***Basic Concept***

* **1: Define the terms plaintext, ciphertext, encryption, and decryption.**

 **Plaintext**  
This is the original, readable message or data that needs to be protected. It is in a form that can be easily understood by humans or machines without any special processing.  
*Example:* "Hello, world!"

 **Ciphertext**  
This is the scrambled or unreadable version of the plaintext after it has been encrypted. Ciphertext is not meant to be understood unless it is decrypted back to plaintext.  
*Example:* "Xx8!@gH$3" (depends on the encryption method)

 **Encryption**  
This is the process of converting plaintext into ciphertext using an algorithm and an encryption key. The goal is to protect the information from unauthorized access.

 **Decryption**  
This is the process of converting ciphertext back into its original plaintext form. It usually requires a decryption key and is the reverse of encryption.

 **2: Explain the difference between symmetric and asymmetric cryptography**

**1. Symmetric Cryptography (Secret-Key Cryptography)**

* **Key Concept**: Uses the **same key** for both encryption and decryption.
* **Speed**: Generally **faster** and more efficient.
* **Key Sharing**: The biggest challenge is **securely sharing the key** between sender and receiver.
* **Use Case Examples**: Encrypting files, securing data at rest, VPNs.
* **Common Algorithms**: AES, DES, RC4.

**Example**:  
Alice and Bob both use the same secret password to encrypt and decrypt messages.

**2. Asymmetric Cryptography (Public-Key Cryptography)**

* **Key Concept**: Uses a **pair of keys** – one **public key** for encryption and one **private key** for decryption.
* **Security**: More secure for communication between parties who’ve never met.
* **Speed**: Typically **slower** than symmetric due to complex math.
* **Use Case Examples**: Secure email, digital signatures, SSL/TLS (used in HTTPS).
* **Common Algorithms**: RSA, ECC, DSA.

**Example**:  
Alice encrypts a message using Bob’s **public key**. Only Bob can decrypt it using his **private key**.

**Quick Comparison Table:**

| **Feature** | **Symmetric** | **Asymmetric** |
| --- | --- | --- |
| Keys Used | One shared key | Public and private key |
| Speed | Fast | Slower |
| Key Distribution | Difficult | Easier (only public key needs sharing) |
| Best For | Encrypting large data | Secure key exchange, digital signatures |
|  |  |  |
|  |  |  |

* **3: List three common goals of cryptography.**

Here are **three common goals of cryptography**, often referred to as the **core principles** of information security:

**1. Confidentiality**

* **Goal**: Ensure that information is only accessible to those who are authorized to see it.
* **How it's achieved**: Encryption.
* **Example**: Encrypting a message so only the intended recipient can read it.

**2. Integrity**

* **Goal**: Ensure that the data has not been altered or tampered with.
* **How it's achieved**: Hashing, checksums, digital signatures.
* **Example**: Verifying a file’s hash to make sure it hasn’t been modified.

**3. Authentication**

* **Goal**: Confirm the identity of the parties involved in communication.
* **How it's achieved**: Digital certificates, passwords, biometrics.
* **Example**: Logging into a secure website using a username and password or two-factor authentication.

Other related goals sometimes included are **non-repudiation** (ensuring someone can't deny their actions) and **availability**, but the three listed above are the most central to cryptography.

***Symmetric Key Cryptography***

* **1- Encrypt the plaintext "HELLO" using a Caesar cipher with a shift of 3.**

To encrypt **"HELLO"** using a **Caesar cipher** with a **shift of 3**, we shift each letter **3 positions forward** in the alphabet.

Here's the step-by-step:

| **Letter** | **Shifted +3** |
| --- | --- |
| H | K |
| E | H |
| L | O |
| L | O |
| O | R |

**✅ Encrypted Ciphertext: KHOOR**

* **2- Decrypt the ciphertext "VWDQGD" using a Caesar cipher with a shift of 3.**

To **decrypt** the ciphertext **"VWDQGD"** using a **Caesar cipher** with a **shift of 3**, we shift each letter **3 positions backward** in the alphabet.

Here’s the breakdown:

| **Letter** | **Shifted -3** |
| --- | --- |
| V | S |
| W | T |
| D | A |
| Q | N |
| G | D |
| D | A |

**✅ Decrypted Plaintext: STANDA**

* **3- Encrypt the plaintext "OPENAI" using a Vigenère cipher with the keyword "CRYPTO".**

**Step 1: Align the keyword with the plaintext**

Repeat the keyword to match the length of the plaintext:

mathematica

Plaintext: O P E N A I

Keyword: C R Y P T O

**Step 2: Convert letters to numbers (A = 0, B = 1, ..., Z = 25)**

css

Plaintext: O(14) P(15) E(4) N(13) A(0) I(8)

Keyword: C(2) R(17) Y(24) P(15) T(19) O(14)

### Step 3: Add corresponding values and take mod 26

matlab

(14 + 2) % 26 = 16 → Q

(15 + 17) % 26 = 6 → G

(4 + 24) % 26 = 2 → C

(13 + 15) % 26 = 2 → C

(0 + 19) % 26 = 19 → T

(8 + 14) % 26 = 22 → W

✅ Encrypted Ciphertext: **QGCCTW**

***Asymmetric Key Cryptography***

**1- Generate a key pair using RSA encryption with a prime modulus of 17 and a public exponent of 5**

Let's generate an **RSA key pair** using:

* Prime modulus **p = 17**
* Public exponent **e = 5**

We’ll go through the steps to find the **public key (e, n)** and **private key (d, n)**.

### ✅ Step 1: Choose two prime numbers

Since only one is given (**p = 17**), let's also choose another small prime for this example. We'll use:

* **q = 11** (another small prime)

### ✅ Step 2: Compute ****n = p × q****

n=17×11=187

### ✅ Step 3: Compute Euler’s totient function:

ϕ(n)=(p−1)×(q−1)=(17−1)(11−1)=16×10=160

### ✅ Step 4: Choose the public exponent ****e = 5****

We already have this. Make sure it's coprime with 160 (✔️ Yes, 5 and 160 are coprime).

### ✅ Step 5: Find the ****private key d****, the modular inverse of ****e mod φ(n)****

We need to find **d** such that:

d×e≡1 mod160

d×5≡1 mod160

Using modular inverse:

d=129 (since 5×129=645≡1 mod  160)

### 🔑 Final RSA Key Pair:

* **Public Key**: (e = 5, n = 187)
* **Private Key**: (d = 129, n = 187)

**2- Encrypt the plaintext "SECRET" using the recipient's public key: (e=5, n=221)**

Let's encrypt the plaintext **"SECRET"** using the recipient’s **RSA public key**:

* **e = 5**
* **n = 221**

**🔣 Step 1: Convert letters to numbers**

We use the **A = 0 to Z = 25** system:

ini

S = 18

E = 4

C = 2

R = 17

E = 4

T = 19

**🔐 Step 2: Encrypt each number using the RSA formula:**

cipher=()mod  n

Using **e = 5** and **n = 221**:

| **Letter** | **Plain #** | **Encrypted: (x⁵ mod 221)** |
| --- | --- | --- |
| S | 18 | 18^5 mod  221=174 |
| E | 4 | 4^5 mod  221=102 |
| C | 2 | 2^5 mod  221=32 |
| R | 17 | 17^5 mod  221=47 |
| E | 4 | 4^5 mod  221=102 |
| T | 19 | 19^5 mod  221=143 |

**✅ Encrypted Ciphertext (as numbers):**

**[174, 102, 32, 47, 102, 143]**

**3- Decrypt the ciphertext "196" using the recipient's private key: (d=53, n=221)**

To decrypt the ciphertext **"196"** using the RSA **private key**:

* **d = 53**
* **n = 221**

### 🔓 Step 1: Use the RSA decryption formula:

plaintext=()mod  n

plaintext=()mod  221

That’s a large exponent, so we’ll use **modular exponentiation** to simplify:

)mod  221 =18

### 🔡 Step 2: Convert the number back to a letter

Using **A = 0, B = 1, ..., Z = 25**:

18→S

### ✅ Decrypted Plaintext: ****S****

**4- Cryptographic Algorithms**

* 1. **Research and compare the strengths and weaknesses of DES and AES encryption algorithms**

Here’s a comparison of **DES (Data Encryption Standard)** and **AES (Advanced Encryption Standard)**, focusing on their **strengths and weaknesses**:

**🔐 1. DES (Data Encryption Standard)**

**🔷 Strengths:**

* **Simple and fast** in hardware implementations.
* Was a **widely adopted** and standardized algorithm in the past.
* Good for **learning purposes** and understanding block ciphers.

**🔻 Weaknesses:**

* **Short key length**: 56-bit key is too small by modern standards and can be **brute-forced**.
* **Vulnerable to modern attacks** like differential and linear cryptanalysis.
* Obsolete: **No longer considered secure** for protecting sensitive data.
* Block size is **only 64 bits**, making it more vulnerable to certain attacks (e.g., birthday attack).

**🔐 2. AES (Advanced Encryption Standard)**

**🔷 Strengths:**

* **Strong security**: Uses key lengths of **128, 192, or 256 bits**, making it highly resistant to brute-force attacks.
* **Efficient**: Fast in both hardware and software, especially on modern processors.
* **Larger block size**: 128-bit block size improves security against certain attacks.
* **Widely trusted**: Used by governments, banks, and across industries worldwide.
* **Supports multiple rounds** (10, 12, or 14 depending on key size), increasing complexity.

**🔻 Weaknesses:**

* **Complexity**: More complex to implement securely than DES.
* **Side-channel attacks**: Like many ciphers, AES can be vulnerable if not implemented properly (e.g., timing or power analysis attacks).
* **Performance in constrained devices**: Slightly more resource-intensive than DES on very limited hardware, but still manageable.

**🔍 Comparison Summary Table**

| **Feature** | **DES** | **AES** |
| --- | --- | --- |
| Key Size | 56 bits | 128, 192, or 256 bits |
| Block Size | 64 bits | 128 bits |
| Security | Weak (obsolete) | Strong (widely used) |
| Speed | Fast in hardware | Fast in both hardware and software |
| Resistance to Attacks | Vulnerable to brute-force | Highly resistant |
| Current Use | Deprecated | Industry standard |

**🏁 Conclusion:**

* **DES** is outdated and **not secure** for modern applications.
* **AES** is the **current standard** for secure encryption and is suitable for almost all use cases today.
  1. **Explain the concept of a hash function and provide an example of a commonly used hash function**

**What is a Hash Function?**

A **hash function** is a mathematical function that takes an input (or **message**) and returns a fixed-size string of characters, which is typically a **digest**. The output is often represented as a **hash value** (or hash code). The main purpose of a hash function is to produce a unique identifier for data (though in theory, two different inputs could produce the same hash, this is rare).

**Key Properties of Hash Functions:**

1. **Deterministic**: The same input always produces the same output.
2. **Fixed Output Length**: Regardless of the input size, the hash value has a fixed length (e.g., 256 bits, 512 bits).
3. **Efficient**: The function should quickly compute the hash for any input.
4. **Pre-image Resistance**: Given a hash value, it should be computationally infeasible to determine the original input.
5. **Collision Resistance**: It should be very difficult to find two different inputs that produce the same hash value.
6. **Avalanche Effect**: A small change in the input should produce a significantly different hash.

**Example of a Commonly Used Hash Function: SHA-256**

**SHA-256** (Secure Hash Algorithm 256-bit) is one of the most popular cryptographic hash functions. It produces a **256-bit hash value** and is part of the **SHA-2 family** of hash functions.

For example:

* **Input**: "Hello, world!"
* **Output (SHA-256 Hash)**:  
  a591a6d40bf420404a011733cfb7b190d62c65bf0bcdaee832f348e7e6d5e6c4

Even if you change a single character in the input, the hash value changes completely.

**Example Use Cases of Hash Functions:**

1. **Data Integrity**: Hash functions are commonly used to verify the integrity of data, e.g., by comparing hashes before and after transmission.
2. **Password Storage**: Instead of storing passwords directly, systems store the hash of the password. When users log in, the system hashes the entered password and compares it to the stored hash.
3. **Digital Signatures**: Hash functions are used to create digital signatures to authenticate documents or transactions.

**Summary of SHA-256 Example:**

* **Input**: "Hello, world!"
* **Hash (SHA-256)**: a591a6d40bf420404a011733cfb7b190d62c65bf0bcdaee832f348e7e6d5e6c4
  1. **Discuss the advantages of using a hybrid cryptosystem that combines symmetric and asymmetric encryption**

A **hybrid cryptosystem** combines the strengths of both **symmetric** and **asymmetric encryption** to offer a more efficient and secure method of protecting data. The hybrid approach is commonly used in modern cryptographic protocols, such as **SSL/TLS** for secure web communication.

**Advantages of Using a Hybrid Cryptosystem:**

**1. Efficiency of Symmetric Encryption**

* **Speed**: Symmetric encryption algorithms (e.g., AES) are **faster** than asymmetric encryption because they use the same key for both encryption and decryption, and the algorithms are simpler.
* **Large Data Handling**: Since symmetric encryption is much faster, it's ideal for encrypting large amounts of data. Encrypting a bulk of data directly with symmetric encryption ensures quick transmission without performance degradation.

**Example**: If you're transmitting a large file, using a symmetric key (such as AES) to encrypt the file ensures minimal latency.

**2. Security of Asymmetric Encryption**

* **Key Distribution**: Asymmetric encryption (e.g., RSA, ECC) addresses the **key exchange problem**. The public key can be freely distributed, and only the private key can decrypt the message. This allows secure key exchange over untrusted networks.
* **Authentication and Non-Repudiation**: Asymmetric encryption can ensure the identity of the sender (via digital signatures), which helps with **authentication** and provides **non-repudiation** (the sender can't deny sending the message).

**Example**: In a hybrid system, the symmetric key is exchanged securely using asymmetric encryption, ensuring both privacy and authenticity.

**3. Scalability**

* **Public Key Infrastructure (PKI)**: The use of asymmetric encryption means that each participant only needs a **pair of public/private keys**, not a separate shared secret with every other participant. This makes it scalable for systems with many users.
* **Reduced Key Management Complexity**: With asymmetric encryption, you don't have to distribute and manage numerous symmetric keys for each pair of users. Instead, you can distribute the public key widely and keep the private key secret.

**4. Practical Use Cases**

* **SSL/TLS**: A classic example is **SSL/TLS** (used in HTTPS). Here's how the hybrid system works:
  1. **Key Exchange**: The server sends its public key to the client.
  2. **Symmetric Key Creation**: The client generates a random symmetric key, encrypts it using the server's public key, and sends it to the server.
  3. **Data Encryption**: Both parties now use the symmetric key to encrypt and decrypt data efficiently.

This approach provides the **security** of asymmetric encryption for key exchange and the **speed** of symmetric encryption for the data transfer.

**5. Reduced Computational Overhead**

* Asymmetric encryption, while secure, is computationally expensive due to its complexity. Using it solely for encrypting large volumes of data would be inefficient.
* A hybrid system allows **asymmetric encryption to only encrypt the small symmetric key**, while the **bulk of the data** is encrypted using a fast symmetric cipher.

**6. Enhanced Security Features**

* **Forward Secrecy**: Some hybrid systems (e.g., Diffie-Hellman key exchange) ensure that even if long-term private keys are compromised, past communications cannot be decrypted because the symmetric key changes for each session.
* **Perfect Forward Secrecy (PFS)**: This prevents an attacker from decrypting previous communications even if the private key of the server is exposed in the future.

**Summary of Hybrid Cryptosystem Advantages:**

| **Feature** | **Symmetric Encryption** | **Asymmetric Encryption** | **Hybrid Cryptosystem** |
| --- | --- | --- | --- |
| **Speed** | Fast | Slow | Fast for data, secure for key exchange |
| **Key Distribution** | Requires secure sharing | Public key can be shared | Asymmetric for key exchange, symmetric for data |
| **Scalability** | Requires separate keys for each pair | Efficient with many users | Efficient for large systems |
| **Security** | Vulnerable to key exposure | Secure key exchange | Combines security and speed |
| **Use Case** | File encryption, data storage | Key exchange, digital signatures | SSL/TLS, email encryption |

**Conclusion:**

A hybrid cryptosystem balances the **speed** of symmetric encryption with the **security** and **scalability** of asymmetric encryption. By using asymmetric encryption to securely exchange a symmetric key, it minimizes the computational overhead while maintaining strong security for communication.

***Practical Application***

**1-Design a simple encryption program in your preferred programming language that encrypts a user's input using a symmetric key**

Here's a simple encryption program in Python that uses the **symmetric encryption** technique (specifically **AES** from the pycryptodome library) to encrypt the user's input. It requires a **key** and **initialization vector (IV)** for encryption.

1. **Python code for encryption**:

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get\_random\_bytes

import base64

# Function to encrypt the input

def encrypt\_input(plaintext, key):

# Generate a random initialization vector (IV)

iv = get\_random\_bytes(AES.block\_size)

# Create AES cipher object with key and IV

cipher = AES.new(key, AES.MODE\_CBC, iv)

# Pad the plaintext to make it a multiple of block size (16 bytes)

padded\_plaintext = pad(plaintext.encode(), AES.block\_size)

# Encrypt the plaintext

ciphertext = cipher.encrypt(padded\_plaintext)

# Return the ciphertext and IV as a base64-encoded string for easy storage/transfer

return base64.b64encode(iv + ciphertext).decode()

# Main program

def main():

# Ask user for input

user\_input = input("Enter the text to encrypt: ")

# Symmetric encryption key (must be 16, 24, or 32 bytes long)

key = b'Sixteen byte key' # Example: 16-byte key

# Encrypt the user's input

encrypted\_text = encrypt\_input(user\_input, key)

# Output the encrypted text

print(f"Encrypted text: {encrypted\_text}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Explanation of Code:**

1. **AES.new(key, AES.MODE\_CBC, iv)**:
   * This creates a new AES cipher in **CBC mode** (Cipher Block Chaining).
   * **Key**: This is the symmetric key used for encryption (in this example, it's a 16-byte key).
   * **IV**: An **initialization vector (IV)** is generated randomly to ensure that encrypting the same plaintext results in different ciphertexts each time.
2. **pad()**:
   * The plaintext must be padded to a multiple of the AES block size (16 bytes). The pad function ensures this padding.
3. **base64.b64encode()**:
   * Since the ciphertext can contain non-printable characters, we encode it in **base64** to make it readable and easily transferable.
4. **get\_random\_bytes(AES.block\_size)**:
   * This generates a random IV for each encryption to prevent patterns in the ciphertext.

### ****Running the Program****:

1. When you run the program, it will ask you for input.
2. It will encrypt the input text using the AES symmetric key and output the encrypted text in **base64** format.

### Example:

pgsql

Enter the text to encrypt: Hello, World!

Encrypted text: PkBBFYFZxGd5qflO5Gp/HT02gAxtxFxlzVbWB3o9vJI=

**2- Implement a function that calculates the MD5 hash of a given input string.**

To calculate the **MD5 hash** of a given input string in Python, i use the built-in hashlib library. The MD5 algorithm produces a **128-bit hash value** that is often represented as a 32-character hexadecimal number.

**Python Code to Calculate MD5 Hash:**

import hashlib

# Function to calculate MD5 hash

def calculate\_md5(input\_string):

# Create an MD5 hash object

md5\_hash = hashlib.md5()

# Update the hash object with the bytes of the input string

md5\_hash.update(input\_string.encode())

# Get the hexadecimal representation of the hash

return md5\_hash.hexdigest()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

input\_string = input("Enter the string to hash: ")

md5\_hash\_value = calculate\_md5(input\_string)

print(f"MD5 hash: {md5\_hash\_value}")

### ****Explanation:****

1. **hashlib.md5()**: This creates a new MD5 hash object.
2. **update(input\_string.encode())**: The update() method feeds the input string (converted to bytes using encode()) into the hash object.
3. **hexdigest()**: This returns the resulting MD5 hash in hexadecimal format.

**Example Output:**

bash

Enter the string to hash: Hello, World!

MD5 hash: fc3ff98e8c6a0d3087d515c0473f8677

The MD5 hash of "Hello, World!" is fc3ff98e8c6a0d3087d515c0473f8677.

**Important Note:**

MD5 is **not considered secure** for cryptographic purposes due to vulnerabilities like **collision attacks**. It should not be used for security-critical applications. For secure hashing, consider using **SHA-256** or another member of the SHA-2 family.

**3- Research and explain the concept of a digital signature and its role in verifying message authenticity**

**What is a Digital Signature?**

A **digital signature** is a cryptographic technique used to validate the authenticity and integrity of a message, software, or digital document. It serves as a **virtual equivalent** of a handwritten signature or a stamped seal, but it offers much more security because it provides proof of both the **origin** of the message (authenticity) and its **integrity** (non-tampering).

Digital signatures use **asymmetric cryptography**, which involves a **pair of keys**: a **private key** for signing the message and a **public key** for verification.

**How Does a Digital Signature Work?**

The process of creating and verifying a digital signature involves the following steps:

**1. Signing the Message (Creating the Digital Signature)**

* **Step 1**: **Hash the Message**: The original message (or document) is first passed through a **hash function** (e.g., SHA-256) to produce a fixed-size hash (digest). This is done because the hash provides a shorter and unique representation of the message.
* **Step 2**: **Encrypt the Hash with the Private Key**: The hash value is then encrypted with the **signer's private key**. This encrypted hash is the **digital signature**.
* **Step 3**: **Send the Message and the Signature**: The message is sent along with the digital signature. Only the signer’s private key could have created that signature.

**2. Verifying the Message (Checking the Digital Signature)**

* **Step 1**: **Decrypt the Digital Signature with the Public Key**: The recipient uses the **signer's public key** to decrypt the digital signature, revealing the hash value that was originally calculated when the message was signed.
* **Step 2**: **Hash the Received Message**: The recipient hashes the received message using the same hash function that was used by the sender.
* **Step 3**: **Compare Hashes**: The recipient compares the decrypted hash with the hash they just calculated:
  + If the two hashes match, the signature is valid, meaning the message has **not been altered** and is indeed from the claimed sender.
  + If the hashes do not match, the message has been **tampered with** and the signature is invalid.

**Key Features of a Digital Signature**

1. **Authentication**:
   * The digital signature ensures that the message was sent by the person who claims to have sent it. It provides proof of the **identity** of the sender.
   * This is accomplished because only the sender's private key could have created the signature, and only the sender knows that private key.
2. **Integrity**:
   * If the content of the message changes in transit, the hash value will not match the decrypted hash during the verification process. This ensures the message has **not been tampered with**.
3. **Non-repudiation**:
   * The sender **cannot deny** having sent the message once it is signed, because only they hold the private key that created the digital signature. This property is called **non-repudiation**, and it is a critical part of ensuring accountability in digital communication.

**Real-World Use Cases of Digital Signatures**

1. **Email Security**:
   * Digital signatures are often used in **secure email protocols** like **S/MIME** or **PGP** (Pretty Good Privacy) to verify the identity of the sender and ensure the email has not been altered.
2. **Software Distribution**:
   * Software companies sign their software and updates with digital signatures to ensure that the software hasn't been tampered with during download or installation. This guarantees that the software comes from the legitimate source and is safe to use.
3. **Legal and Financial Documents**:
   * Digital signatures are used to sign **contracts**, **agreements**, or **forms** online. These signatures are legally binding in many countries and offer a secure alternative to paper-based signatures.
4. **Blockchain**:
   * Digital signatures play a crucial role in blockchain technologies, where transactions are signed to ensure that they are legitimate and from the rightful user, preventing unauthorized transactions.

**Example of How Digital Signatures Work in Practice:**

Let's assume **Alice** wants to send a signed message to **Bob**.

1. Alice creates the message: "I, Alice, agree to the terms."
2. Alice generates a hash of the message, say **abc123hash**.
3. Alice encrypts this hash with her **private key** to create the digital signature: **signature**.
4. Alice sends both the message and the digital signature to Bob.

When Bob receives the message, he:

1. Generates a hash of the message: **abc123hash**.
2. Uses Alice’s **public key** to decrypt the digital signature and obtain Alice's original hash.
3. Compares the two hashes:
   * If they match, Bob knows that the message is authentic and hasn't been altered.
   * If they don't match, Bob knows the message has been tampered with.

**Advantages of Digital Signatures:**

1. **High Security**: Since the private key is kept secret, and only the public key is used for verification, digital signatures are highly secure.
2. **Efficiency**: Digital signatures allow for the secure exchange of documents and messages without the need for physical signatures or third-party intermediaries.
3. **Legality**: In many jurisdictions, digital signatures are **legally binding**, equivalent to handwritten signatures.

**Conclusion**

A **digital signature** is a powerful cryptographic tool that ensures **authenticity**, **integrity**, and **non-repudiation** of messages or documents. It uses **asymmetric encryption** to secure communication and ensure that data has not been tampered with, providing strong assurances for both the sender and recipient.