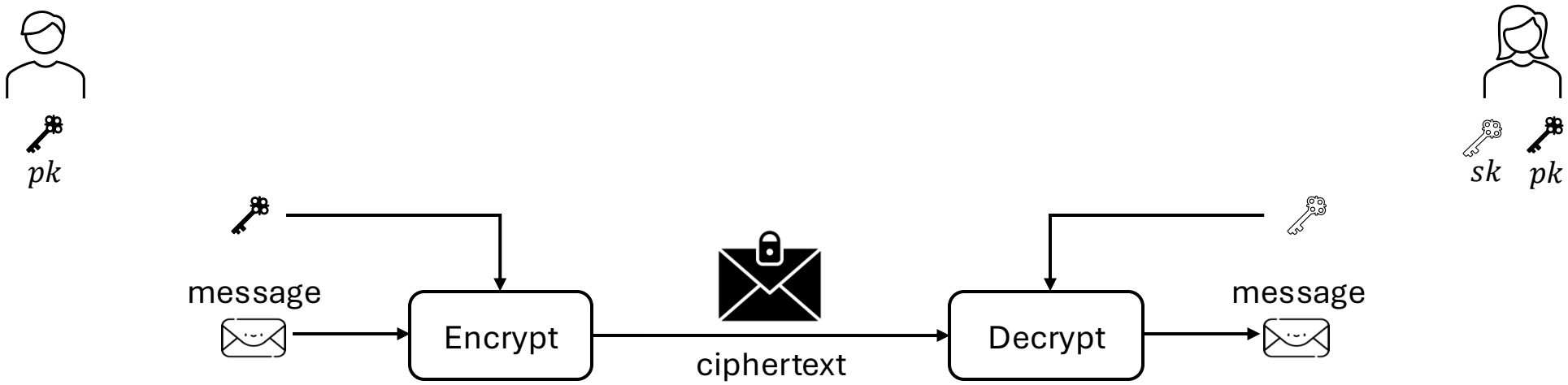
Cryptography Engineering

- Lecture 4 (Nov 12, 2025)
- Today's notes:
  - Public-key Encryption
  - Key Encapsulation Mechanism
  - ElGamal encryption
  - Hashed ElGamal KEM
  - DHIES

Public-key Encryption

- Public-key Encryption **(PKE)**: Asymmetric setting, Encryption/Decryption
  - Encrypt messages



Public-key Encryption

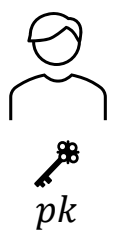
- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)

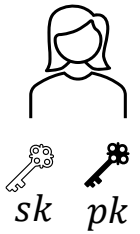
- PKE: Asymmetric setting, Encryption/Decryption
  - Encrypt messages

- KEM: Asymmetric setting, **Encapsulation/Decapsulation**
  - “Encrypt” keys
  - Have attracted increasing attention recently!

Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)

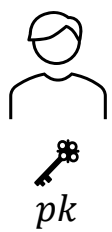


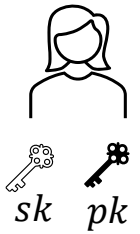


- PKE:

Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)





- PKE:

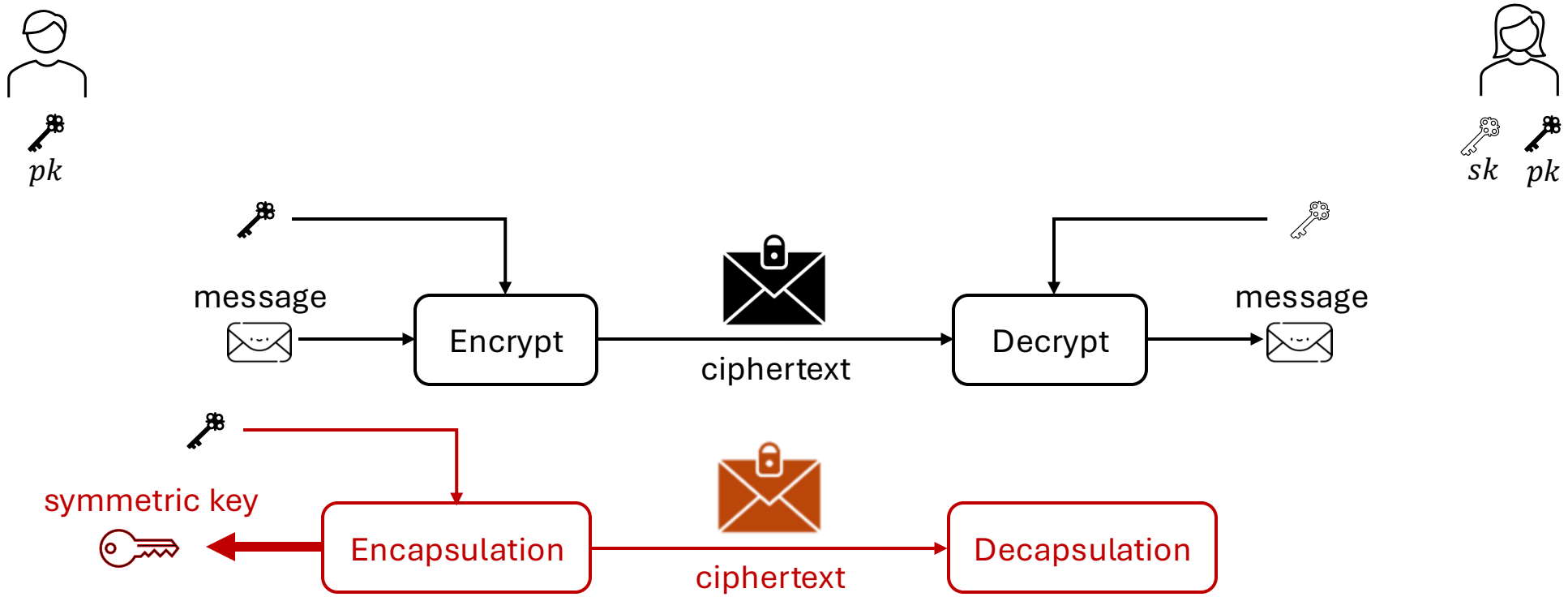
- KEM:

Encapsulation

Decapsulation

Key Encapsulation Mechanism

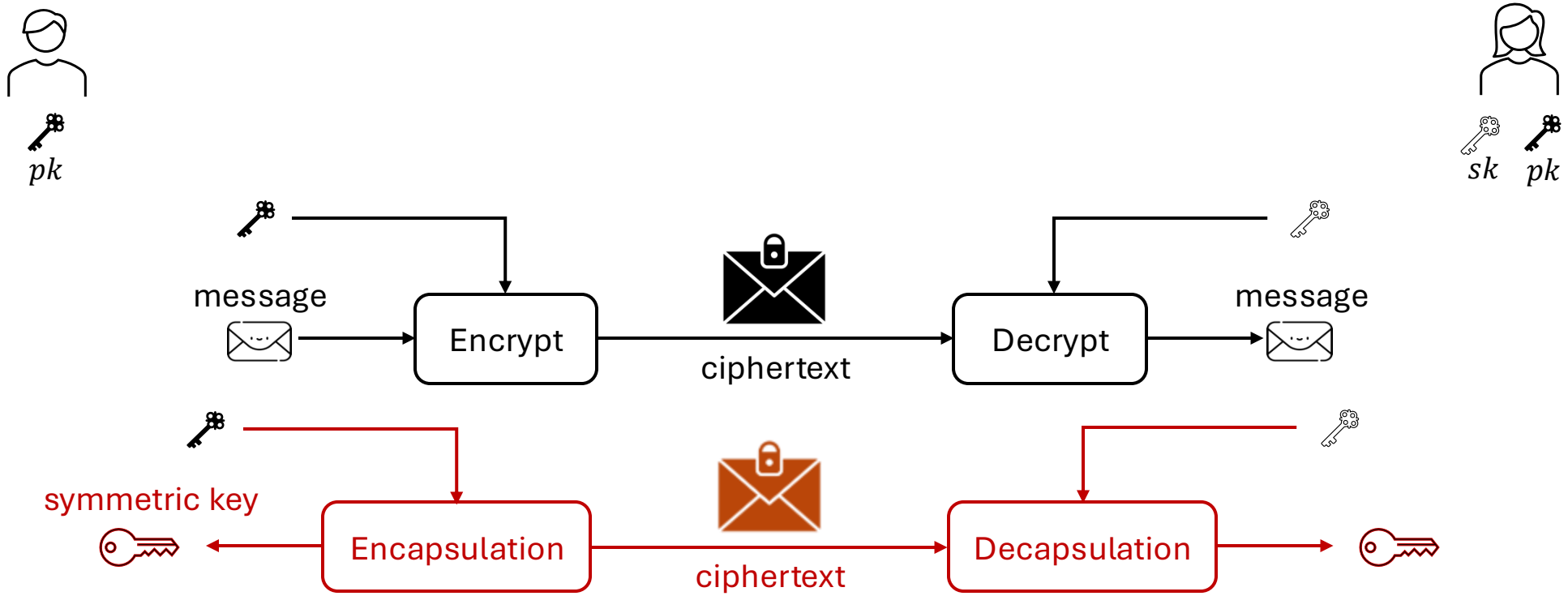
- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)



- PKE:
- KEM:

Key Encapsulation Mechanism

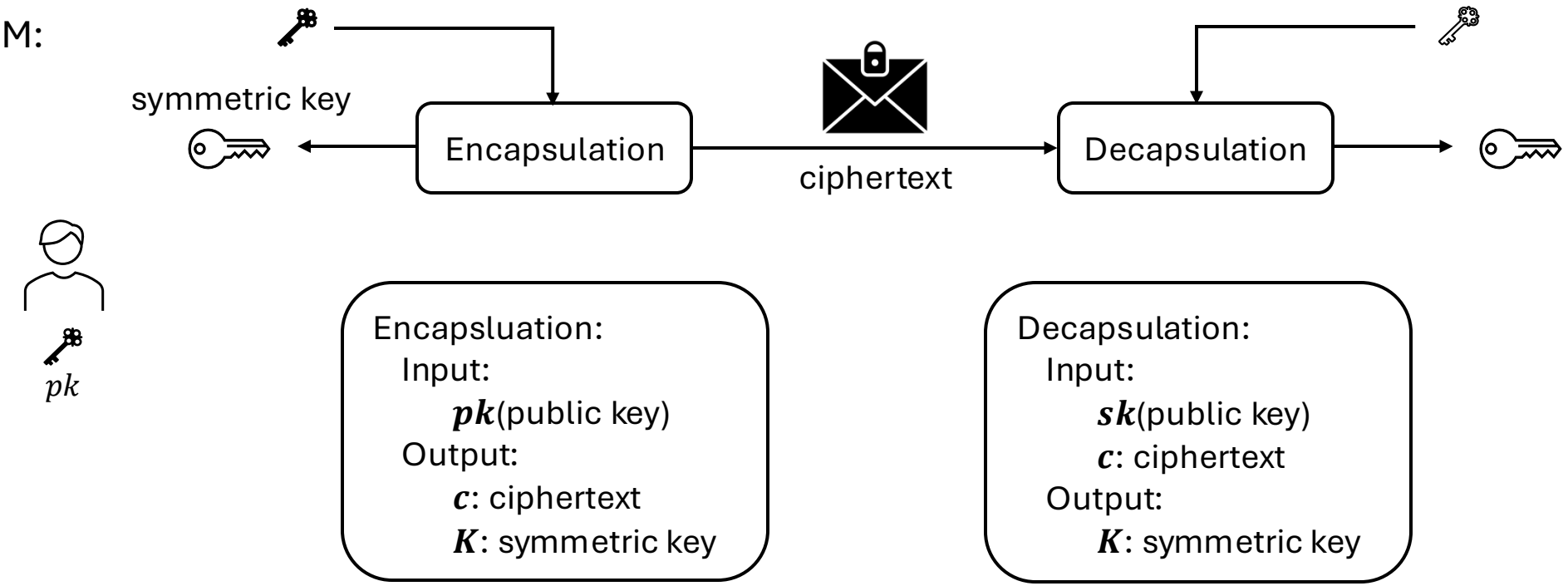
- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)



Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

- KEM:

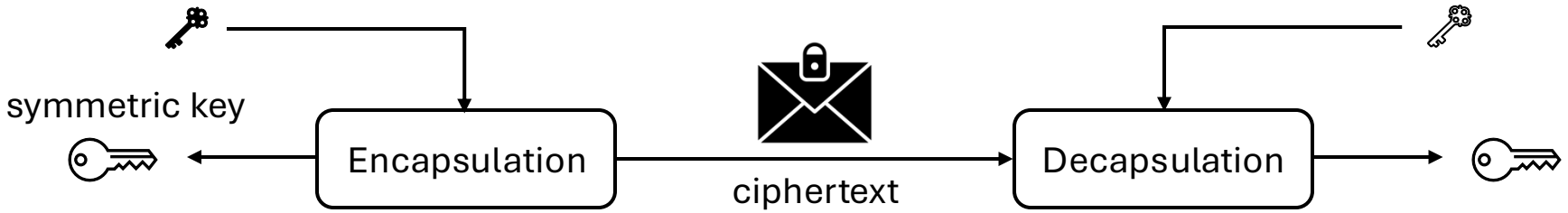


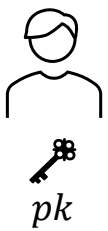
Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

Security:
No sk => Cannot decrypt c
or can decrypt but get the wrong K

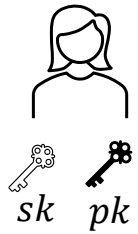
- KEM:





Encapsluation:
Input:
pk(public key)
Output:
c: ciphertext
K: symmetric key

Decapsulation:
Input:
sk(public key)
c: ciphertext
Output:
K: symmetric key

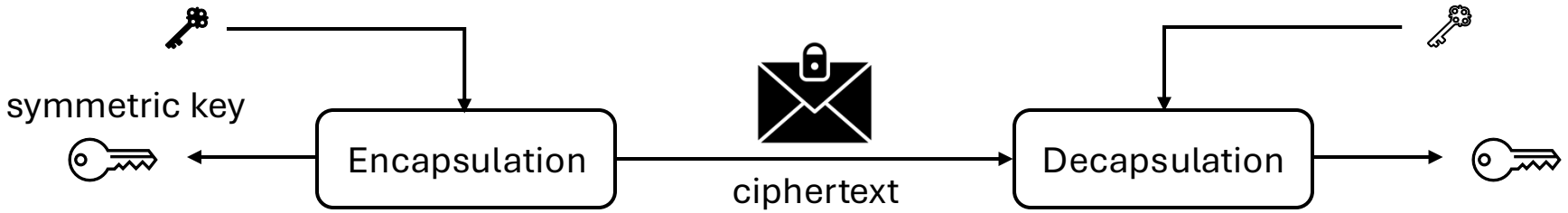


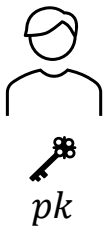
Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

Security:
No sk => Cannot decrypt c
or can decrypt but get the wrong K

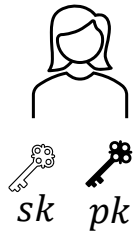
- KEM:





Encapsluation:
Input:
pk(public key)
Output:
c: ciphertext
K: symmetric key

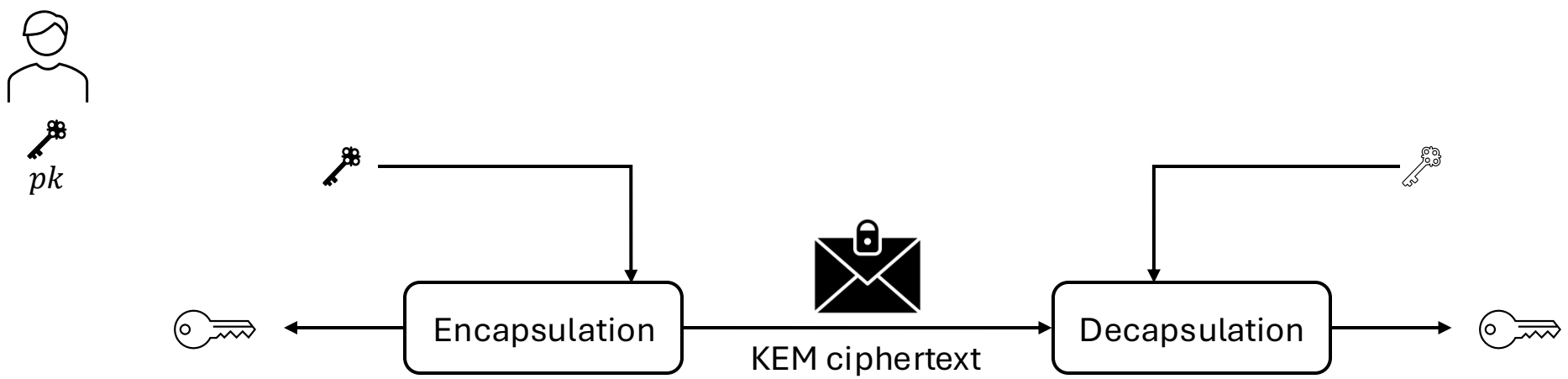
Decapsulation:
Input:
sk(public key)
c: ciphertext
Output:
K: symmetric key



One can use K to do
symmetric-key encryption

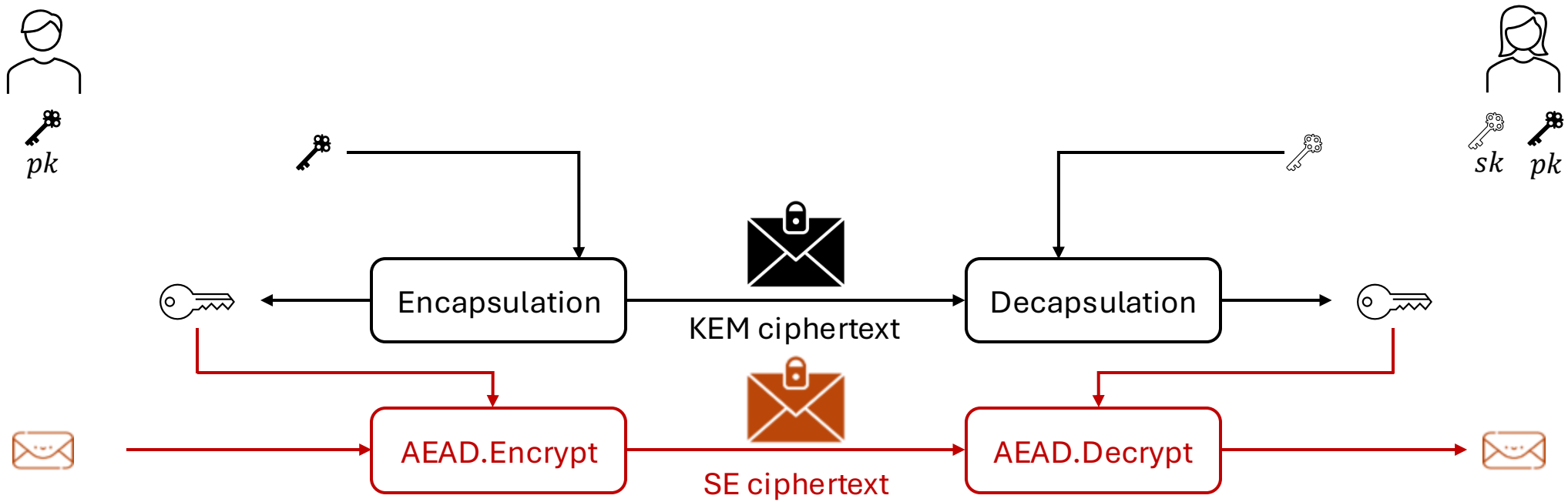
PKE vs KEM

- Relation between KEM and PKE



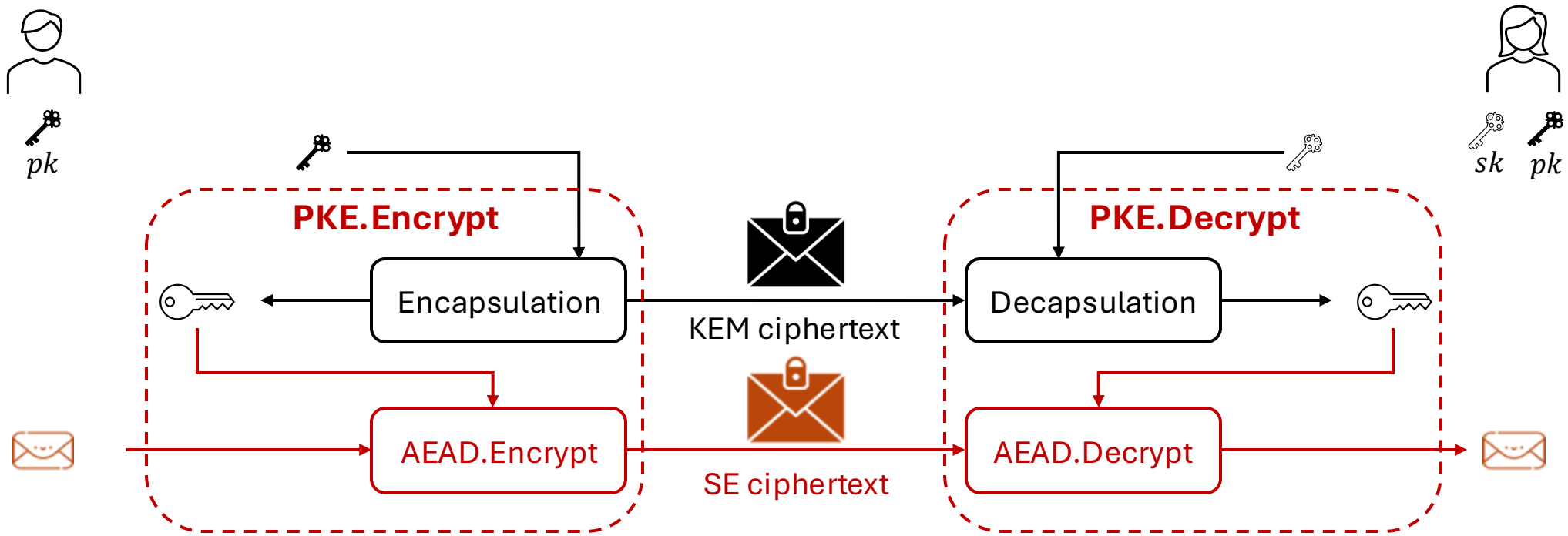
PKE vs KEM

- Relation between KEM and PKE
  - **Build PKE from KEM** (using symmetric encryption, e.g., AEAD)



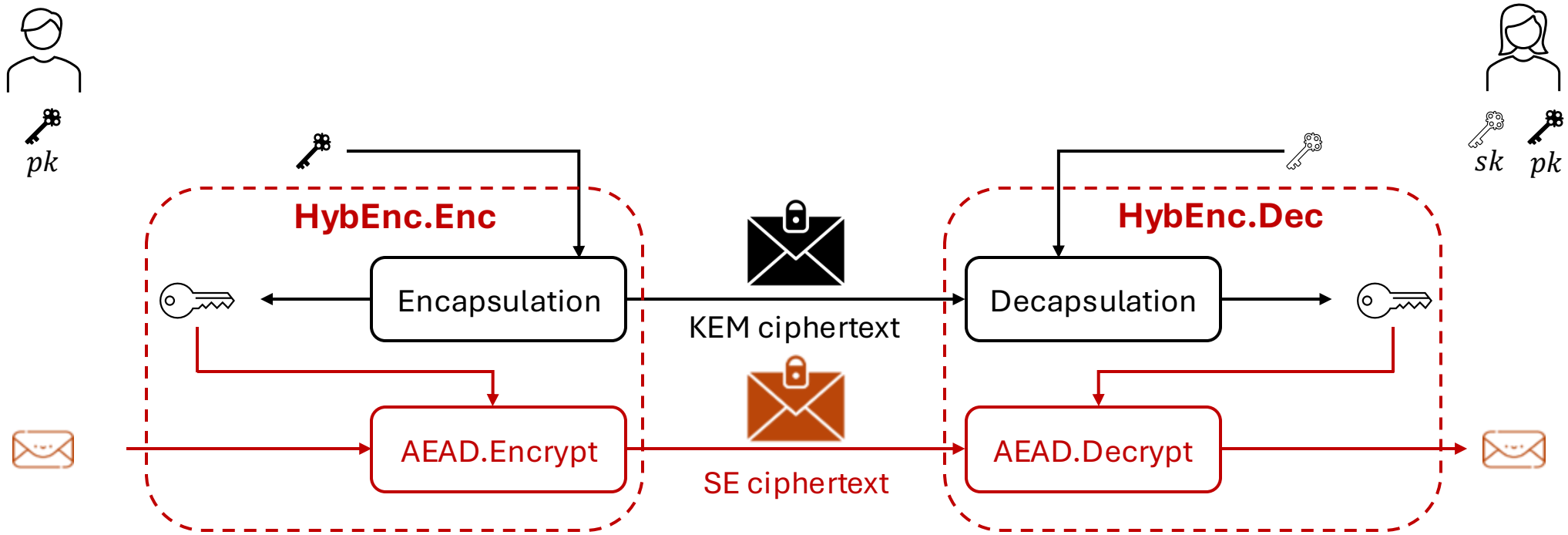
PKE vs KEM

- Relation between KEM and PKE
  - **Build PKE from KEM** (using symmetric encryption, e.g., AEAD)



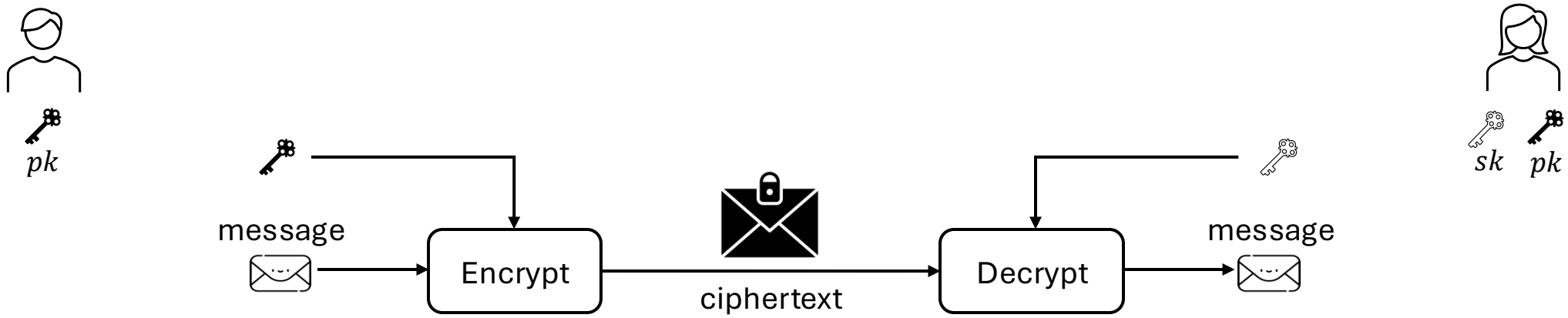
PKE vs KEM

- Relation between KEM and PKE
  - Build PKE from KEM (using symmetric encryption, e.g., AEAD)
  - **Hybrid encryption (HybEnc) scheme**



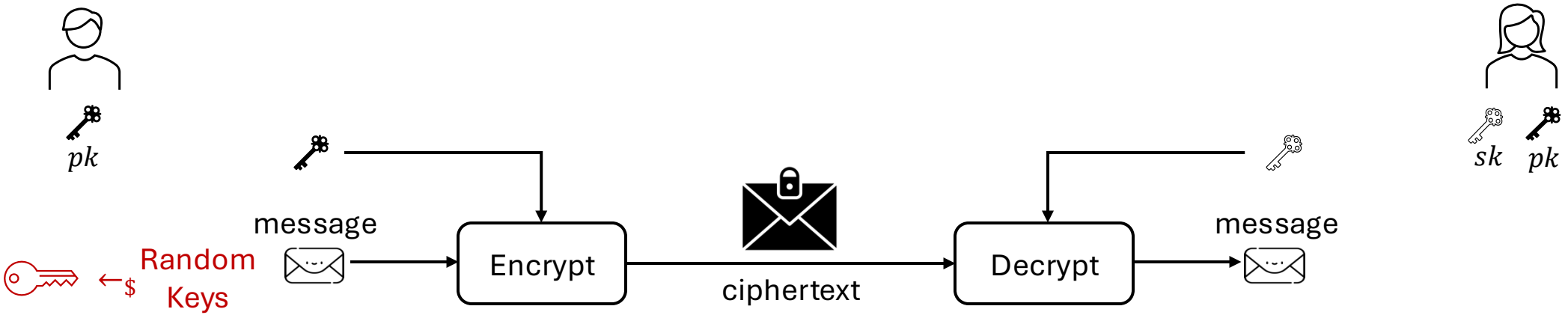
PKE vs KEM

- Relation between KEM and PKE
  - **Build KEM from PKE**



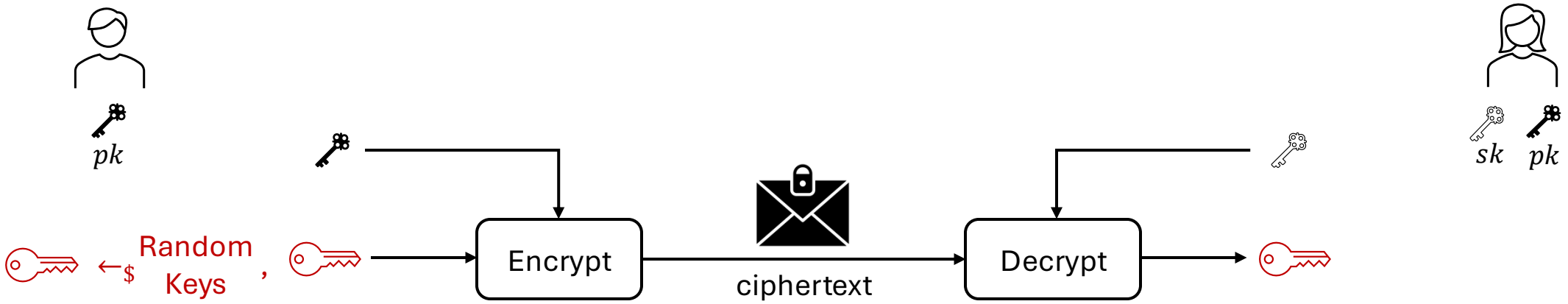
PKE vs KEM

- Relation between KEM and PKE
  - **Build KEM from PKE**



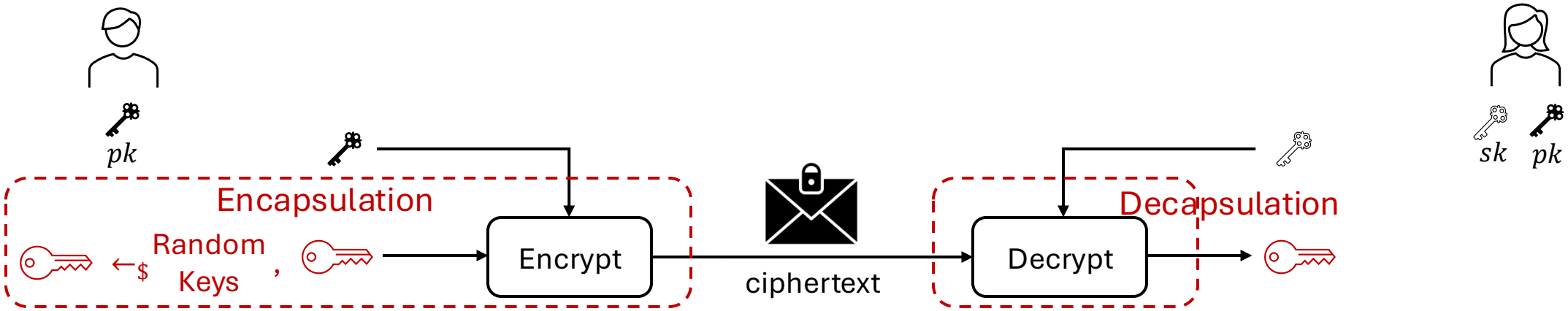
PKE vs KEM

- Relation between KEM and PKE
  - **Build KEM from PKE**



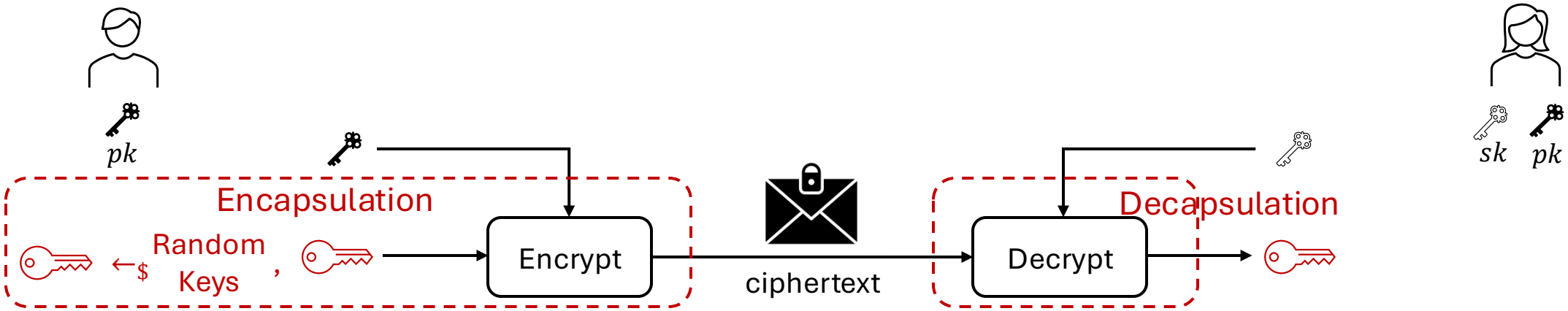
PKE vs KEM

- Relation between KEM and PKE
  - **Build KEM from PKE**



PKE vs KEM

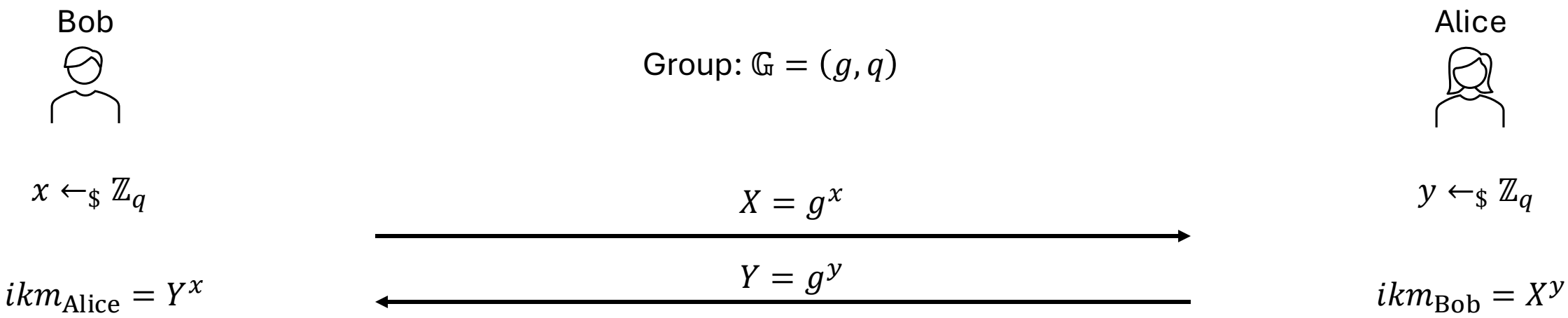
- Relation between KEM and PKE
  - **Build KEM from PKE**



- Message space of PKE = Key space of KEM
- If the message space has an algebraic structure, then the key space does as well.

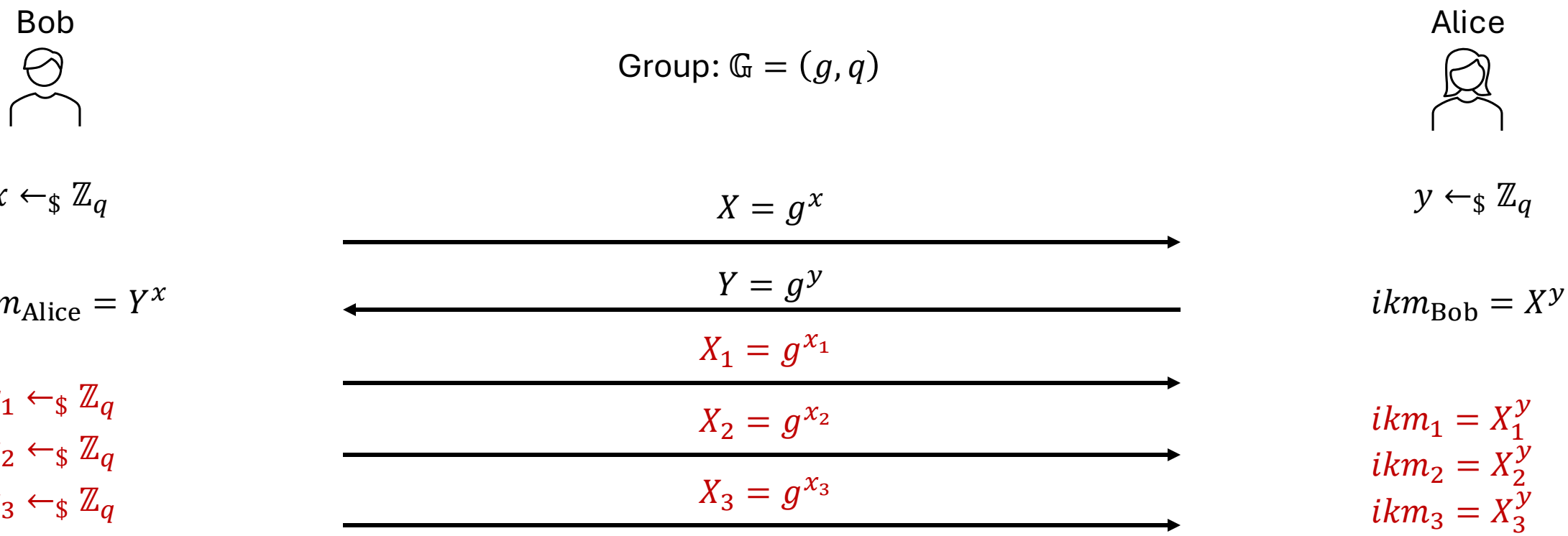
ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE



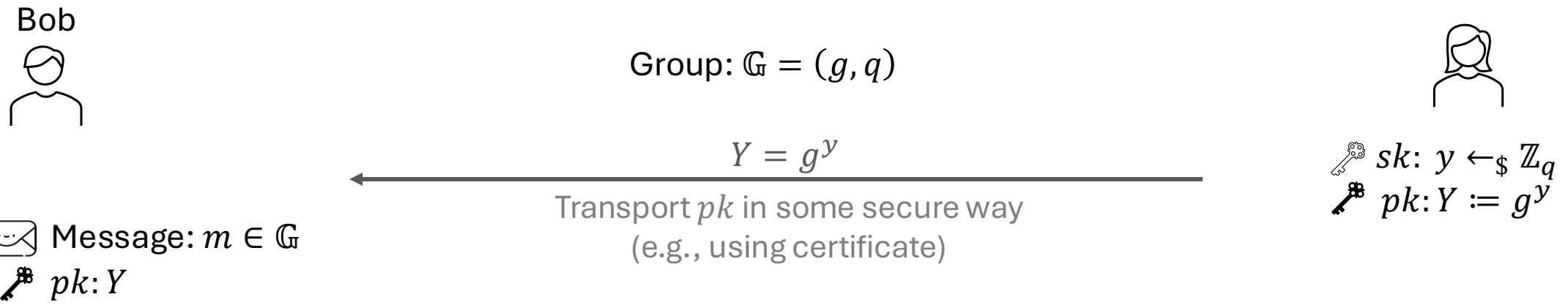
ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)



ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)



ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)

Bob

Group: $\mathbb{G} = (g, q)$

$Y = g^y$

Transport pk in some secure way
(e.g., using certificate)

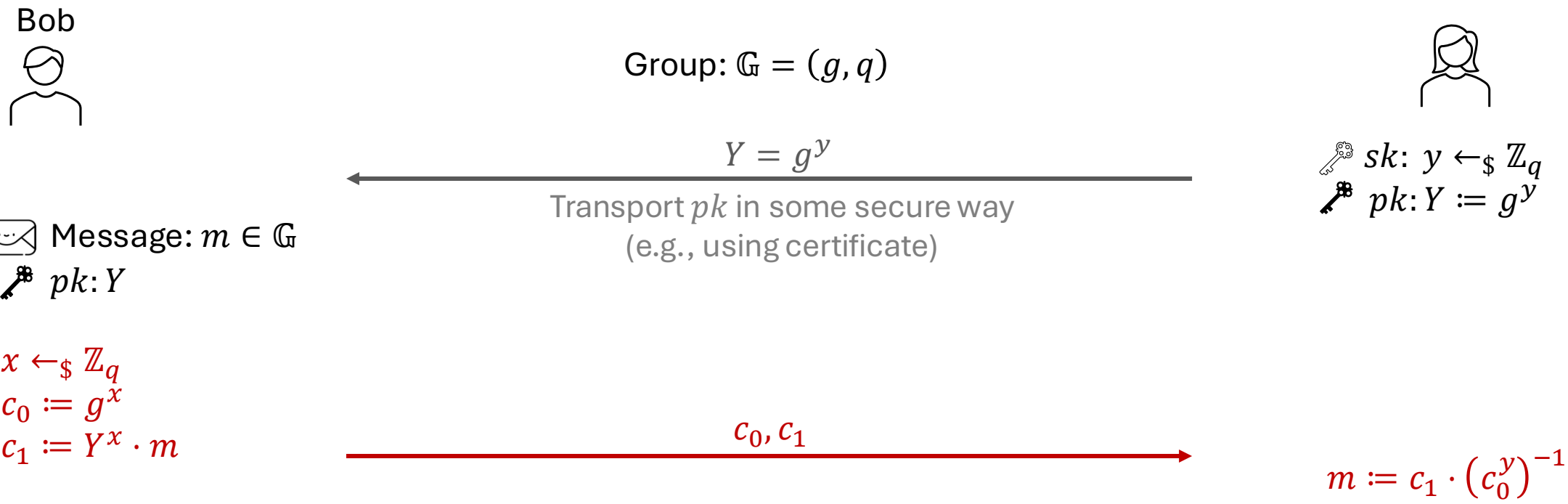
sk: $y \leftarrow_\$ \mathbb{Z}_q$
pk: $Y := g^y$

Message: $m \in \mathbb{G}$
pk: Y

$x \leftarrow_\$ \mathbb{Z}_q$
$c_0 := g^x$
$c_1 := Y^x \cdot m$

ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)



ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)

How can we build a KEM scheme from it?

Bob

Group: $\mathbb{G} = (g, q)$

$Y = g^y$

Transport pk in some secure way
(e.g., using certificate)

sk: $y \leftarrow_{\$} \mathbb{Z}_q$
pk: $Y := g^y$

Message: $m \in \mathbb{G}$
pk: Y

$x \leftarrow_{\$} \mathbb{Z}_q$
$c_0 := g^x$
$c_1 := Y^x \cdot m$

c_0, c_1

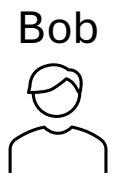
$m := c_1 \cdot (c_0^y)^{-1}$

ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)

How can we build a KEM scheme from it?

Option 1: Let m be the symmetric key



Bob

Group: $\mathbb{G} = (g, q)$

sk: $y \leftarrow_{\$} \mathbb{Z}_q$

pk: $Y := g^y$

$Y = g^y$

Transport pk in some secure way
(e.g., using certificate)

$k \in \mathbb{G}$

pk: Y

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

$c_0 := g^x$

$c_1 := Y^x \cdot k$

c_0, c_1

$k := c_1 \cdot (c_0^y)^{-1}$

ElGamal Encryption

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)

How can we build a KEM scheme from it?

Option 2: HKDF the DH shared secret

Bob

Group: $\mathbb{G} = (g, q)$

$Y = g^y$

Transport pk in some secure way
(e.g., using certificate)

sk: $y \leftarrow_\$ \mathbb{Z}_q$

pk: $Y := g^y$

pk: Y

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

$c_0 := g^x$

$c_1 := Y^x$

$c_0, \cancel{c_1}$

$k := H(c_0, c_1)$

$k := H(c_0, c_0^y)$

Hashed ElGamal KEM

- **ElGamal Encryption**
  - Interpret it from the perspective of DHKE (Reuse Y)

How can we build a KEM scheme from it?

Hashed ElGamal KEM

Bob

Group: $\mathbb{G} = (g, q)$

sk: $y \leftarrow_{\$} \mathbb{Z}_q$

pk: $Y := g^y$

$Y = g^y$

Transport pk in some secure way
(e.g., using certificate)

pk: Y

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

$c_0 := g^x$

$c_1 := Y^x$

c_0, c_1

$k := H(c_0, c_1)$

$k := H(c_0, c_0^y)$

Hashed ElGamal KEM

- Quick exercise: Compare efficiency

	Option 1: Let m be the symmetric key	**Hashed ElGamal KEM**
Encapsulation	$k \in \mathbb{G}$ $x \leftarrow_{\$} \mathbb{Z}_q$ $c_0 := g^x$ $c_1 := Y^x \cdot k$	$x \leftarrow_{\$} \mathbb{Z}_q$ $c_0 := g^x$ $c_1 := Y^x$ $k := H(c_0, c_1)$
Decapsulation	$k := c_1 \cdot (c_0^y)^{-1}$	$k := H(c_0, c_0^y)$

DHIES

- **DHIES**
  - **Hybrid encryption** based on Hashed ElGamal KEM + Symmetric Encryption + MAC

Bob

$pk: Y$

Message: m

DHIES.Encaps

$x \leftarrow_{\$} \mathbb{Z}_q, c := g^x$

$k[1, \ldots,$ SEnc_Len,
SEnc_Len + 1,..., $L] := H(Y, Y^x)$

Alice

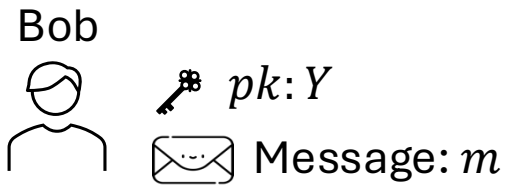
$sk: y \leftarrow_{\$} \mathbb{Z}_q$

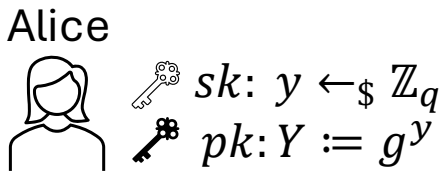
$pk: Y := g^y$

DHIES.Decaps

DHIES

- **DHIES**
  - **Hybrid encryption** based on Hashed ElGamal KEM + Symmetric Encryption + MAC





DHIES.Encaps

$x \leftarrow_{\$} \mathbb{Z}_q, c := g^x$

$k[1, \dots, \text{SEnc\_Len}, \text{SEnc\_Len} + 1, \dots, L] := H(Y, Y^x)$

$k_{\text{enc}} := k[1, \dots, \text{SEnc\_Len}]$

$ct := \text{SE}(k_{\text{enc}}, m)$

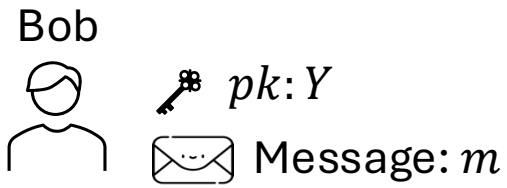
$k_{\text{mac}} := k[\text{SEnc\_Len} + 1, \dots, L]$

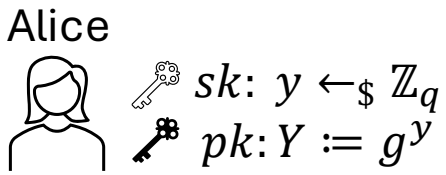
$tag := \text{HMAC}(k_{\text{mac}}, ct)$

DHIES.Decaps

DHIES

- **DHIES**
  - **Hybrid encryption** based on Hashed ElGamal KEM + Symmetric Encryption + MAC





DHIES.Encaps

$x \leftarrow_{\$} \mathbb{Z}_q, c := g^x$

$k[1, \dots, \text{SEnc\_Len}, \text{SEnc\_Len} + 1, \dots, L] := H(Y, Y^x)$

$k_{\text{enc}} := k[1, \dots, \text{SEnc\_Len}]$

$ct := \text{SE}(k_{\text{enc}}, m)$

$k_{\text{mac}} := k[\text{SEnc\_Len} + 1, \dots, L]$

$tag := \text{HMAC}(k_{\text{mac}}, ct)$

$c, ct, tag \longrightarrow$

DHIES.Decaps

DHIES

- **DHIES**
  - **Hybrid encryption** based on Hashed ElGamal KEM + Symmetric Encryption + MAC

Bob

$pk: Y$

Message: m

Alice

$sk: y \leftarrow_{\$} \mathbb{Z}_q$

$pk: Y := g^y$

DHIES.Encaps

$x \leftarrow_{\$} \mathbb{Z}_q, c := g^x$

$k[1, \ldots, \text{SEnc\_Len}, \text{SEnc\_Len} + 1, \ldots, L] := H(Y, Y^x)$

$k_{\text{enc}} := k[1, \ldots, \text{SEnc\_Len}]$

$ct := \text{SE}(k_{\text{enc}}, m)$

$k_{\text{mac}} := k[\text{SEnc\_Len} + 1, \ldots, L]$

$tag := \text{HMAC}(k_{\text{mac}}, ct)$

c, ct, tag →

DHIES.Decaps

Exercise:
Complete the decapsulation.

Hint: DHKE, get the key, and then...

Exercises

- Implement Hashed ElGamal and DHIES
  - Rust: Use the x25519-dalek crate. Use the StaticSecret struct as key pairs.

- Thinking question:
  - In DHIES, we split the hashed key into a symmetric key and a MAC key. Now we want to use HKDF.Extract and HKDF.Expand to get these two keys. How can we do this?