Cryptography Engineering

- Lecture 12 (Feb 05, 2025)
- Today's notes:
  - Key Encapsulation Mechanism
  - CRYSTAL-Kyber
  - CRYSTAL-Dilithium
  - From Pre-Quantum to Post-Quantum

Key Encapsulation Mechanism

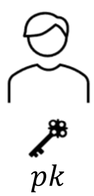
- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)

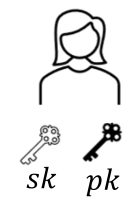
- PKE: Asymmetric setting, Encryption/Decryption
  - Encrypt messages

- KEM: Asymmetric setting, **Encapsulation/Decapsulation**
  - “Encrypt” keys

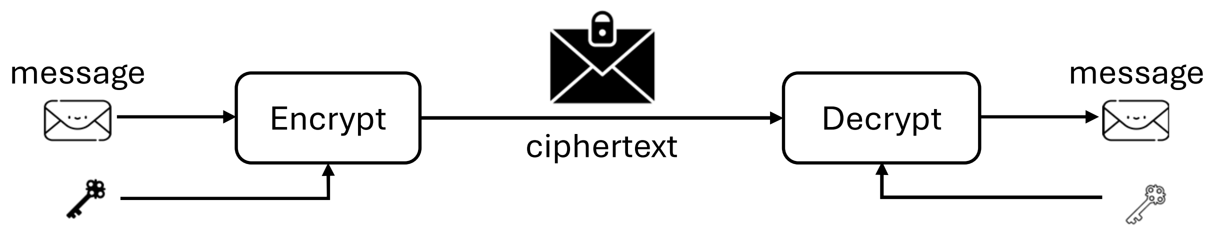
Key Encapsulation Mechanism

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



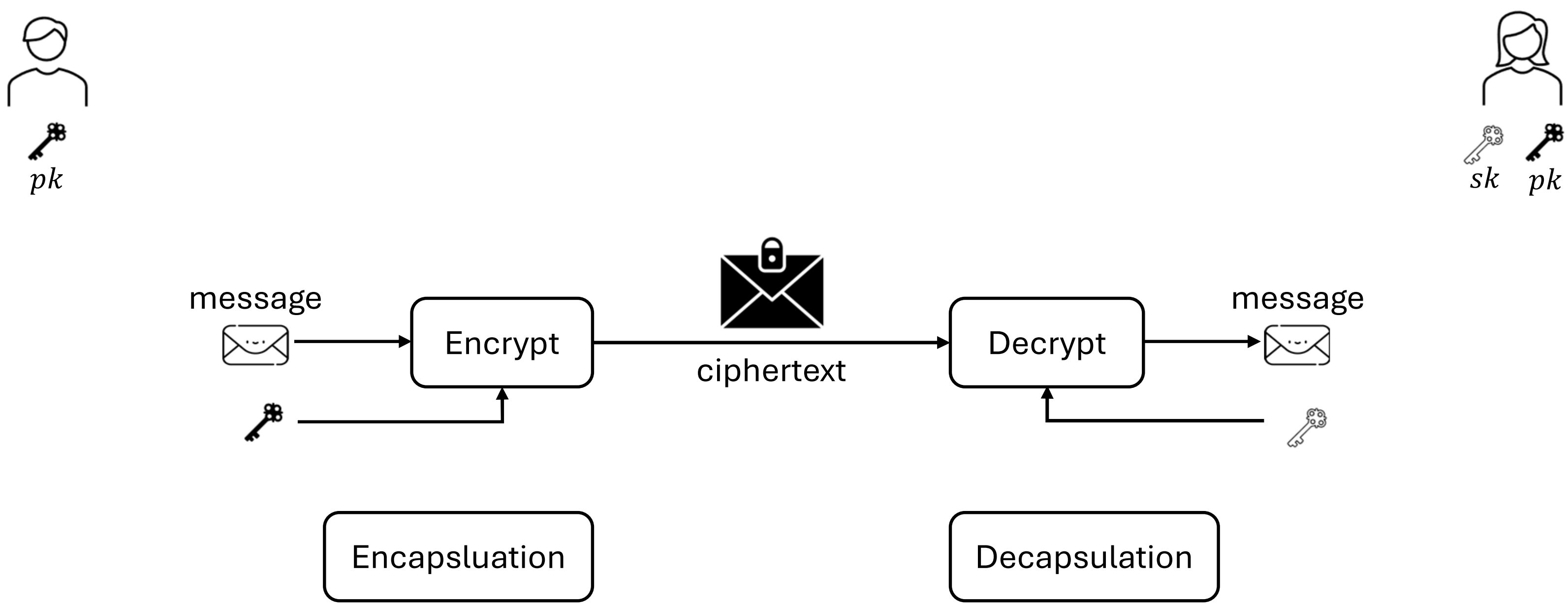


- PKE:



Key Encapsulation Mechanism

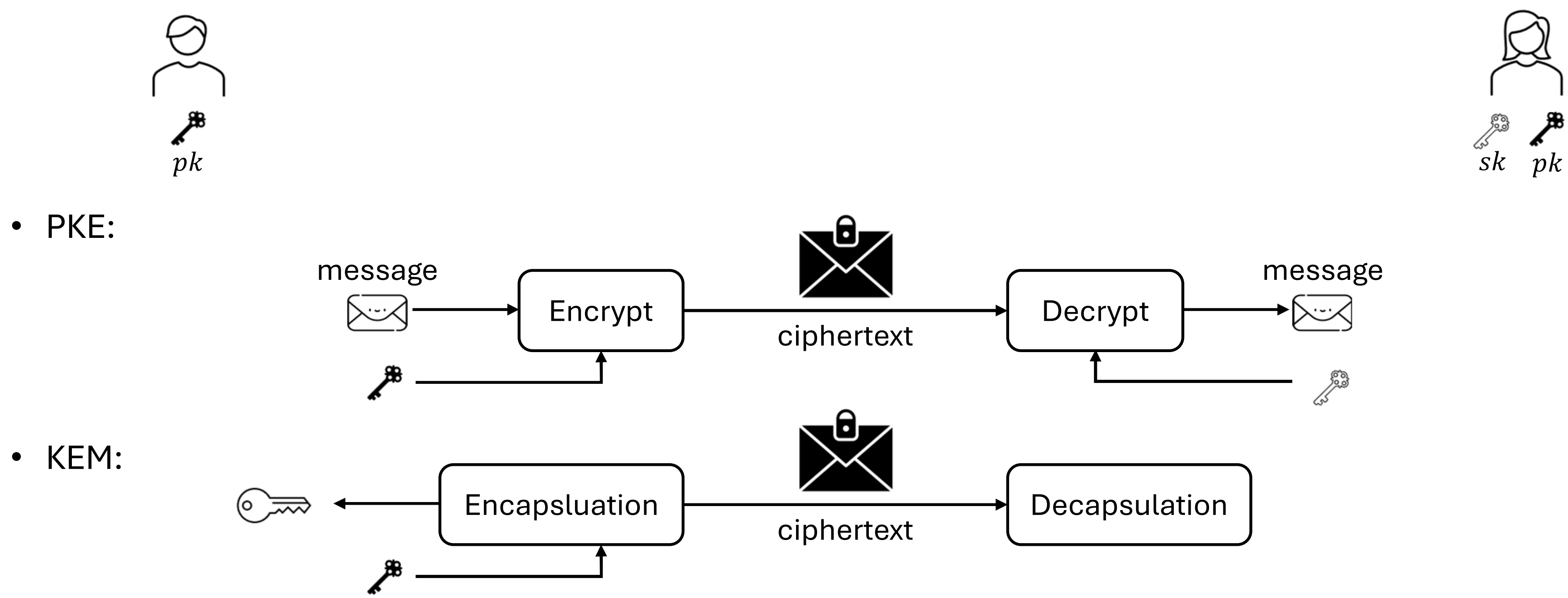
- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)



- PKE:
- KEM:

Key Encapsulation Mechanism

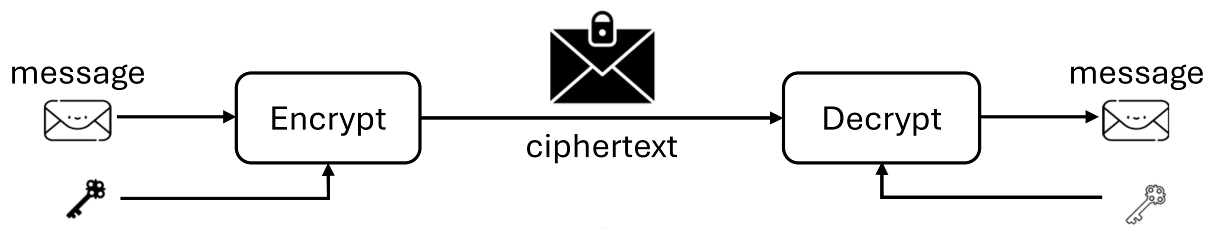
- Key Encapsulation Mechanism (KEM) v.s. Public-key Encryption (PKE)



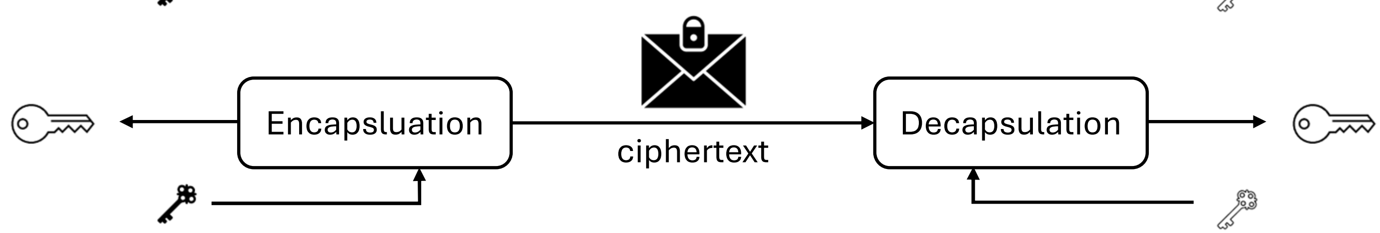
Key Encapsulation Mechanism

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

- PKE:



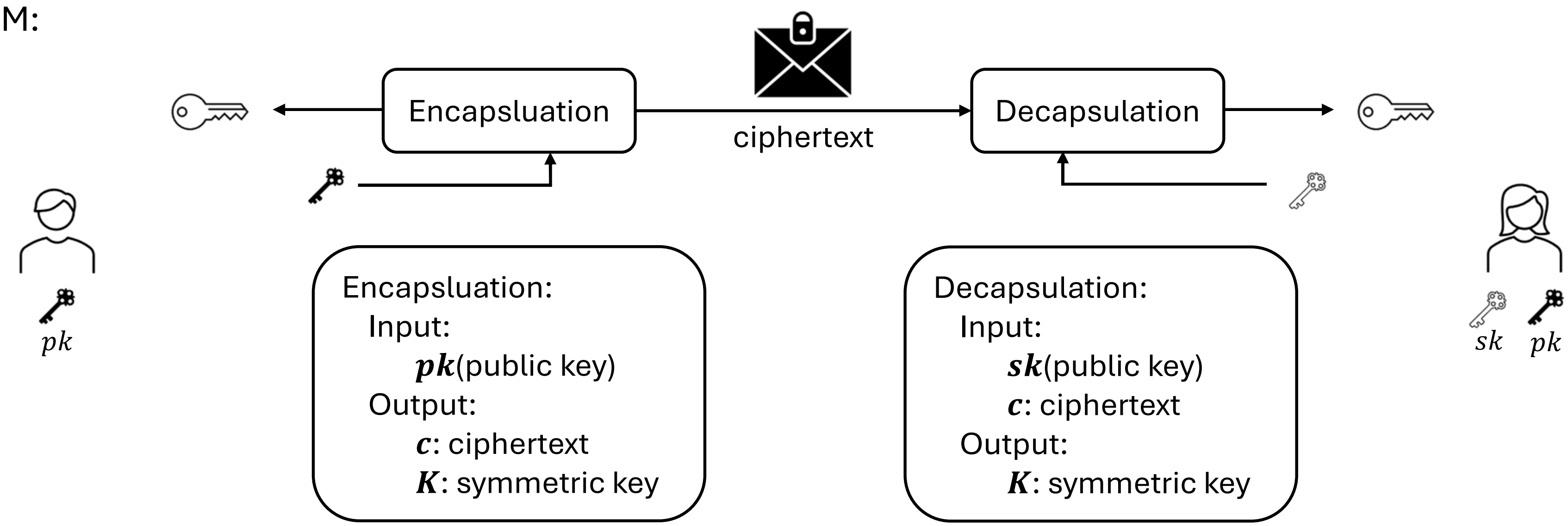
- KEM:



Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

- KEM:

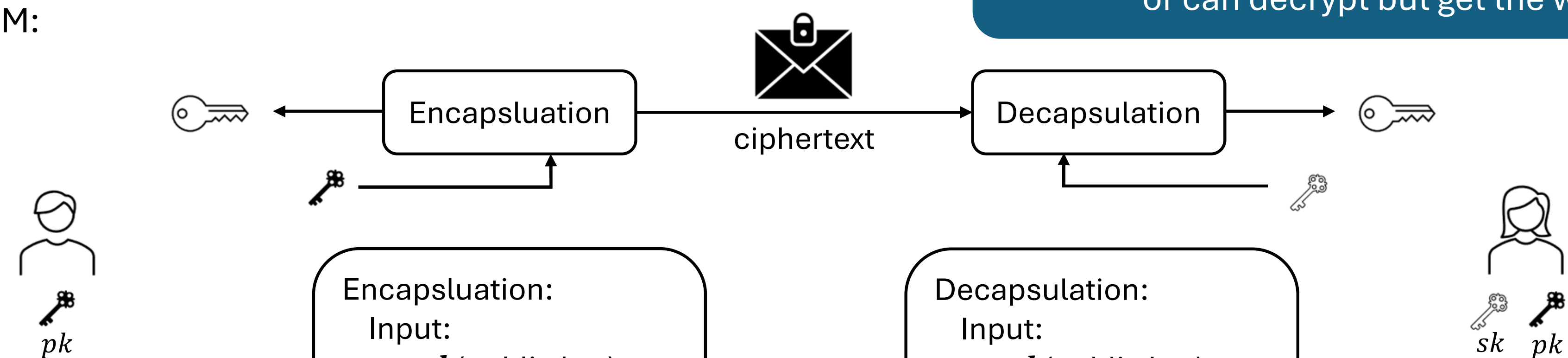


Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

Security:
No $\boldsymbol{sk}$ => Cannot decrypt $\boldsymbol{c}$
or can decrypt but get the wrong $\boldsymbol{K}$

- KEM:



Encapsluation:
Input:
$\boldsymbol{pk}$(public key)
Output:
$\boldsymbol{c}$: ciphertext
$\boldsymbol{K}$: symmetric key

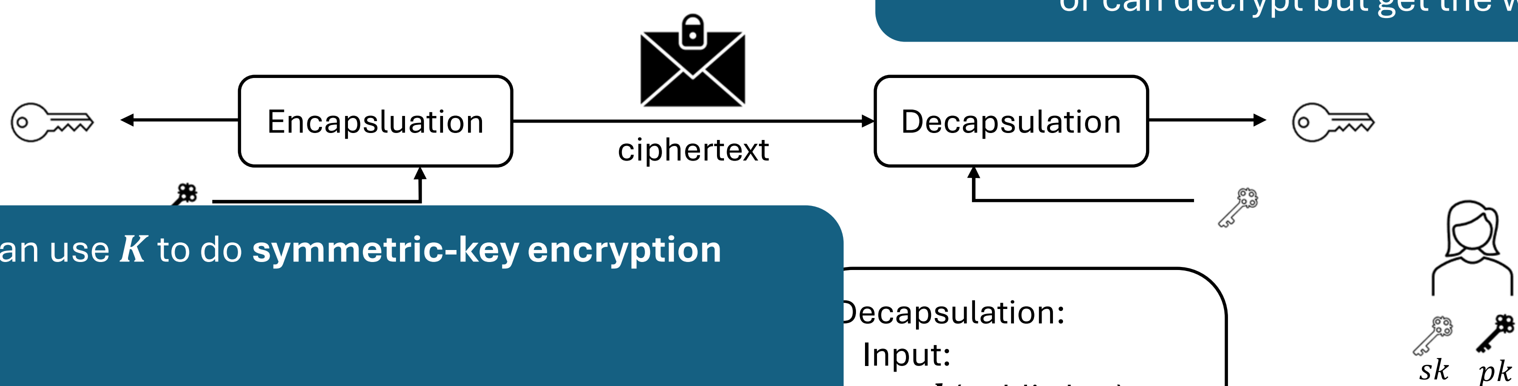
Decapsulation:
Input:
$\boldsymbol{sk}$(public key)
$\boldsymbol{c}$: ciphertext
Output:
$\boldsymbol{K}$: symmetric key

Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

Security:
No $\boldsymbol{sk}$ => Cannot decrypt $\boldsymbol{c}$
or can decrypt but get the wrong $\boldsymbol{K}$

- KEM:



One can use $\boldsymbol{K}$ to do **symmetric-key encryption**

Output:
$\boldsymbol{c}$: ciphertext
$\boldsymbol{K}$: symmetric key

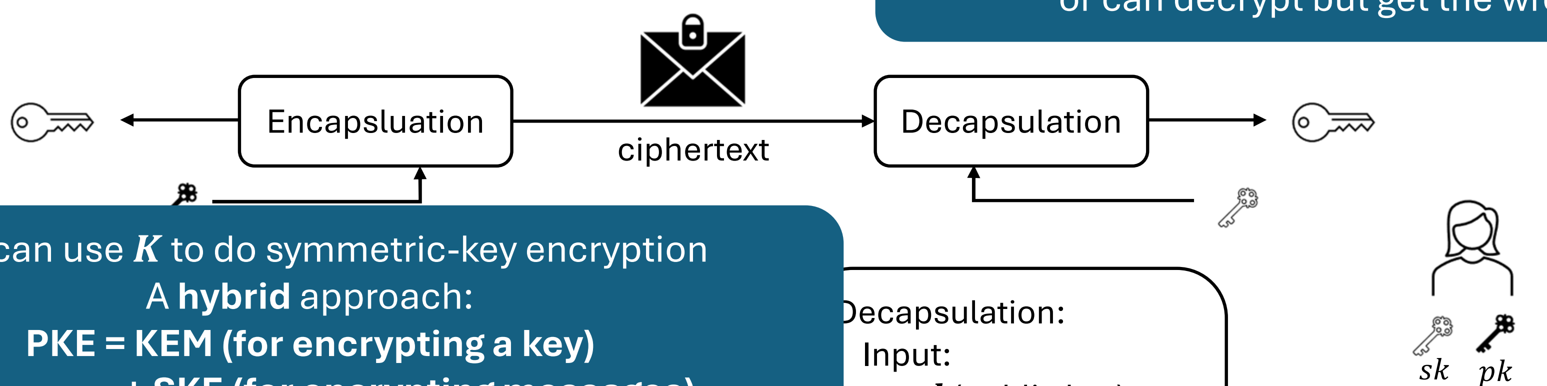
Decapsulation:
Input:
$\boldsymbol{sk}$(public key)
$\boldsymbol{c}$: ciphertext
Output:
$\boldsymbol{K}$: symmetric key

Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

Security:
No $\boldsymbol{sk}$ => Cannot decrypt $\boldsymbol{c}$
or can decrypt but get the wrong $\boldsymbol{K}$

- KEM:



One can use $\boldsymbol{K}$ to do symmetric-key encryption
A **hybrid** approach:
PKE = KEM (for encrypting a key)
+ SKE (for encrypting messages)

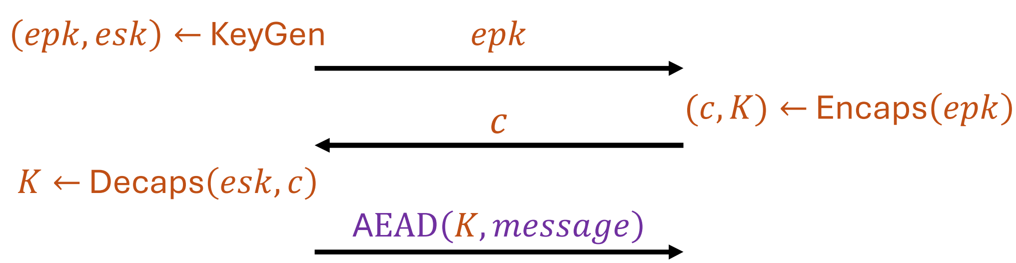
Output:
$\boldsymbol{c}$: ciphertext
$\boldsymbol{K}$: symmetric key

Decapsulation:
Input:
$\boldsymbol{sk}$(public key)
$\boldsymbol{c}$: ciphertext
Output:
$\boldsymbol{K}$: symmetric key

Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

A simple Key Exchange
based on KEM

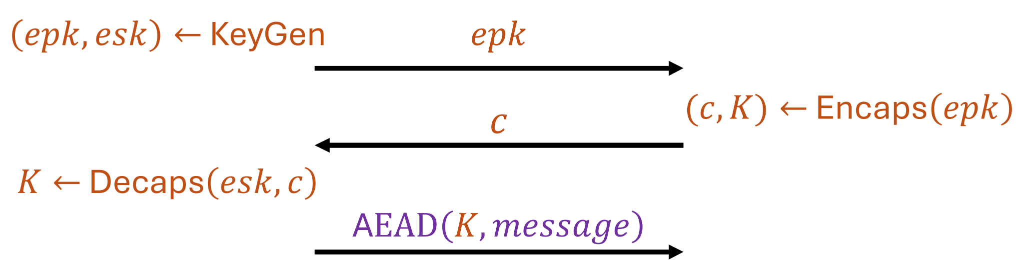


- Post-quantum secure KEM: **Construct post-quantum PKE, KE, …**

Key Encapsulation Mechanism

- Key Encapsulation Mechanism (KEM)

A simple Key Exchange
based on KEM



- Post-quantum secure KEM: **Construct post-quantum PKE, KE, …**
- Why we prefer using KEM to do encryption in the post-quantum cryptography:
  - Encrypt long messages, simpler structure (compared to pure PKE), known secure generic constructions…

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - Based on MLWE
  - ML-KEM [FIPS203]: based on CRYSTALS-Kyber

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **key pair**? What does it look like?
  - In DH-based schemes: $(\mathrm{sk} = a, \mathrm{pk} = g^a)$ with some group generator g

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **key pair**? What does it look like?
  - In DH-based schemes: $(\mathrm{sk} = a, \mathrm{pk} = g^a)$ with some group generator g
  - In some lattice-based schemes: $\left(\mathrm{sk} = \boldsymbol{s}, \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b})\right)$, where

$$\boldsymbol{s} = \left\{\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}\right\}$$

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **key pair**? What does it look like?
  - In DH-based schemes: $(\mathrm{sk} = a, \mathrm{pk} = g^a)$ with some group generator g
  - In some lattice-based schemes: $\big(\mathrm{sk} = \boldsymbol{s}, \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b})\big)$, where

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \qquad \boldsymbol{A} = \begin{pmatrix} a_{11} \cdots a_{1m} \\ a_{21} \cdots a_{2m} \\ \vdots \quad \vdots \quad \vdots \\ a_{n1} \cdots a_{nm} \end{pmatrix} \qquad \boldsymbol{b} = \boldsymbol{A}^T \boldsymbol{s} + \boldsymbol{e}, \text{ where } \boldsymbol{e} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{pmatrix} \text{ is a short vector}$$

(n < m, a “big-fat” random matrix)

  - Distribution of the short vectors: Discrete Gaussian, centered binomial distribution (Kyber)...

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **key pair**? What does it look like?
  - In DH-based schemes: $(\mathrm{sk} = a, \mathrm{pk} = g^a)$ with some group generator g
  - In some lattice-based schemes: $\left(\mathrm{sk} = \boldsymbol{s}, \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b})\right)$, where

$$\boldsymbol{s} = \begin{Bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{Bmatrix} \qquad \boldsymbol{A} = \begin{Bmatrix} a_{11} \cdots a_{1m} \\ a_{21} \cdots a_{2m} \\ \vdots \quad \vdots \quad \vdots \\ a_{n1} \cdots a_{nm} \end{Bmatrix} \qquad \boldsymbol{b} = \boldsymbol{A}^T\boldsymbol{s} + \boldsymbol{e}, \text{ where } \boldsymbol{e} = \begin{Bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{Bmatrix} \text{ is a short vector}$$

(n < m, a "big-fat" random matrix)

  - Distribution of the short vectors: Discrete Gaussian, centered binomial distribution (Kyber)…

  - In CRYSTAL-Kyber: Similar structure, but more compact and over module lattice…

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **ciphertext**? (Suppose the message is m)
  - In Hashed ElGamal: $c = (g^r, H(X^r) \oplus m)$

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **ciphertext**? (Suppose the message is m)
  - In Hashed ElGamal: $c = (g^r, H(X^r) \oplus m)$
  - (Simplified) Kyber.PKE's key pair: $\left(\mathrm{sk} = \boldsymbol{s}, \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b})\right)$
  - (Simplified) Kyber.PKE's ciphertext: $(\boldsymbol{c_0}, c_1)$, where:

$\boldsymbol{c_0} = \boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{r}, c_1 = \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{r} + \boldsymbol{r}' + m$, where $\boldsymbol{r}, \boldsymbol{r}'$ are fresh generated short vectors

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - What is the format of the **ciphertext**? (Suppose the message is m)
  - (Simplified) Kyber.PKE's key pair: $\left(\mathrm{sk} = \boldsymbol{s}, \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b})\right)$
  - (Simplified) Kyber.PKE's ciphertext: $(\boldsymbol{c_0}, c_1)$, where:

$$\boldsymbol{c_0} = \boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{r}, c_1 = \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{r} + \boldsymbol{r}' + m, \text{ where } \boldsymbol{r}, \boldsymbol{r}' \text{ are fresh generated short vectors}$$

  - Optimizations:
    - All multiplications are over NTT (Number-theoretic transform)
    - Ciphertext compression techniques of lattice-based schemes…
  - Security guarantee: Someone learns m from $(\boldsymbol{c_0}, c_1)$ => It breaks the MLWE problem…

Key Encapsulation Mechanism

- CRYSTALS-Kyber
  - (Simplified) Kyber.PKE's key pair: $\left(\mathrm{sk} = \boldsymbol{s}, \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b})\right)$
  - (Simplified) Kyber.PKE's ciphertext: $(\boldsymbol{c_0}, c_1)$, where:

$$\boldsymbol{c_0} = \boldsymbol{A^T r}, c_1 = \boldsymbol{b^T r} + \boldsymbol{r}' + m, \text{ where } \boldsymbol{r}, \boldsymbol{r}' \text{ are fresh generated short vectors}$$

  - Optimizations:
    - All multiplications are over NTT (Number-theoretic transform)
    - Ciphertext compression techniques of lattice-based schemes…
  - Security guarantee: Someone learns m from $(\boldsymbol{c_0}, c_1)$ => It breaks the MLWE problem…

  - **Kyber.KEM:** The KEM scheme we actually use. Based on Kyber.PKE…

Signature

- CRYSTALS-Dilithium
  - Format of the **key**: $\left(\mathrm{sk} = (\boldsymbol{s_1}, \boldsymbol{s_2}), \mathrm{pk} = (\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s_1} + \boldsymbol{s_2})\right)$
    (where $\boldsymbol{A}$ is a random matrix and $\boldsymbol{s_1}, \boldsymbol{s_2}$ are short vectors)
  - Format of the signature of a message M: $(c, \boldsymbol{z}, \boldsymbol{h})$,
    - c: A challenge related to M
    - $\boldsymbol{z}$: A short vector (can be only generated by using $(\boldsymbol{s_1}, \boldsymbol{s_2})$
    - $\boldsymbol{h}$: A hint vector with low hamming weight

  - Optimizations: All multiplications are over NTT (Number-theoretic transform)

  - Security guarantee: Someone forges a signature => It breaks the MSIS problem…

From Pre-Quantum to Post-Quantum

- Hybrid Cryptography
  - Classical algorithms + post-quantum algorithms
  - Example: ECDH in TLS 1.3 -> ECDH + Kyber in TLS

From Pre-Quantum to Post-Quantum

- Hybrid Cryptography
  - Classical algorithms + post-quantum algorithms
  - Example: ECDH in TLS 1.3 -> ECDH + Kyber in TLS

The ECDH in TLS 1.3

g^x, client_nonce →

← g^y, server_nonce, …

⋮

A simple KE based on Kyber KEM

$(epk, esk) \leftarrow$ KeyGen

epk →

$(c, K) \leftarrow$ Encaps(epk)

← c

$K \leftarrow$ Decaps(esk, c)

- Advantages: Classical security provided by ECDH + Quantum security provided by Kyber

From Pre-Quantum to Post-Quantum

- Hybrid Cryptography
  - Classical algorithms + post-quantum algorithms
  - Example: ECDH in TLS 1.3 -> ECDH + Kyber in TLS (still use the classical signature scheme)

ECDH+ Kyber KEM

$(epk, esk) \leftarrow$ KeyGen $\quad g^x$, epk, client_nonce $\longrightarrow$

$\longleftarrow$ g^y, c, server_nonce, … $\quad (c, K) \leftarrow$ Encaps(epk)

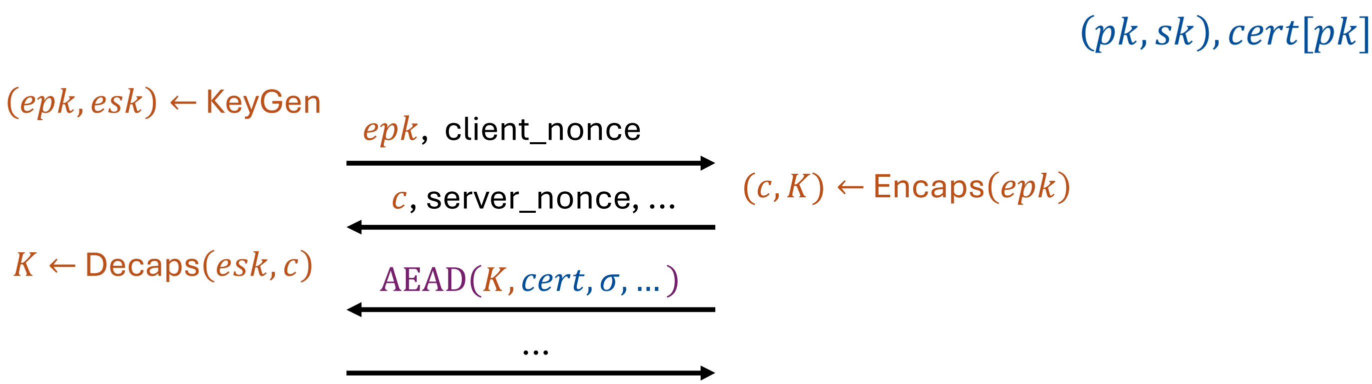
$K \leftarrow$ Decaps(esk, c)

⋮

Keys = KeySchedule(…|| g^{xy} || K ||…)

From Pre-Quantum to Post-Quantum

- PQTLS = PQKEM + PQSign
  - (Only show the difference to TLS 1.3...)



Keys = KeySchedule(...|| K ||...)

From Pre-Quantum to Post-Quantum

- Another PQ-secure variant of TLS: KEM-TLS
  - Do the TLS handshake purely on KEM scheme, without signature
  - Pros: PQ signature is slower than PQ KEM.
  - Cons: Most servers still need signature scheme, and normally, signature schemes can do more things than KEM…

- Some other PQ replacements (or need to be replaced):
  - X3DH -> PQXDH -> (fully PQ-secure X3DH-style protocols…)
  - PQ-secure Double Ratchet (Unknown)
  - PQ-secure Password-based AKE (Unknown)
  - PQ-secure OPRF (Unknown)
  - …

Many open problems!

Exercises

- Find available python implementation of Kyber and Dilithium.
  - https://github.com/GiacomoPope/dilithium-py
  - https://github.com/GiacomoPope/kyber-py
- (3 points) Implement the PQTLS protocol using Kyber and Dilithium.
  - Hint: Use the similar key schedule algorithm yourself

Further Reading

- Page of CRYSTALS-Kyber: https://pq-crystals.org/kyber/
- Page of CRYSTALS-Dilithium: https://pq-crystals.org/dilithium/
- KEMTLS: https://kemtls.org/
- The PQXDH protocol: https://signal.org/docs/specifications/pqxdh/
- iMessage with PQ3: https://security.apple.com/blog/imessage-pq3/