Cryptography Engineering

- Lecture 7 (Dec 3, 2025)

- Case study: **E2EE-secure messaging** (2 lectures)
  - Secure Messaging
  - X3DH Protocol
  - Symmetric-key Ratchet
  - Forward/Backward Secrecy
  - Diffie-Hellman Ratchet

“Case Studies” in the Course

- So far, we know digital signature (certificate), TLS (and PQTLS, KEM-TLS)…
- Real-world cryptographic applications are far more complex

- We will study several real-world cryptographic applications in this course

- Your final project (to be decided) **may use** some techniques from these real-world applications…

Secure Messaging

- Text Messages/Instant Messaging

WhatsApp **Signal** **iMessage**

End-to-End Encryption

- End-to-End Encryption (E2EE)
  - Only sender and recipient can decrypt messages...
  - **The server cannot decrypt messages** (if it does not tamper with the conversation...
  - Confidentiality and Privacy
  - In practice, the server will help relaying/forwarding messages...

End-to-End Encryption

- End-to-End Encryption (E2EE)
  - Only sender and recipient can decrypt messages…
  - **The server cannot decrypt messages** (if it does not tamper with the conversation…
  - Confidentiality and Privacy
  - In practice, the server will help relaying/forwarding messages…

E2EE (by default)
Examples

WhatsApp Signal iMessage Element

Non-E2EE (by default)
Examples

Discord Facebook Messager Telegram

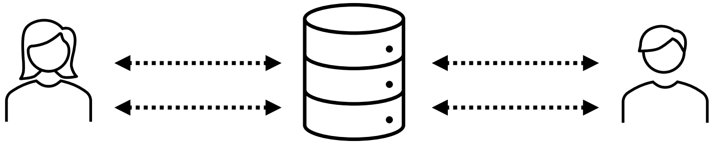
End-to-End Encryption

Initialization

Including logging in, sharing users' information, …

Messaging

E2EE

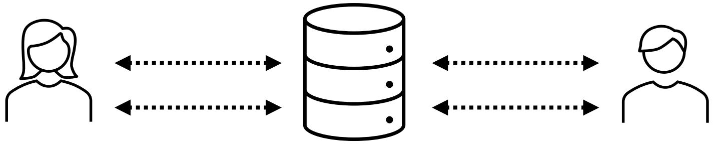


Alice

Server

Bob

Non-E2EE

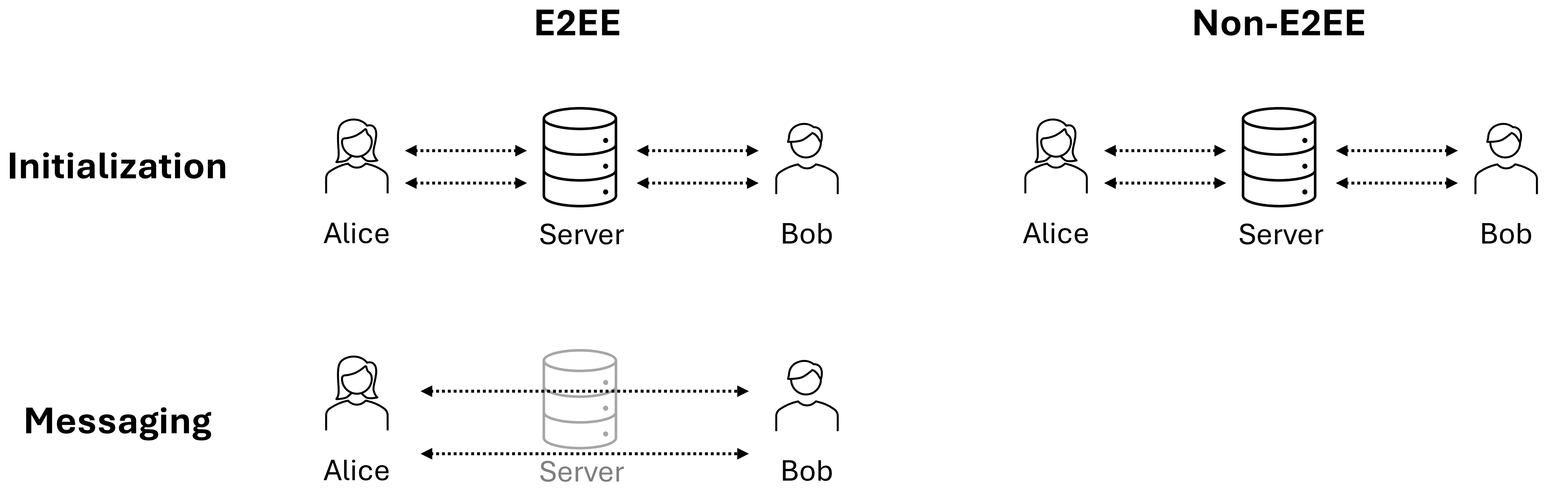


Alice

Server

Bob

End-to-End Encryption



E2EE | **Non-E2EE**

Initialization

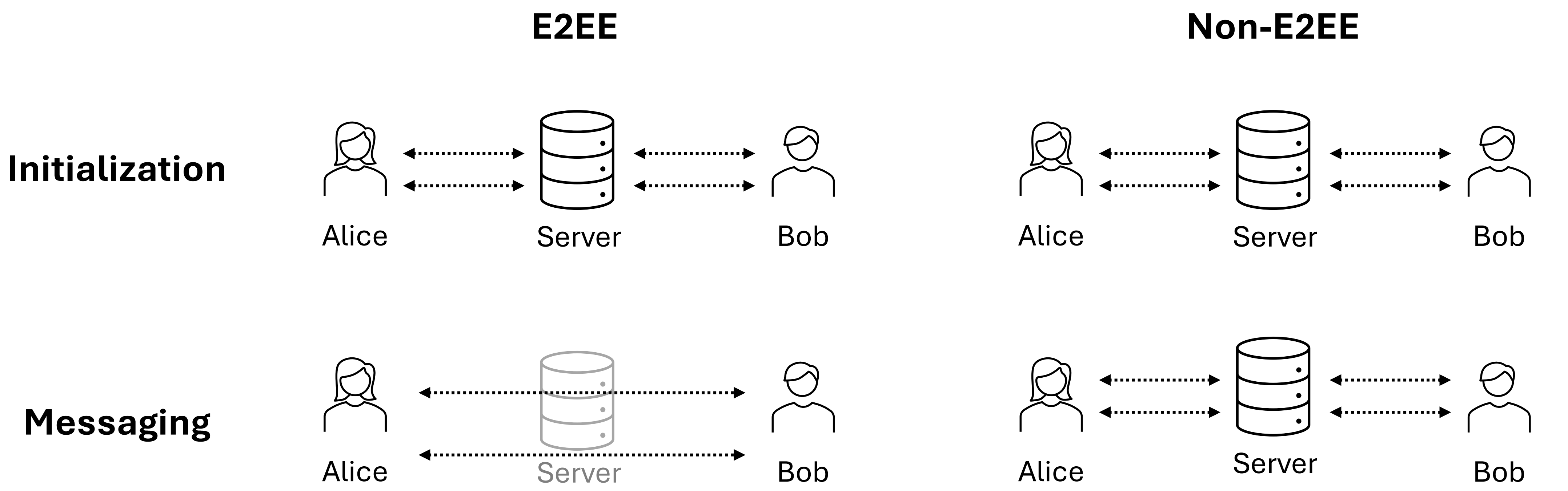
Alice Server Bob | Alice Server Bob

Messaging

Alice Server Bob

The server **only relays** the
encrypted messages, no storage
(or just short-term storage).

End-to-End Encryption



E2EE | **Non-E2EE**

Initialization

Alice — Server — Bob | Alice — Server — Bob

Messaging

Alice — Server — Bob | Alice — Server — Bob

The server **only relays** the encrypted messages, no storage (or just short-term storage).

Encrypted communication (e.g., via TLS) with the server, **but the messages may be stored (in plaintext)…**

Signal Secure Messaging Protocol

Signal

Hey Now

Hey Now

How's it going? Now

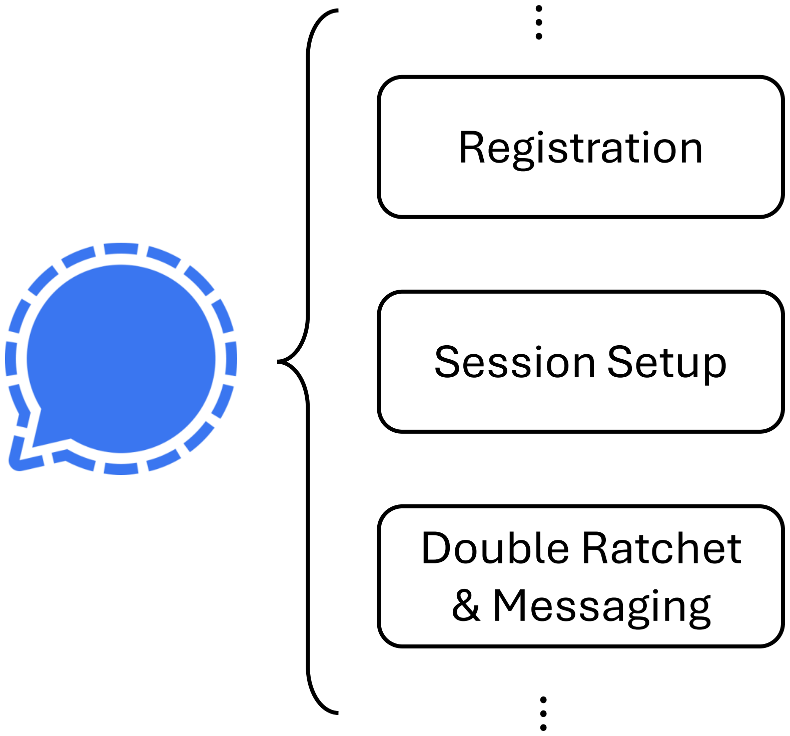
Good

How are you? Now

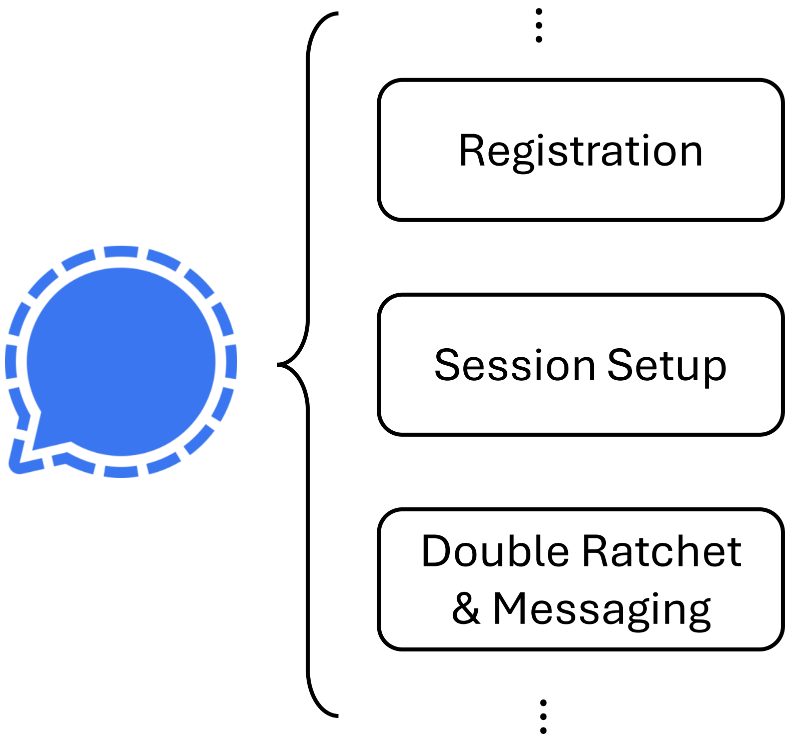
Fine Now

- One of the most secure instant messaging app
- End-to-end encryption (E2EE)
- WhatsApp also uses the Signal protocol

Signal Secure Messaging Protocol



Signal Secure Messaging Protocol



- Identity keys, signed pre-keys, one-time pre-keys, …
- X3DH (Extended Triple Diffie-Hellman) protocol **(Today)**
- Double Ratchet Algorithm:
  - Symmetric Ratchet **(Today)**
  - Diffie-Hellman Ratchet

The X3DH Protocol

- Address *How to Establish Secure Initial Shared Secret*
  - It needs the server to help sharing pre-information
- Based on (EC)DH
- Mutual Authentication:
  - Two communication parties have long-term key pairs
- Forward Secrecy

The X3DH Protocol – Key Pairs

- Key pairs of each party:
  - For simplicity, we define 'XPK' always equals to 'g^{xk}'
  - All public keys (along with the user identity) will be stored in the server

Public parameters: $(\mathbb{G}, g, q)$:

A q-order EC group $\mathbb{G}$ with a generator g

	Alice	Bob
Identity secret key (IK)	$ik_{\mathrm{A}} \in_\$ \mathbb{Z}_q$	$ik_{\mathrm{B}} \in_\$ \mathbb{Z}_q$
Identity public key (IPK)	$IPK_{\mathrm{A}} (= g^{ik_A})$	IPK_{B}
Signing secret pre-key (SK)	$sk_{\mathrm{A}} \in_\$ \mathbb{Z}_q$	$sk_{\mathrm{B}} \in_\$ \mathbb{Z}_q$
Signing public pre-key (SPK)	SPK_{A}	SPK_{B}
One-time secret pre-keys (OK)	$\{ok_{\mathrm{A}}^1, ok_{A}^2, \dots\} \subseteq_\$ \mathbb{Z}_q$	$\{ok_{\mathrm{B}}^1, ok_{\mathrm{B}}^2, \dots\} \subseteq_\$ \mathbb{Z}_q$
One-time public pre-keys (OPK)	$(OPK_{\mathrm{A}}^1, OPK_{\mathrm{A}}^2, \dots)$	$(OPK_{\mathrm{B}}^1, OPK_{\mathrm{B}}^2, \dots)$

The X3DH Protocol – Key Pairs

- Key pairs of each party:
  - For simplicity, we define $'XPK'$ always equals to $'g^{xk}'$
  - All public keys (along with the user identity) will be stored in the server

Identity keys

- Generated during registration
- Will be used for **Key Exchange and Signing**

Public parameters: $(\mathbb{G}, g, q)$:

A q-order EC group $\mathbb{G}$ with a generator g

	Alice	Bob
Identity secret key (IK)	$ik_\mathrm{A} \in_\$ \mathbb{Z}_q$	$ik_\mathrm{B} \in_\$ \mathbb{Z}_q$
Identity public key (IPK)	$IPK_\mathrm{A} (= g^{ik_A})$	IPK_B
Signing secret pre-key (SK)	$sk_\mathrm{A} \in_\$ \mathbb{Z}_q$	$sk_\mathrm{B} \in_\$ \mathbb{Z}_q$
Signing public pre-key (SPK)	SPK_A	SPK_B
One-time secret pre-keys (OK)	$\{ok_\mathrm{A}^1, ok_A^2, \dots\} \subseteq_\$ \mathbb{Z}_q$	$\{ok_\mathrm{B}^1, ok_B^2, \dots\} \subseteq_\$ \mathbb{Z}_q$
One-time public pre-keys (OPK)	$(OPK_\mathrm{A}^1, OPK_\mathrm{A}^2, \dots)$	$(OPK_\mathrm{B}^1, OPK_\mathrm{B}^2, \dots)$

The X3DH Protocol – Key Pairs

- Key pairs of each party:
  - For simplicity, we define 'XPK' always equals to 'g^{xk}'
  - All public keys (along with the user identity) will be stored in the server

Public parameters: $(\mathbb{G}, g, q)$:

A q-order EC group $\mathbb{G}$ with a generator g

	Alice	Bob
Identity secret key (IK)	$ik_{\mathrm{A}} \in_{\$} \mathbb{Z}_q$	$ik_{\mathrm{B}} \in_{\$} \mathbb{Z}_q$
Identity public key (IPK)	$IPK_{\mathrm{A}}(= g^{ik_A})$	IPK_{B}
Signing secret pre-key (SK)	$sk_{\mathrm{A}} \in_{\$} \mathbb{Z}_q$	$sk_{\mathrm{B}} \in_{\$} \mathbb{Z}_q$
Signing public pre-key (SPK)	SPK_{A}	SPK_{B}
One-time secret pre-keys (OK)	$\{ok_{\mathrm{A}}^1, ok_{A}^2, \ldots\} \subseteq_{\$} \mathbb{Z}_q$	$\{ok_{\mathrm{B}}^1, ok_{B}^2, \ldots\} \subseteq_{\$} \mathbb{Z}_q$
One-time public pre-keys (OPK)	$(OPK_{\mathrm{A}}^1, OPK_{\mathrm{A}}^2, \ldots)$	$(OPK_{\mathrm{B}}^1, OPK_{\mathrm{B}}^2, \ldots)$

Signing Pre-keys

- Generated during registration
- Updated periodically (e.g., once a week, or once a month)
- Will be used for **Key Exchange and Signing**

The X3DH Protocol – Key Pairs

- Key pairs of each party:
  - For simplicity, we define 'XPK' always equals to 'g^{xk}'
  - All public keys (along with the user identity) will be stored in the server

Public parameters: $(\mathbb{G}, g, q)$:

A q-order EC group $\mathbb{G}$ with a generator g

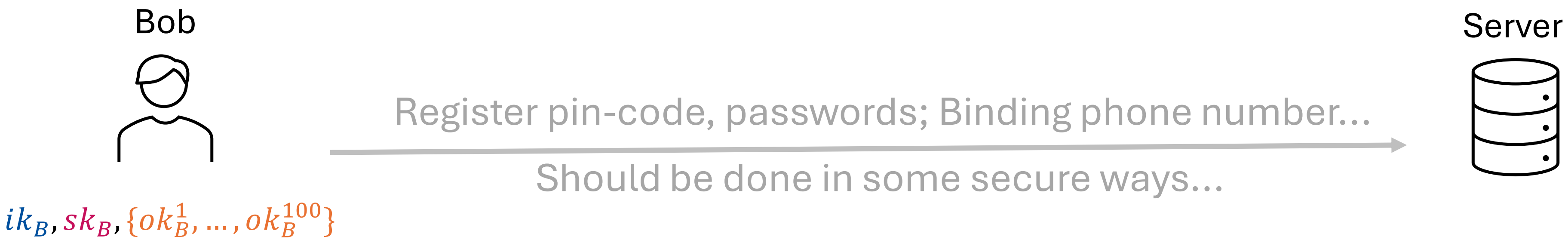
	Alice	Bob
Identity secret key (IK)	$ik_A \in_\$ \mathbb{Z}_q$	$ik_B \in_\$ \mathbb{Z}_q$
Identity public key (IPK)	$IPK_A (= g^{ik_A})$	IPK_B
Signing secret pre-key (SK)	$sk_A \in_\$ \mathbb{Z}_q$	$sk_B \in_\$ \mathbb{Z}_q$
Signing public pre-key (SPK)	SPK_A	SPK_B
One-time secret pre-keys (OK)	$\{ok_A^1, ok_A^2, \dots\} \subseteq_\$ \mathbb{Z}_q$	$\{ok_B^1, ok_B^2, \dots\} \subseteq_\$ \mathbb{Z}_q$
One-time public pre-keys (OPK)	$(OPK_A^1, OPK_A^2, \dots)$	$(OPK_B^1, OPK_B^2, \dots)$

One-time Pre-keys

- Generated as a batch during registration
- Each key is used once for each new session; Deleted after use
- Re-generated when used up (or the supply is low)

The X3DH Protocol – Pre-key Bundles

- When Bob registers (we only focus on the cryptographic parts)…
  - For simplicity, we define 'XPK' always equals to 'g^{xk}'



Bob

Server

Register pin-code, passwords; Binding phone number…

Should be done in some secure ways…

$ik_B, sk_B, \{ok_B^1, \ldots, ok_B^{100}\}$

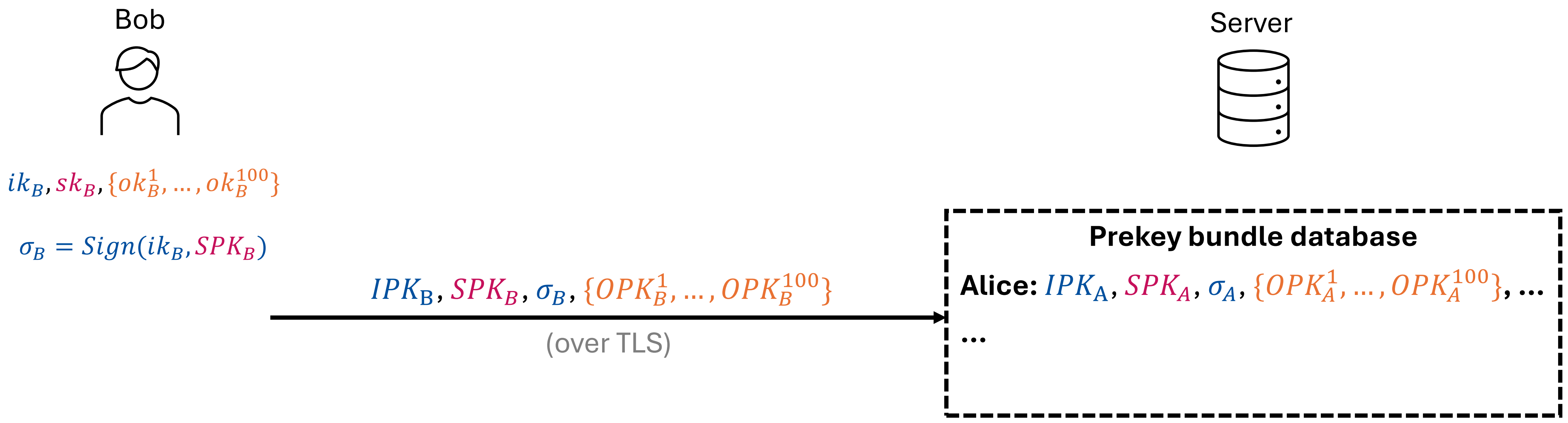
Prekey bundle database

Alice: $IPK_A, SPK_A, \sigma_A, \{OPK_A^1, \ldots, OPK_A^{100}\}$**, …**

…

The X3DH Protocol – Pre-key Bundles

- When Bob registers (we only focus on the cryptographic parts)…
  - For simplicity, we define ‘XPK’ always equals to ‘g^{xk}’



Bob

Server

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

$\sigma_B = Sign(ik_B, SPK_B)$

$IPK_\mathrm{B}, SPK_B, \sigma_B, \{OPK_B^1, \dots, OPK_B^{100}\}$

(over TLS)

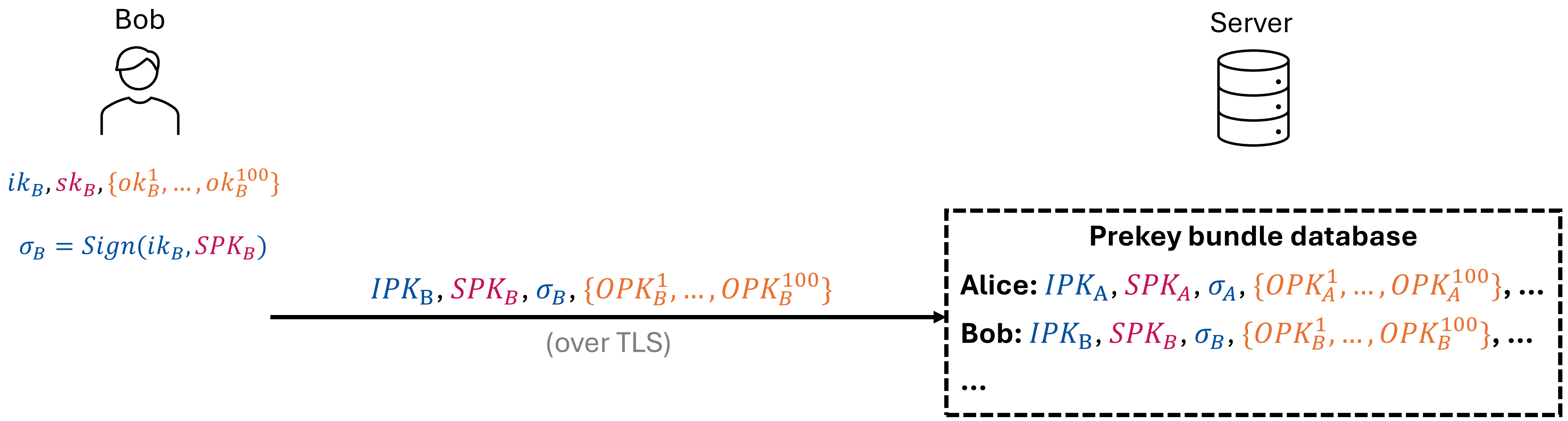
Prekey bundle database

Alice: $IPK_\mathrm{A}, SPK_A, \sigma_A, \{OPK_A^1, \dots, OPK_A^{100}\}$**, …**

…

The X3DH Protocol – Pre-key Bundles

- When Bob registers (we only focus on the cryptographic parts)…
  - For simplicity, we define ‘XPK’ always equals to ‘g^{xk}’



$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

$\sigma_B = Sign(ik_B, SPK_B)$

$IPK_\mathrm{B}, SPK_B, \sigma_B, \{OPK_B^1, \dots, OPK_B^{100}\}$

(over TLS)

Prekey bundle database

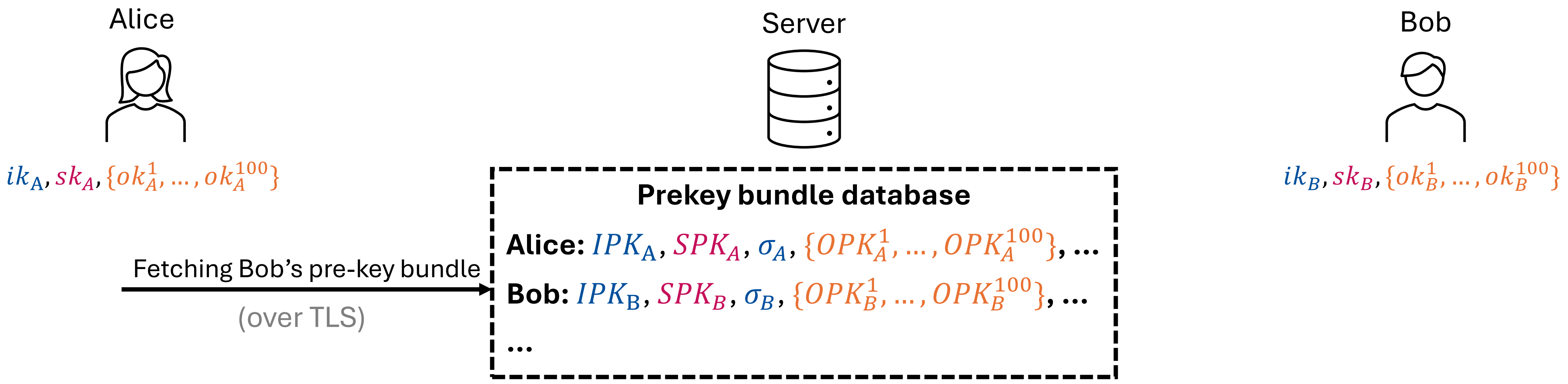
Alice: $IPK_\mathrm{A}, SPK_A, \sigma_A, \{OPK_A^1, \dots, OPK_A^{100}\}$**, …**

Bob: $IPK_\mathrm{B}, SPK_B, \sigma_B, \{OPK_B^1, \dots, OPK_B^{100}\}$**, …**

…

The X3DH Protocol – Pre-key Bundles

- When Alice communicates with Bob…



Alice

$ik_{\mathrm{A}}, sk_A, \{ok_A^1, \dots, ok_A^{100}\}$

Server

Bob

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

Fetching Bob’s pre-key bundle

(over TLS)

Prekey bundle database

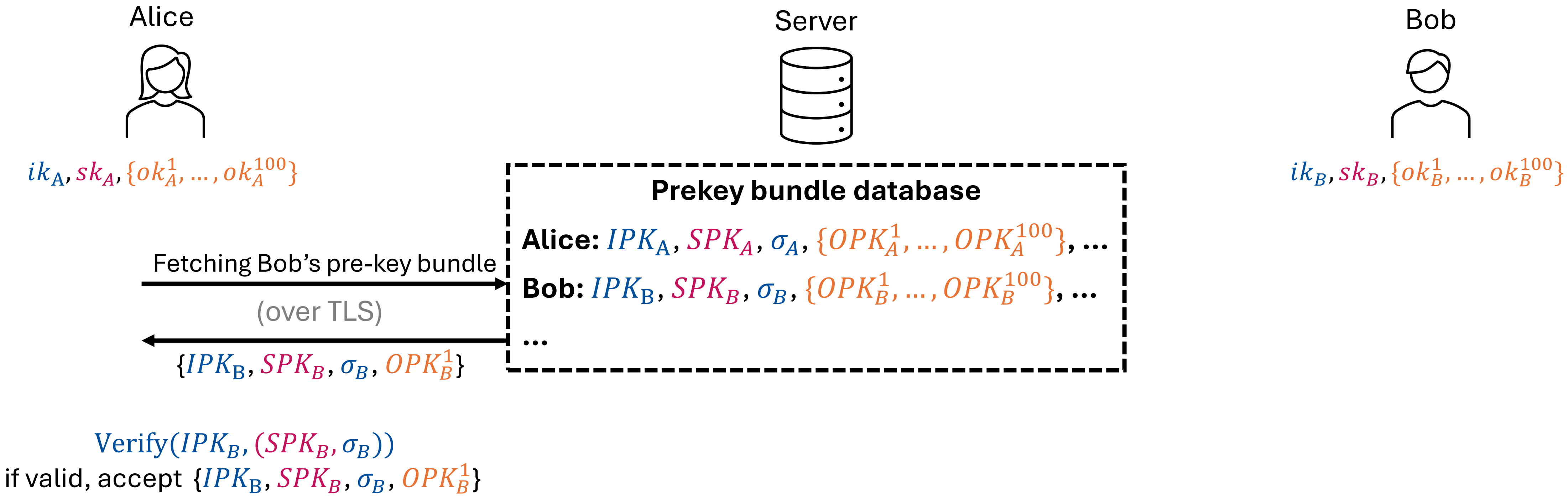
Alice: $IPK_{\mathrm{A}}, SPK_A, \sigma_A, \{OPK_A^1, \dots, OPK_A^{100}\}$**, …**

Bob: $IPK_{\mathrm{B}}, SPK_B, \sigma_B, \{OPK_B^1, \dots, OPK_B^{100}\}$**, …**

…

The X3DH Protocol – Pre-key Bundles

- When Alice communicates with Bob…



Alice

Server

Bob

$ik_A, sk_A, \{ok_A^1, \dots, ok_A^{100}\}$

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

Prekey bundle database

Alice: $IPK_A, SPK_A, \sigma_A, \{OPK_A^1, \dots, OPK_A^{100}\}$, **…**

Bob: $IPK_B, SPK_B, \sigma_B, \{OPK_B^1, \dots, OPK_B^{100}\}$, **…**

…

Fetching Bob's pre-key bundle

(over TLS)

$\{IPK_B, SPK_B, \sigma_B, OPK_B^1\}$

$\mathrm{Verify}(IPK_B, (SPK_B, \sigma_B))$

if valid, accept $\{IPK_B, SPK_B, \sigma_B, OPK_B^1\}$

The X3DH Protocol

- When Alice communicates with Bob...

Alice

$ik_{\mathrm{A}}, sk_A, \{ok_A^1, \dots, ok_A^{100}\}$

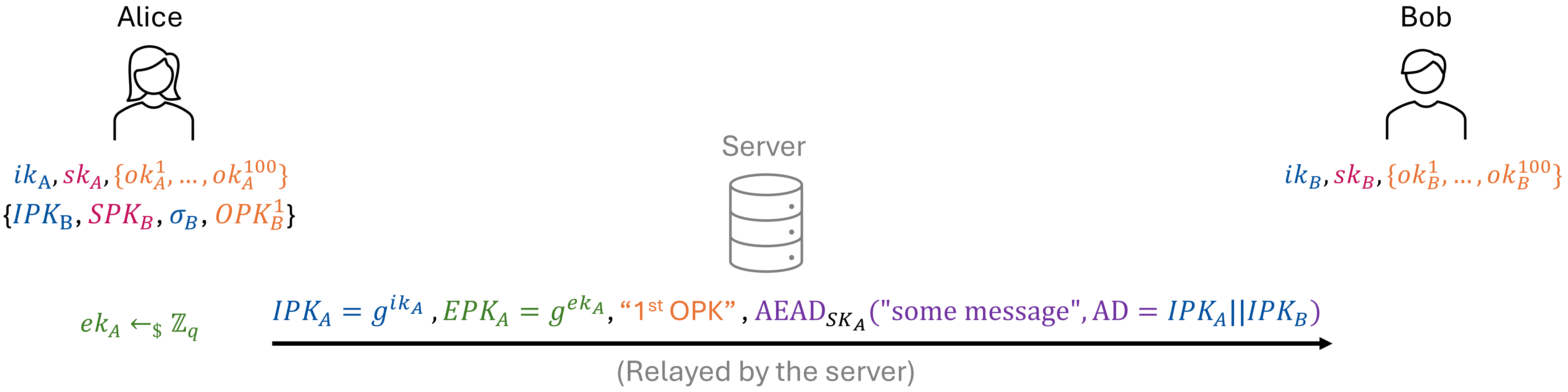
$\{IPK_{\mathrm{B}}, SPK_B, \sigma_B, OPK_B^1\}$

Bob

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

The X3DH Protocol

- When Alice communicates with Bob…



Alice

Server

Bob

$ik_A, sk_A, \{ok_A^1, \dots, ok_A^{100}\}$

$\{IPK_B, SPK_B, \sigma_B, OPK_B^1\}$

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

$ek_A \leftarrow_\$ \mathbb{Z}_q$

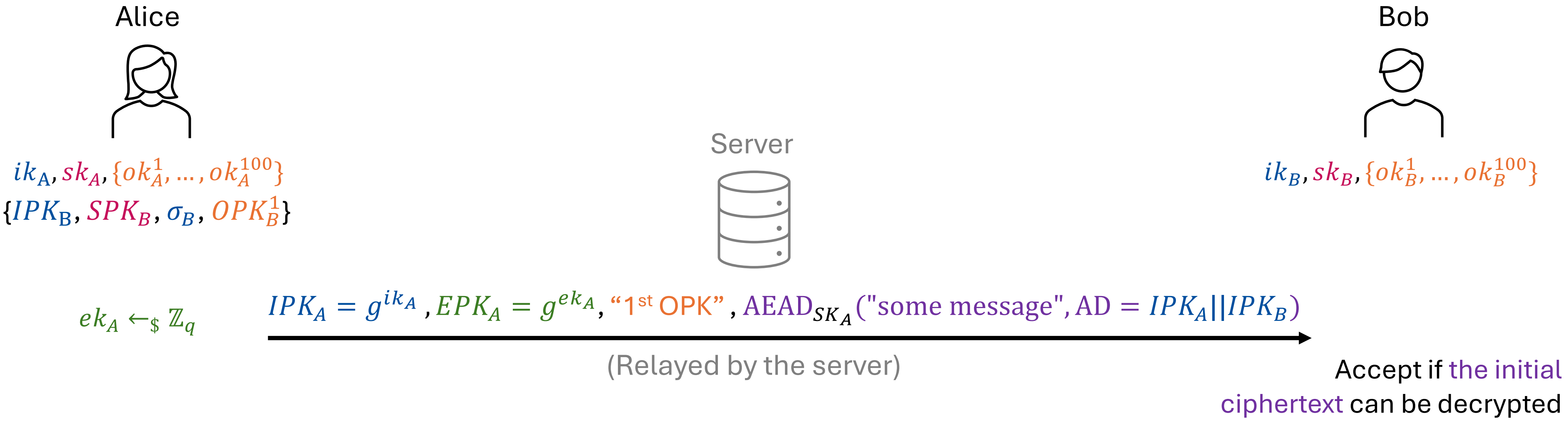
$IPK_A = g^{ik_A}, EPK_A = g^{ek_A}$, "1st OPK", $\mathrm{AEAD}_{SK_A}(\text{"some message"}, \mathrm{AD} = IPK_A || IPK_B)$

(Relayed by the server)

$SK_A =$ X3DH_Key_Alice($ik_A, ek_A, IPK_B, SPK_B, OPK_B^1$)
(…will be explained later)

The X3DH Protocol

- When Bob receives messages (which is actually relayed by the server) from Alice...



Alice

$ik_A, sk_A, \{ok_A^1, \dots, ok_A^{100}\}$

$\{IPK_B, SPK_B, \sigma_B, OPK_B^1\}$

Server

Bob

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

$ek_A \leftarrow_\$ \mathbb{Z}_q$

$IPK_A = g^{ik_A}, EPK_A = g^{ek_A}$, “1st OPK” , $\mathrm{AEAD}_{SK_A}(\text{"some message"}, \mathrm{AD} = IPK_A || IPK_B)$

(Relayed by the server)

Accept if the initial ciphertext can be decrypted

SK_A = X3DH_Key_Alice($ik_A, ek_A, IPK_B, SPK_B, OPK_B^1$)
(...will be explained later)

SK_B = X3DH_Key_Bob($IPK_A, EPK_A, ik_B, sk_B, ok_B^1$)
(...will be explained later)

The X3DH Protocol

- How the X3DH protocol computes a shared secret...

Alice

```
X3DH_Key_Alice($ik_A, ek_A, IPK_B, SPK_B, OPK_B$)
1. $\mathrm{DH}_1 = SPK_B^{ik_A}$
2. $\mathrm{DH}_2 = IPK_B^{ek_A}$
3. $\mathrm{DH}_3 = SPK_B^{ek_A}$
4. $\mathrm{DH}_4 = (OPK_B)^{ek_A}$
5. return $SK_A = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$
```

Bob

```
$SK_B$ = X3DH_Key_Bob($IPK_A, EPK_A, ik_B, sk_B, ok_B$)
1. $\mathrm{DH}_1 = IPK_A^{sk_B}$
2. $\mathrm{DH}_2 = EPK_A^{ik_B}$
3. $\mathrm{DH}_3 = EPK_A^{sk_B}$
4. $\mathrm{DH}_4 = EPK_A^{ok_B}$
5. return $SK_B = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$
```

The X3DH Protocol

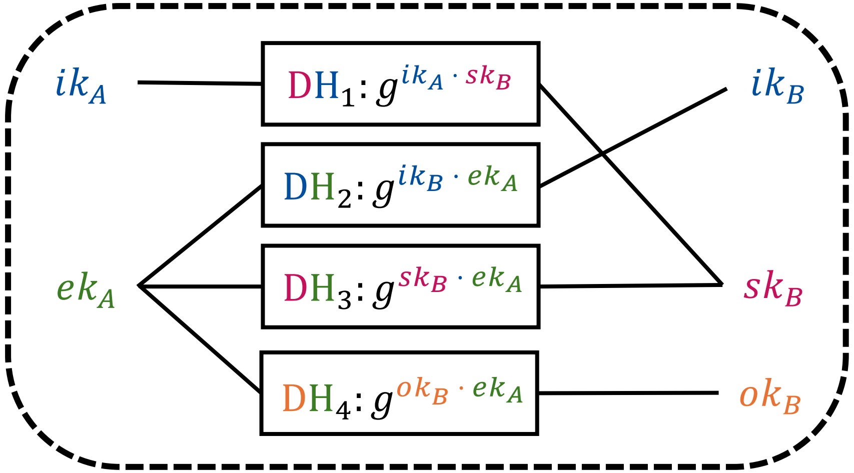
- How the X3DH protocol computes a shared secret...

Alice

Bob

X3DH_Key_Alice(ik_A, ek_A, IPK_{B}, SPK_B, OPK_B)

1. $\mathrm{DH}_1 = SPK_B^{ik_A}$
2. $\mathrm{DH}_2 = IPK_{\mathrm{B}}^{ek_A}$
3. $\mathrm{DH}_3 = SPK_B^{ek_A}$
4. $\mathrm{DH}_4 = (OPK_B)^{ek_A}$
5. $SK_{\mathrm{A}} = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$



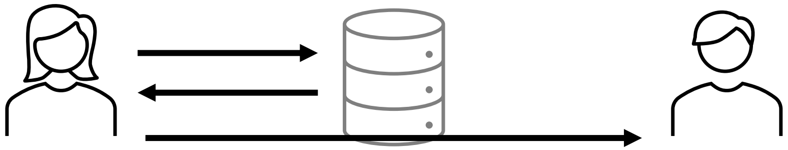
ik_A
ek_A
$\mathrm{DH}_1: g^{ik_A \cdot sk_B}$
$\mathrm{DH}_2: g^{ik_B \cdot ek_A}$
$\mathrm{DH}_3: g^{sk_B \cdot ek_A}$
$\mathrm{DH}_4: g^{ok_B \cdot ek_A}$
ik_B
sk_B
ok_B

X3DH_Key_Bob(IPK_{A}, EPK_A, ik_{B}, sk_B, ok_B)

1. $\mathrm{DH}_1 = IPK_A^{sk_B}$
2. $\mathrm{DH}_2 = EPK_A^{ik_B}$
3. $\mathrm{DH}_3 = EPK_A^{sk_B}$
4. $\mathrm{DH}_4 = EPK_A^{ok_B}$
5. $SK_{\mathrm{B}} = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

The X3DH Protocol

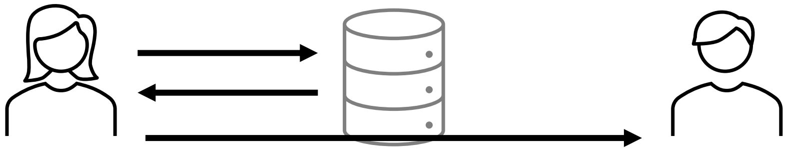
- Based on (EC)DH
- Trusted server required
    - Store public keys, relay messages, ...
    - But it cannot decrypt ciphertexts...
- 0-RTT (Zero round-trip time)
    - Immediate message sending without waiting for a response
- Support offline communication
    - Can be executed even if Bob (the receiver) is offline
    - Offline messages (encrypted) will be stored in the server until Bob is online again
- Mutual Authentication, Forward Secrecy, ...
    - In this course, we focus on *How it works* rather than *Why it is secure*...



X3DH

The X3DH Protocol

- Based on (EC)DH
- Trusted server required
  - Store public keys, relay messages, ...
  - But it cannot decrypt ciphertexts...
- 0-RTT (Zero round-trip time)
  - Immediate message sending without waiting for a response
- Support offline communication
  - Can be executed even if Bob (the receiver) is offline
  - Offline messages (encrypted) will be stored in the server unt
- Mutual Authentication, Forward Secrecy, ...
  - In this course, we focus on *How it works* rather than *Why it is secure*...



X3DH

A note: Do not confuse X3DH with TLS

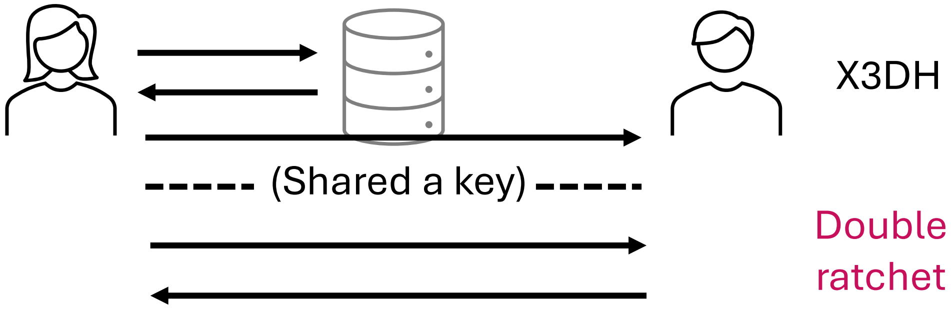
Different primary goals/settings:

X3DH: secure messaging between users, rely on trusted pre-shared public keys...

TLS: secure connections with a server, rely on trusted CAs and use certificates...

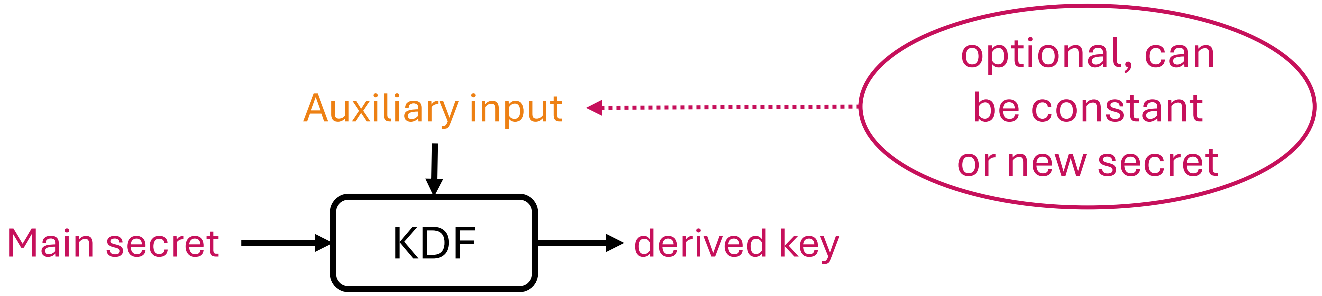
Double Ratchet

- After completing X3DH…
- … we use **Double Ratchet** to:
  - Encrypt messages + updates the shared key
  - ~~Encrypt messages using the same shared key~~
  - **Diffie-Hellman Ratchet** + **Symmetric-key Ratchet**
- Essential for forward/backward secrecy



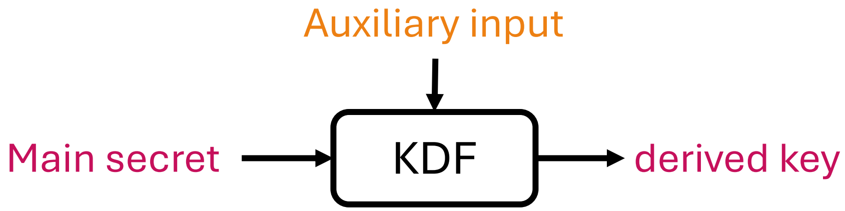
Symmetric-key Ratchet

- KDF chain
  - KDF: Key derivation function

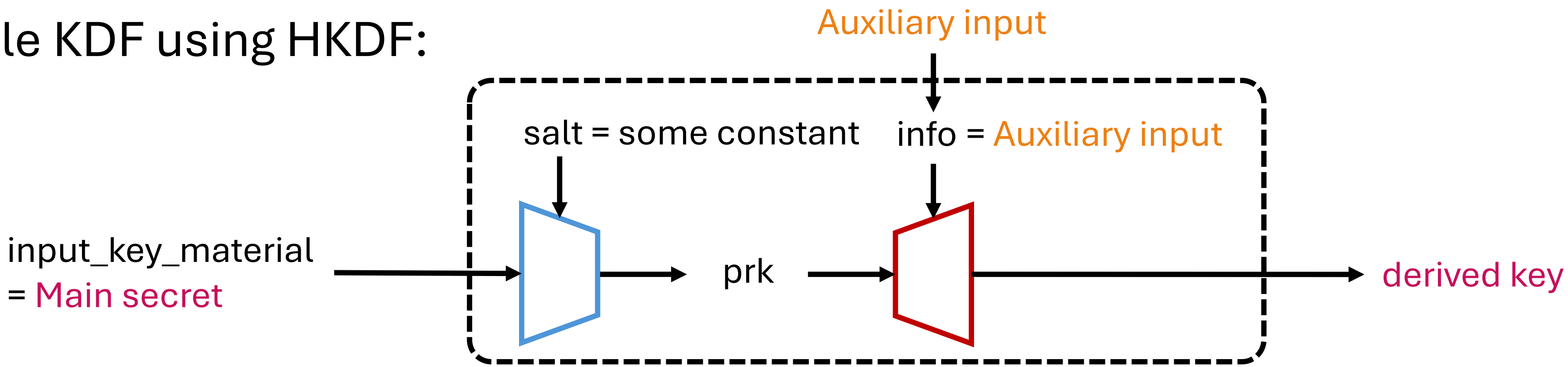


Symmetric-key Ratchet

- KDF chain
  - KDF: Key derivation function



- Example KDF using HKDF:

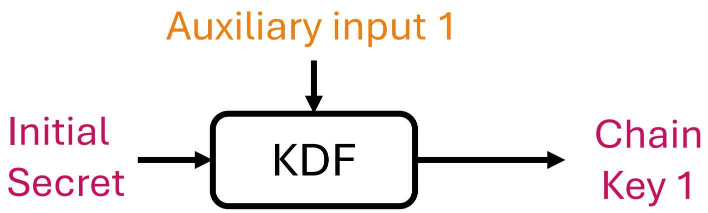


1. prk = HKDF.**Extract**(input_key_material = Main secret, salt = some constant)

2. derived key = HKDF.**Expand**(prk, Auxiliary input)

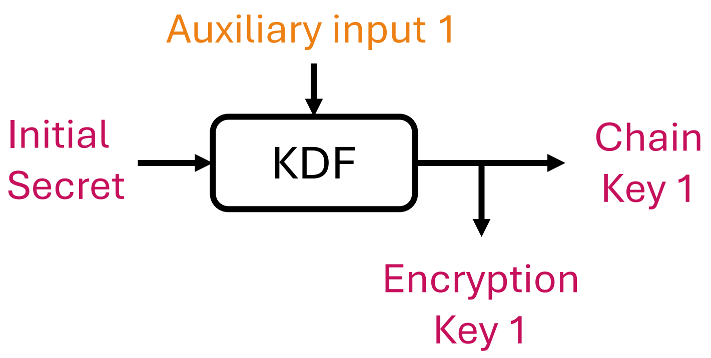
Symmetric-key Ratchet

- KDF chain
  - KDF: Key derivation function



Symmetric-key Ratchet

- KDF chain
  - KDF: Key derivation function

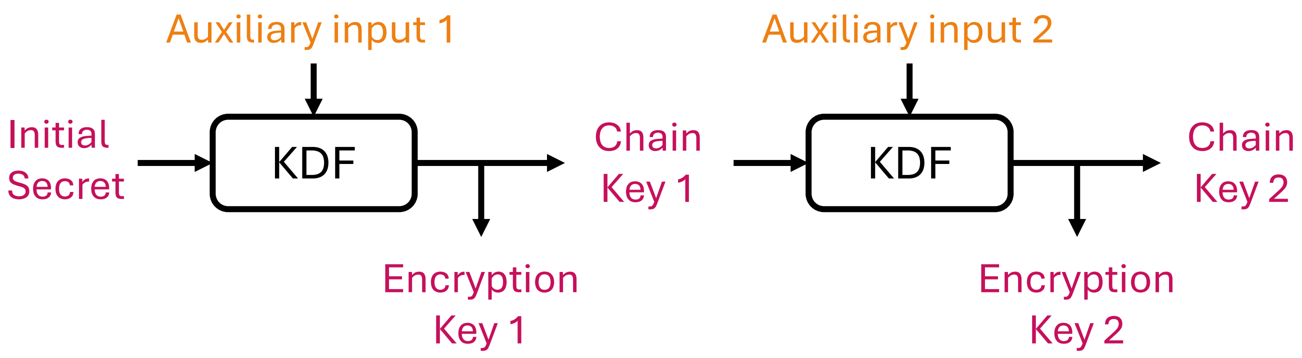


For key chaining…

For encryption…

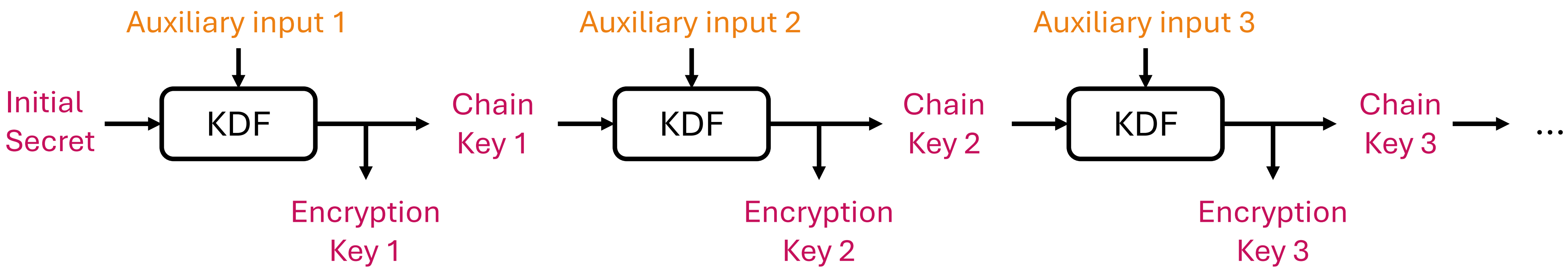
Symmetric-key Ratchet

- KDF chain
  - KDF: Key derivation function



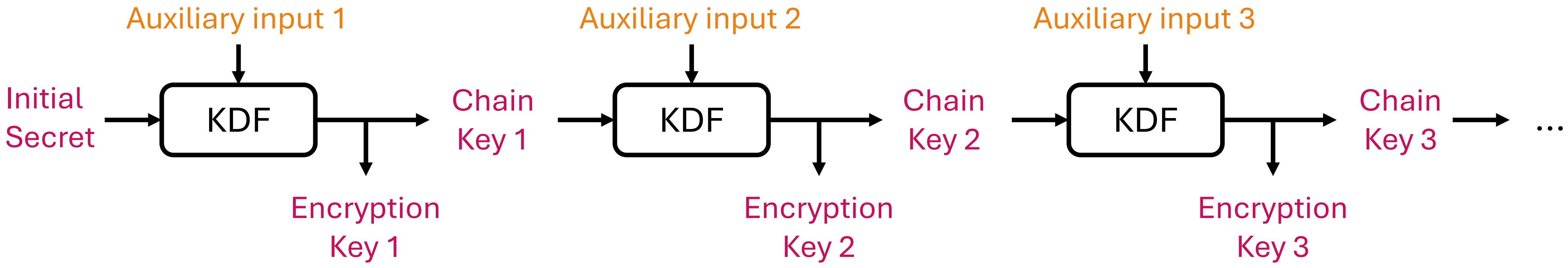
Symmetric-key Ratchet

- KDF chain
  - KDF: Key derivation function



Symmetric-key Ratchet

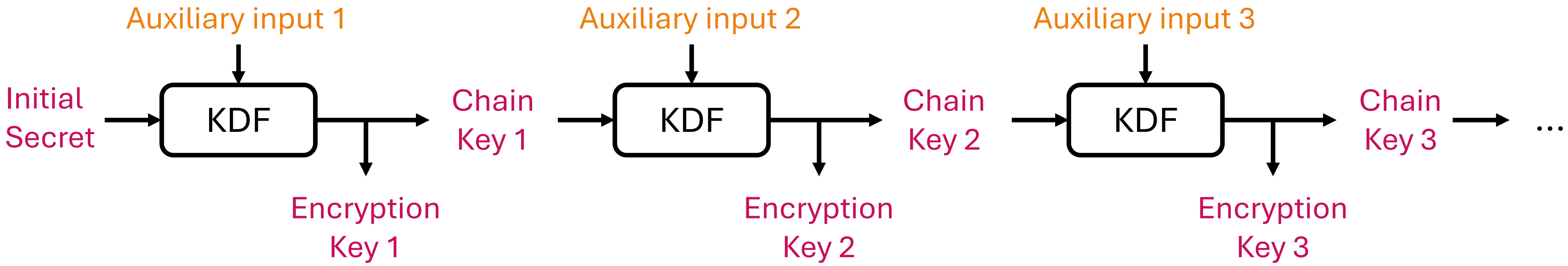
- KDF chain
  - KDF: Key derivation function



- Use Key Chain to encrypt messages

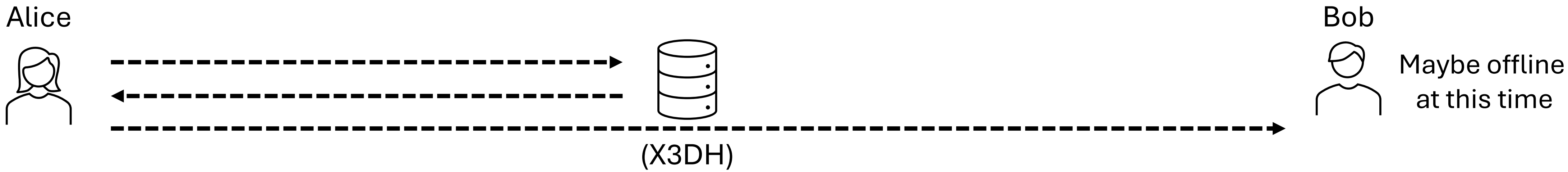
Symmetric-key Ratchet

- KDF chain
    - KDF: Key derivation function



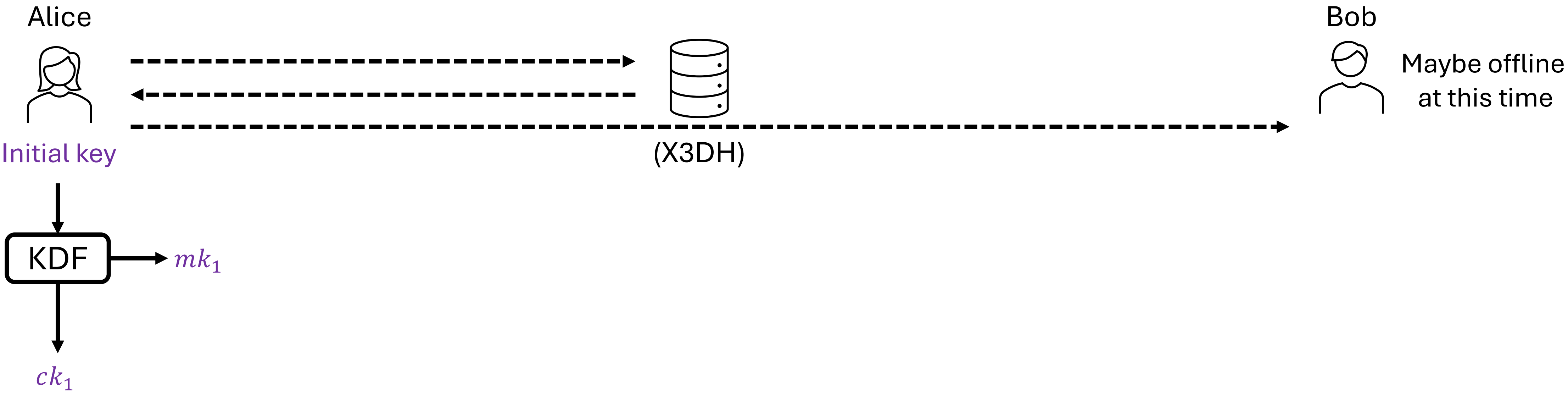
Symmetric-key Ratchet

- A toy example of secure messaging using symmetric-key ratchet



Symmetric-key Ratchet

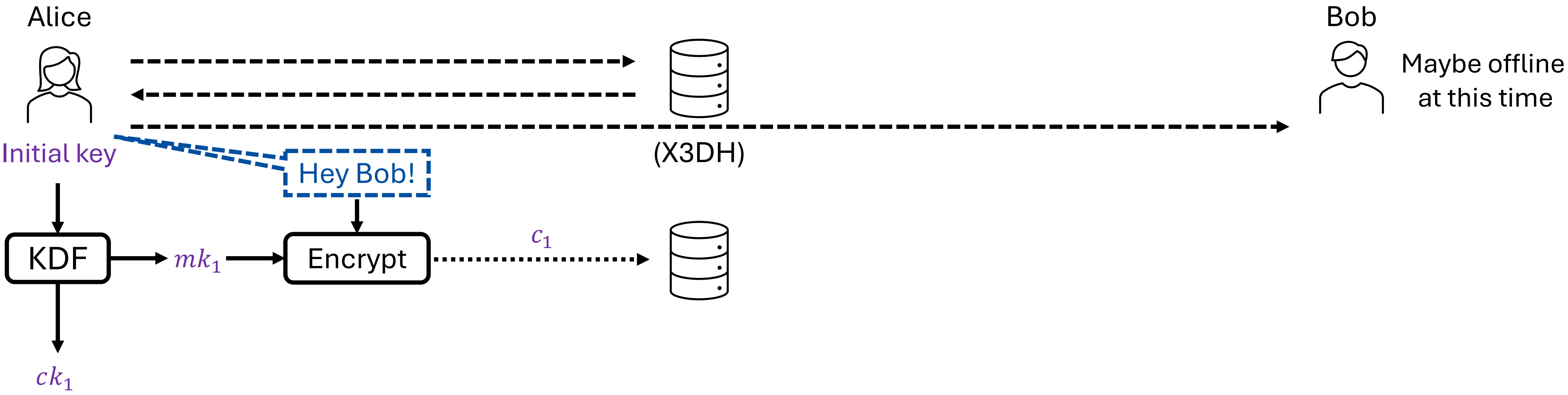
- A toy example of secure messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

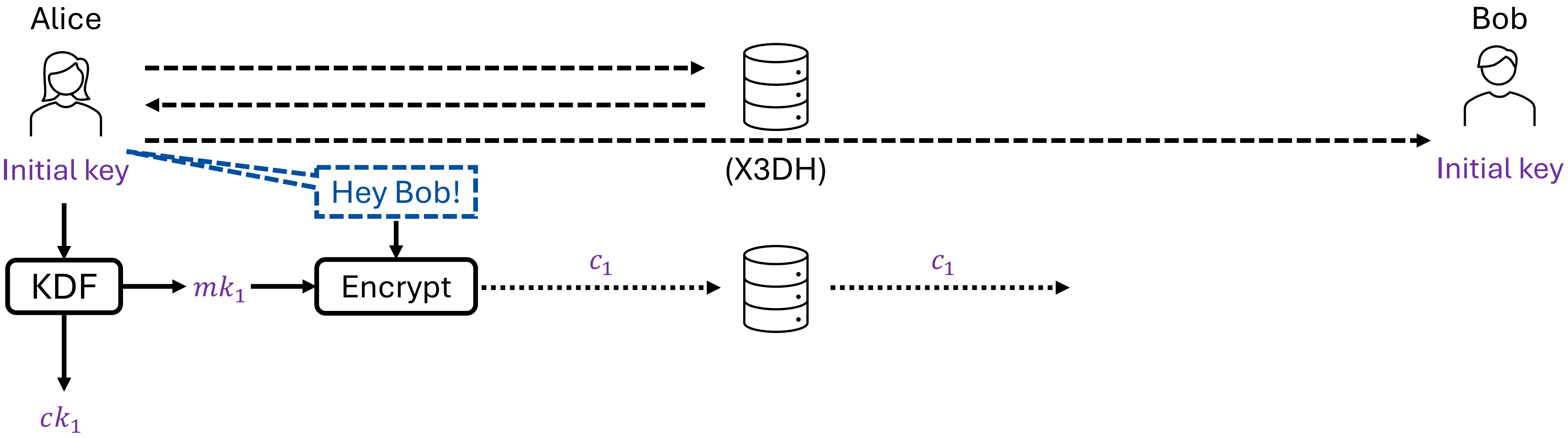
- A toy example of secure messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

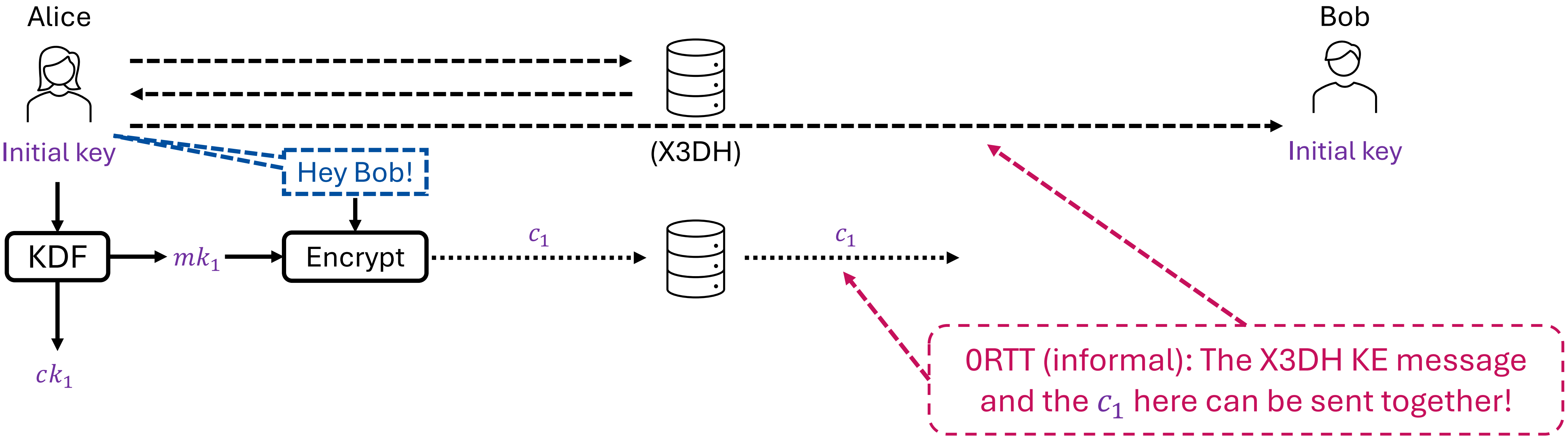
- A toy example of secure messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

- A toy example of secure messaging using symmetric-key ratchet

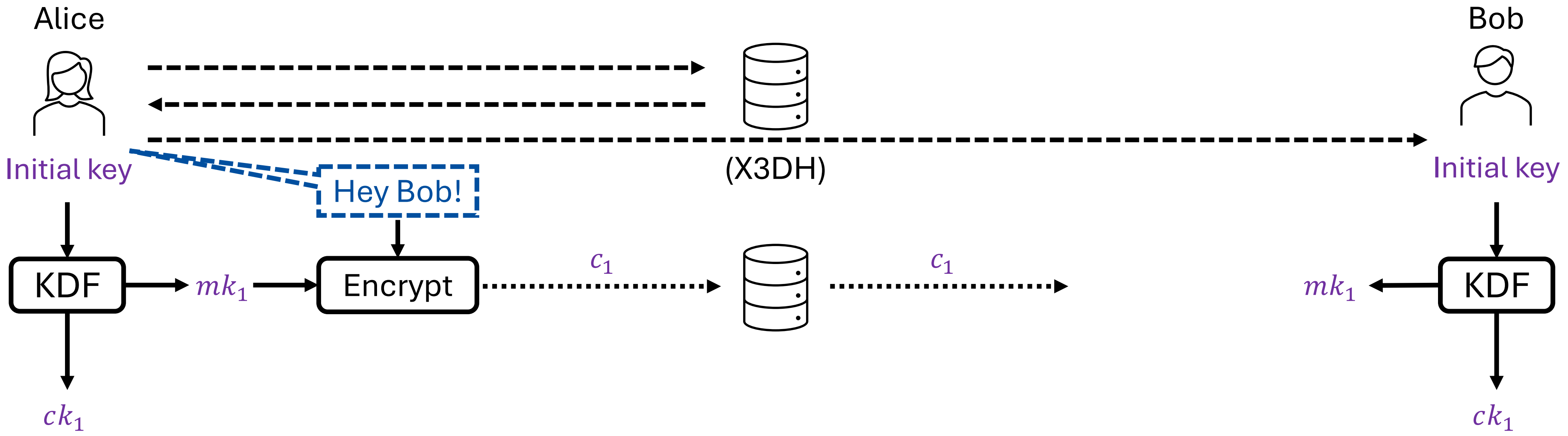


0RTT (informal): The X3DH KE message and the c_1 here can be sent together!

*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

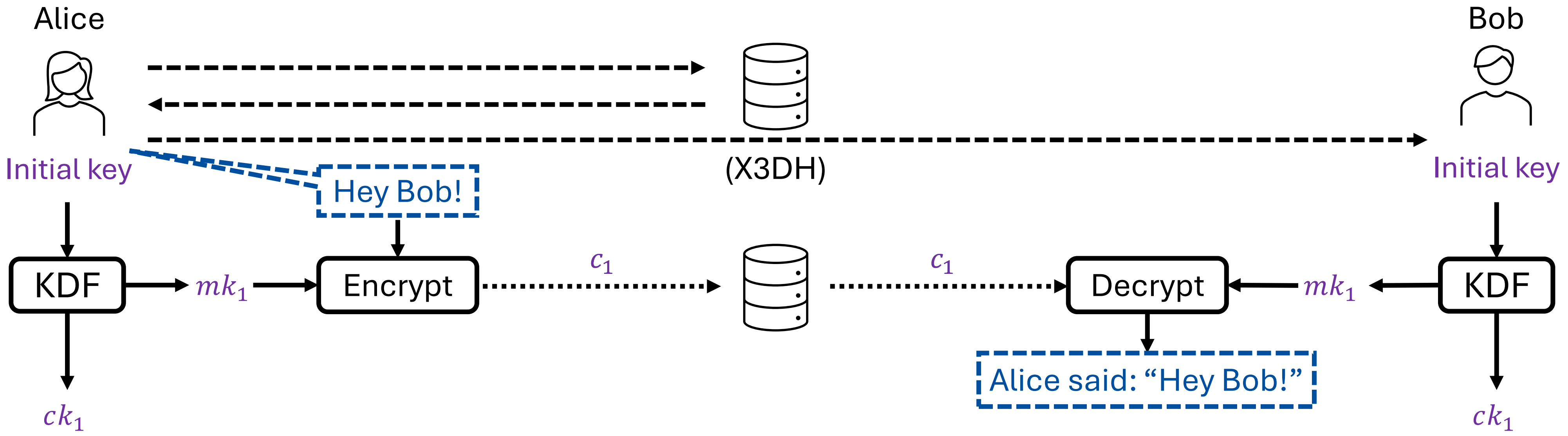
- A toy example of instant messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

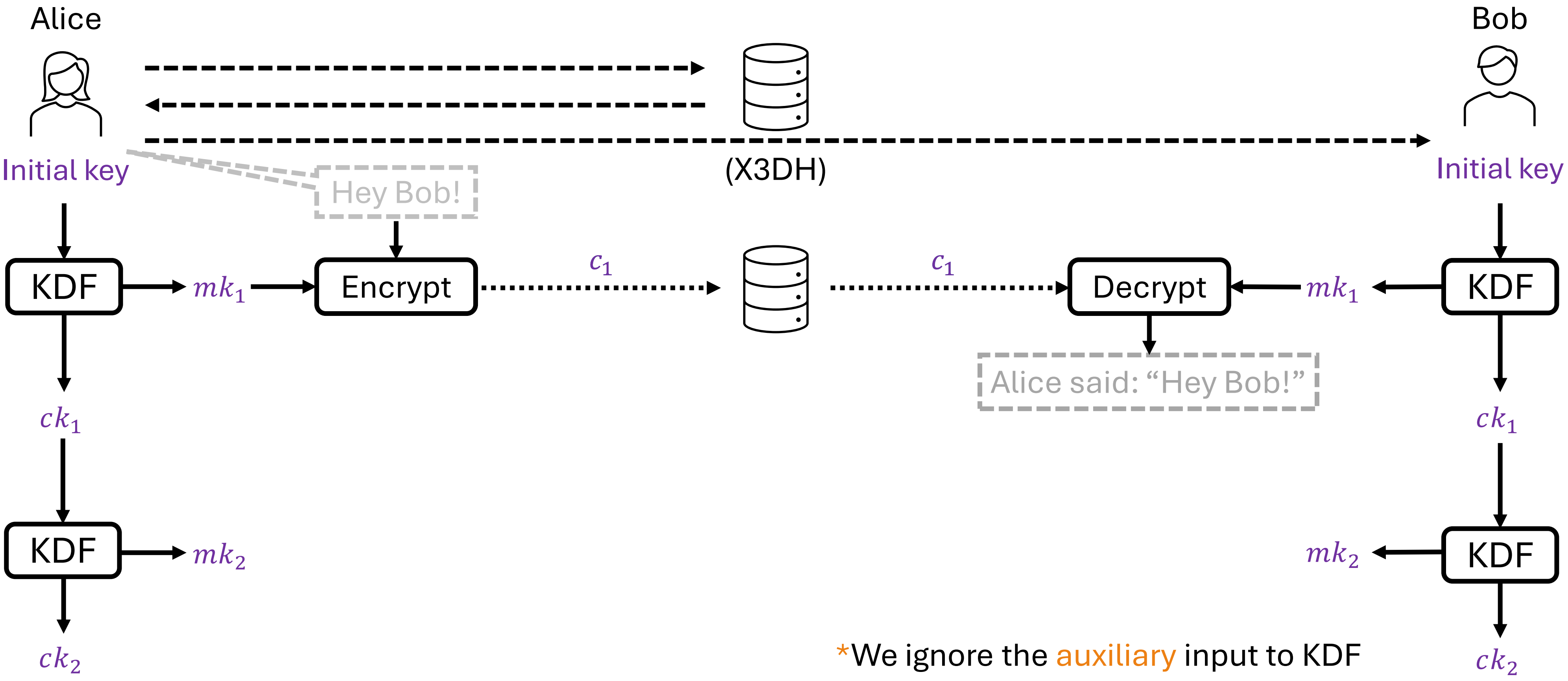
- A toy example of secure messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

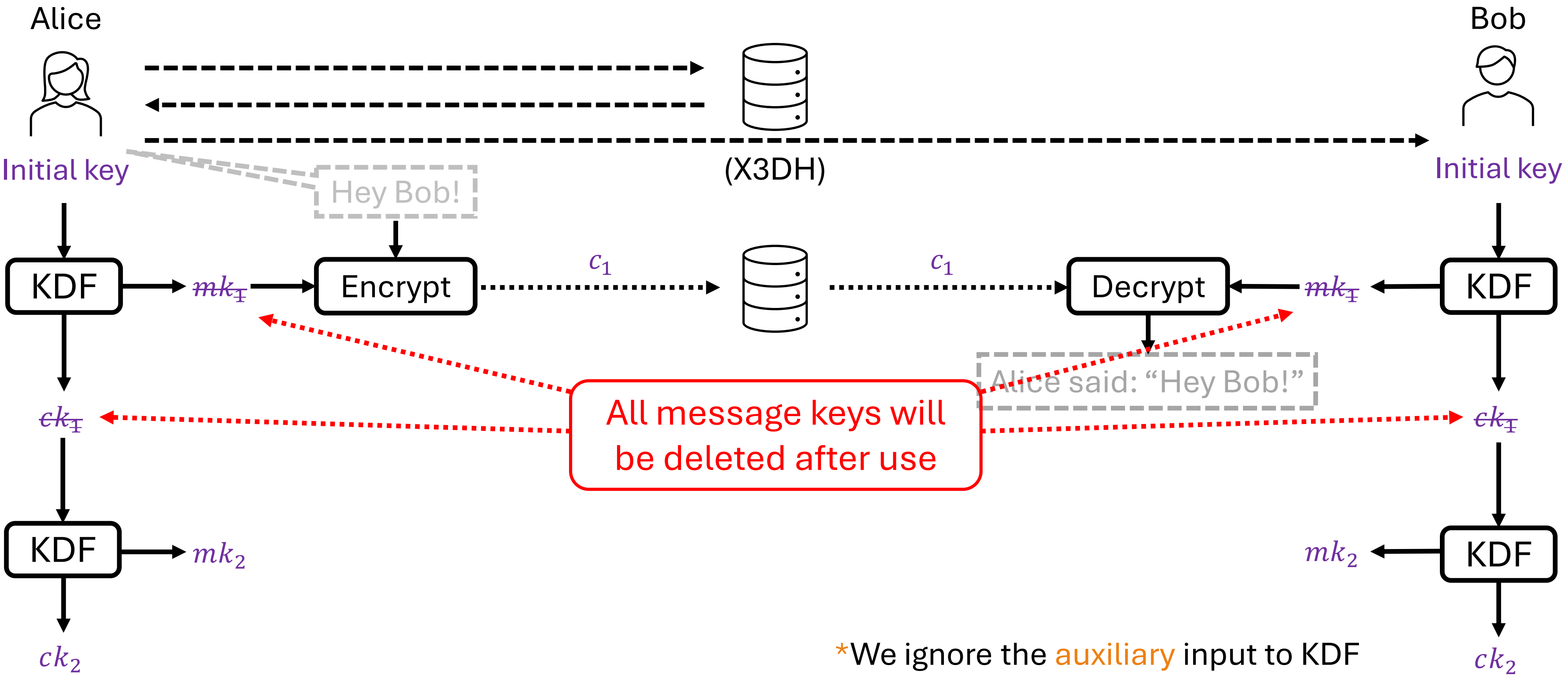
- A toy example of secure messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

- A toy example of secure messaging using symmetric-key ratchet

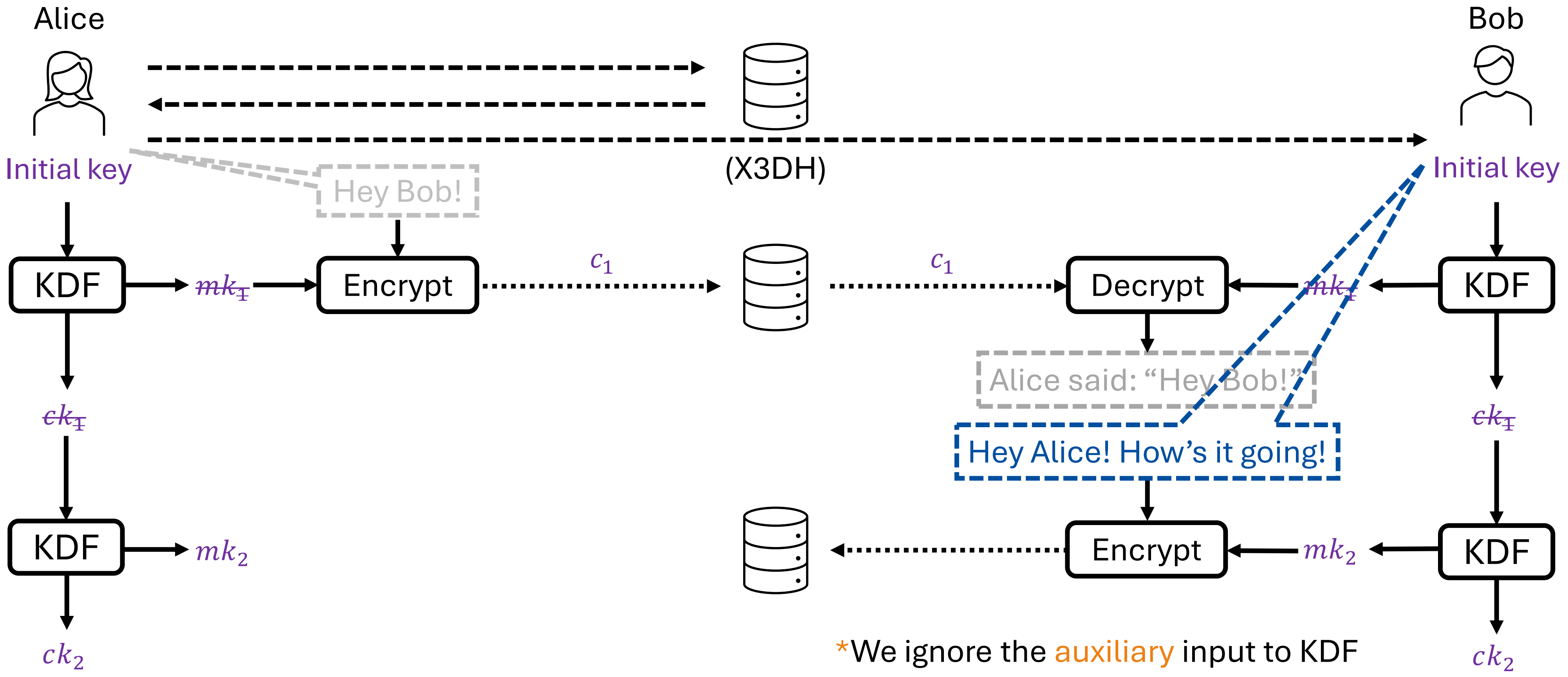


All message keys will be deleted after use

*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

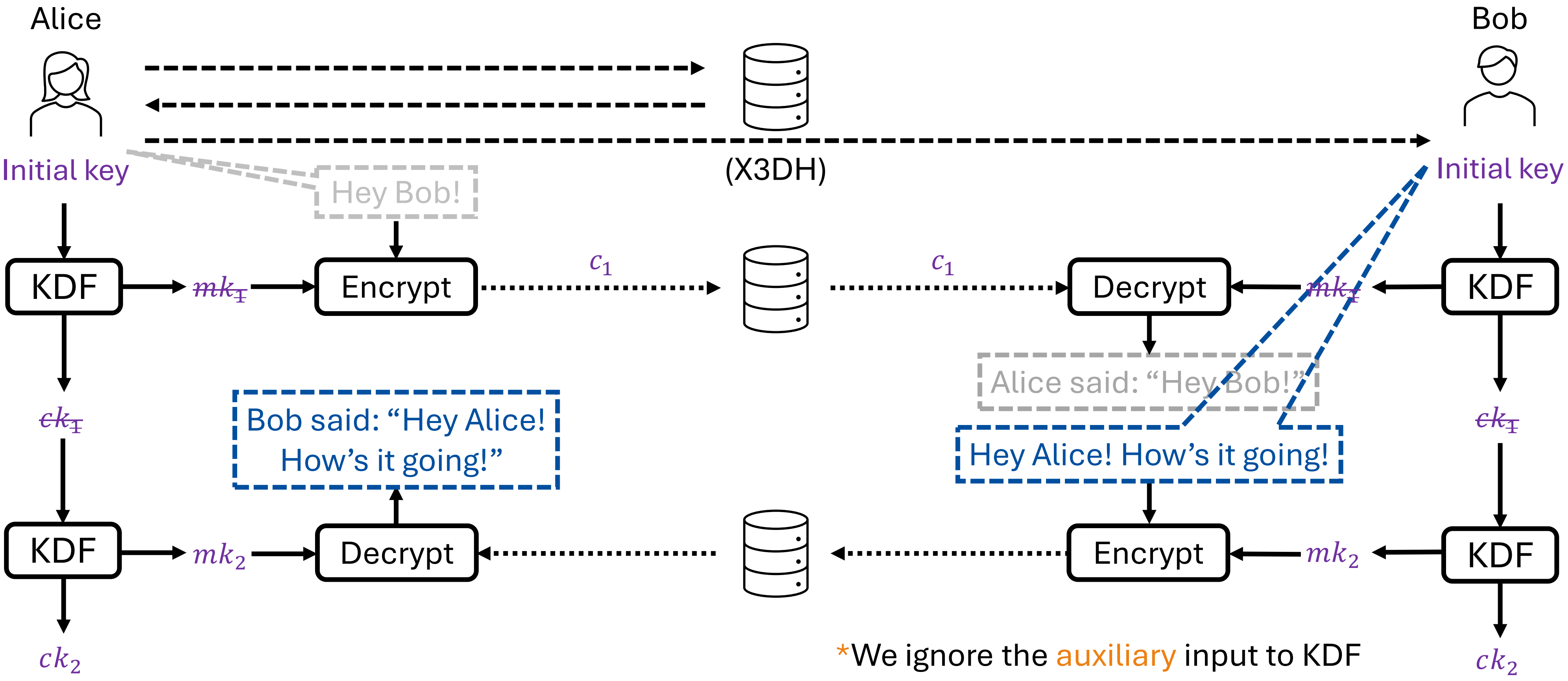
- A toy example of instant messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

Symmetric-key Ratchet

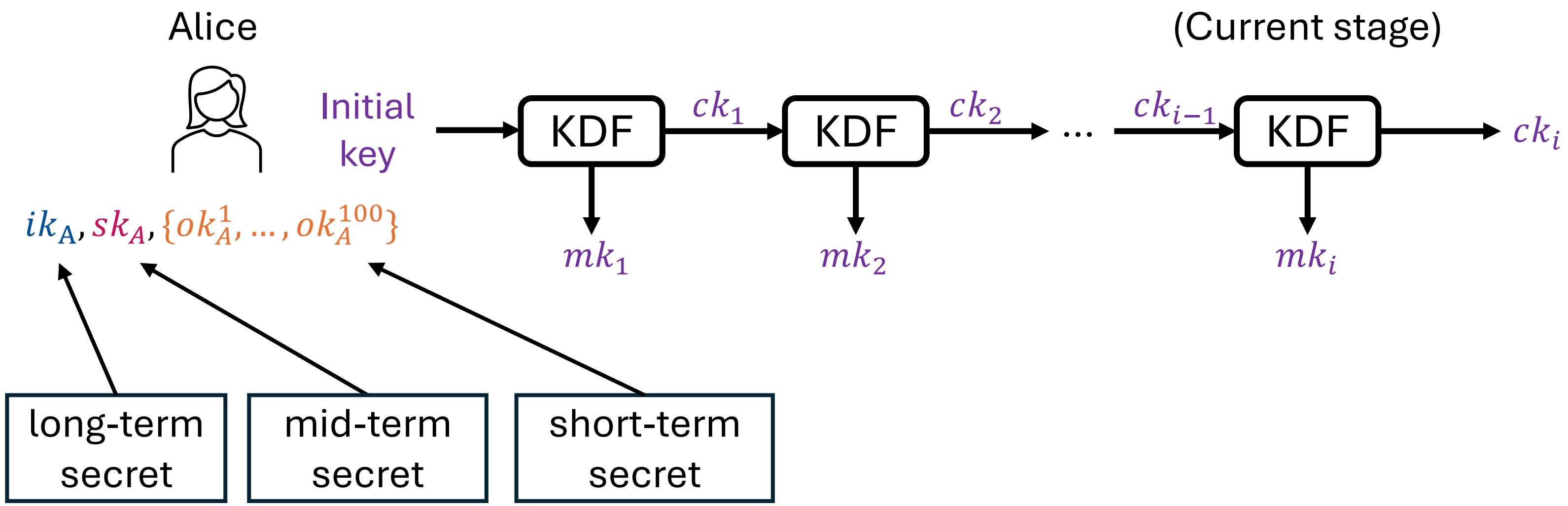
- A toy example of secure messaging using symmetric-key ratchet



*We ignore the auxiliary input to KDF

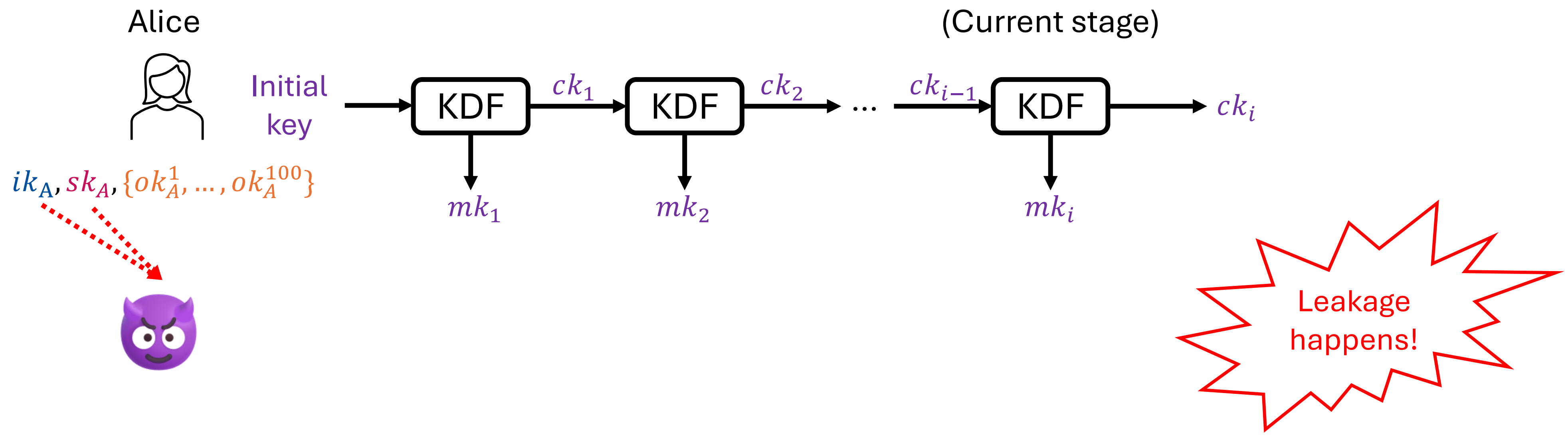
Forward Secrecy

- Long-term secret keys are compromised, but past communication remains secure…



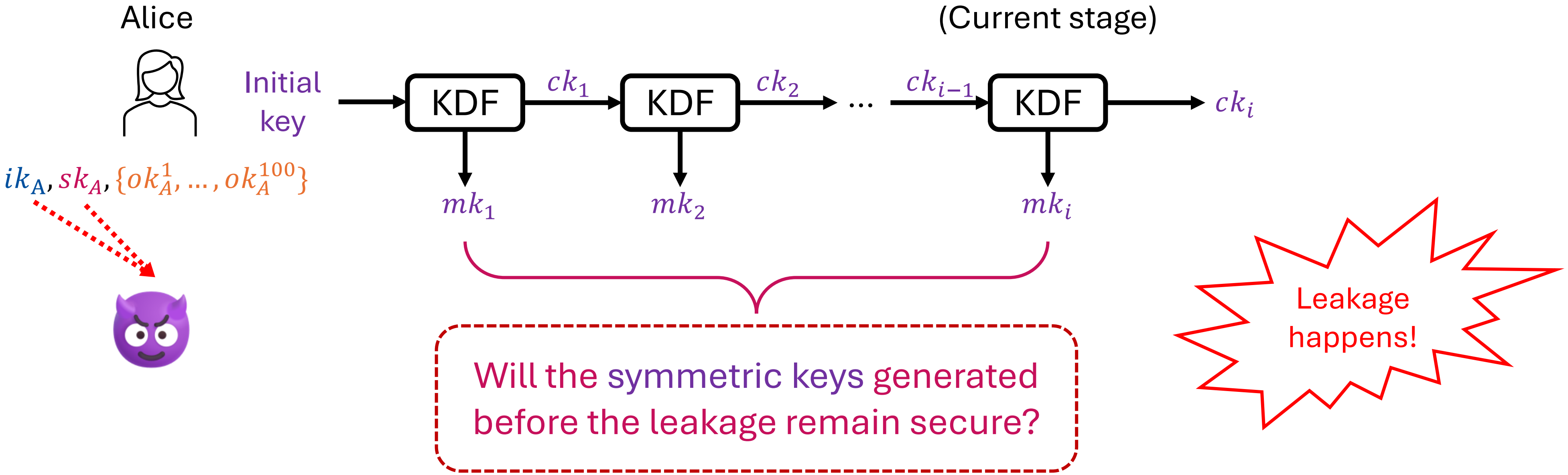
Forward Secrecy

- Long-term secret keys are compromised, but past communication remains secure…



Forward Secrecy

- Long-term secret keys are compromised, but past communication remains secure…



Will the symmetric keys generated before the leakage remain secure?

Forward Secrecy

- Recall: How the X3DH protocol computes a shared secret...

Alice
Leakage!
Bob

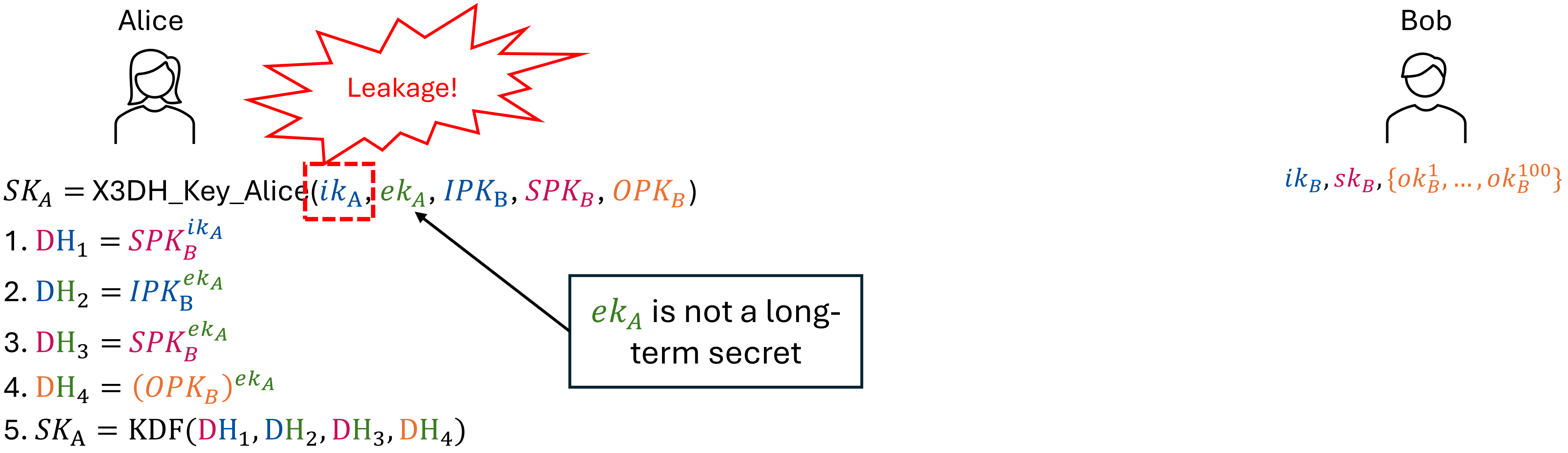
$SK_A = \text{X3DH\_Key\_Alice}(\boxed{ik_A,} ek_A, IPK_B, SPK_B, OPK_B)$

$ik_B, sk_B, \{ok_B^1, \ldots, ok_B^{100}\}$

1. $\mathrm{DH}_1 = SPK_B^{ik_A}$
2. $\mathrm{DH}_2 = IPK_B^{ek_A}$
3. $\mathrm{DH}_3 = SPK_B^{ek_A}$
4. $\mathrm{DH}_4 = (OPK_B)^{ek_A}$
5. $SK_A = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

Forward Secrecy

- Recall: How the X3DH protocol computes a shared secret...



Alice

Leakage!

$SK_A = \text{X3DH\_Key\_Alice}(\boxed{ik_A,}\ ek_A, IPK_B, SPK_B, OPK_B)$

1. $\mathrm{DH}_1 = SPK_B^{ik_A}$
2. $\mathrm{DH}_2 = IPK_B^{ek_A}$
3. $\mathrm{DH}_3 = SPK_B^{ek_A}$
4. $\mathrm{DH}_4 = (OPK_B)^{ek_A}$
5. $SK_A = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

ek_A is not a long-term secret

Bob

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

Forward Secrecy

- Recall: How the X3DH protocol computes a shared secret...

Alice

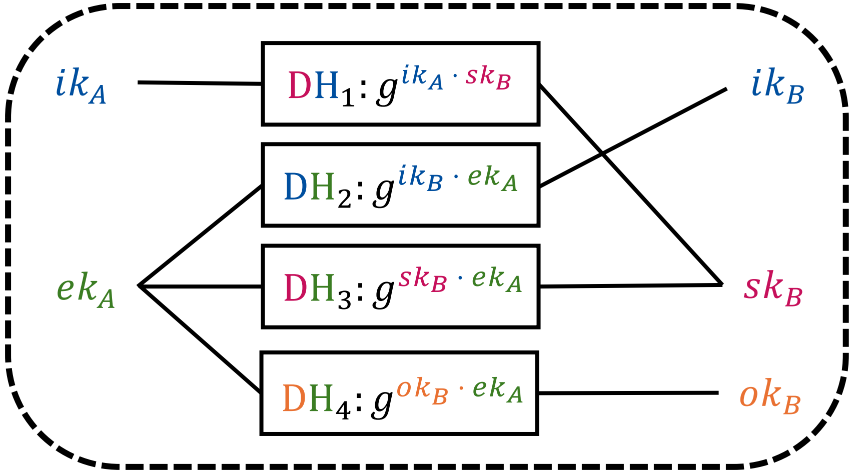
Bob

Leakage!

$SK_A = \text{X3DH\_Key\_Alice}(\boxed{ik_A,} ek_A, IPK_B, SPK_B, OPK_B)$

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

1. $\mathrm{DH}_1 = SPK_B^{ik_A}$
2. $\mathrm{DH}_2 = IPK_B^{ek_A}$
3. $\mathrm{DH}_3 = SPK_B^{ek_A}$
4. $\mathrm{DH}_4 = (OPK_B)^{ek_A}$
5. $SK_A = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$


ik_A
ek_A
$\mathrm{DH}_1: g^{ik_A \cdot sk_B}$
$\mathrm{DH}_2: g^{ik_B \cdot ek_A}$
$\mathrm{DH}_3: g^{sk_B \cdot ek_A}$
$\mathrm{DH}_4: g^{ok_B \cdot ek_A}$
ik_B
sk_B
ok_B

Forward Secrecy

- Recall: How the X3DH protocol computes a shared secret...

Alice

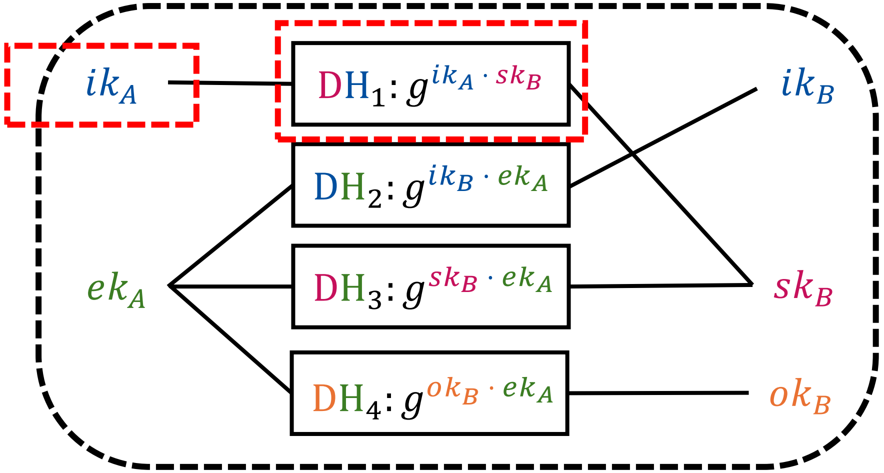
Leakage!

Bob

$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$

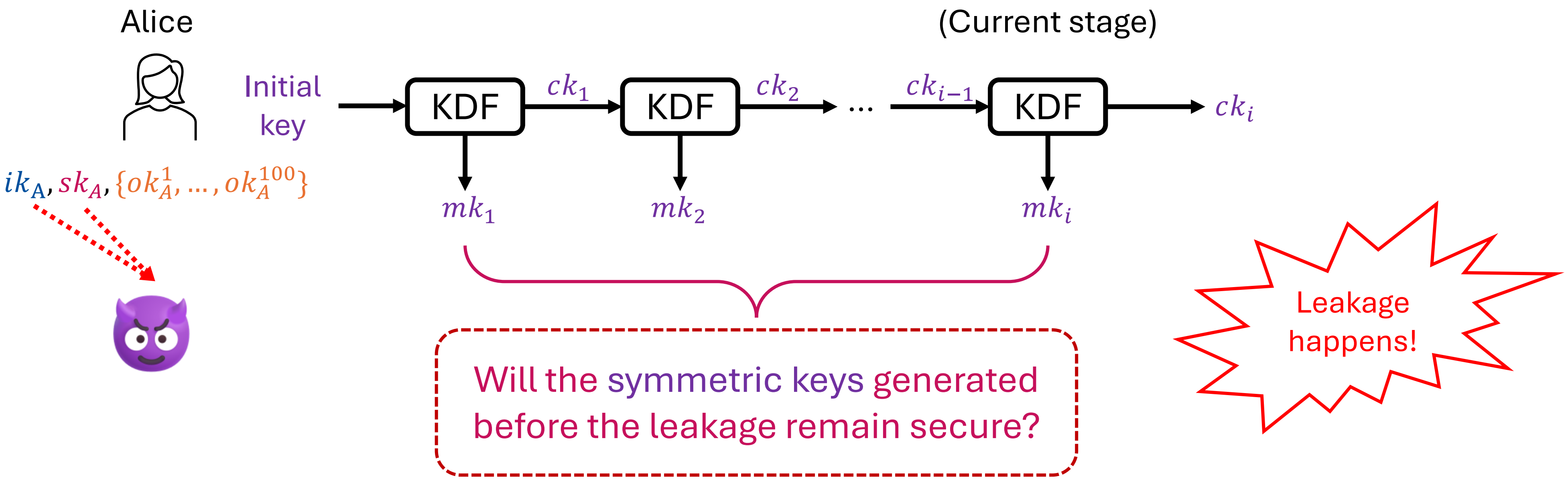
SK_A = X3DH_Key_Alice(ik_A, ek_A, IPK_B, SPK_B, OPK_B)

1. $\mathrm{DH}_1 = SPK_B^{ik_A}$
2. $\mathrm{DH}_2 = IPK_B^{ek_A}$
3. $\mathrm{DH}_3 = SPK_B^{ek_A}$
4. $\mathrm{DH}_4 = (OPK_B)^{ek_A}$
5. $SK_A = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$


ik_A
$\mathrm{DH}_1 : g^{ik_A \cdot sk_B}$
ik_B
$\mathrm{DH}_2 : g^{ik_B \cdot ek_A}$
ek_A
$\mathrm{DH}_3 : g^{sk_B \cdot ek_A}$
sk_B
$\mathrm{DH}_4 : g^{ok_B \cdot ek_A}$
ok_B

Forward Secrecy

- Long-term secret keys are compromised, but past communication remains secure…

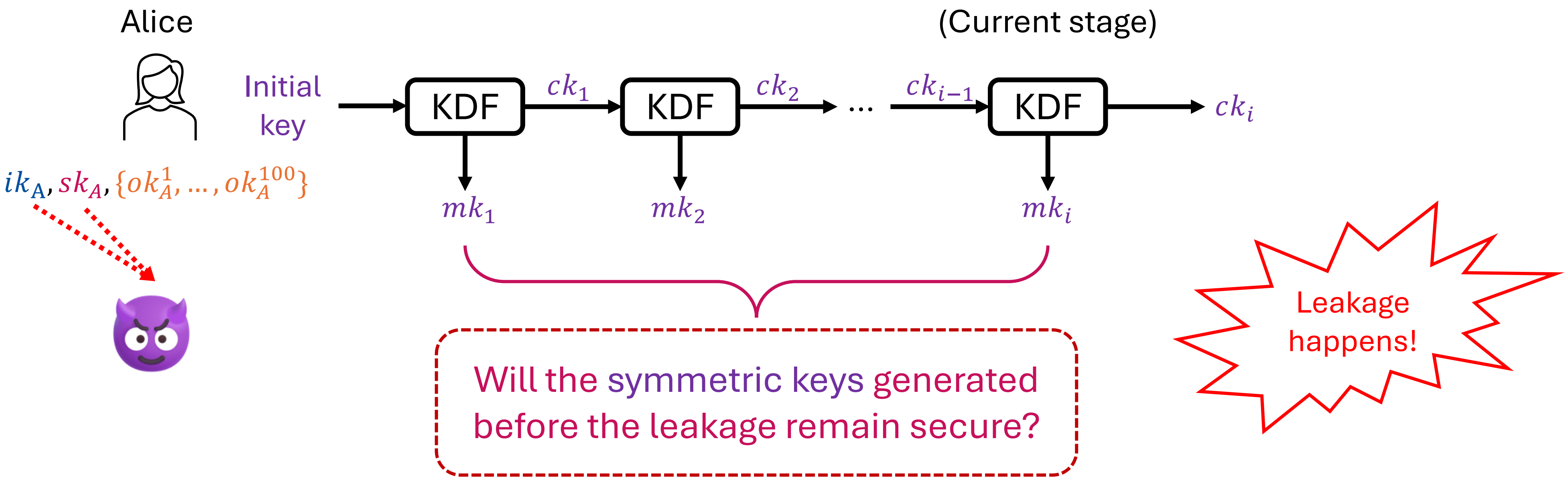


Will the symmetric keys generated before the leakage remain secure?

Initial_key (of X3DH)= $\mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

Forward Secrecy

- Long-term secret keys are compromised, but past communication remains secure…



Will the symmetric keys generated before the leakage remain secure?

Leakage happens!

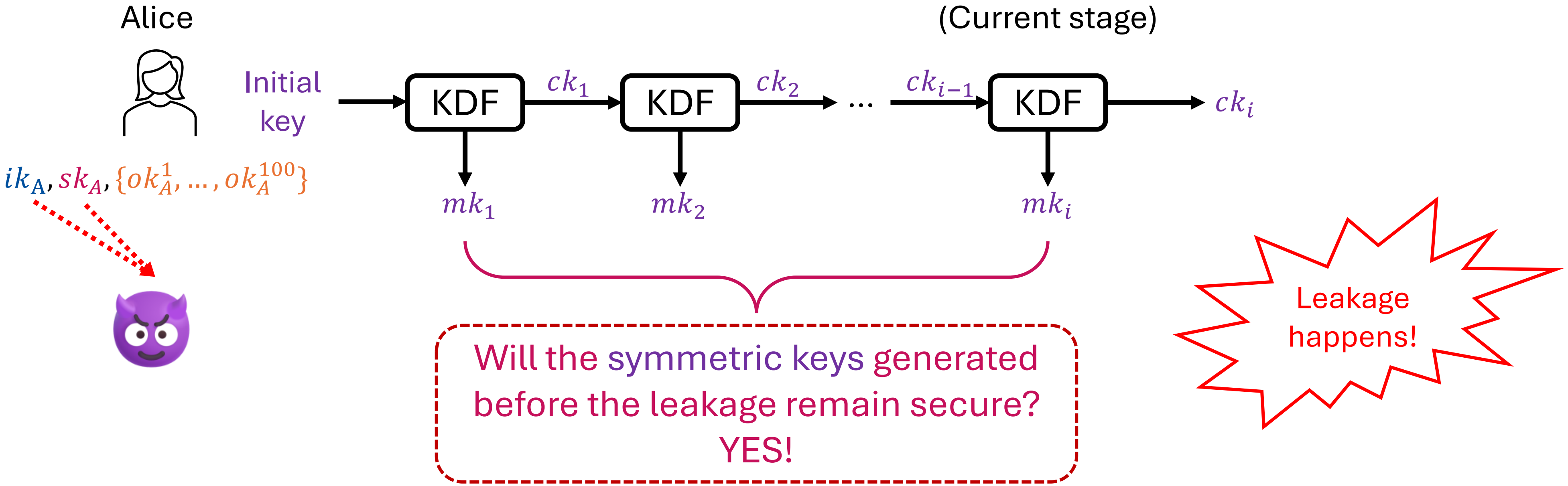
Initial_key (of X3DH)= $\mathrm{KDF}(\boxed{\mathrm{DH}_1}, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

DH_1 is not secure

But DH_2,3,4 remain secret

Forward Secrecy

- Long-term secret keys are compromised, but past communication remains secure...



Will the symmetric keys generated before the leakage remain secure? YES!

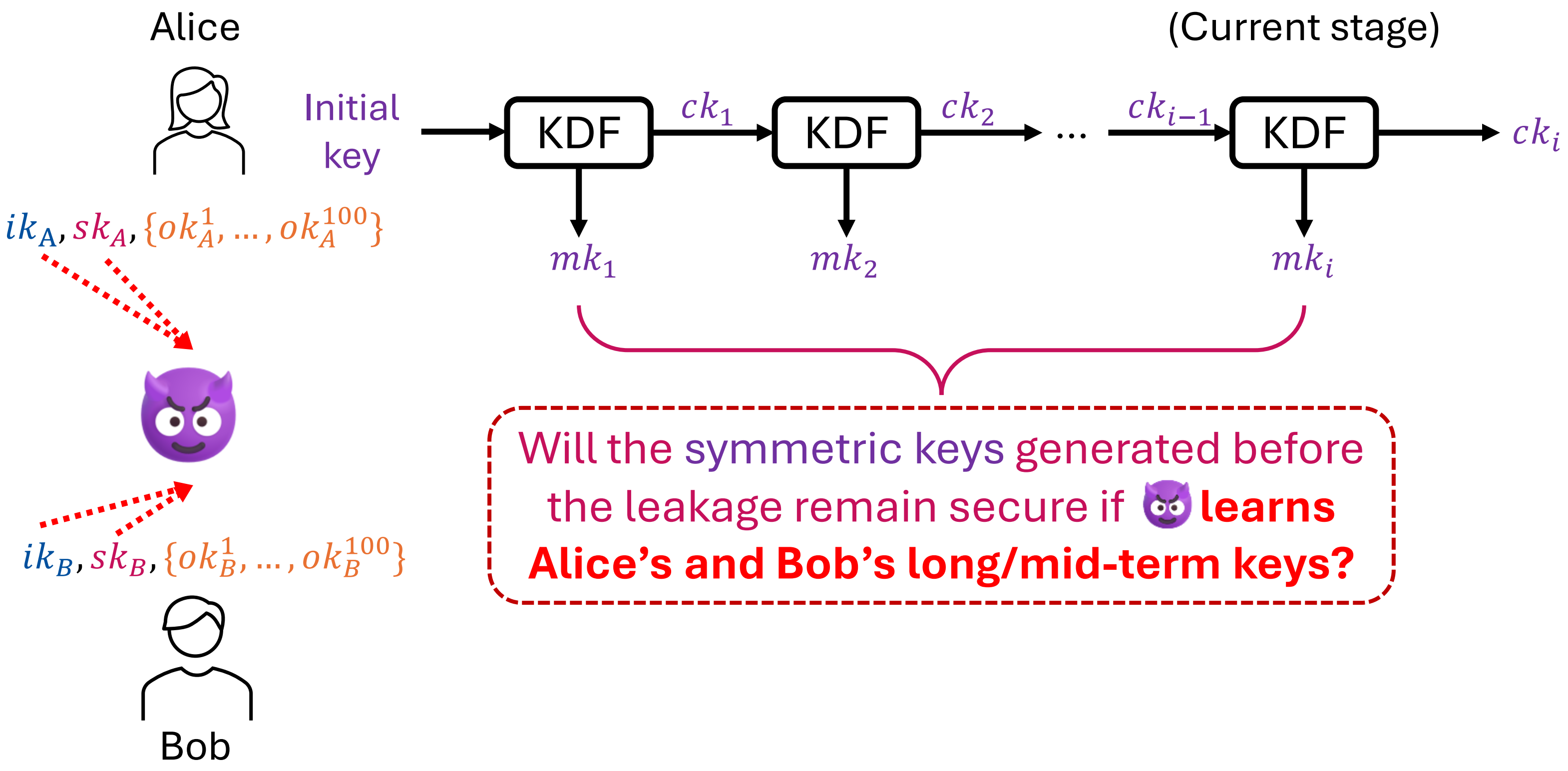
Initial_key (of X3DH)= $\mathrm{KDF}(\boxed{\mathrm{DH}_1}, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

DH_1 is not secure

But DH_2,3,4 remain secret

Forward Secrecy

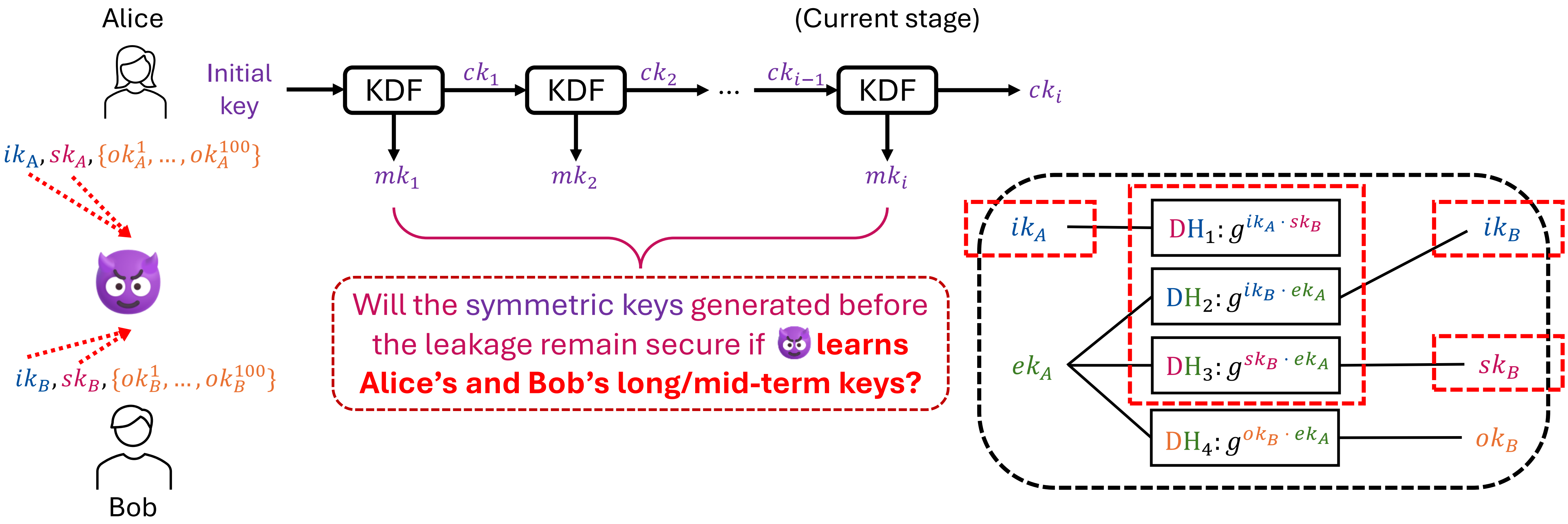
- Long-term secret keys are compromised, but past communication remains secure…



Will the symmetric keys generated before the leakage remain secure if 😈 **learns Alice's and Bob's long/mid-term keys?**

Forward Secrecy

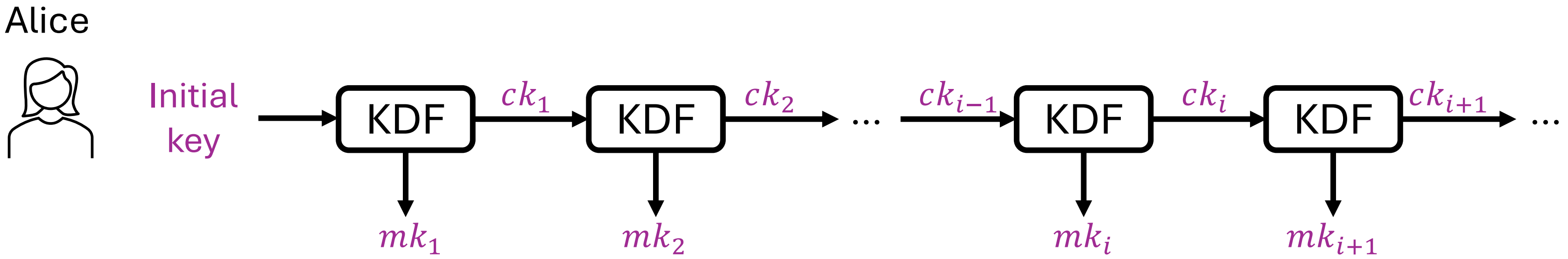
- Long-term secret keys are compromised, but past communication remains secure...



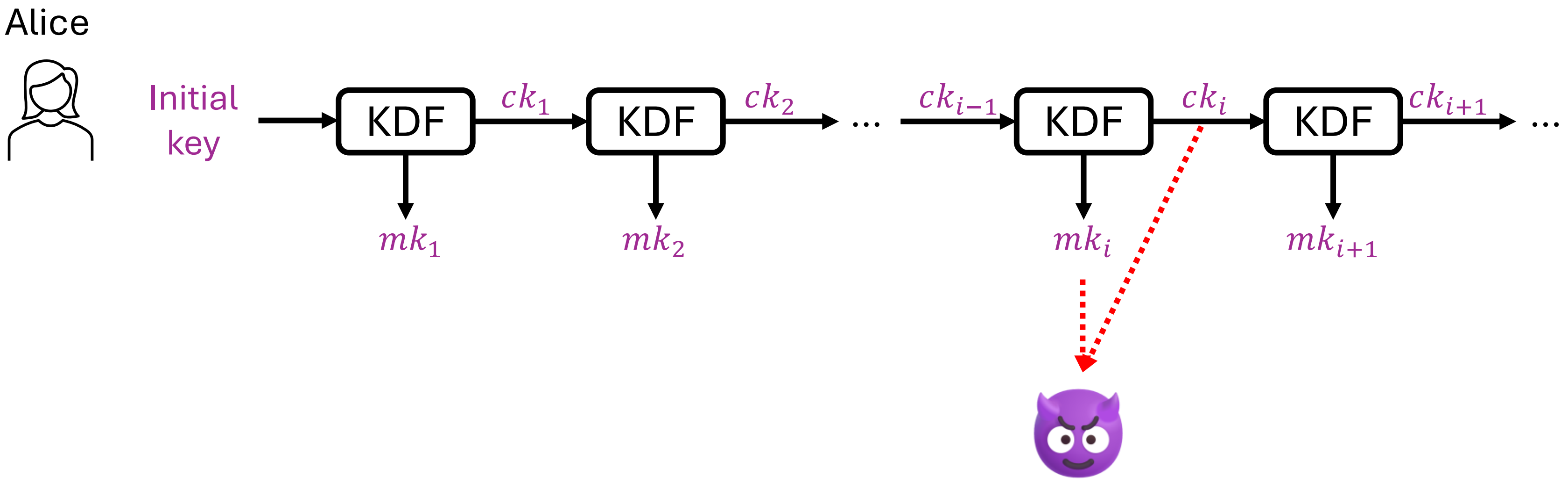
Alice
(Current stage)
Initial key
KDF
KDF
KDF
ck_1
ck_2
…
ck_{i-1}
ck_i
mk_1
mk_2
mk_i
$ik_A, sk_A, \{ok_A^1, \dots, ok_A^{100}\}$
$ik_B, sk_B, \{ok_B^1, \dots, ok_B^{100}\}$
Bob
ik_A
$\mathrm{DH}_1: g^{ik_A \cdot sk_B}$
ik_B
$\mathrm{DH}_2: g^{ik_B \cdot ek_A}$
ek_A
$\mathrm{DH}_3: g^{sk_B \cdot ek_A}$
sk_B
$\mathrm{DH}_4: g^{ok_B \cdot ek_A}$
ok_B

Will the symmetric keys generated before the leakage remain secure if 😈 **learns Alice’s and Bob’s long/mid-term keys?**

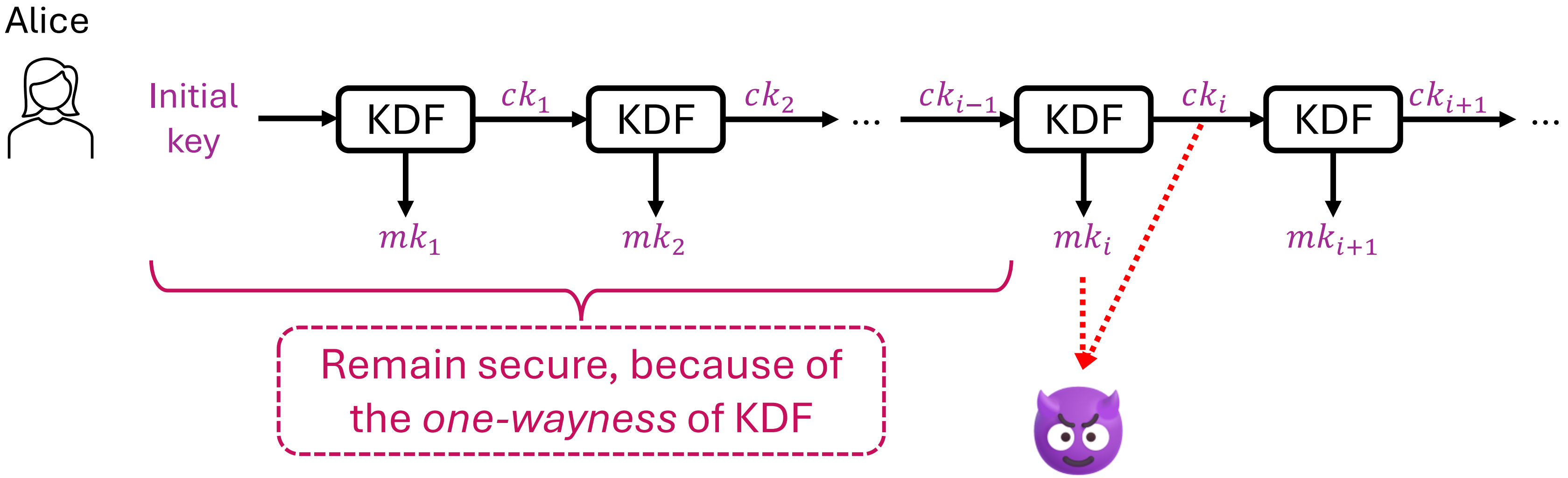
Backward Secrecy



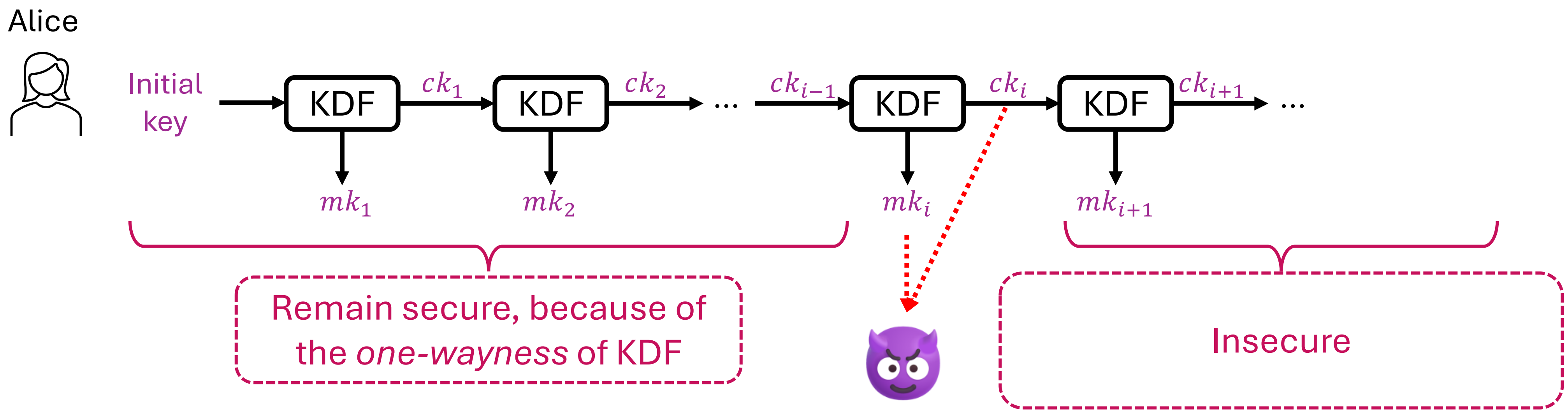
Backward Secrecy



Backward Secrecy



Backward Secrecy

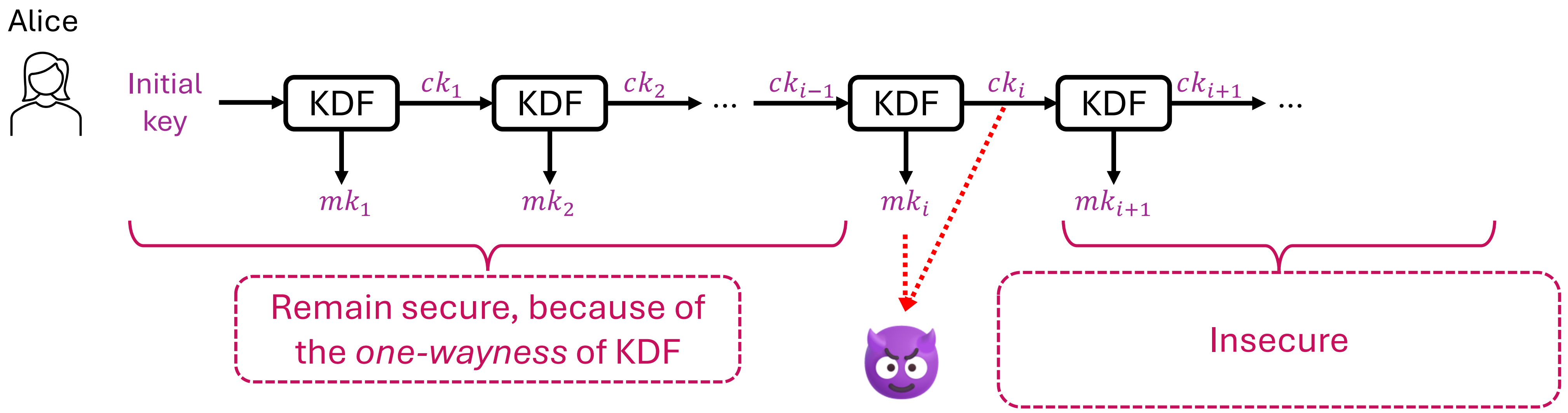


Remain secure, because of the *one-wayness* of KDF

Insecure

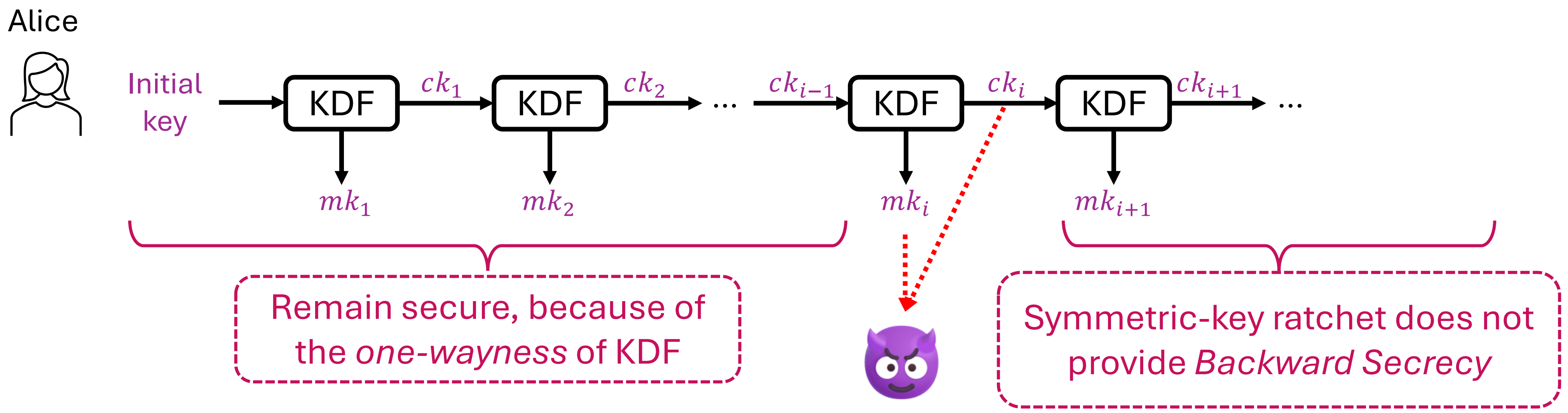
Backward Secrecy

- Future communication remains secure even if a current session key is compromised



Backward Secrecy

- Future communication remains secure even if a current session key is compromised



Remain secure, because of the *one-wayness* of KDF

Symmetric-key ratchet does not provide *Backward Secrecy*

Diffie-Hellman Ratchet

- X3DH + Symmetric-key Ratchet
  - X3DH provides *Forward Secrecy*
  - Current session key compromises does not lead to the compromise of previous session keys
    - (by the one-wayness of KDF in Symmetric-key Ratchet)
  - No Backward Secrecy

Diffie-Hellman Ratchet

- X3DH + Symmetric-key Ratchet
  - X3DH provides *Forward Secrecy*
  - Current session key compromises does not lead to the compromise of previous session keys
    - (by the one-wayness of KDF in Symmetric-key Ratchet)
  - No Backward Secrecy

- Solution: Diffie-Hellman Ratchet (Next lecture)

Exercise

- Implement an X3DH demo
  - **Registration:** Alice and Bob register their key pairs with the server
  - **Fetch pre-key bundles:** To run the key exchange with Bob, Alice first fetches his pre-key bundle from the server
  - **Key Exchange:** After receiving Bob’s pre-key bundle, Alice runs the X3DH key exchange to get the shared secret
- Implement the KDF chain (use the X3DH shared secret as the initial secret)
  - Example (using HMAC-KDF-chain): $ck_{i+1} = HMAC(ck_i, \text{"chain key"}), mk_i = HMAC(ck_i, \text{"message key"})$
- Implement the secure messaging demo using X3DH and KDF
  - Alice starts the conversation
  - If Bob is offline, Alice’s protocol messages are stored in the server
  - Alice starts to derive the initial chain key and root key using $(rk, ck_0) = HKDF([X3DH\ secret], "X3DH")$, and send encrypted messages to Bob. (Root key is ignored now)
  - Once Bob is online, the server sends Alice’s protocol messages to Bob, and Bob decrypts Alice’s messages and responds…