Public Key Infrastructure and the Full Picture

DSC 4110/6111: Modern Cryptography and AI/ML Security

Dr. Laltu Sardar

School of Data Science,
Indian Institute of Science Education and Research Thiruvananthapuram (IISER TVM)

August 7, 2026

Table of Content

Recommended Reading From textbook: Chapter *

Revisiting the Full Picture of Secure Communication

Secure Communication: All four phases

1. **Setting Up Trust**
   - A trusted Certificate Authority (CA) establishes the root of trust.
   - Each party generates a public/private key pair.
   - The CA verifies the party's identity and issues a digital certificate binding the identity to its public key.
2. **Establishing Trust Between Parties**
   - Parties exchange their certificates over the public channel.
   - Each party verifies the CA's signature using a trusted CA public key.
   - This authenticates the public keys and prevents Man-in-the-Middle attacks.
3. **Establishing a Secure Session**
   - Parties perform an authenticated key exchange such as DH/ECDH.
   - Both derive the same shared secret without transmitting it.
   - A KDF converts the shared secret into suitable session keys.
4. **Secure Communication**
   - Symmetric encryption provides efficient confidentiality for long messages.
   - AEAD provides confidentiality, integrity, and authentication; AAD is authenticated but not encrypted.
   - A fresh nonce is used for each message; keys may be refreshed periodically and discarded when the session ends.

Phase 1: CA Establishes the Root of Trust

The Certificate Authority (CA) first generates its own key pair

$$(pk_{CA}, sk_{CA}).$$

The CA creates a root certificate containing its public key and identity.

$$\boxed{\text{Root Certificate} = \text{Identity}_{CA} \parallel pk_{CA} \parallel \text{Validity} \parallel \cdots}$$

The root certificate is distributed through a trusted mechanism and the CA public key becomes a

$$\boxed{\text{Root of Trust.}}$$

Examples of trust stores: Operating Systems Browsers Applications

Informally, CAs are the organizations from which we purchase domains, such as Let's Encrypt (ISRG), GlobalSign, Sectigo, and GoDaddy.

Phase 1: Party Generates Its Key Pair

Suppose party A wants to communicate securely.
A generates

$$\boxed{(pk_A, sk_A) \leftarrow \mathsf{KeyGen}(1^\lambda).}$$

- pk_A is the public key and may be distributed.
- sk_A is the secret key and must remain under A's control.
- The secret key is never sent to the CA or to other parties.

A now requests a certificate binding

$$\boxed{\mathsf{Identity}_A \longleftrightarrow pk_A.}$$

Phase 1: Certificate Signing Request and Issuance

A sends a Certificate Signing Request (CSR) containing

$$(\text{Identity}_A, pk_A, \text{additional information})$$

to the CA.

The CA verifies that A controls the claimed identity.

The CA then creates and signs a certificate

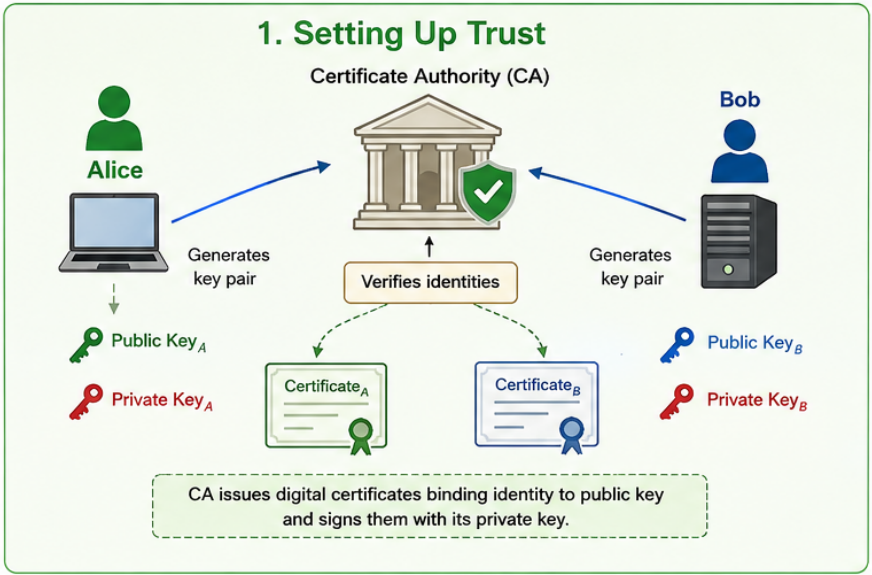
$$\boxed{Cert_A = \text{Sign}_{sk_{CA}}(\text{Identity}_A \parallel pk_A \parallel \text{Validity} \parallel \cdots).}$$

The CA returns $Cert_A$ to A.

Important

The CA does not need to know sk_A. A proves possession of the corresponding secret key during the certificate enrollment process.

Phase 1: Summary



Phase 2: Certificate Exchange

When A initiates communication with B, A presents

$$\boxed{Cert_A}$$

to B.

The certificate contains

$$(\text{Identity}_A, \quad pk_A, \quad \text{Validity}, \quad \text{CA signature.})$$

B **already trusts the CA public key**

$$pk_{CA}.$$

Therefore, B can verify

$$\text{Verify}_{pk_{CA}}(Cert_A) = 1.$$

$$\boxed{\text{Certificate} \Longrightarrow \text{Authenticated Public Key}}$$

Phase 2: Preventing the Man-in-the-Middle Attack

Without certificate verification, an attacker could replace

$$pk_A$$

with the attacker's public key

$$pk_E.$$

The certificate prevents this because the attacker cannot modify

$$\text{Identity}_A \parallel pk_A$$

without forging the CA's signature.

B accepts pk_A only if

$$\boxed{\text{Verify}_{pk_{CA}}(\text{Identity}_A \parallel pk_A, \sigma_{CA}) = 1.}$$

Result

The public key is not merely public; it is **authenticated**.

Phase 2: Mutual Authentication

If both parties need authentication, B also possesses

$$(pk_B, sk_B, Cert_B).$$

Then both parties verify each other's certificates.

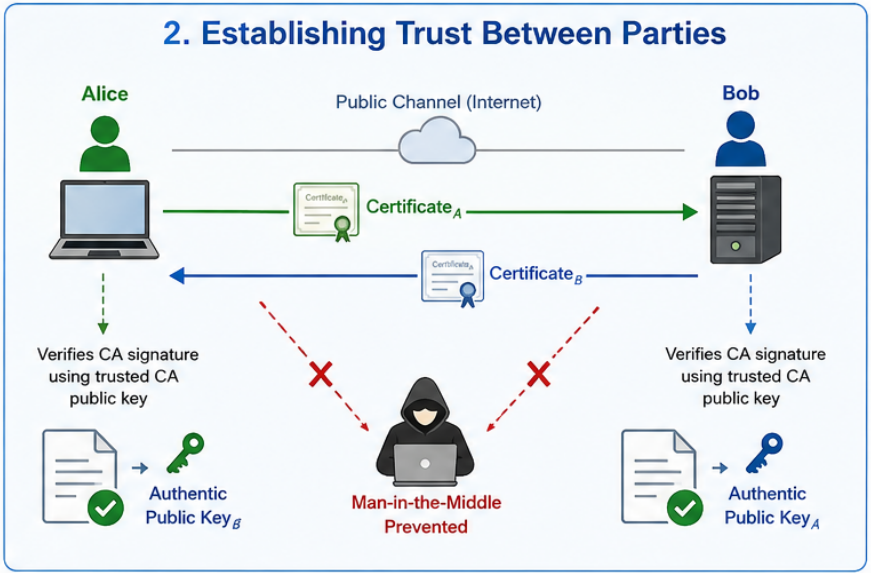
$$A \xleftrightarrow[Cert_B]{Cert_A} B$$

Thus,

$$\boxed{A \text{ authenticates } B \quad \text{and} \quad B \text{ authenticates } A.}$$

This is commonly called
mutual authentication.

Phase 2: Summary



Phase 3: Key Exchange

After authentication, A and B establish a shared secret.
For Diffie–Hellman:

$$a, b \xleftarrow{\$} \mathbb{Z}_q$$

and compute

$$A = g^a, \qquad B = g^b.$$

They exchange A and B over the public channel.

$$\boxed{K = B^a = A^b = g^{ab}.}$$

The values g^a and g^b do **not** need to be encrypted.
They need to be **authenticated** so that an attacker cannot replace them.

Phase 3: Authentication of the Key Exchange

The ephemeral key-exchange values must be bound to the authenticated identities. Conceptually,

$$A = g^a, \qquad \sigma_A = \mathrm{Sign}_{sk_A}(A \,\|\, \text{context}),$$

and similarly

$$B = g^b, \qquad \sigma_B = \mathrm{Sign}_{sk_B}(B \,\|\, \text{context}).$$

The receiver verifies the signatures using the already authenticated public keys.

$$\boxed{\text{Authenticated Public Keys} + \text{Authenticated DH Values}}$$

$$\Longrightarrow \boxed{\text{Authenticated Key Establishment}}$$

Phase 3: From Shared Secret to Session Keys

The DH shared secret

$$K = g^{ab}$$

is generally not used directly as an encryption key.

Both parties apply a Key Derivation Function (KDF):

$$\boxed{K_{\text{session}} = \text{KDF}(K, \text{context})}$$

The KDF can derive multiple independent keys:

$$K \longrightarrow \begin{cases} K_{\text{enc}} \\ K_{\text{auth}} \\ K_{\text{other}} \end{cases}$$

The resulting keys have the required length and are suitable for symmetric cryptographic algorithms.

Phase 3: Secure Session Established

After key derivation,

$$\boxed{\text{Public-Key Cryptography} \longrightarrow \text{Shared Secret} \longrightarrow \text{KDF} \longrightarrow \text{Session Keys}}$$

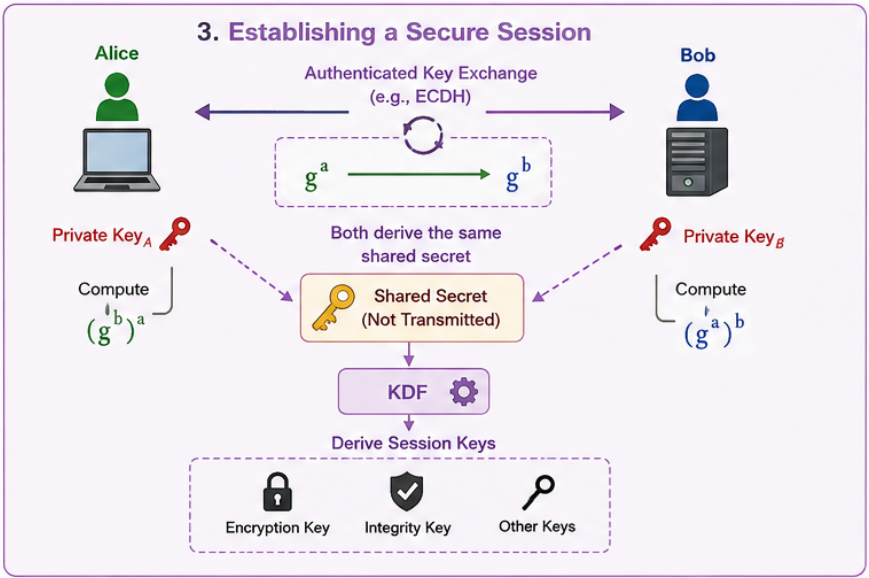
The expensive public-key operations are now finished.

For subsequent communication, A and B use efficient symmetric cryptography (Block Cipher / Stream Cipher).

$$\boxed{\text{Session Key} \longrightarrow \text{AES-GCM / ChaCha20-Poly1305}}$$

IISER THIRUVANANTHAPURAM

Phase 3: Summary



Phase 4: Session Key to Secure Communication

For every message M_i, a fresh nonce N_i is selected.
An AEAD scheme computes

$$(C_i, T_i) = \mathrm{Enc}_{K_{\mathrm{session}}}(N_i, A_i, M_i).$$

The transmitted data is conceptually

$$\boxed{N_i \parallel A_i \parallel C_i \parallel T_i.}$$

where

- N_i : nonce,
- A_i : associated data,
- C_i : ciphertext,
- T_i : authentication tag.

The nonce is normally public; its critical requirement is appropriate uniqueness under the relevant key.

Phase 4: Decryption and Verification

The receiver obtains

$$(N, A, C, T).$$

The AEAD algorithm first verifies the authentication tag.

$$\text{Verify}(K, N, A, C, T)$$

If verification fails,

$$\boxed{\text{Reject}}$$

and the ciphertext must not be trusted.
If verification succeeds,

$$\boxed{M = \text{Dec}_K(N, A, C, T).}$$

Thus AEAD provides

$$\boxed{\text{Confidentiality} + \text{Integrity} + \text{Authentication}.}$$

AAD is authenticated but remains unencrypted.

Phase 4: Rekeying and Session Termination

A long-lived session may periodically refresh its keys.

$$K_1 \longrightarrow K_2 \longrightarrow K_3 \longrightarrow \cdots$$

Rekeying limits the amount of data protected under a single key and can provide additional security properties.

When communication terminates,

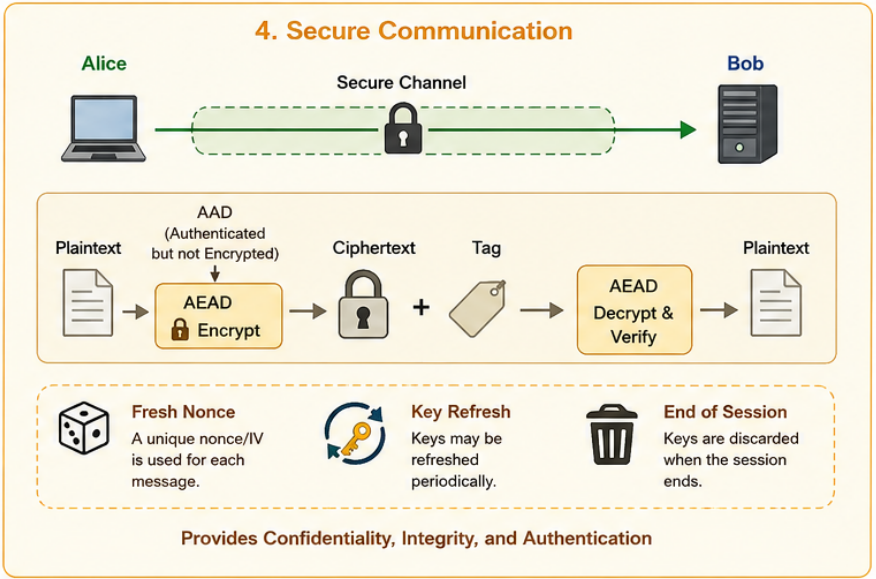
$$\boxed{\text{Session Keys} \longrightarrow \text{Discarded}}$$

For ephemeral Diffie–Hellman, destruction of ephemeral secret values also supports

$$\boxed{\text{Perfect Forward Secrecy (PFS)}}$$

when the protocol is designed appropriately.

Phase 4: Summary



Fundamental Cryptographic Primitives

Primitive	Purpose	Examples
Block Cipher	Confidentiality for fixed-size blocks; used with modes of operation for long messages	AES, SM4
Stream Cipher	Confidentiality by generating a pseudorandom keystream	ChaCha20, Salsa20
Hash Function	Fixed-length digest for integrity and fingerprinting	SHA-256, SHA-3
Public-Key Encryption	Securely transmit secrets without a pre-shared key	RSA, ElGamal, ML-KEM
Digital Signature	Authentication, integrity, and non-repudiation	DSA, ECDSA, ML-DSA

Observation

Modern cryptographic protocols are built by combining these primitives. For example,

Public-Key Encryption $\rightarrow$ Key Exchange $\rightarrow$ Session Key $\rightarrow$ Block/Stream Cipher + AEAD.

Man-in-the-Middle Attack on Diffie–Hellman

Alice and Bob perform Diffie–Hellman over an insecure channel.

Alice and Bob want to establish a shared key.

Man-in-the-Middle Attack on Diffie–Hellman

Alice and Bob perform Diffie–Hellman over an insecure channel.

Alice sends her DH public value $A = g^a$.

Man-in-the-Middle Attack on Diffie–Hellman

Alice and Bob perform Diffie–Hellman over an insecure channel.

Eve intercepts g^a and sends her own value g^e to Bob.

Man-in-the-Middle Attack on Diffie–Hellman

Alice and Bob perform Diffie–Hellman over an insecure channel.

Bob sends his DH public value $B = g^b$.

Man-in-the-Middle Attack on Diffie–Hellman

Alice and Bob perform Diffie–Hellman over an insecure channel.

Eve intercepts g^b and sends g^e to Alice.

MITM
Man-in-the-middle Attack

DH MITM: Two Different Shared Keys

Alice receives g^e instead of Bob's g^b and computes

$$K_{AE} = (g^e)^a = g^{ae}.$$

Bob receives g^e instead of Alice's g^a and computes

$$K_{EB} = (g^e)^b = g^{be}.$$

Eve knows e and can compute both keys:

$$K_{AE} = (g^a)^e, \qquad K_{EB} = (g^b)^e.$$

The Problem

Alice believes she shares a key with Bob, and Bob believes he shares a key with Alice. In reality,

$$\boxed{A \leftrightarrow E \quad \text{and} \quad E \leftrightarrow B}$$

are two separate secure channels.

Authenticating the Diffie–Hellman Exchange

To prevent the MITM attack, Alice authenticates her DH public value using her digital signature.
Alice computes

$$A = g^a$$

and signs it using her secret signing key sk_A:

$$\sigma_A = \mathrm{Sign}_{sk_A}(A \parallel \text{context}).$$

Bob verifies

$$\mathrm{Verify}_{pk_A}(A \parallel \text{context}, \sigma_A) = 1.$$

But Bob must first know that pk_A really belongs to Alice.
Therefore, pk_A must be obtained through a valid certificate:

$$Cert_A = \mathrm{Sign}_{sk_{CA}}(\mathrm{Identity}_A \parallel pk_A \parallel \mathrm{Validity}).$$

Bob verifies $Cert_A$ using the trusted CA public key pk_{CA}.

Complete Trust Chain

$$\boxed{Cert_A \rightarrow \text{Authenticated } pk_A \rightarrow \mathrm{Verify}_{pk_A}(\sigma_A) \rightarrow \text{Authenticated } g^a}$$

Discussion on Important Points

1. Hash Function
2. Symmetric Key encryption
3. Public-key Encryption
4. Digital Signature

Hash Function

Where Do Hash Functions Appear?

We have mainly seen two kinds of hash functions:

Cryptographic Hash Function Universal Hash Function

Hash functions are often not directly visible in the communication flow, but are used internally in several cryptographic mechanisms.

1. **Digital Signatures:** A cryptographic hash compresses a long message before signing.

$$m \longrightarrow H(m) \longrightarrow \mathrm{Sign}_{sk}(H(m)).$$

2. **Key Derivation:** Cryptographic hash functions are used inside KDFs to derive session keys. For example:
   - **HKDF-SHA-256**: uses SHA-256 inside HKDF to derive keys.
   - **TLS 1.3**: uses HKDF with SHA-256/SHA-384 to derive handshake and application traffic keys.
3. **AEAD Authentication:** A keyed universal hash is used inside constructions such as AES-GCM for authentication.

Hash Functions in the Complete Communication Flow

Hash functions appear at different stages of the system.

Phase	Use	Hash Type
Trust / Certificates	Hash data before digital signing	Cryptographic hash
Key Establishment	Hashing may be used in the authenticated handshake	Cryptographic hash
Key Derivation	Derive session keys from shared secret	Cryptographic hash
Secure Communication	Authentication tag in AEAD like AES-GCM	Universal hash (GHASH)

Big Picture

A hash function is a **building block**, not a security service by itself. Its security role depends on the construction in which it is used.

Symmetric Key Encryption

Symmetric-Key Encryption for Secure Communication

After key establishment and key derivation, the resulting session keys are used for efficient communication.

- **Confidentiality:** For long messages, we use either a **stream cipher** or a **block cipher with a suitable mode of operation**.

$$M \xrightarrow{K} C$$

- **Fundamental Primitives:** Block and stream ciphers are fundamental symmetric-key primitives, designed to provide efficient confidentiality using a shared secret key.
- **Authentication and Integrity:** Encryption alone does not detect modification of the ciphertext. Therefore, an authentication mechanism generates a short **authentication tag** that is transmitted with the ciphertext.

$$(C, T)$$

- **Modern Practice:** Instead of using separate encryption and authentication mechanisms, we commonly use **AEAD** constructions such as AES-GCM or ChaCha20-Poly1305.

Key Idea

$$\boxed{\text{Session Key} \rightarrow \text{AEAD} \implies \text{Ciphertext} + \text{Authentication Tag}}$$

Authentication

Authentication: Before and After Key Establishment

Authentication is needed at two different stages.

1. **Before a Shared Secret Exists: Digital Signature**
   A party uses its private key to authenticate a message or handshake transcript:
   $$\sigma = \mathsf{Sign}_{sk}(m).$$
   Anyone with the authenticated public key can verify:
   $$\mathsf{Verify}_{pk}(m, \sigma) = 1.$$

2. **After a Shared Secret Exists: Message Authentication Code**
   Once Alice and Bob share a secret key K, they can authenticate messages using a keyed authentication mechanism:
   $$T = \mathsf{MAC}_K(m).$$
   The receiver recomputes the tag and accepts only if it matches.

Key Difference

Digital signatures use a **private/public key pair**; MACs use a **shared secret key**. A MAC provides authentication and integrity, but not public verifiability.

Keyed Hash and MAC Constructions

A MAC can be constructed using different underlying primitives.

- **HMAC:** Uses a cryptographic hash function.

$$\mathrm{HMAC}_K(m) = H\big((K' \oplus opad) \parallel H((K' \oplus ipad) \parallel m)\big).$$

- **CMAC:** Uses a block cipher, typically AES, to construct a secure MAC.

$$T = \mathrm{CMAC}_K(m).$$

- **Keyed Universal Hash:** Uses a secret hash key and a universal hash construction.

$$T = \mathrm{UH}_K(m).$$

A well-designed construction can provide efficient authentication with a short tag.

Examples

HMAC-SHA-256 | AES-CMAC | GHASH / Poly1305

Modern Practice

In AEAD schemes such as AES-GCM and ChaCha20-Poly1305, authentication is integrated with encryption, so a separate MAC is normally not needed.

Where Are MACs Still Used with AEAD?

AEAD already provides encryption, integrity, and message authentication, so a separate MAC is normally unnecessary for the encrypted data.
However, MAC constructions such as HMAC and CMAC are still useful in other settings.

- **Authentication without Encryption:** When confidentiality is not required, a MAC can authenticate a message efficiently.
$$T = \mathrm{MAC}_K(m).$$
- **Key Derivation:** HMAC is used as a building block in KDFs such as HKDF.
- **Protocol Authentication:** Some protocols use MACs to authenticate control messages, session information, or protocol state.
- **Specialized Constructions:** CMAC and keyed universal hashes can be used where their particular performance or construction properties are desirable.

Key Point

For ordinary encrypted application data, AEAD replaces the need for a separate encryption + MAC construction.

Public Key Encryption

Public-Key Cryptography: Where Is It Used?

Public-key cryptography uses a key pair

$$(pk, sk),$$

where pk can be publicly distributed while sk remains secret.

- **Digital Signatures:** The secret key is used to generate a signature, while the public key is used for verification.

$$\sigma = \mathrm{Sign}_{sk}(m), \qquad \mathrm{Verify}_{pk}(m, \sigma).$$

- **Key Establishment:** Public-key techniques allow parties to establish a shared secret over an insecure channel. Examples include DH/ECDH and public-key KEMs such as ML-KEM.
- **Public-Key Encryption:** Anyone can encrypt using the recipient's public key, while only the recipient can decrypt using the corresponding secret key.

$$c = \mathrm{Enc}_{pk}(m), \qquad m = \mathrm{Dec}_{sk}(c).$$

- **Authentication and Trust:** Public keys are used to verify digital certificates and establish authenticated identities in PKI.

Key Observation

Public-key cryptography is computationally more expensive than symmetric cryptography and is therefore mainly used for **authentication and key establishment**, not bulk data encryption.

Popular Public-Key Encryption Schemes

Public-key encryption schemes have been constructed from different mathematical hardness assumptions.

Scheme	Hardness Assumption	Type
RSA-OAEP	Integer Factorization	Classical
ElGamal	Discrete Logarithm (DLP)	Classical
ECIES	Elliptic Curve DLP	Classical
Paillier	Composite Residuosity	Classical
McEliece	Syndrome Decoding	Post-Quantum
NTRU	Lattice Problems	Post-Quantum
LWE-based PKE	Learning With Errors	Post-Quantum
ML-KEM (Kyber)	Module-LWE	Post-Quantum KEM

Modern Direction

Classical PKE based on RSA and DLP is vulnerable to quantum attacks. Modern key establishment is moving toward post-quantum KEMs such as **ML-KEM (Kyber)**.

We will study some of the above.

Key Agreement

Key Exchange: One-Sided vs. Two-Sided Contribution

Earlier key-establishment methods based on public-key encryption can have a **one-sided contribution**.

PKE-Based Key Transport

Alice chooses the session secret

$$K \xleftarrow{\$} \{0,1\}^{\ell}.$$

She encrypts it using Bob's public key:

$$C = \mathsf{Enc}_{pk_B}(K).$$

Bob decrypts and obtains K.

$$\boxed{K_A = K}$$

Only Alice contributes the secret randomness.

Key Distinction

PKE-based key transport: **one party chooses the secret**. It is no more used.
DHKE: **both parties contribute to the shared secret**.

Why Does DHKE Work?

Correctness follows from

$$(B)^a = (g^b)^a = g^{ab} = (g^a)^b = (A)^b.$$

Security Assumption

Given

$$(g, g^a, g^b),$$

an adversary should not be able to compute

$$g^{ab}.$$

This is known as the **Computational Diffie-Hellman (CDH) Assumption**.

The underlying hardness comes from the Discrete Logarithm Problem (DLP).

Choosing the Group

The security of DHKE depends entirely on the choice of the group.

Group	Assumption
$\mathbb{Z}_p^*$	DLP
Elliptic Curve Groups	ECDLP
Class Groups	Class Group DLP

The group should satisfy

- Efficient group operations.
- Efficient exponentiation.
- Computationally hard Discrete Logarithm Problem.
- No known efficient classical attacks.

Typical choices include

- Multiplicative group $\mathbb{Z}_p^*$,
- Elliptic curve groups.

Observation

The same protocol can be instantiated over different algebraic groups, provided the DLP remains computationally hard.

Examples of Classical Key Agreement Protocols

Most classical key exchange protocols rely on the hardness of the Discrete Logarithm Problem.

Protocol	**Underlying Assumption**
Diffie-Hellman (DH)	DLP / CDH
Elliptic Curve DH (ECDH)	Elliptic Curve DLP
MQV / HMQV	DLP
SIGMA	DLP + Digital Signatures
TLS 1.2 (ECDHE)	Elliptic Curve DLP

Other key establishment mechanisms are based on different assumptions.

$$\boxed{\text{RSA} \rightarrow \text{Integer Factorization}}$$

The Quantum Challenge

Classical key agreement protocols rely on mathematical assumptions such as

- Discrete Logarithm Problem,
- Integer Factorization.

Large-scale quantum computers can solve both problems efficiently using

Shor's Algorithm.

Consequently,

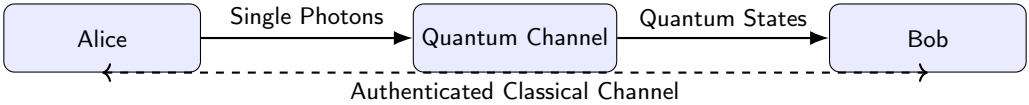
- Diffie-Hellman becomes insecure,
- ECDH becomes insecure,
- RSA key exchange becomes insecure.

Next Step

Can we design key agreement protocols that remain secure even against quantum computers?

Solution 1: Quantum Key Distribution (QKD)

Instead of relying on computational hardness assumptions, Quantum Key Distribution (QKD) derives its security from the laws of quantum mechanics.



Idea

Alice encodes random bits into quantum states. Bob measures them. Any eavesdropping disturbs the quantum states and is therefore detectable.

Limitations of QKD

Although QKD provides information-theoretic security, practical deployment remains challenging.

- Requires dedicated quantum communication channels.
- Requires specialized quantum hardware.
- Limited transmission distance without quantum repeaters.
- Expensive deployment and maintenance.
- Still requires an authenticated classical communication channel.

Observation

QKD is suitable for specialized high-security applications, but is currently impractical for securing the global Internet.

Solution 2: Post-Quantum Key Exchange

Instead of changing the communication medium, we redesign the cryptographic algorithms.

$$\boxed{\text{Classical Network} + \text{New Mathematics}}$$

Post-Quantum Cryptography (PQC)

- Runs on existing computers.
- Uses existing Internet infrastructure.
- Resists attacks from both classical and quantum computers.
- Requires no quantum communication channel.

Modern standards replace Diffie-Hellman by **Post-Quantum Key Encapsulation Mechanisms (KEMs)**.

IISER THIRUVANANTHAPURAM

Hardness Assumptions for Post-Quantum Cryptography

Unlike classical cryptography, PQC relies on mathematical problems for which no efficient classical or quantum algorithms are currently known.

Family	**Representative Assumption**
Lattice-based	LWE, MLWE, SIS
Code-based	Syndrome Decoding
Multivariate	MQ Problem
Hash-based	Hash Function Security
Isogeny-based	Isogeny Problems

Today

The NIST post-quantum standards primarily rely on **lattice-based cryptography**.

What If a Hardness Assumption Is Broken?

Suppose a future breakthrough shows that a widely used hardness assumption (e.g., lattice problems) can be solved efficiently.

Immediate Consequence

All cryptographic primitives whose security relies on that assumption would need to be replaced.

Fortunately, post-quantum cryptography is based on multiple independent families of hard problems.

Family	Example
Code-based	McEliece
Hash-based	XMSS, SPHINCS$^+$
Multivariate	MQ-based Schemes
Isogeny-based	SIKE (Research)
Lattice-based	ML-KEM, ML-DSA

Future Directions

- Develop cryptography from new mathematical assumptions.
- Explore alternative computational paradigms (e.g., DNA computing, molecular cryptography, etc.).
- Continuously analyze, standardize, and diversify cryptographic assumptions.

Next Topic

Next Topic

Public Key Encryption: and its usage in Digital Signatures.

References

Textbook:

1. Douglas R. Stinson and Maura Paterson. *Cryptography: Theory and Practice*. 4th Edition. Chapman & Hall/CRC Press, 2019. ISBN: 9780367148591.

Acknowledgment:

- The figures in this work were generated with the assistance of ChatGPT (OpenAI).

IISER THIRUVANANTHAPURAM

Dr. Laltu Sardar, Assistant Professor, IISER Thiruvananthapuram
laltu.sardar@iisertvm.ac.in, laltu.sardar.crypto@gmail.com
Course webpage: https://laltu-sardar.github.io/courses/26_crypto.html.