Cryptography Engineering

- Lecture 4 (Nov 13, 2024)

- Today’s notes:
  - Secure Messaging
  - X3DH Protocol
  - Key Ratcheting
  - Diffie-Hellman Ratcheting

- Today’s coding tasks (and homework):
  - Implement X3DH using sockets

Cryptography Engineering

- First Part of this Course: Key Exchange, Signature, and Handshake
  - Diffie-Hellman Key Exchange, and MitM attacks
  - Digital Signature and Certificate
  - Handshake Protocol
  - We addressed *How to Build a Secure Channel* over an open network…
    - *e.g., share a secure key, …*

- Second Part:
  - How to communicate securely over an open network…
  - Main Topic: Secure Messaging

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

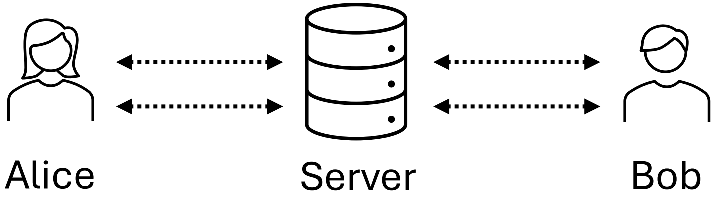
End-to-End Encryption

Initialization

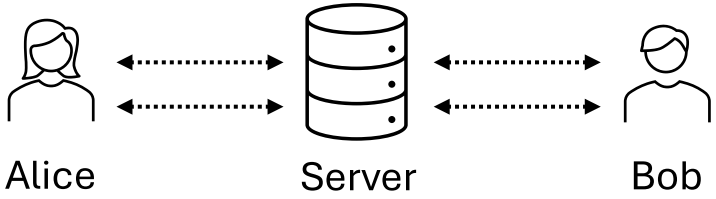
Including logging in, sharing users' information, …

Messaging

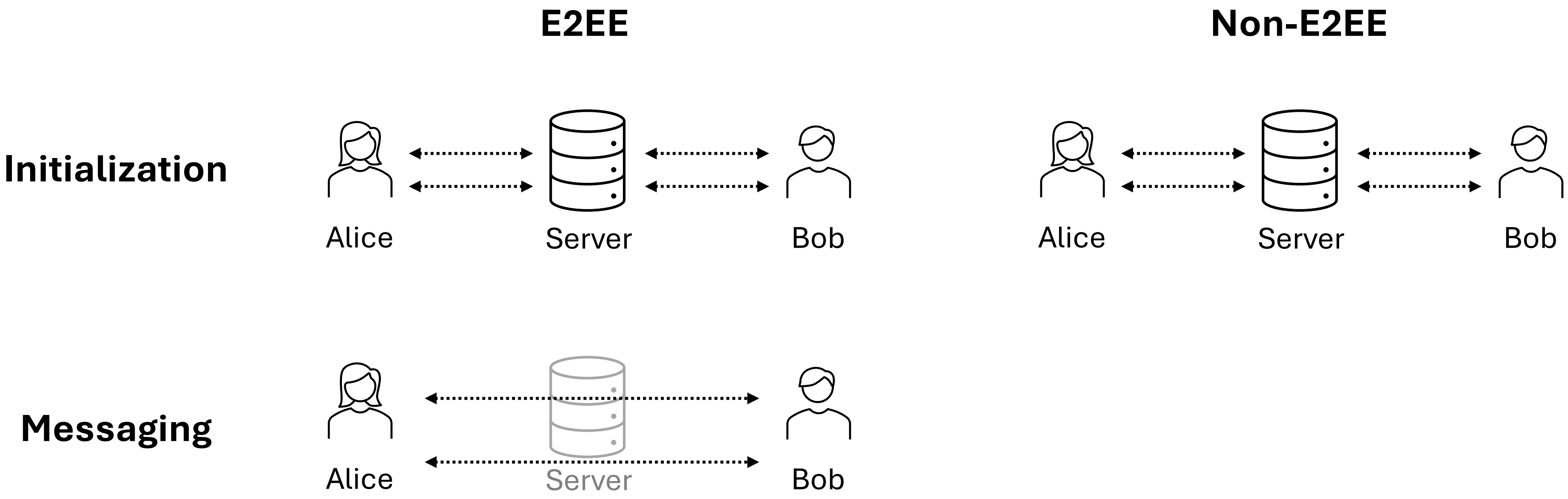
E2EE



Non-E2EE

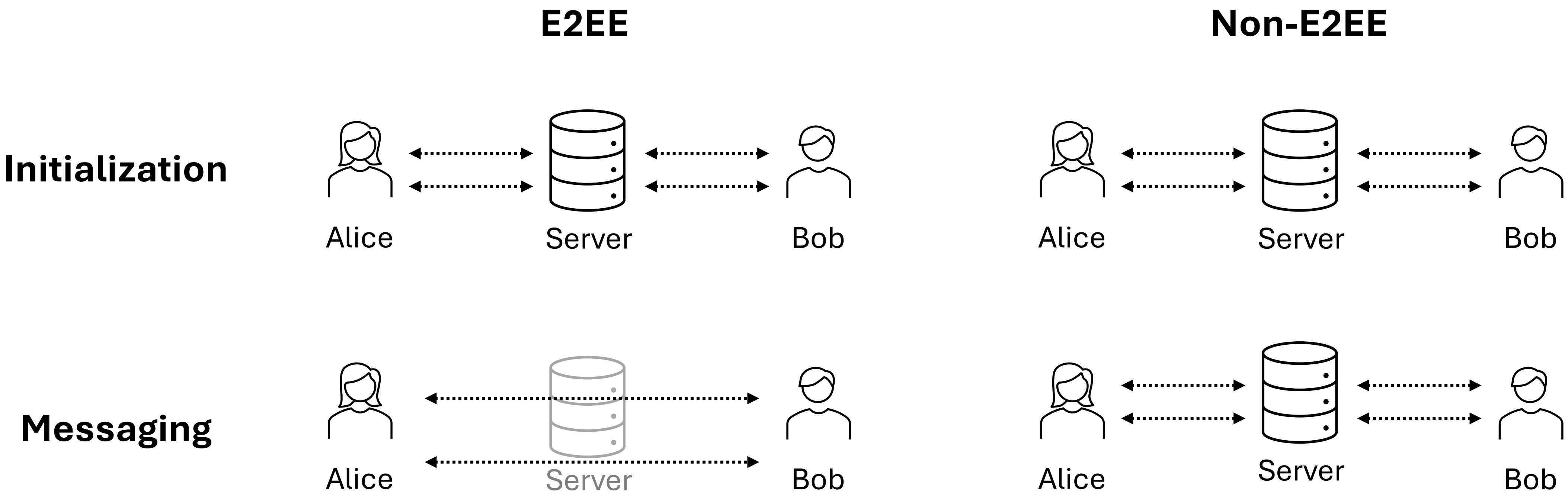


End-to-End Encryption



The server **only relays** the encrypted messages, no storage.

End-to-End Encryption



The server **only relays** the encrypted messages, no 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

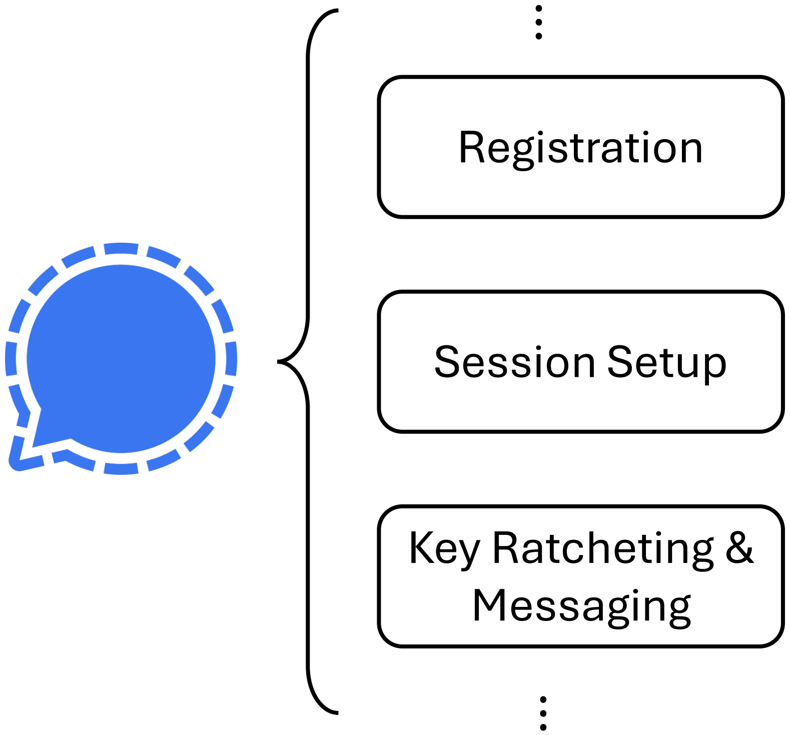
Good

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



⋮

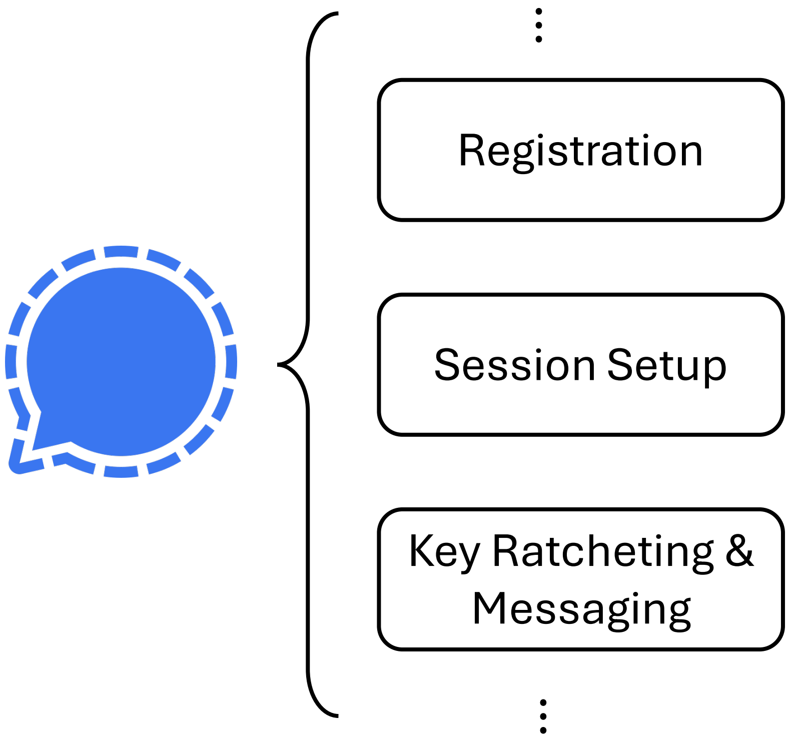
Registration

Session Setup

Key Ratcheting & Messaging

⋮

Signal Secure Messaging Protocol



⋮

Registration

Session Setup

Key Ratcheting & Messaging

⋮

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

The X3DH Protocol

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

The X3DH Protocol

- 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)	$(OPK_{\mathrm{A}}^1, OPK_{\mathrm{A}}^2, \ldots)$	$(OPK_{\mathrm{B}}^1, OPK_{\mathrm{B}}^2, \ldots)$

The X3DH Protocol

- 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)	$\{ok_{\mathrm{A}}^1, ok_A^2, \ldots\} \subseteq_{\$} \mathbb{Z}_q$	$\{ok_{\mathrm{B}}^1, ok_B^2, \ldots\} \subseteq_{\$} \mathbb{Z}_q$
One-time public pre-keys (OPK)	$(OPK_{\mathrm{A}}^1, OPK_{\mathrm{A}}^2, \ldots)$	$(OPK_{\mathrm{B}}^1, OPK_{\mathrm{B}}^2, \ldots)$

The X3DH Protocol

- 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

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**

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 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

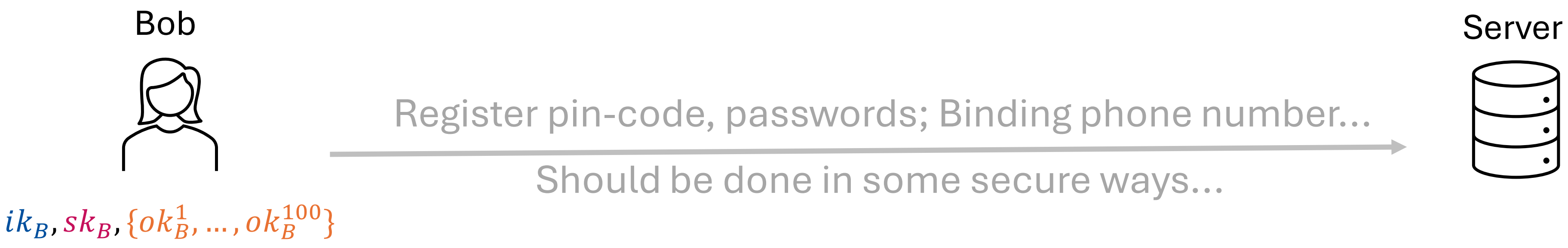
	Alice	Bob
Identity secret key (IK)	$ik_{\mathrm{A}} \in_{\$} \mathbb{Z}_q$	$ik_{\mathrm{B}} \in_{\$} \mathbb{Z}_q$
Identity public key (IPK)	$IPK_{\mathrm{A}}(= g^{ik_A})$	IPK_{B}
Signing secret pre-key (SK)	$sk_{\mathrm{A}} \in_{\$} \mathbb{Z}_q$	$sk_{\mathrm{B}} \in_{\$} \mathbb{Z}_q$
Signing public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

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

- 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…

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

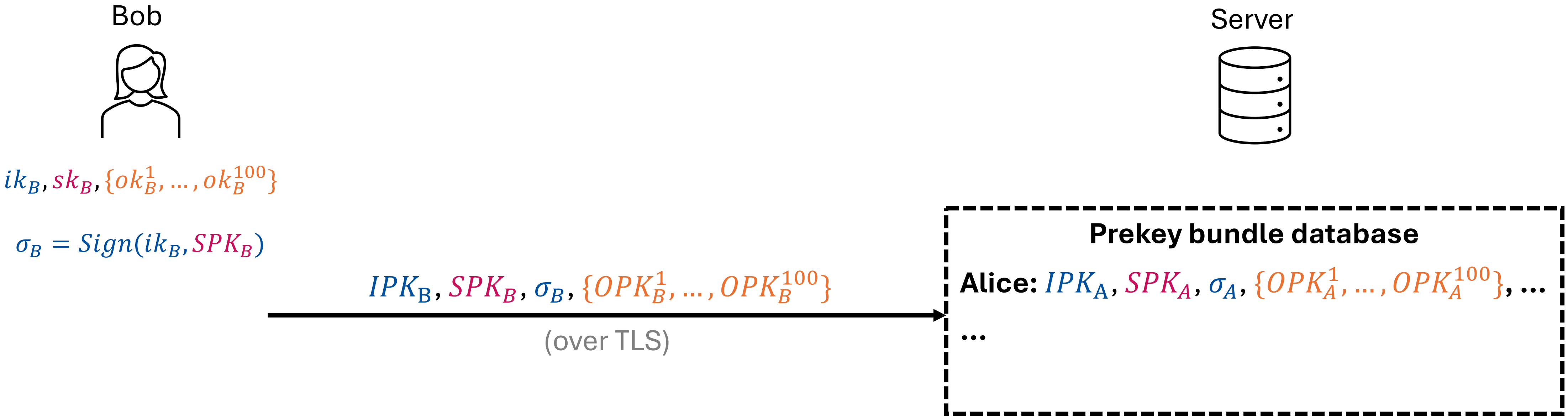
Prekey bundle database

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

…

The X3DH Protocol

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



Bob

Server

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

$\sigma_B = Sign(ik_B, SPK_B)$

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

(over TLS)

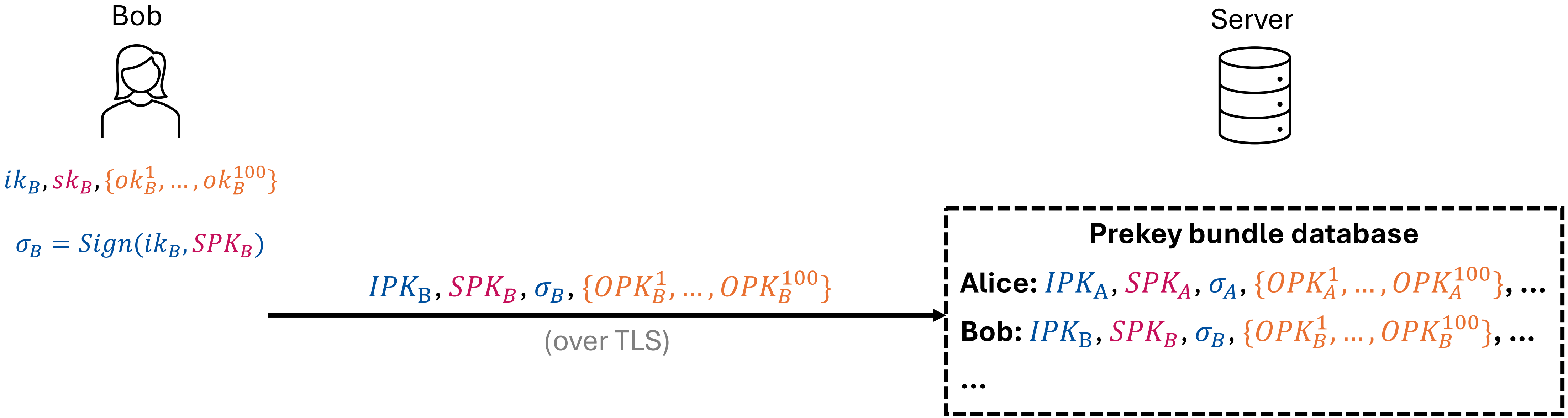
Prekey bundle database

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

…

The X3DH Protocol

- 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

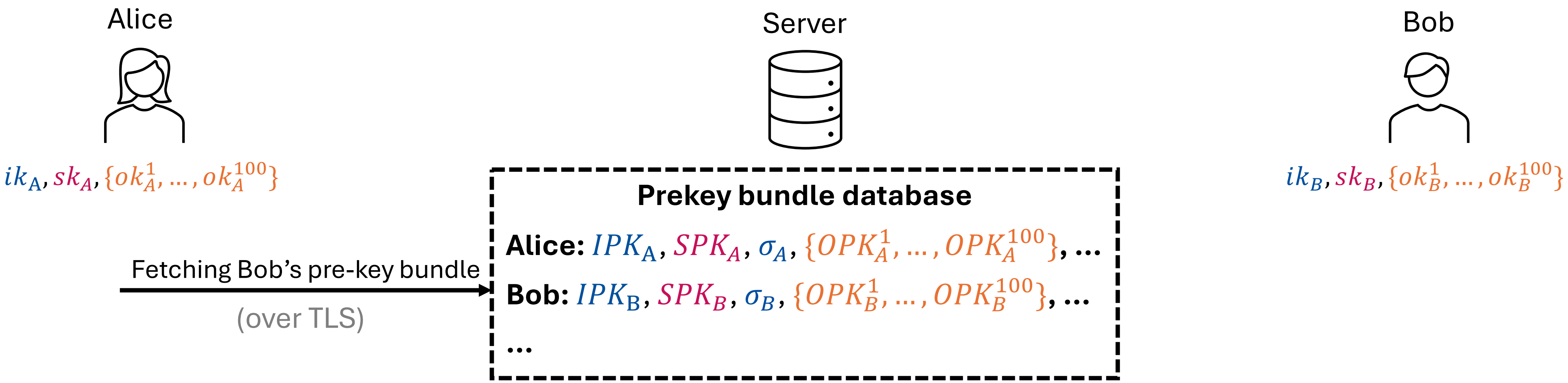
Alice: $IPK_\mathrm{A}, SPK_A, \sigma_A, \{OPK_A^1, \dots, OPK_A^{100}\}$, …

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

…

The X3DH Protocol

- 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

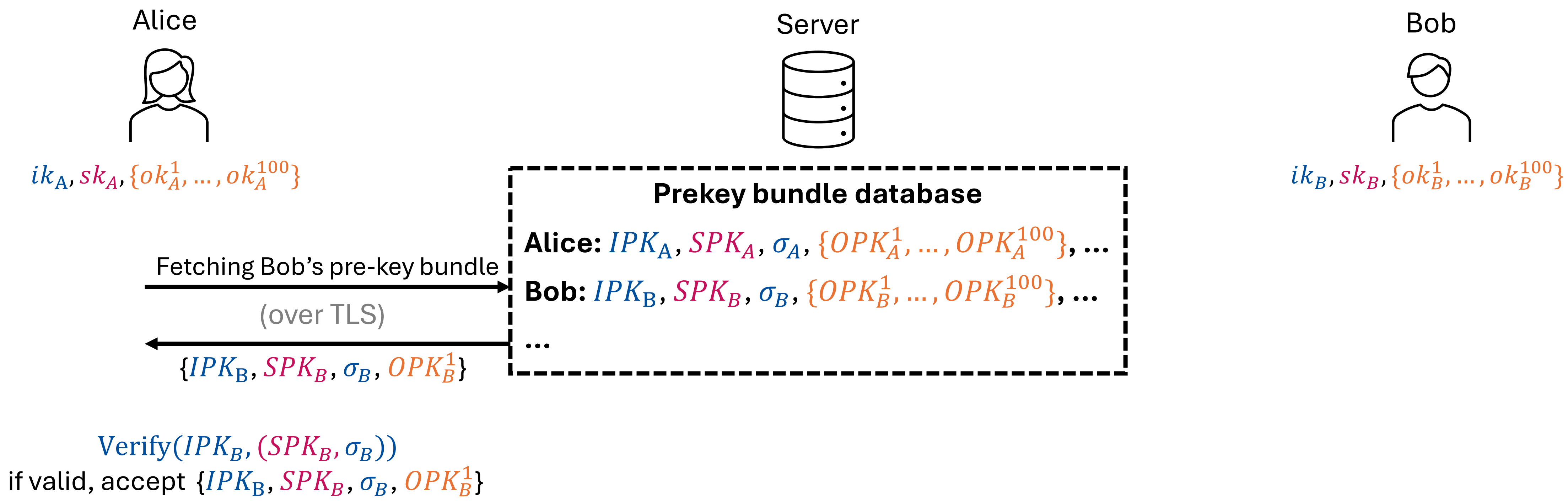
Alice: $IPK_{\mathrm{A}}, SPK_A, \sigma_A, \{OPK_A^1, \dots, OPK_A^{100}\}$**, …**

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

…

The X3DH Protocol

- When Alice communicates with Bob…



Alice

Server

Bob

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

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

Prekey bundle database

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

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

…

Fetching Bob’s pre-key bundle

(over TLS)

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

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

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

The X3DH Protocol

- When Alice communicates with Bob…

Alice

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

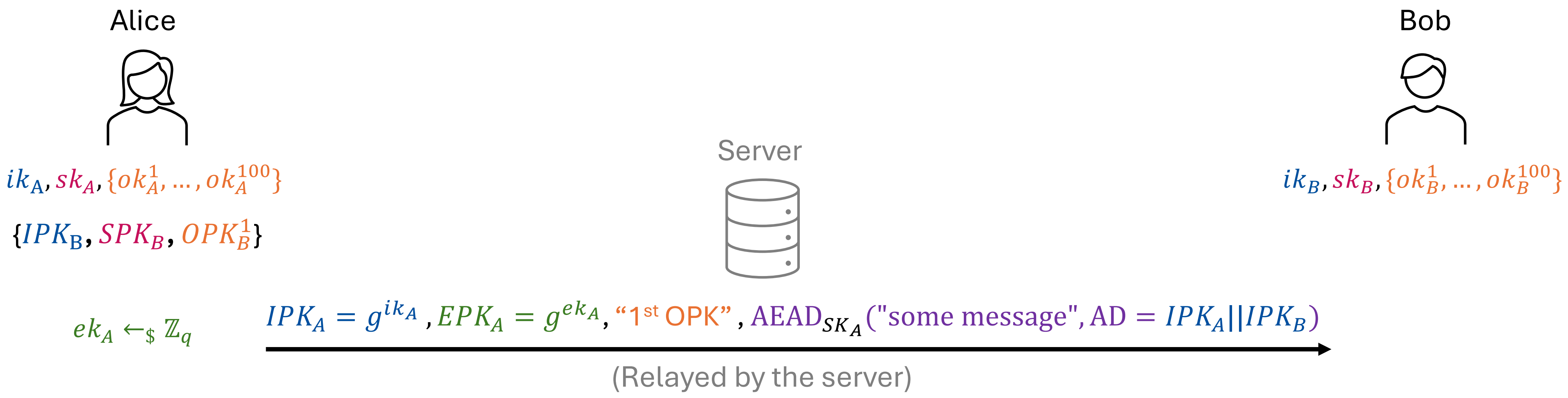
$\{IPK_{\mathrm{B}}, SPK_B, OPK_B^1\}$

Bob

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

The X3DH Protocol

- When Alice communicates with Bob…



Alice

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

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

Server

Bob

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

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

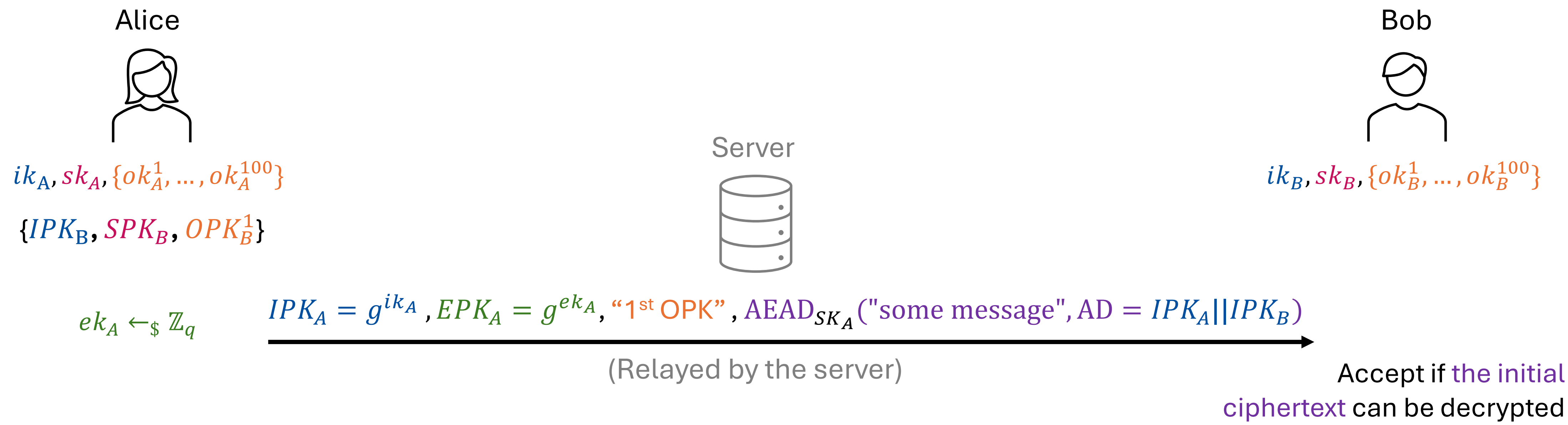
$IPK_A = g^{ik_A}$, $EPK_A = g^{ek_A}$, “1st OPK” , $\mathrm{AEAD}_{SK_A}(\text{"some message"}, \mathrm{AD} = IPK_A || IPK_B)$

(Relayed by the server)

$SK_A = \text{X3DH\_Key\_Alice}(ik_A, ek_A, IPK_B, SPK_B, OPK_B^1)$
(…will be explained later)

The X3DH Protocol

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



Alice

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

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

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

Server

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

(Relayed by the server)

Bob

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

Accept if the initial ciphertext can be decrypted

$SK_A =$ X3DH_Key_Alice($ik_{\mathrm{A}}, ek_A, IPK_{\mathrm{B}}, SPK_B, OPK_B^1$)
(...will be explained later)

$SK_B =$ X3DH_Key_Bob($IPK_{\mathrm{A}}, EPK_A, ik_{\mathrm{B}}, sk_B, ok_B^1$)
(...will be explained later)

The X3DH Protocol

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

Alice

$SK_A = \text{X3DH\_Key\_Alice}(ik_{\mathrm{A}}, ek_A, IPK_{\mathrm{B}}, SPK_B, OPK_B)$

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. $SK_{\mathrm{A}} = \mathrm{KDF}(\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4)$

// Using $\mathrm{DH}_1, \mathrm{DH}_2, \mathrm{DH}_3, \mathrm{DH}_4$ as the main secret

Bob

$SK_B = \text{X3DH\_Key\_Bob}(IPK_{\mathrm{A}}, EPK_A, ik_{\mathrm{B}}, sk_B, ok_B)$

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

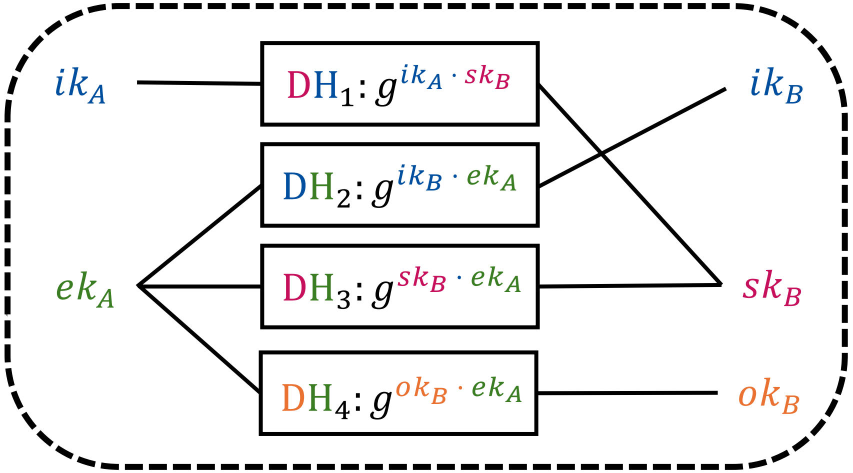
- How the X3DH protocol computes a shared secret…

Alice

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

1. $DH_1 = SPK_B^{ik_A}$
2. $DH_2 = IPK_B^{ek_A}$
3. $DH_3 = SPK_B^{ek_A}$
4. $DH_4 = (OPK_B)^{ek_A}$
5. $SK_A = KDF(DH_1, DH_2, DH_3, DH_4)$

// Using DH_1, DH_2, DH_3, DH_4 as the m



ik_A

ek_A

$DH_1: g^{ik_A \cdot sk_B}$

$DH_2: g^{ik_B \cdot ek_A}$

$DH_3: g^{sk_B \cdot ek_A}$

$DH_4: g^{ok_B \cdot ek_A}$

ik_B

sk_B

ok_B

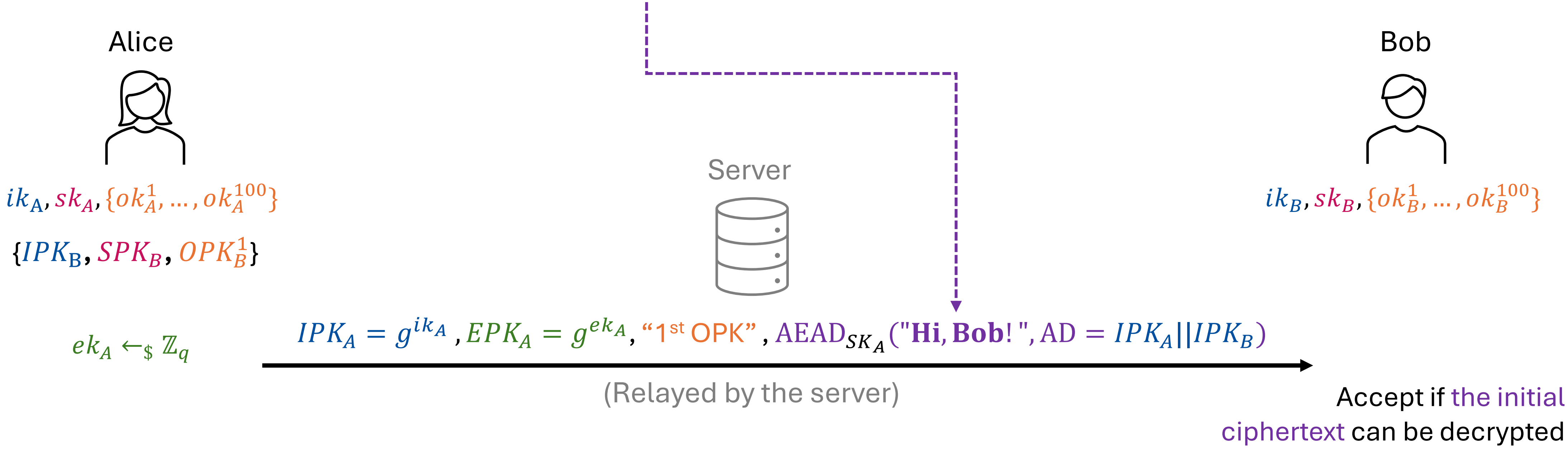
Bob

$SK_B = \text{X3DH\_Key\_Bob}(IPK_A, EPK_A, ik_B, sk_B, ok_B)$

1. $DH_1 = IPK_A^{sk_B}$
2. $DH_2 = EPK_A^{ik_B}$
3. $DH_3 = EPK_A^{sk_B}$
4. $DH_4 = EPK_A^{ok_B}$
5. $SK_B = KDF(DH_1, DH_2, DH_3, DH_4)$

The X3DH Protocol

- **0-RTT** (Zero Round-Trip Time): Send message instantly without waiting response



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

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

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

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

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

(Relayed by the server)

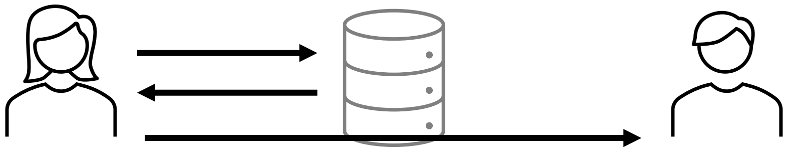
Accept if the initial ciphertext can be decrypted

SK_A = X3DH_Key_Alice($ik_{\mathrm{A}}, ek_A, IPK_{\mathrm{B}}, SPK_B, OPK_B^1$)
(...will be explained later)

SK_B = X3DH_Key_Bob($IPK_{\mathrm{A}}, EPK_A, ik_{\mathrm{B}}, sk_B, ok_B^1$)
(...will be explained later)

The X3DH Protocol

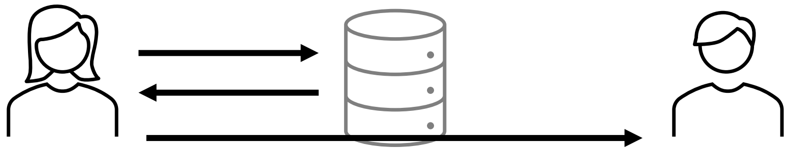
- Based on (EC)DH
- Trusted server required
  - Store public keys, relay messages, ...
  - Cannot decrypt ciphertexts...
- 0-RTT
  - Immediate message sending without waiting for a response
- Support offline communication
  - Can be executed even if Bob (the receiver) is offline
  - Offline messages (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, ...
  - Cannot decrypt ciphertexts...
- 0-RTT
  - 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

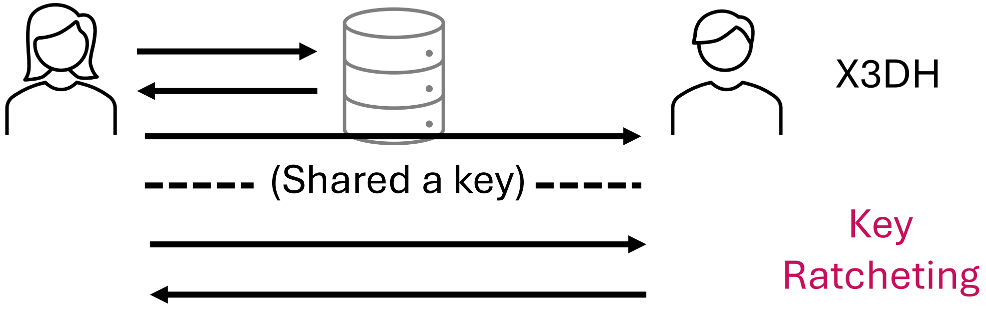
Different primary goals/settings:

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...

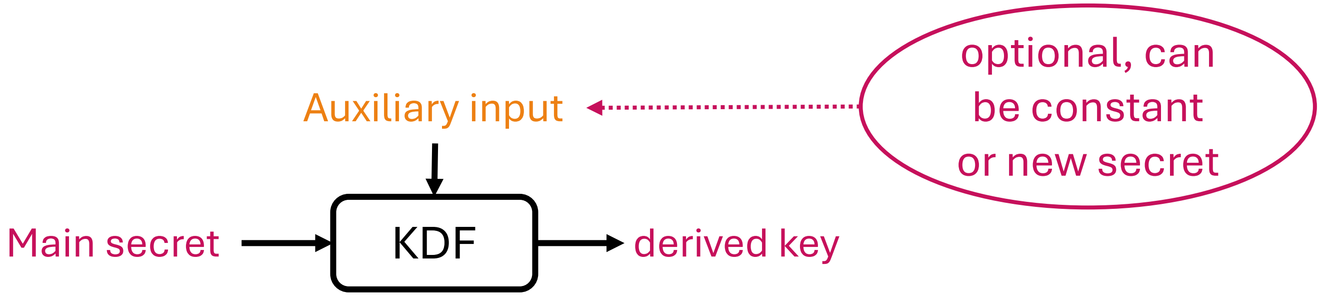
Symmetric Key Ratcheting

- After completing X3DH…
- … we use **Double Ratcheting** to:
  - Encrypt messages + updates the shared key
  - ~~Encrypt messages using the same shared key~~
  - Diffie-Hellman Ratcheting + Symmetric-key Ratcheting
- Essential for forward/backward secrecy (next lecture)
- Today: Symmetric-key ratcheting



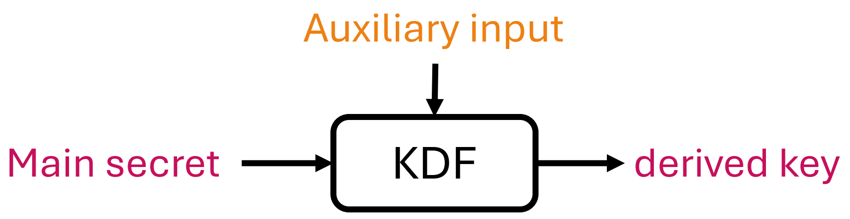
Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function

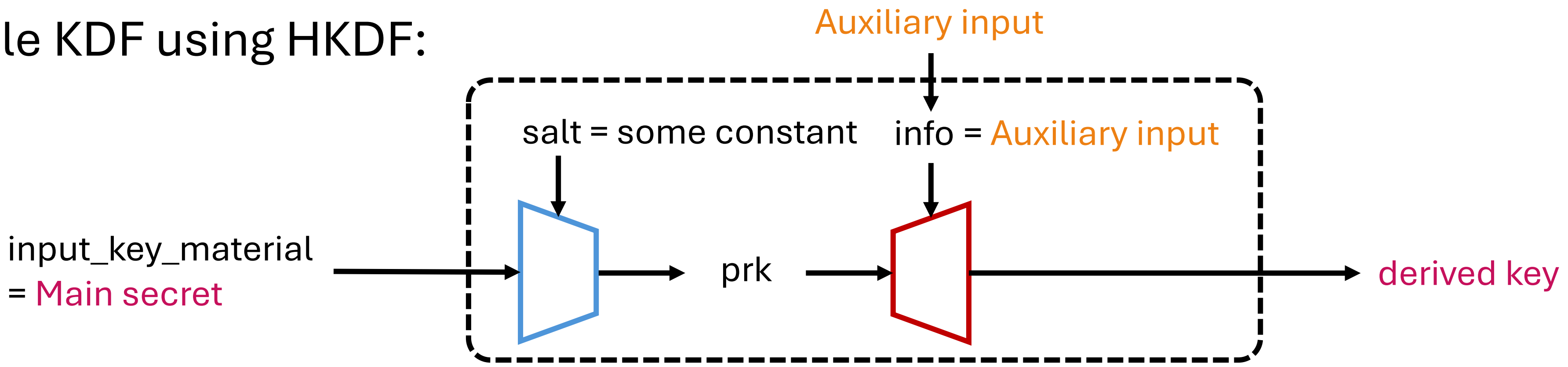


Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function



- Example KDF using HKDF:

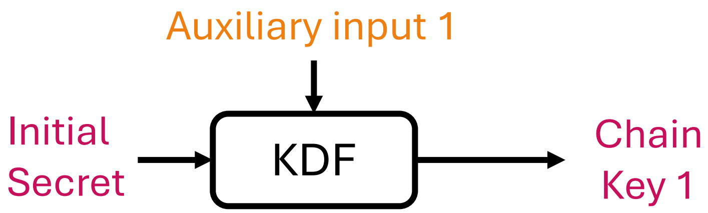


1. prk = HKDF.**Extract**(input_key_material = Main secret, salt = some constant)

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

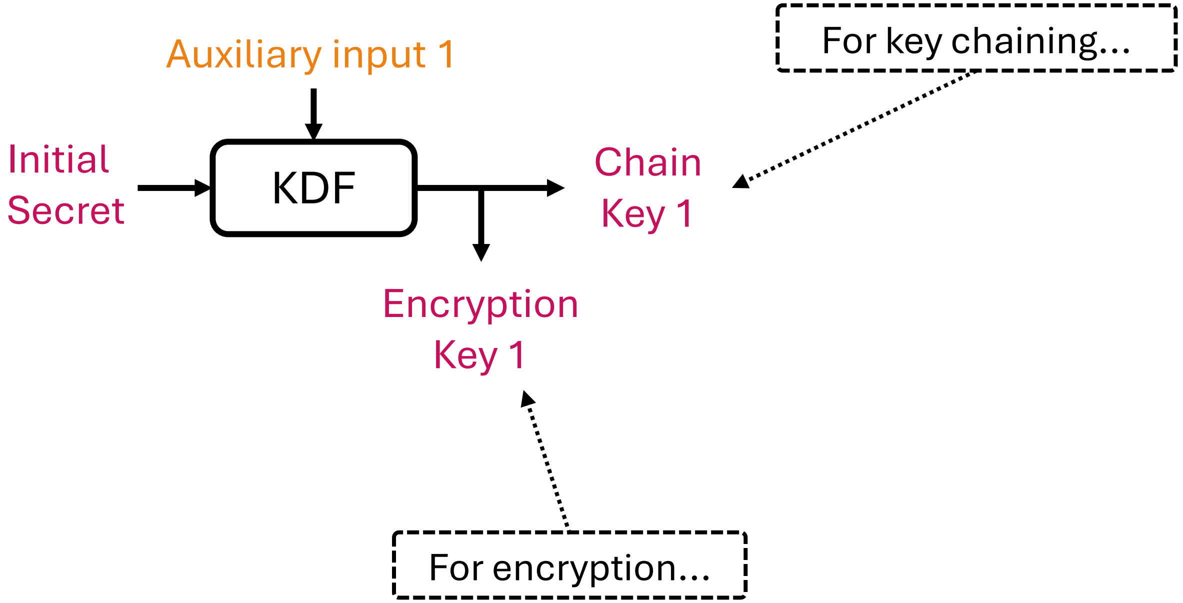
Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function



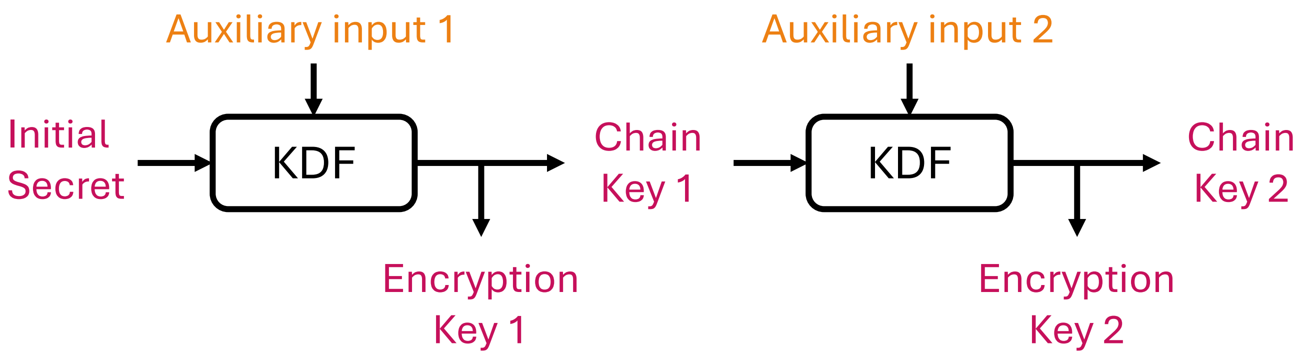
Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function



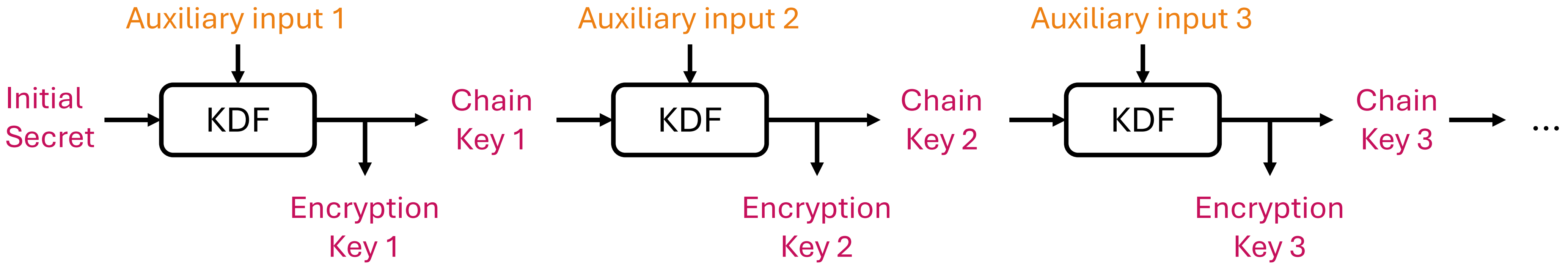
Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function



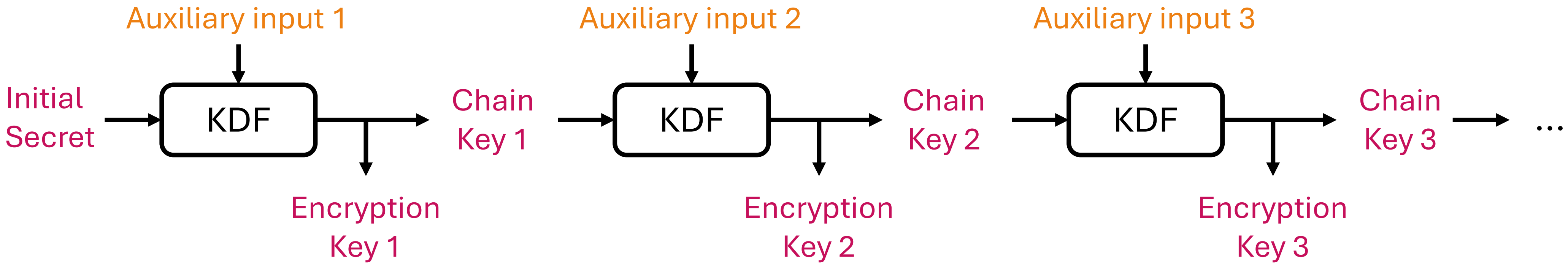
Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function



Symmetric Key Ratcheting

- KDF chain
  - KDF: Key derivation function



- Use Key Chain to encrypt messages (next lecture)

Coding Tasks

1. Implement a KDF chain based on HKDF.
    - You can learn how to use HKDF in the example code “HKDF.py” of Lecture 3.
    - To split a KDF output into Encryption Key and Chain Key, you can first specify the “length” parameter of hkdf_expand, and then truncate it into two byte-strings.

Homework

- **Homework:** Try implementing X3DH using sockets:

  1. Suppose that Alice and Bob have registered with the server. Namely, the server has stored prekey bundles of Alice and Bob.

  2. Alice wants to communicate with Bob, it first fetches prekey bundle of Bob from the server.

  3. Upon receiving the prekey bundle of Bob, Alice verifies the bundle. If it is valid, then Alice follows the X3DH protocol and compute a shared key. After computing a shared key, it sends the protocol message (see the X3DH protocol in this lecture note) to the server.

  4. The server forwards the message from Alice to Bob.

  5. Upon receiving the message from Alice, Bob also compute the X3DH session key.

- **Bonus:** Upgrade your implementation of X3DH so that it allows the recipient user to be offline.

Further Reading

- Old news -- *WhatsApp's Signal Protocol integration is now complete:* *https://signal.org/blog/whatsapp-complete/*
- Technical Documentations of Signal: https://signal.org/docs/
- Cohn-Gordon et al’s security analysis of Signal: https://eprint.iacr.org/2016/1013