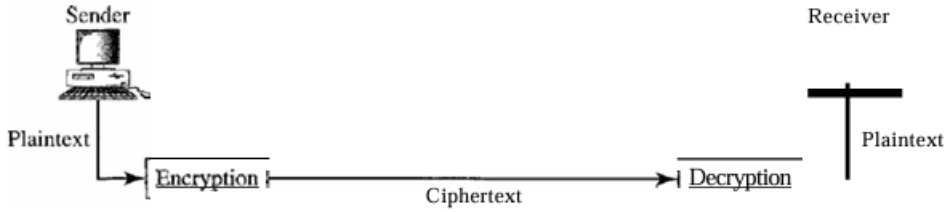
UNIT – 5

Cryptography Summary

Cryptography is the practice of converting plain text into a coded format to keep messages secure.

- **Plaintext**
  The original message before it is changed into a secret format.
- **Ciphertext**
  The message after it has been transformed into a secret format using an encryption process.
- **Encryption**
  The process of converting plaintext into ciphertext to make it secure.
- **Decryption**
  The process of converting ciphertext back into plaintext to read the original message.
- **Cipher**
  A method or algorithm used for encryption and decryption.
- **Key**
  A number or set of numbers used by the cipher to encrypt and decrypt messages. It ensures the message is transformed and restored correctly.

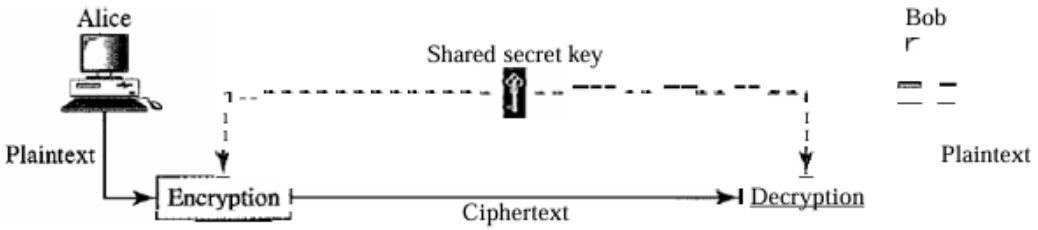
Figure 30.1 *Cryptography components*



Types of Cryptography:

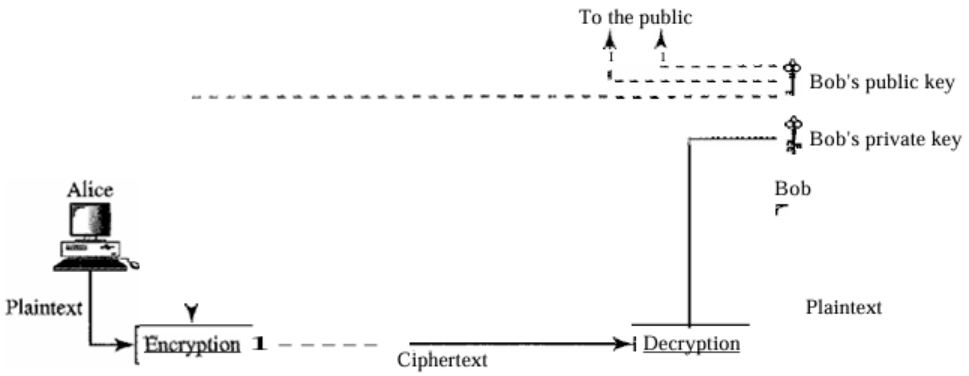
1. **Symmetric-Key Cryptography**:
   - Uses **one shared key** for both encryption and decryption.
   - Example: If Alice sends a message to Bob, both use the same key to encrypt and decrypt.
   - **Advantage**: Faster encryption.
   - **Disadvantage**: Key must be shared securely between parties.

Figure 30.3 *Symmetric-key cryptography*



2. **Asymmetric-Key Cryptography** (Public-Key Cryptography):
    - Uses **two keys**: a **public key** (available to everyone) and a **private key** (kept secret by the receiver).
    - Example: Alice uses Bob's public key to encrypt a message. Only Bob can decrypt it using his private key.
    - **Advantage**: More secure since keys are different.
    - **Disadvantage**: Slower than symmetric-key encryption.

Figure 30.4 *Asymmetric-key cryptography*



Comparison:

- **Symmetric-Key**: One shared key for both encryption and decryption.
- **Asymmetric-Key**: Public key for encryption and private key for decryption.

In essence, symmetric-key cryptography is like having one key that both sender and receiver use, while asymmetric-key cryptography uses a pair of keys — one for locking (encrypting) and another for unlocking (decrypting).

Figure 30.5 *Keys used in cryptography*

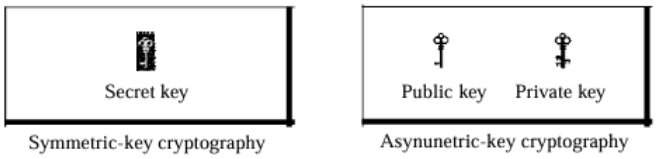
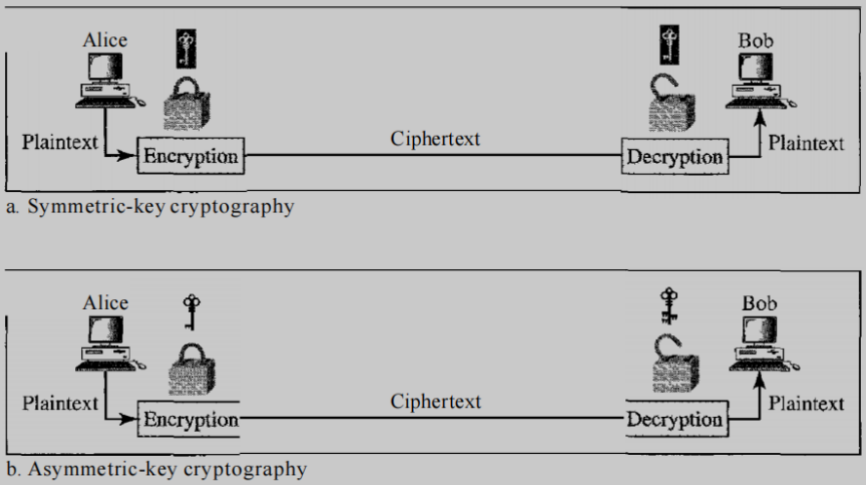


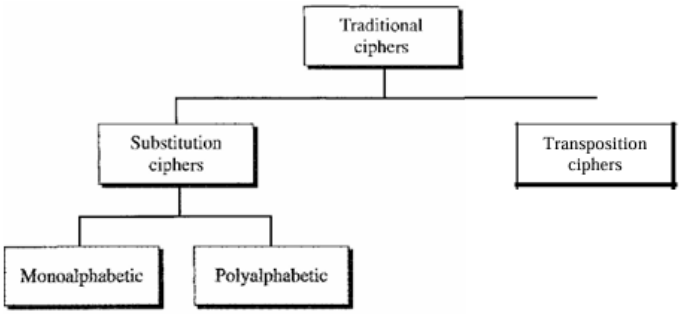
Figure 30.6 *Comparison between two categories ofcryptography*



Symmetric-Key Cryptography

Symmetric-Key Cryptography uses the same key for both encryption (locking the message) and decryption (unlocking the message). It has been used for thousands of years, and while modern methods are more complex, the basic idea is the same.

Traditional Ciphers (Old Methods)



1. **Substitution Ciphers**:
   - **Idea**: Replace each letter or symbol in the plaintext with another letter/symbol.
   - **Types**:
     - **Monoalphabetic**: One-to-one replacement. For example, every "A" becomes "D".
     - **Polyalphabetic**: One-to-many replacement. For example, "A" might become "D" once, and "N" another time.
   - **Example**:
     - Plaintext: "HELLO"
     - Monoalphabetic Ciphertext: "KHOOR" (same "L" is replaced by "O").
     - Polyalphabetic Ciphertext: "ABNZF" (different replacements for each "L").
2. **Shift Cipher (Caesar Cipher)**:
   - **Idea**: Shift each letter by a fixed number of places in the alphabet.
   - **Example**: With a shift key of 3:
     - Plaintext: "HELLO"
     - Encrypted Text: "KHOOR"
     - Shift key of 15:
     - "HELLO" becomes "WTAAD".
3. **Transposition Ciphers**:
   - **Idea**: Rearrange (shuffle) the positions of the characters instead of replacing them.
   - **Example**:
     - Key Order: 2-4-1-3
     - Plaintext: "HELL" → Rearranged: "ELHL".
   - Encrypting "HELLO MY DEAR":
     - Rearrange in blocks: "HELL OMYD EARZ" → Ciphertext: "ELHLMDOYAZER".

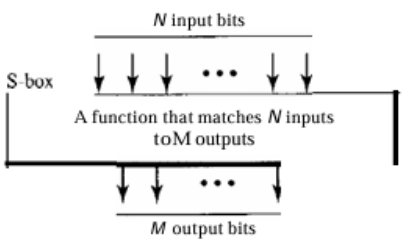
Modern Ciphers Overview

Modern ciphers work at the **bit level**, allowing them to encrypt various data types like text, images, and audio. They integrate multiple simple ciphers to enhance overall security. Key components include XOR, rotation, substitution (S-box), and transposition (P-box) ciphers.

Core Components and Examples

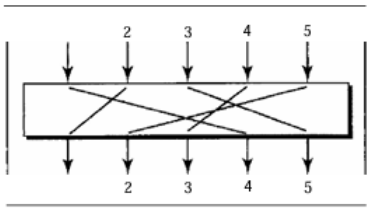
1. **XOR Cipher**:
    - **Definition**: Uses the XOR operation between plaintext and a key for encryption and decryption.
    - **Example**:
        - Plaintext: 1010
        - Key: 1100
        - Ciphertext: 1010 XOR 1100 = 0110
    - **Use**: Basic, fast encryption, but vulnerable if the key is reused.
2. **Rotation Cipher**:
    - **Definition**: Rotates bits of plaintext to the left or right. Can be keyed (variable rotation based on the key) or keyless (fixed rotation).
    - The rotation cipher can be keyed or keyless.
    - In keyed rotation, the value of the key defines the number of rotations; in keyless rotation the number of rotations is fixed.
    - **Example**:
        - Left rotate 1011 by 1 bit results in 0111.
    - **Use**: Obscures data by shifting bits, often used in data scrambling.
3. **Substitution Cipher (S-box)**:
    - **Definition**: Transforms input bits into output bits based on a predefined substitution table.
    - An S-box (substitution box) parallels the traditional substitution cipher for characters.
    - The input to an S-box is a stream of bits with length N; the result is another stream of bits with length M.
    - And Nand M are not necessarily the same.
    - **Example**:
        - Input 1010 -> Output 1101 using an S-box table.
    - **Use**: Adds non-linear complexity to encryption algorithms like AES and DES.

S-box

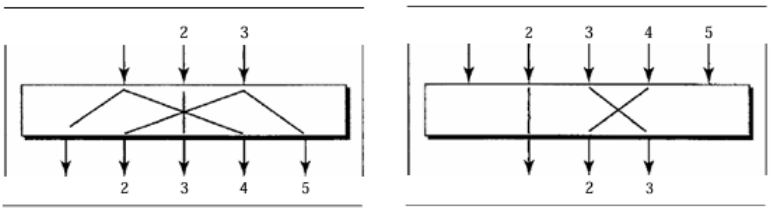


4. **Transposition Cipher (P-box)**:
    - **Definition**: Permutes (rearranges) bits to increase diffusion.
    - **Example**:
        - Input 1010 rearranged to 0101.
        - A straight permutation cipher or a straight P-box has the same number of inputs as outputs. In other words, if the number of inputs is *N,* the number of outputs is also *N.*
        - In an expansion permutation cipher, the number of output ports is greater than the number of input ports.
        - In a compression permutation cipher, the number of output ports is less than the number of input ports.
    - **Use**: Enhances complexity without altering the actual bits, used in various block ciphers.

Figure 30.12 *P-boxes: straight, expansion, and compression*



a. Straight



b. Expansion c. Compression

Modern Round Ciphers

Modern ciphers, called **round ciphers**, encrypt data through multiple steps called **rounds**. Each round applies a mix of simpler encryption techniques.

- A **round key**, a variation of the main key, is used for each round. For example, if there are 10 rounds, the system generates 10 round keys (Key 1, Key 2, ..., Key 10), each used in its respective round.
- These ciphers are also known as **block ciphers** because they break the message (plaintext) into fixed-sized blocks and use the same key to encrypt and decrypt these blocks.
- Two important symmetric-key ciphers are:
    1. **DES (Data Encryption Standard):** Used widely in the past as a standard for encryption.
    2. **AES (Advanced Encryption Standard):** The current standard, offering better security.

Data Encryption Standard (DES)

DES is a block cipher that encrypts 64 bits of plaintext using a 64-bit key. It was developed by IBM and became a widely-used encryption standard.

How DES Works

1. **Input**: The plaintext (64 bits) is processed using a key (64 bits, though only 56 bits are actually used; 8 bits are for error checking).
2. **Initial Permutation**: The plaintext bits are shuffled in a predefined way (keyless).
3. **16 Rounds of Processing**: DES applies the following operations repeatedly 16 times:
    - **Split into two halves**: The data is split into a left half (32 bits) and a right half (32 bits).
    - **Key Mixing**: A subkey (48 bits) is derived from the main key and combined with the right half using the DES function.
    - **DES Function**: The right half undergoes expansion, XOR with the subkey, substitution (using S-boxes), and permutation.
    - **Swap and Combine**: After processing, the two halves are swapped, then combined for the next round.
4. **Final Permutation**: The bits are shuffled again (inverse of the initial permutation).
5. **Output**: The result is a 64-bit ciphertext.

Figure 30.13 *DES*

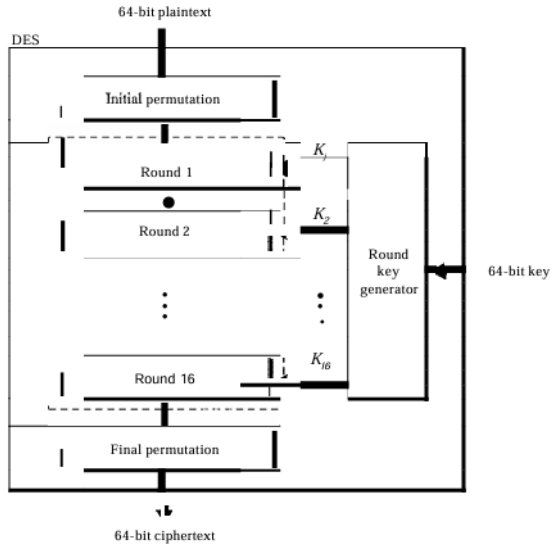
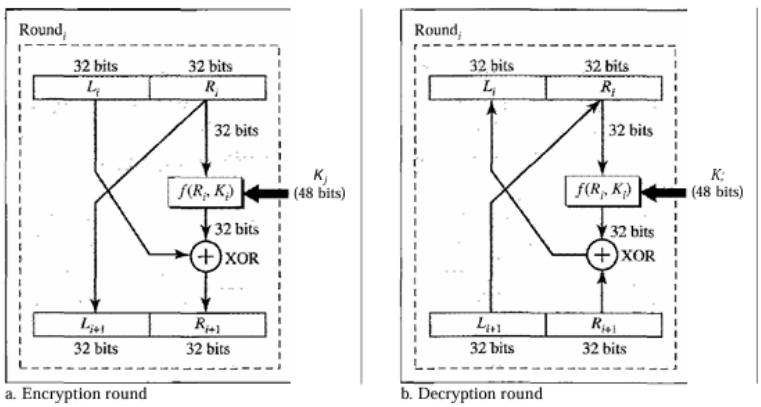
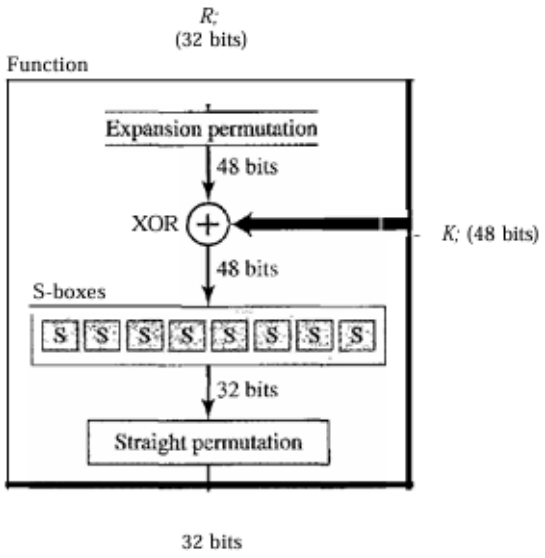


Figure 30.14 *One round in DES ciphers*



DES function



DES Function

The DES function is the "heart" of the encryption:

1. The right half (32 bits) is expanded to 48 bits.
2. XOR with a 48-bit subkey.
3. Pass through 8 S-boxes, reducing it back to 32 bits.
4. Perform a straight permutation on the 32 bits.

Decryption

Decryption works in the reverse order, but the subkeys are applied in reverse.

Triple DES (3DES):

- **Definition**: Applies the DES algorithm three times with different keys for enhanced security.
- The original DES (Data Encryption Standard) was considered insecure because its key (56 bits) was too short, making it vulnerable to attacks. To strengthen it, Triple DES was introduced.
- Critics found DES's 64-bit key too short, making it vulnerable to attacks.

How does it work?

- 3DES applies DES three times in either an **encrypt-decrypt-encrypt (EDE)** or **decrypt-encrypt-decrypt (DED)** sequence.
- Triple DES encrypts data by applying the DES algorithm three times in a specific sequence. There are two main versions of this process:

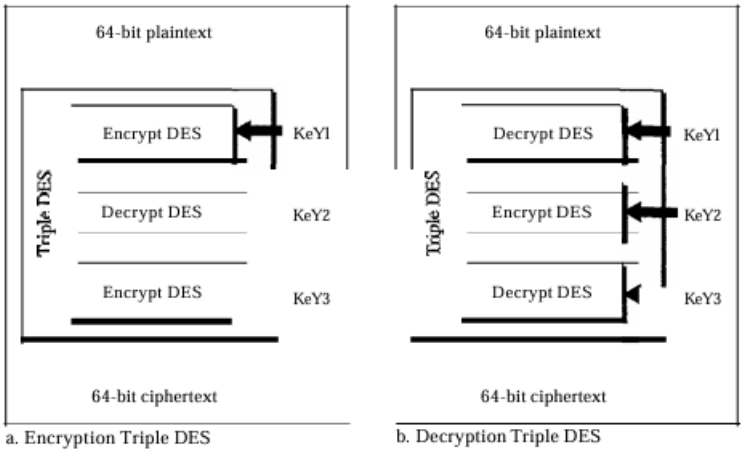
3DES with Two Keys:

- First encryption with Key 1, Decryption with Key 2, Encryption again with Key 1.
  This setup gives a total key length of 112 bits (2 keys × 56 bits).

3DES with Three Keys:

- Encryption with Key 1, Decryption with Key 2, Encryption with Key 3.
  This increases the total key length to 168 bits (3 keys × 56 bits).

Figure 30.16 *Triple DES*



a. Encryption Triple DES

b. Decryption Triple DES

Benefits of 3DES:

- If a single DES block is used, it can still work with 3DES if all three keys are set the same (Key 1 = Key 2 = Key 3). This makes it compatible with older systems.
- The three-step encryption process makes it much harder for attackers to break compared to plain DES.

Man-in-the-Middle Attack Defense:
Using two or three different keys protects against specific types of attacks, like the "man-in-the-middle" attack.

Advanced Encryption Standard (AES):

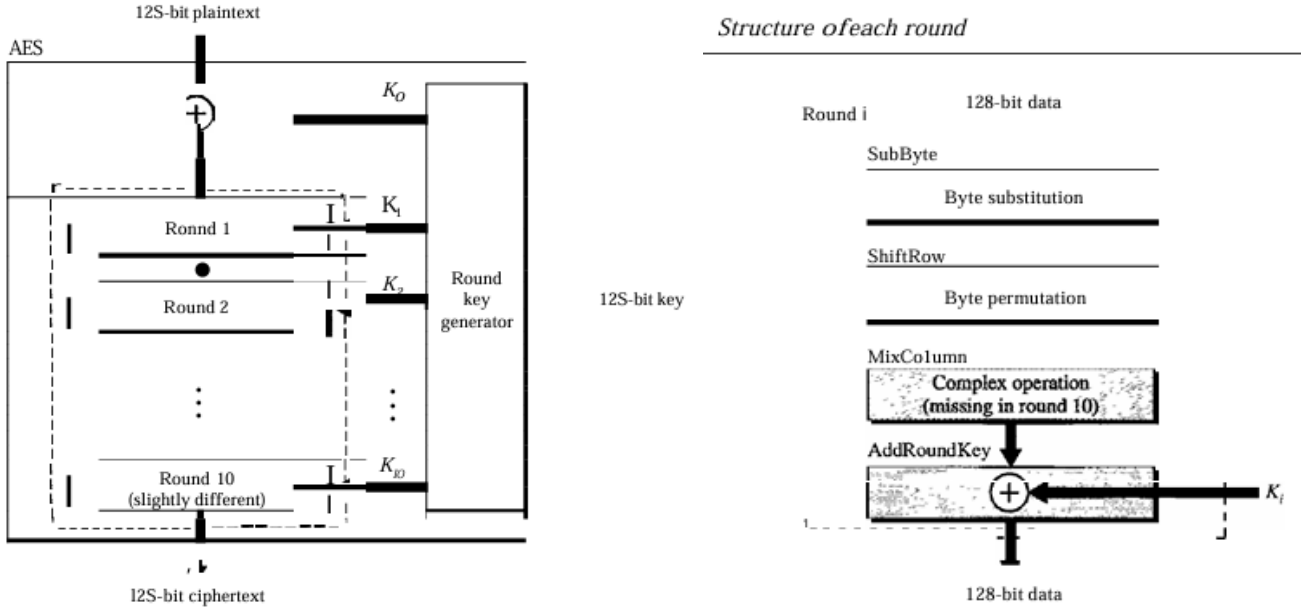
- **Problem with DES**: Its key size was too small, making it less secure. Triple DES (3DES) made it more secure but was too slow.
- **Key Features of AES**
  - **Key Sizes**: AES supports three key sizes: 128 bits, 192 bits, and 256 bits.
  - **Rounds**: The number of encryption steps (rounds) depends on the key size:
    - **128-bit key**: 10 rounds
    - **192-bit key**: 12 rounds
    - **256-bit key**: 14 rounds

Table 30.1 *AES configuration*

Size ofData Block	*Number ofRounds*	*Key Size*
128 bits	10	128 bits
	12	192 bits
	14	256 bits

- **How AES Works**
  - **Initial Step**: The plaintext is combined with an initial key using XOR (exclusive OR).
  - **Rounds**: AES performs multiple rounds of encryption.
    - Each round uses a **round key** derived from the original key.
    - The last round is slightly different and skips one of the operations.
  - **Final Output**: After all rounds, the encrypted data (ciphertext) is produced.
  - Each round has **four complex operations** that can be reversed during decryption.
  - The last round skips one of these operations.
- **Why AES is Better**

- AES is faster and more secure than DES and 3DES.
- It is now the **encryption standard** widely used for securing sensitive data.
    - **Use**: Adopted in secure communications like Wi-Fi encryption and HTTPS.

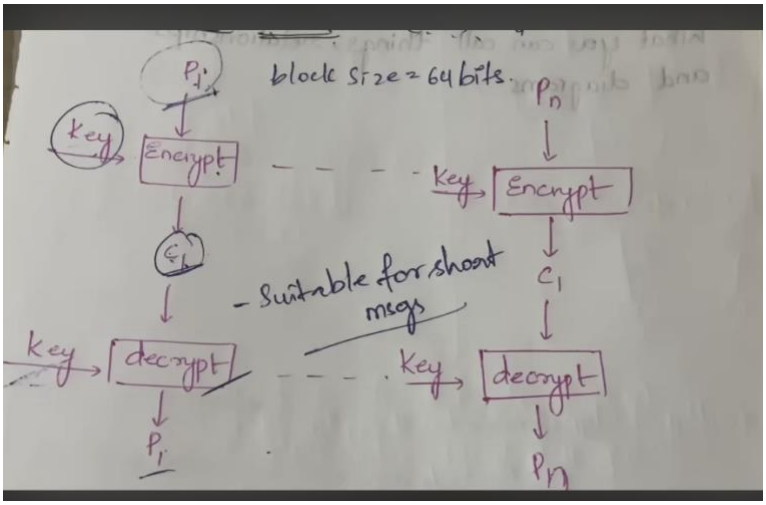


Modes of Operation – ECB, CBC, CFB, OFB

- A mode of operation defines how block ciphers are used to encrypt and decrypt data, especially when the data is larger or smaller than the block size.

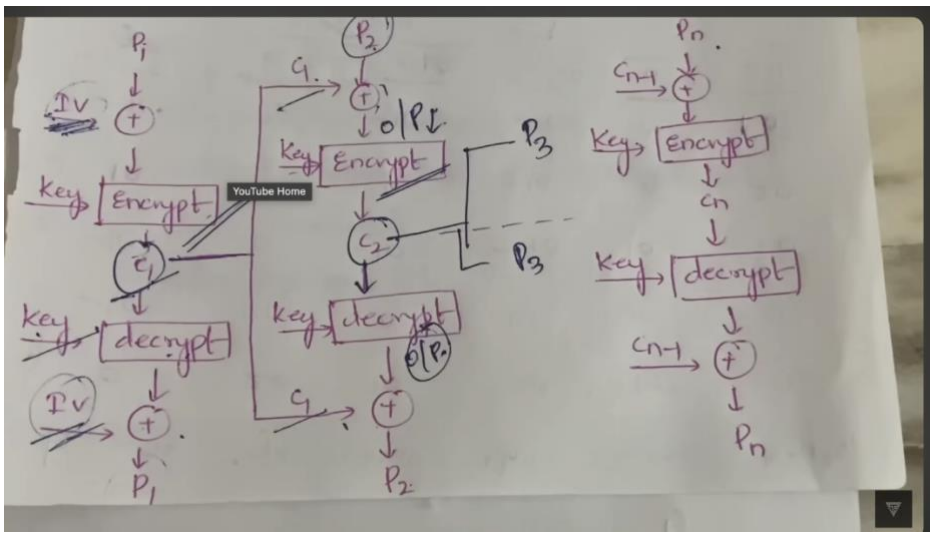
1. Electronic Code Book (ECB)

- **How it works**: Each plaintext block is encrypted separately to produce a ciphertext block.
- **Characteristics**:
    - Equal plaintext blocks produce equal ciphertext blocks (security flaw).
    - Blocks can be rearranged in ciphertext, which will reorder plaintext.
    - Independent encryption of blocks; errors in one block don't affect others.
    - Transmission errors affect only the corresponding block, not others.
- **Good for**: Small, independent data like encryption keys.



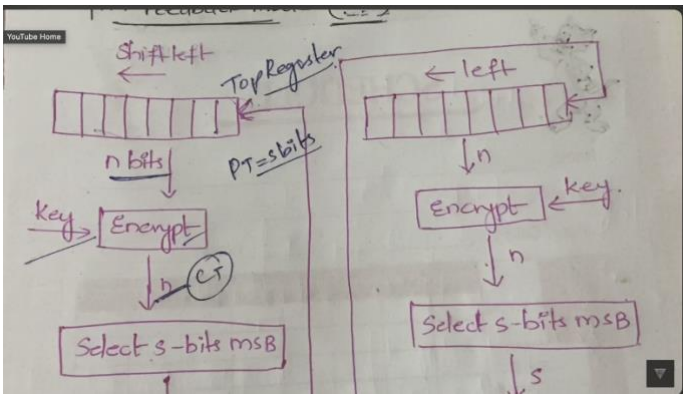
2. Cipher Block Chaining (CBC)

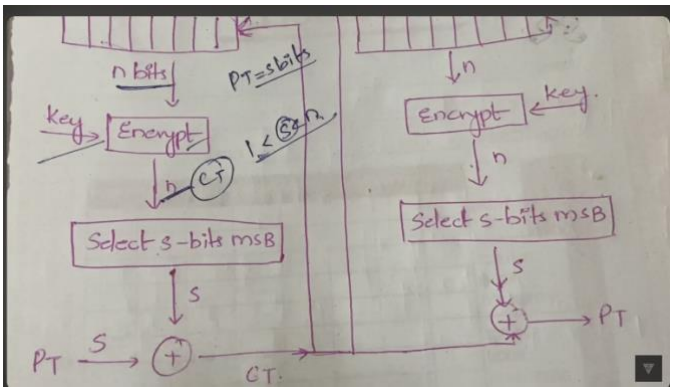
- **How it works**: Each plaintext block is XORed with the previous ciphertext block before encryption. An **Initialization Vector (IV)** is used for the first block.
- **Characteristics**:
  - Equal plaintext blocks result in different ciphertext blocks (better security).
  - Blocks are dependent; an error in one block affects others during decryption.
  - Transmission errors propagate, affecting subsequent blocks.
- **Good for**: Secure data transmission.



3. Cipher Feedback (CFB)

- **How it works**: Encrypts small chunks of data (e.g., 1, 4, or 8 bits) using feedback from the previous ciphertext. The cipher generates a stream key for encryption.
- **Characteristics**:
  - Changing the IV results in different ciphertext for the same plaintext.
  - Errors in one ciphertext block affect subsequent blocks.
  - Useful for streaming data.
- **Good for**: Encrypting small amounts of data or streaming applications.

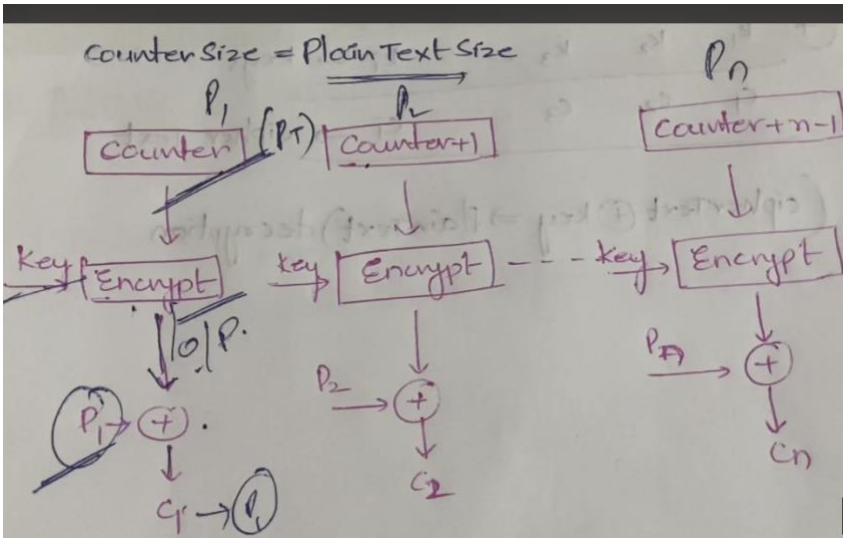




4. Output Feedback (OFB)

- **How it works**: Similar to CFB, but feedback comes from the **key stream** instead of the ciphertext, avoiding error propagation.
- **Characteristics**:
  - Changing the IV produces different ciphertext for the same plaintext.
  - Errors in ciphertext don't propagate to future blocks.
- **Good for**: Scenarios where error propagation needs to be avoided, like noisy communication channels.

5. Counter Mode(CTR):



Key Takeaways

- **ECB**: Fast but insecure for repeated patterns.
- **CBC**: Secure but vulnerable to error propagation.
- **CFB**: Flexible for small data chunks but prone to error propagation.
- **OFB**: Error-resilient but slightly more complex.

Asymmetric-Key Cryptography (Public Key Cryptography)

Asymmetric-key cryptography uses two keys: a **public key** for encryption (shared with everyone) and a **private key** for decryption (kept secret).

This method ensures secure communication because even if the public key is known, only the private key can decrypt the message.

Two common algorithms are **RSA** and **Diffie-Hellman**.

RSA Algorithm

Applications of RSA

- **Secure Key Exchange**: RSA is mainly used to securely exchange keys for symmetric-key cryptography.
- **Digital Signatures**: For verifying the authenticity of messages.
- **Authentication**: Ensuring only the intended recipient can decrypt the message.

Note: RSA is efficient for small data like keys or digital signatures but slow for large messages. Thus, it's often used to encrypt keys instead of entire messages.

Steps to Generate RSA Keys

1. **Select Prime Numbers**: Choose two large prime numbers, **p** and **q**.
2. **Calculate n**: Compute $n = p \times q$. This n is used for encryption and decryption.
3. **Calculate Totient (ϕ)**: Compute $\phi = (p-1) \times (q-1)$.
4. **Choose Public Key (e)**: Select a public key e such that $1 < e < \phi$ and e is co-prime to ϕ.
5. **Calculate Private Key (d)**: Determine d such that $(d \times e) \mod \phi = 1$.
6. **Public and Private Keys**: The **public key** is (e, n), and the **private key** is (d, n).

3. Encryption

- Plaintext message M is encrypted using the public key (e, n): $C = M^e \mod n$, where C is the ciphertext.

4. Decryption

- Ciphertext C is decrypted using the private key (d, n): $M = C^d \mod n$, where M is the original plaintext.

Example

- Let $p = 7$ and $q = 11$.
- Calculate $n = 7 \times 11 = 77$ and $\phi = (7-1)(11-1) = 60$.
- Choose $e = 13$. Calculate $d = 37$ because $(13 \times 37) \mod 60 = 1$.
- Public key: $(13, 77)$, Private key: $(37, 77)$.

Encryption:

- Suppose Alice wants to send the plaintext **5**.
- Ciphertext $C = P^e \mod n = 5^{13} \mod 77 = 26$.
- Alice sends **26** to Bob.

Decryption:

- Bob receives **26** and uses his private key to decrypt:
- Plaintext $P = C^d \mod n = 26^{37} \mod 77 = 5$.
- Bob gets the original plaintext **5**.

MOD IN CALCI – (13 X 37 = 481) calculate 481%60

481/60 = 8.0166… → 8.0166… - 8 = 0.166… → 0.166… X 60 = 1

Divide → Subtract integral part of quotient → multiply with divisor again

Diffie-Hellman Key Exchange (Simple Explanation)

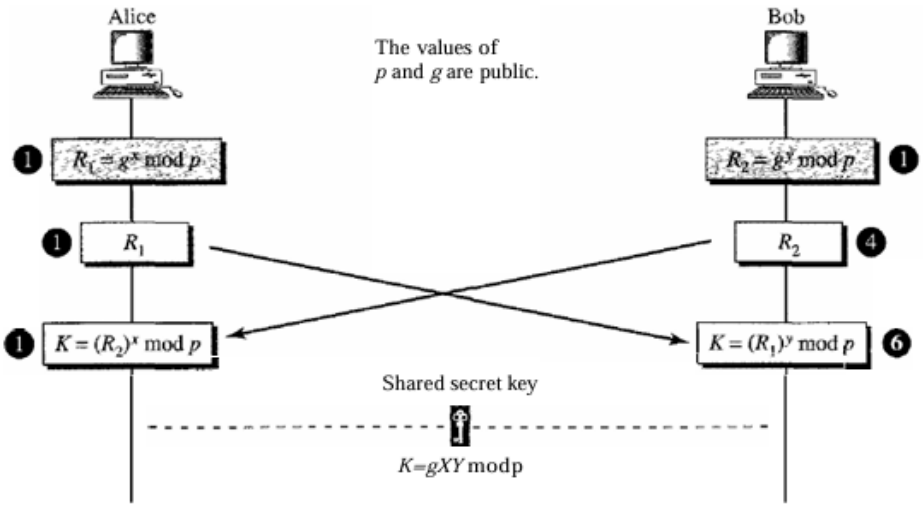
The **Diffie-Hellman** algorithm is used to securely exchange a **shared secret key** between two parties (like Alice and Bob) over the internet. This shared key can later be used for symmetric encryption to secure communication. Here's how it works:

Steps of the Diffie-Hellman Algorithm:

1. **Choose Public Values**:
   - Alice and Bob agree on two public numbers, p (a large prime number) and g (a base number). These numbers are not secret and can be shared openly.
2. **Generate Private Keys**:
   - Alice picks a **private random number** x.
   - Bob picks a **private random number** y.
   - These private numbers are **not shared** with anyone.
3. **Calculate Public Keys**:
   - Alice calculates her public key: R1 = g^x mod p.
   - Bob calculates his public key: R2 = g^y mod p.
   - They exchange these public keys (R1 and R2).
4. **Calculate Shared Secret Key**:
   - Alice uses Bob's public key (R2) to compute the shared secret: K = R2^x mod p.
   - Bob uses Alice's public key (R1) to compute the same shared secret: K = R1^y mod p.

Both Alice and Bob end up with the **same shared key** K, without ever sending their private numbers (x or y) over the internet.

Figure 30.26 *Diffie-Hellman method*



The symmetric (shared) key in the Diffie-Hellman protocol is $K = g^{xy} \bmod p$.

Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume $g = 7$ and $p = 23$. The steps are as follows:

1. Alice chooses $x = 3$ and calculates $R_1 = 7^3 \bmod 23 = 21$.
2. Bob chooses $y = 6$ and calculates $R_2 = 7^6 \bmod 23 = 4$.
3. Alice sends the number 21 to Bob.
4. Bob sends the number 4 to Alice.
5. Alice calculates the symmetric key $K = 4^3 \bmod 23 = 18$.
6. Bob calculates the symmetric key $K = 21^6 \bmod 23 = 18$.

The value of K is the same for both Alice and Bob; $g^{xy} \bmod p = 7^{18} \bmod 23 = 18$.

Man-in-the-Middle Attack

A potential issue with Diffie-Hellman is the **man-in-the-middle attack**:

- An attacker (Eve) intercepts the public keys exchanged between Alice and Bob.
- Eve sends her own keys to both parties, pretending to be the other person.
- Alice and Bob think they are sharing a key with each other, but they are actually sharing a key with Eve.
- Eve can decrypt, read, and modify messages between them.

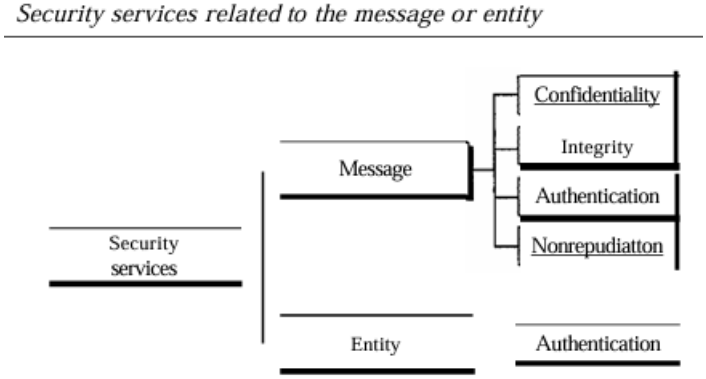
Preventing Man-in-the-Middle Attacks:

- To prevent this, **authentication** is needed. Alice and Bob need a way to verify each other's identity before exchanging keys.

In summary, **Diffie-Hellman** helps create a secure shared key for encryption without directly sharing the key, but it needs additional measures like authentication to protect against attacks.

Security Services in Network Security

Network security provides **five main services** to protect communication and users. These services ensure that messages are transmitted securely and that the identity of users is verified. Here's a breakdown of the five security services:



1. Message Confidentiality

- This ensures that **only the intended receiver** can read the message.
- The message is encrypted (made unreadable) by the sender and decrypted (made readable) by the receiver.

- For example, when you communicate with your bank online, the information you send must be kept private and not exposed to anyone else, like hackers.

How to Achieve Confidentiality?

- Use cryptography:
  - **Symmetric-Key Cryptography**: Both sender and receiver use the same secret key.
  - **Asymmetric-Key Cryptography**: A pair of keys (public and private) are used, with the public key for encryption and the private key for decryption.

Symmetric-Key Cryptography – draw image:

1. **How it Works:**
   - Sender and receiver share a secret key.
   - The sender encrypts the message with the key, and the receiver decrypts it using the same key.
2. **Challenges:**
   - Sharing the secret key securely, especially over long distances, is difficult.
   - For example, someone in the US cannot meet a person in China to exchange the key.
3. **Solution:**
   - Use a **session key**:
     - A temporary key for one communication session.
     - The session key is exchanged securely using asymmetric-key cryptography.
4. **Advantage:**
   - Symmetric-key cryptography is fast and efficient for encrypting large messages.

Asymmetric-Key Cryptography: - draw image

1. **How it Works:**
   - No need to share a secret key.
   - Every user has:
     - A **public key** (shared with everyone) for encryption.
     - A **private key** (kept secret) for decryption.
2. **Example:**
   - If Alice wants to send a message to Bob:
     - Alice uses Bob's public key to encrypt the message.
     - Bob uses his private key to decrypt it.
3. **Two-Way Communication:** bob does same
4. **Challenges:**
   - **Efficiency**: Asymmetric cryptography is slow and inefficient for long messages.
   - **Trust**: The sender must ensure they are using the genuine public key of the receiver to avoid interception by attackers (e.g., someone pretending to be Bob).

2. Message Integrity

What is Message Integrity?

- Ensures that the contents of a message or document have not been tampered with during transmission or storage.
- It is different from confidentiality; it focuses on preventing unauthorized changes, not hiding the content.

Message and message digest

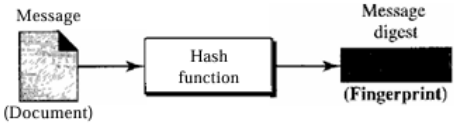
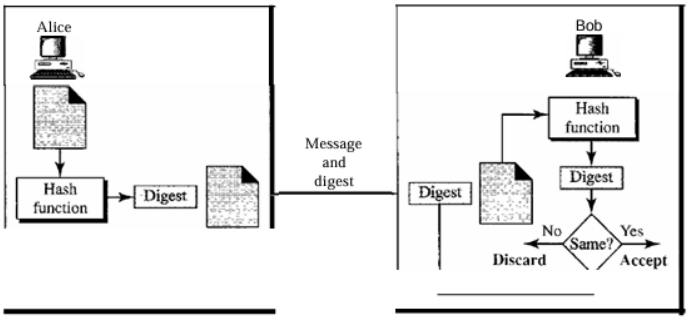


Figure 31.5 *Checking integrity*



How Message Integrity Works:

1. **Fingerprint Analogy:**
    - Just as a physical fingerprint can verify the authenticity of a document, a **message digest** acts as a digital fingerprint for verifying the integrity of a message.
2. **Message Digest Creation:**
    - A special algorithm called a **hash function** processes the message and generates a unique fixed-size output, called the **message digest**.
    - If the message changes even slightly, the digest will change completely.
3. **Sending the Message:**
    - The sender creates a message digest and sends it along with the message to the receiver.
4. **Verification by the Receiver:**
    - The receiver runs the received message through the same hash function to generate a new digest.
    - The new digest is compared with the one received:
        - **If they match**, the message is intact.
        - **If they don't match**, the message has been tampered with.

Hash Function Properties:

1. **One-Way Function:**
    - The message digest cannot be reversed to recreate the original message.
    - Similar to how you cannot recreate a full fingerprint from just a partial one.
2. **Weak Collision Resistance:**

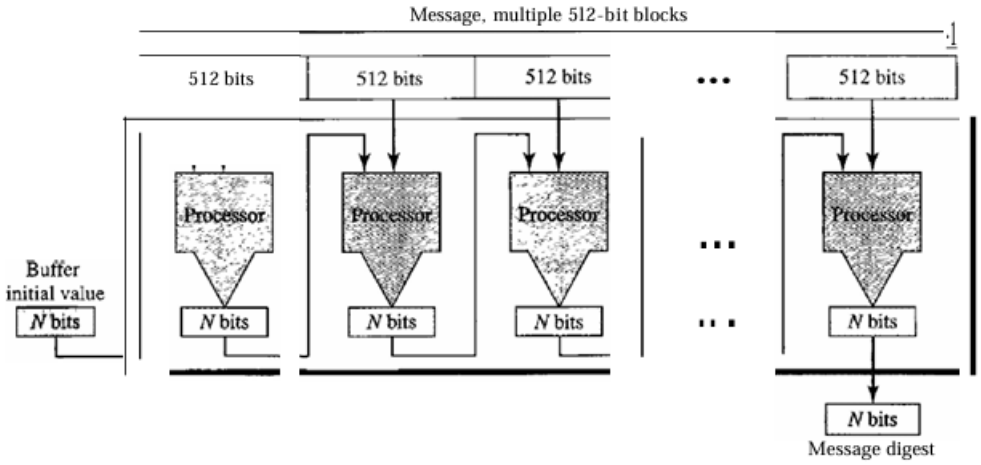
- It is difficult to create another message that produces the same digest as the original message.

3. **Strong Collision Resistance:**
    - It is nearly impossible to find two completely different messages with the same digest.

Popular Hash Algorithm: SHA-1

1. **How it Works:**
    - Breaks the message into 512-bit blocks.
    - Processes each block using a series of 80 steps to produce a **160-bit digest**.
    - If the last block is smaller than 512 bits, it is padded with extra bits.
2. **Output:**
    - The result is a 160-bit (32-character) unique fingerprint for the message.
3. **Purpose:**
    - Ensures the integrity of the message. Even the smallest change in the message will produce a completely different digest.

Figure 31.7 *Message digest creation*



3.Message Authentication

What is Message Authentication?

- Ensures that the message is not only intact (unchanged) but also confirms the identity of the sender.
- Without authentication, a message might appear valid but could have been sent by someone pretending to be the real sender.

How Message Authentication Works:

1. Hash Function Limitations:

- A hash function can detect message tampering but **cannot verify the sender's identity**.

- The output of a hash function is called a **Modification Detection Code (MDC)**, which confirms if a message has been altered but not who sent it.

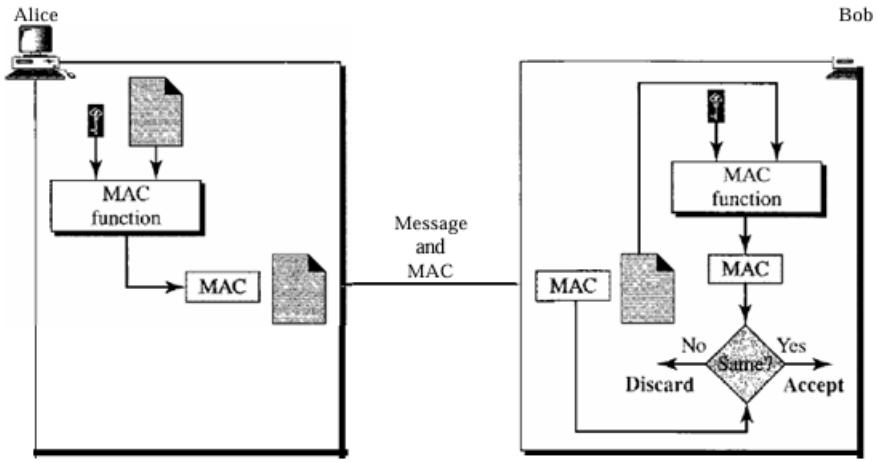
2. Message Authentication Code (MAC):

- To authenticate the sender, a **keyed hash function** is used. This adds a secret key (shared between the sender and receiver) to the hash function.
- The result is called a **Message Authentication Code (MAC)**.

How it Works:

1. **Alice (Sender):**
    - Combines the message with a secret key shared with Bob (KAB).
    - Runs a keyed hash function to generate a **MAC**.
    - Sends the message and MAC to Bob.
2. **Bob (Receiver):**
    - Separates the message and MAC.
    - Uses the same secret key (KAB) and hash function to create a new MAC from the message.
    - Compares his MAC with Alice's MAC:
        - **If they match:** The message is intact and truly from Alice.
        - **If they don't match:** The message is tampered with or from an imposter.

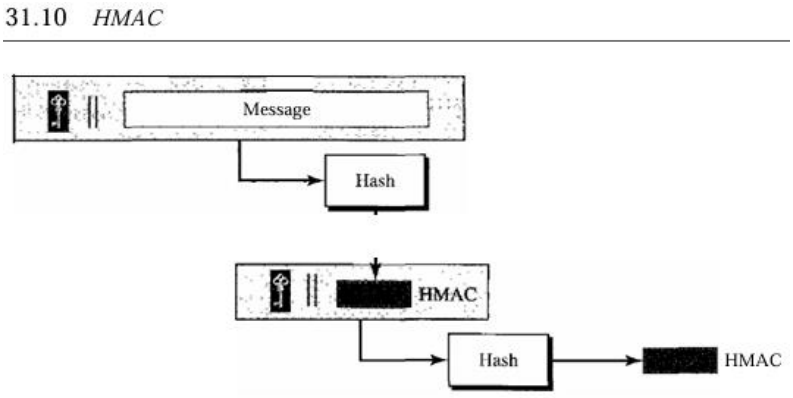
Figure 31.9 *MAC, created by Alice and checked by Bob*



3. HMAC (Hashed MAC):

- A modern and more secure version of MAC based on keyless hash functions like SHA-1.
- **Steps in HMAC:**
    1. Prepend the secret key to the message.
    2. Apply a keyless hash function (e.g., SHA-1) to generate an **intermediate HMAC**.
    3. Prepend the same secret key again and rehash it to generate the final HMAC.

- **Verification:**
    - The receiver generates their own HMAC and compares it with the sender's HMAC to ensure message integrity and sender authentication.



4. Message Nonrepudiation

- This ensures that the **sender cannot deny** having sent the message.
- For example, if you request a money transfer, the bank must have proof that **you** made that request, so you cannot deny it later.

5. Entity Authentication

- This involves verifying the **identity of the user** before granting access to resources.
- For example, when a student logs into their university system, the university must verify that the person logging in is indeed the correct student.
- It protects both the user and the system from unauthorized access.

In summary, these security services protect messages from being tampered with, ensure the identity of users, and prevent any denial of sending important messages.

Digital Signature

A **digital signature** is like signing a document, but electronically. It is used to:

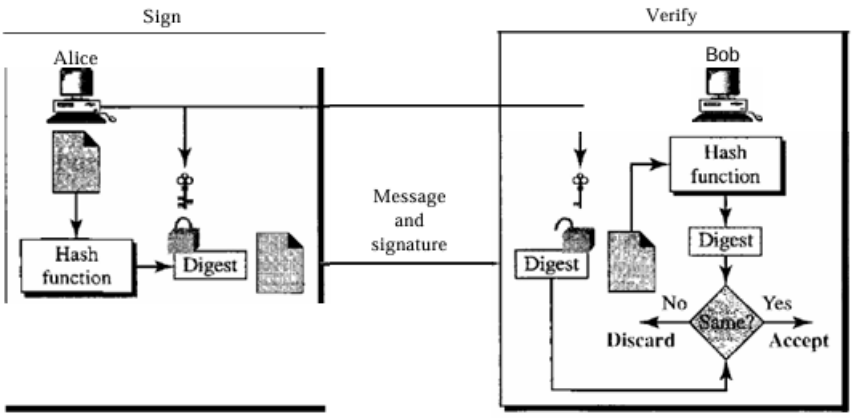
1. **Verify the sender's identity** (e.g., the message truly came from Alice).
2. **Ensure message integrity** (the message wasn't changed during transit).
3. **Provide non-repudiation** (the sender can't deny they sent the message).

Why Not Just Use MAC?

A **Message Authentication Code (MAC)** requires a shared secret key between the sender and receiver. Digital signatures, however, use **asymmetric keys**:

- A **private key** (known only to the sender) is used to sign.
- A **public key** (shared with everyone) is used to verify.

Figure 31.12 *Signing the digest in a digital signature*



Comparison: Digital vs. Conventional Signature

Feature	Conventional Signature	Digital Signature
Inclusion	Signature is part of the document.	Signature is a separate document sent with the message.
Verification	Compare signature with one on file.	Use public key and signature to verify authenticity.
Relationship	One signature can be used for many documents.	One signature is unique to each message.
Duplicity	Copies can be distinguished from the original.	Copies cannot be distinguished unless timestamped.
Keys	A stored signature acts as a public key.	Uses sender's private key for signing, and public key for verification.

How Digital Signatures Work

1. **Signing the Document:**
   - Alice encrypts (signs) the message using her private key.
   - Bob decrypts (verifies) the message using Alice's public key.
   - This ensures the message is from Alice and hasn't been altered.
2. **Signing the Digest (More Efficient):**
   - A digest (summary) of the message is created using a hash function.
   - Alice signs the digest with her private key and sends it along with the message.
   - Bob recreates the digest from the message and verifies it using Alice's public key.

Services Provided by Digital Signatures

1. **Message Integrity:**
   - Any changes to the message result in a different signature, ensuring the message is unchanged.
2. **Message Authentication:**

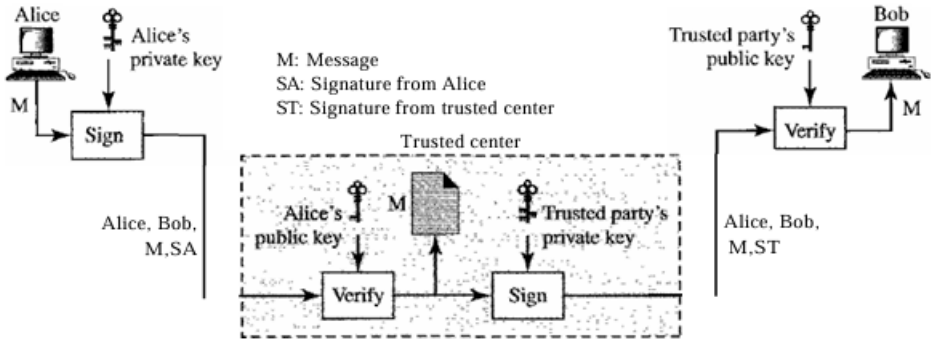
- Since Alice's public key is needed to verify the signature, Bob knows the message is from Alice.

3. **Non-Repudiation:**
   - Alice cannot deny sending the message because only her private key could have created the signature.

Non-Repudiation with a Trusted Third Party

- If a dispute arises (e.g., Alice denies sending the message), a **trusted third party** can act as a mediator:
  1. Alice sends the message and signature to the trusted center.
  2. The center verifies the signature and stores the message with a timestamp.
  3. If a dispute occurs, the center provides proof that Alice sent the message.

Figure 31.13 *Using a trusted center for nonrepudiation*



Examples of Digital Signature Schemes

- **RSA**: One of the first digital signature systems.
- **DSS (Digital Signature Standard)**: A widely used standard for digital signatures.

Network Security Services (Brief Overview)

1. **Message Confidentiality**
   Ensures that only the intended recipient can read a message. This is achieved using encryption, which makes the message unreadable to unauthorized parties.
   - **Symmetric Key Cryptography**: Both the sender and receiver use the same secret key for encryption and decryption. It's efficient but requires secure key exchange, often handled by using a temporary session key.
   - **Asymmetric Key Cryptography**: Uses two keys: a public key (for encryption) and a private key (for decryption). It's secure but slower and requires a trust system to verify the authenticity of the public key.
2. **Message Integrity**
   Ensures the message has not been altered during transmission.

- A **message digest** (created using a hash function) acts as a fingerprint for the message. The sender generates the digest, and the receiver checks if it matches a newly created digest to confirm the message hasn't been tampered with.
- Hash functions should meet criteria like **one-wayness** (difficult to recreate the message from the digest), **weak collision resistance** (hard to forge another message with the same digest), and **strong collision resistance** (ensures no two different messages have the same digest).

3. **Message Authentication**
   Confirms the identity of the sender and ensures the message wasn't sent by an imposter.

- This goes beyond integrity to verify who sent the message, ensuring it's from a trusted source.

4. **Message Nonrepudiation**
   Prevents the sender from denying they sent a message.

- Ensures there is proof (such as a digital signature) that the sender indeed sent the message, preventing future denial of the transaction or communication.

5. **Entity Authentication**
   Verifies the identity of users or entities before allowing access to systems or resources, like logging into a university system.

- This helps protect against unauthorized access and ensures that only legitimate users are granted access.

Message Authentication (Brief Overview)

- **Hash Functions**: Ensure message **integrity** by detecting any changes in the message, but they don't prove the **sender's identity**.
- **Modification Detection Code (MDC)**: A hash function creates an MDC that detects modifications but doesn't authenticate the sender.
- **Message Authentication Code (MAC)**: To authenticate the sender, a **keyed hash function** is used. Both the sender (Alice) and receiver (Bob) share a **symmetric key**. Alice creates a MAC using the key and sends the message with the MAC to Bob. Bob verifies the MAC to ensure the message is from Alice and hasn't been altered.
- **HMAC (Hashed MAC)**: An enhanced MAC method that uses keyless hash functions (e.g., SHA-1) for better security. It applies the key to the message in multiple steps to create an HMAC, which is then verified by the receiver to authenticate the sender and ensure message integrity.

Digital Signature (Brief Overview)

- **Purpose**: A **digital signature** proves the authenticity of a message, ensuring that it came from the correct sender (e.g., Alice) and hasn't been tampered with.
- **Difference from MAC**: Unlike a **Message Authentication Code (MAC)**, which uses a shared symmetric key, a digital signature uses **asymmetric keys** (a private key for signing and a public key for verification).
- **How It Works**:

- **Signing**: Alice uses her **private key** to sign a message or its digest (a unique summary of the message).
- **Verification**: Bob uses Alice's **public key** to verify that the message was signed by Alice and hasn't been altered.

- **Key Differences from Conventional Signatures**:
- **Conventional Signature**: The signature is part of the document, and it’s verified by comparing it to a stored version.
- **Digital Signature**: The signature is separate from the message. The recipient uses a verification process to ensure the message's authenticity.

- **Types of Digital Signature**:
- **Signing the Document**: Alice signs the entire document with her private key.
- **Signing the Digest**: A more efficient method where Alice signs a digest (a short summary) of the message instead of the entire message.
- **Services Provided**:
- **Message Integrity**: Ensures that the message hasn't been altered.
- **Message Authentication**: Confirms that the message came from the claimed sender (Alice).
- **Non-repudiation**: Prevents the sender from denying that they sent the message.
- **Trusted Third Party**: To ensure **non-repudiation**, a trusted party can store the signed message and provide evidence later if the sender denies sending it.
- **Key Point**: Digital signatures use asymmetric cryptography (public and private keys) and provide more secure and verifiable authentication than symmetric keys like those used in MACs.