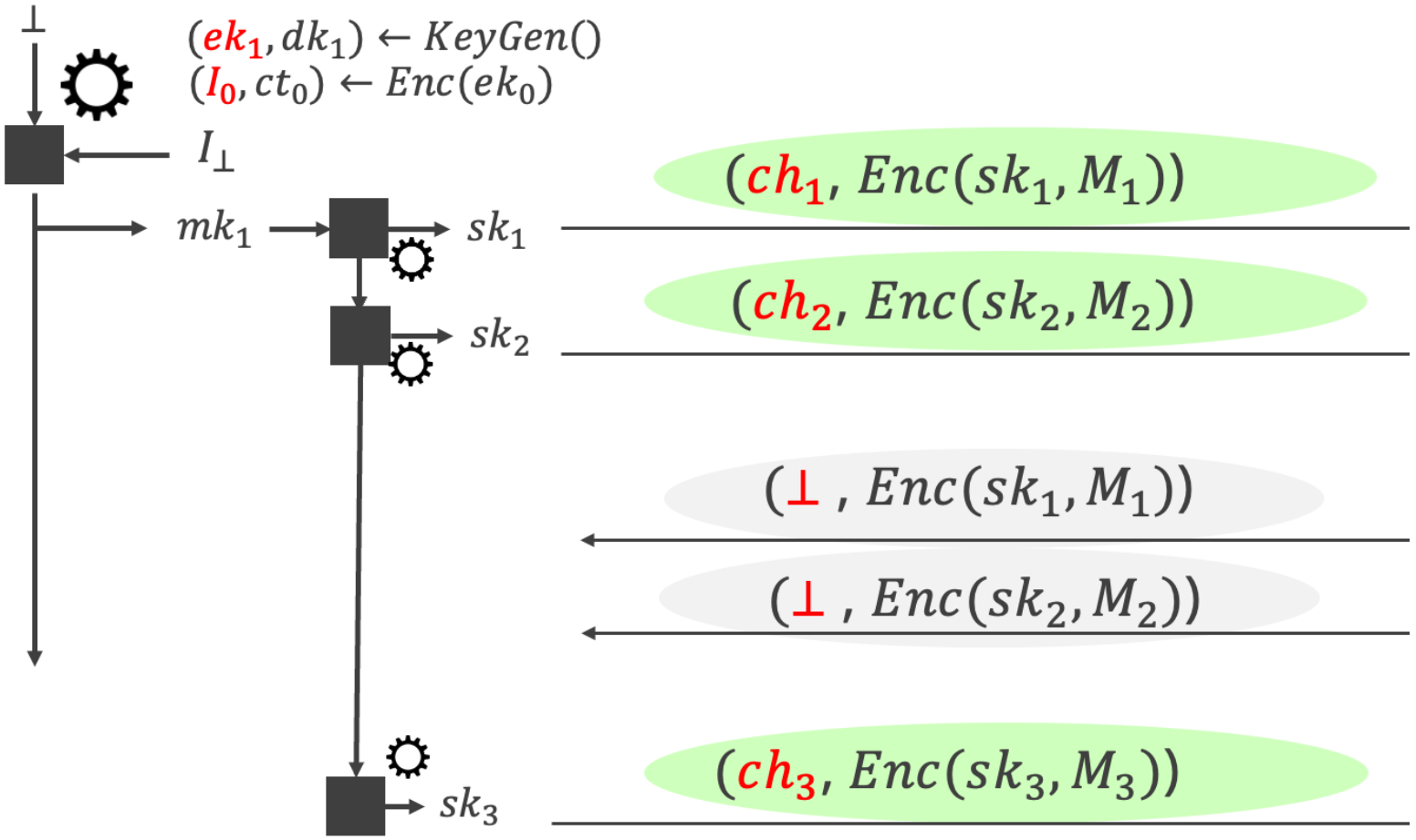
Idea 3: Hybrid Messaging



- Parties mix in keys in different order!

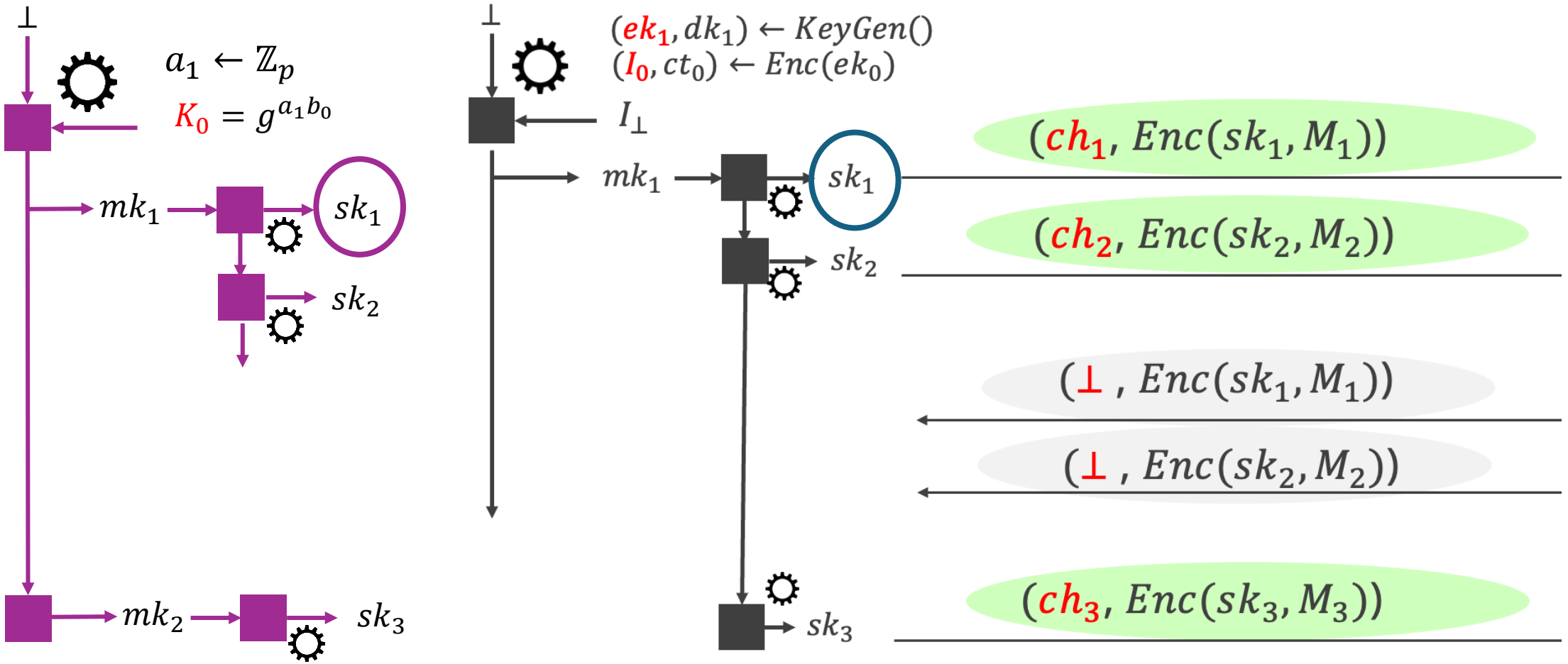
Idea 3: Hybrid Messaging

Approach 3: Run two complete ratchets in parrallel



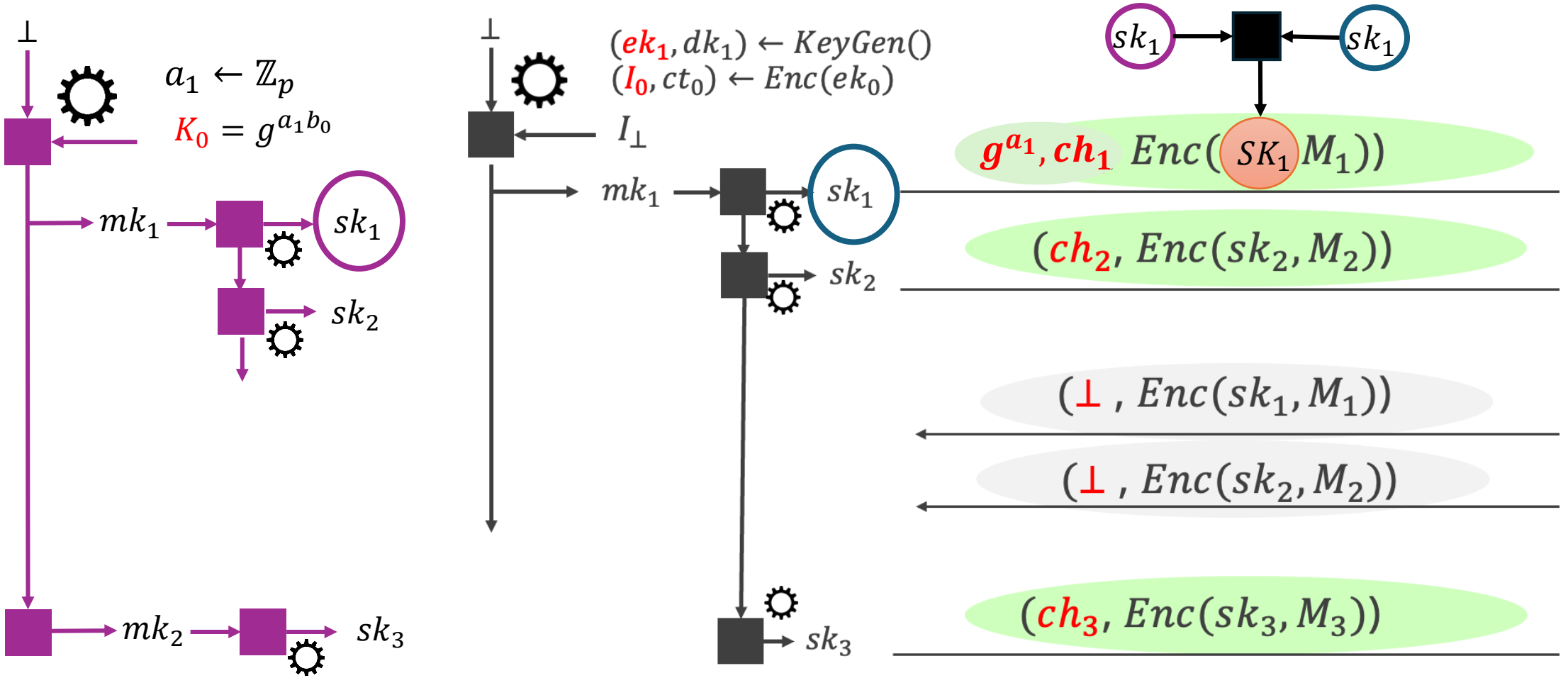
Idea 3: Hybrid Messaging

Approach 3: Run two complete ratchets in parrallel



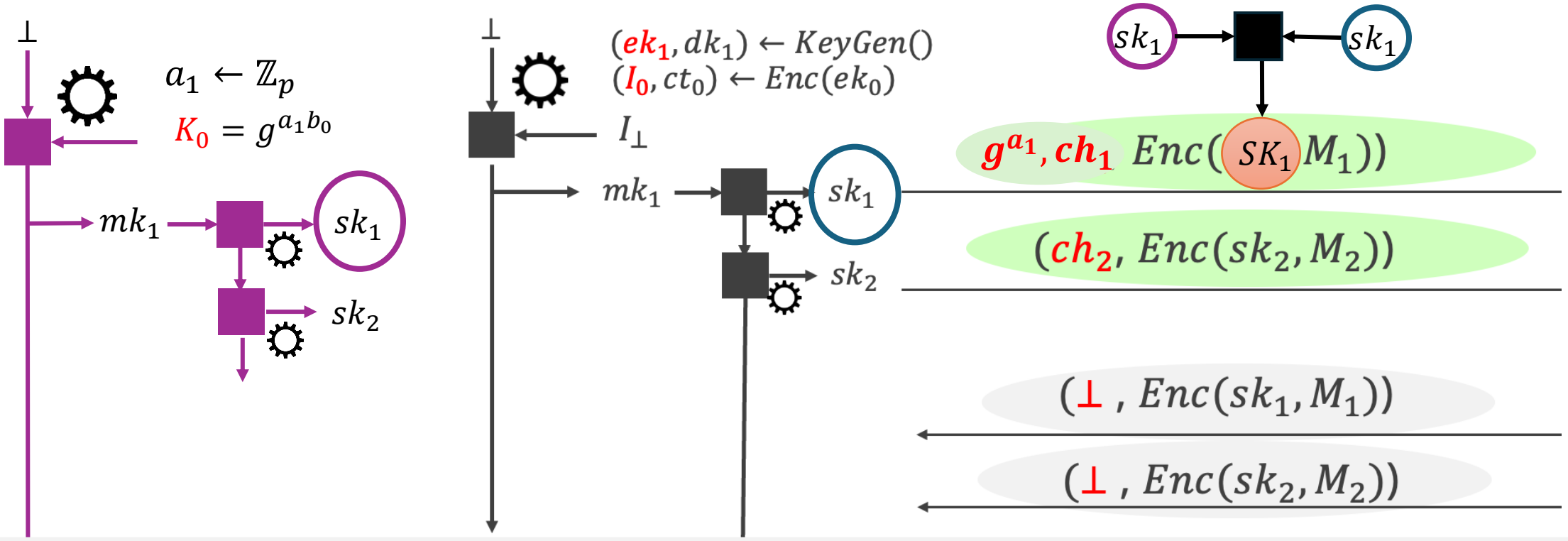
Idea 3: Hybrid Messaging

Approach 3: Run two complete ratchets in parrallel



Idea 3: Hybrid Messaging

Approach 3: Run two complete ratchets in parrallel



- Maintaining two separate key-chains ensures DH-ratchet can advance at natural speed

Thank you!

Signal's Double Ratchet

+

PQ Ratchet

=

Triple Ratchet

Eprint link to [EC:DJKPS25]

- **Our *bandwidth efficient* PQ Ratchet relies on…**
  - A lattice-based *Ratcheting* KEM called KATANA ⚔️
  - Error correcting codes (aka "chunking")
- **Triple Ratchet is a natural composition of the Double and PQ Ratchets ☺**

From Signal

- **A PQ Ratchet IS in development!**
- Current plans include many of today's ideas, but with slight differences.
- **Key difference**: Use an ML-KEM based protocol for a first step.
- Hope to use optimizations like Katana ⚔️ in the future ☺

(*) Apple's PQ3 Protocol

- ✓ Apple rolled out a PQ continuous key agreement.
- ✓ It uses ***amortization*** to save bandwidth.
  - Only send PQ ratchet keys *every 50 messages*.
- ✓ BUT, **not compatible with Signal's restrictions.**
  - Under restricted bandwidth, even sending one PQ ratchet key deteriorates performance.
  - Lose immediate decryption, unless we continuously resend PQ key.
- ✓ On a bad network + device offline for a while ⇨