Unit 4

Cryptographic Hash Functions and Digital Signatures

(8 Hours)

4.4. Message Authentication, Message Authentication Functions, Message Authentication Codes

4.5. Hash Functions, Properties of Hash functions, Applications of Hash Functions

4.6. Message Digests: MD4 and MD5

4.7. Secure Hash Algorithms: SHA-1 and SHA-2

4.8. Digital Signatures: Direct Digital Signatures, Arbitrated Digital Signature

4.9. Digital Signature Standard: The DSS Approach, Digital Signature Algorithm

4.10. Digital Signature Standard: The RSA Approach

Sanjeev Thapa, Er. DevOps, SRE, CKA, RHCSA, RHCE, RHCSA-Openstack, MTCNA, MTCTCE, UBSRS, HEv6, Research Evangelist

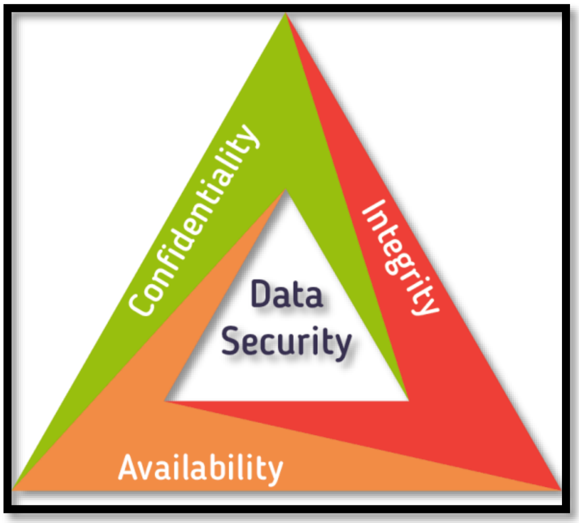
CIA

Quick relation:

- **Confidentiality:** prevents *reading*
- **Integrity:** prevents *modifying*
- **Authentication:** prevents *impersonation*

Confidentiality

- Means **keeping information secret** from unauthorized people.
- Ensures only intended users can **read** the data.
- Achieved mainly using **encryption** (AES, RSA, TLS, etc.).
- Example: Encrypting messages in WhatsApp so outsiders can't read them.



Data Integrity

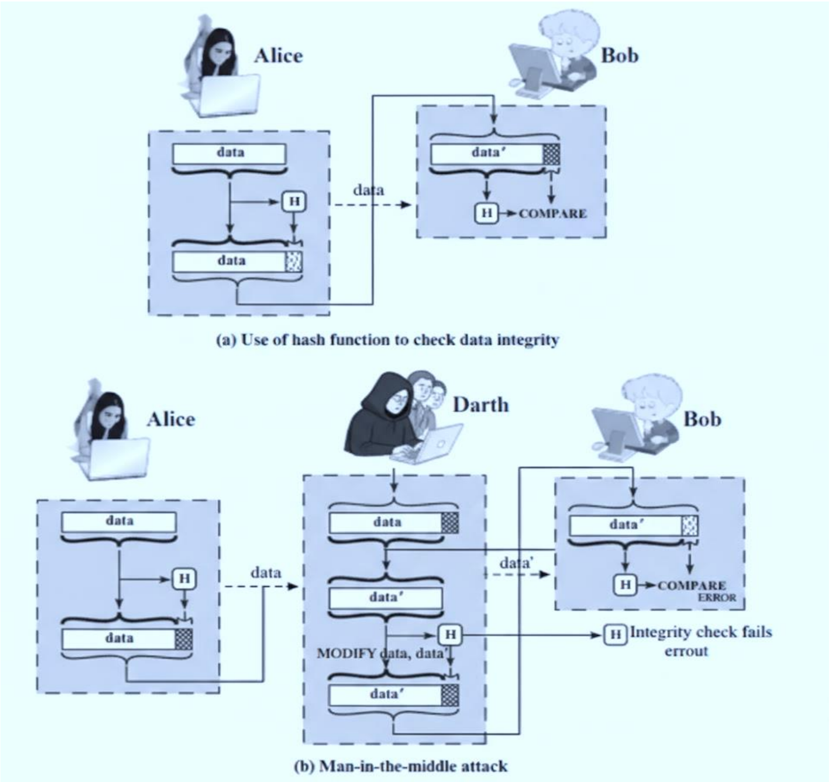
- Means **data is not changed** (accidentally or maliciously) during storage or transmission.
- Ensures the received data is **exactly the same** as the sent data.
- Achieved using **hash functions** (SHA-256), **MAC**, **digital signatures**.
- Example: If a file is modified, its hash value changes.

Authentication

- Means **verifying identity** (who is sending/receiving) or confirming the message is from a genuine source.
- Types:
  - **User authentication:** password, OTP, biometrics.
  - **Message authentication:** MAC / digital signatures verify sender + integrity.
- Example: Logging into Gmail with password + OTP confirms it's really you.

Message Authentication:

- Message authentication is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).
- When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**.
- The essence of the use of a hash function for message integrity is as follows.
  - The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message.
  - The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.
  - If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered (Figure a).
  - The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver. This type of attack is shown in Figure b.



(a) Use of hash function to check data integrity

(b) Man-in-the-middle attack

Authentication Functions

Authentication -> Verifying user Identity

Authentication is the process of confirming the identity of a user, device, or system by validating provided credentials before granting access to a network or its resources.

- It prevents unauthorized usage of systems
- It ensures secure interaction within a network environment
- It maintain security, reliability, and trust within the system

Authentication Functions / 3 ways to produce message authentication.

1) **Message Encryption (Ciphertext acts as authenticator)**
2) **MAC (Message Authentication Code)**
3) **Hash Function (H) / Message Digest**

Legend

M = message

E / D = encryption / decryption

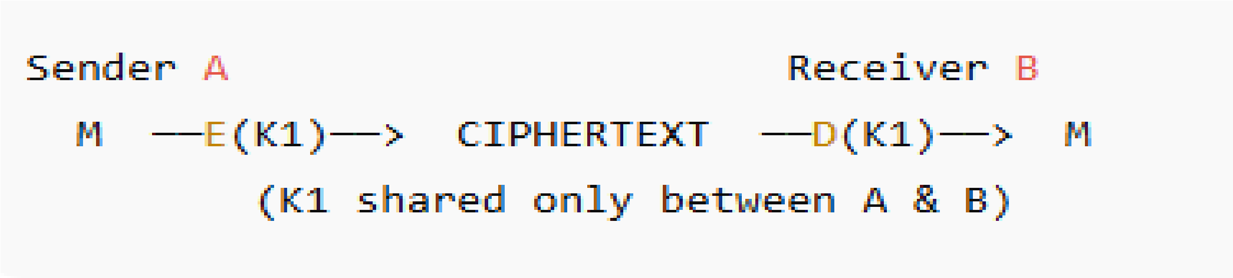
K1 = shared secret key (symmetric)

PubX / PrivX = public/private key of user X

Authenticator = value used to prove message is genuine (MAC/tag/signature)

1) Message Encryption (Ciphertext acts as authenticator)

(a) Symmetric encryption (shared key)

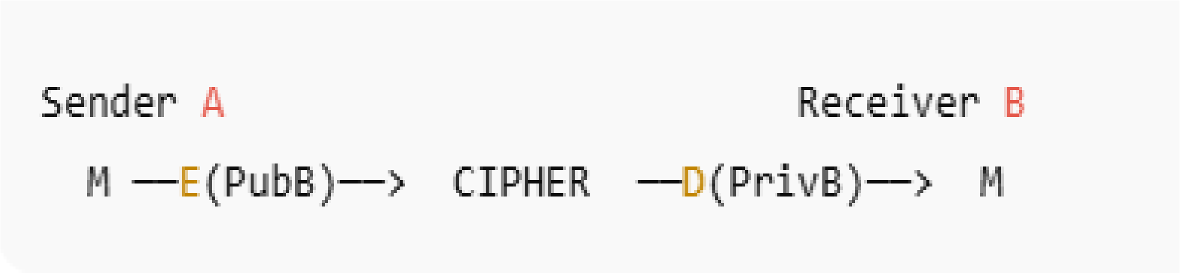


```
Sender A                          Receiver B
  M  ——E(K1)——>  CIPHERTEXT  ——D(K1)——>  M
          (K1 shared only between A & B)
```

✅ Confidentiality
✅ Authentication (only users with K1 can create valid ciphertext)
⚠️ Non-repudiation ✗ (because both share K1)

(b) Public-key encryption for confidentiality

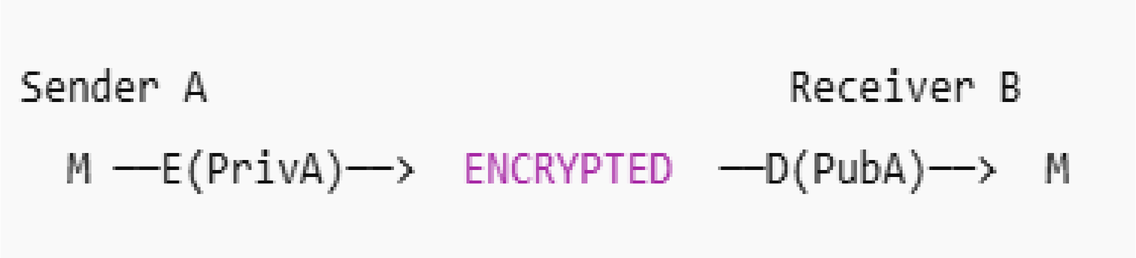
✅ Confidentiality



```
Sender A                          Receiver B
  M ——E(PubB)——>  CIPHER  ——D(PrivB)——>  M
```

❌ Authentication (anyone can use PubB to encrypt)

(c) "Private-key encryption" (signature idea) for authentication



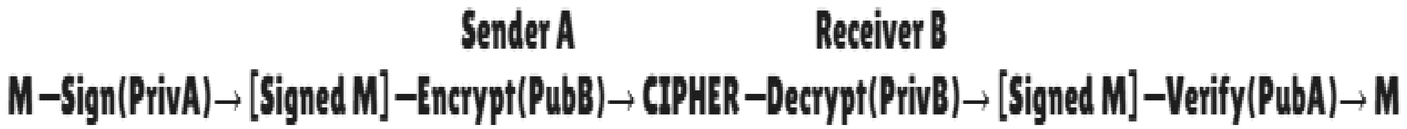
```
Sender A                         Receiver B

  M —E(PrivA)—>  ENCRYPTED  —D(PubA)—>  M
```

✅ Authentication (only A has PrivA)
❌ Confidentiality (anyone with PubA can decrypt)

In practice we don't encrypt the full message with PrivA; we **sign the hash** of the message (digital signature).

(d) To get BOTH (Sign then Encrypt)

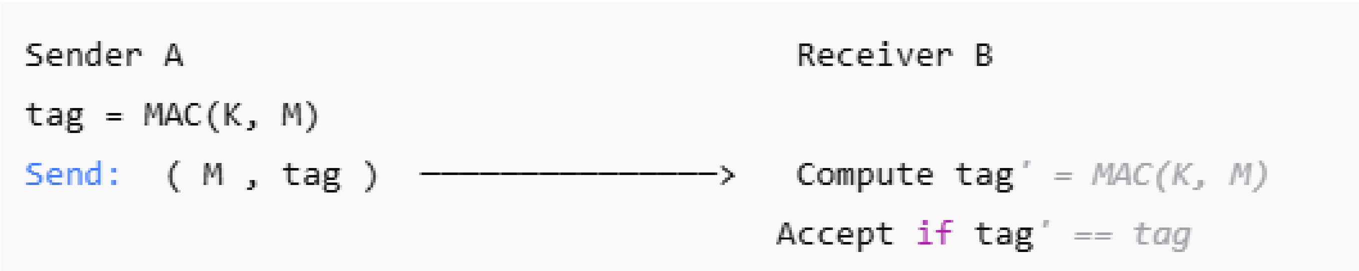


```
                          Sender A               Receiver B
M –Sign(PrivA)→ [Signed M] –Encrypt(PubB)→ CIPHER –Decrypt(PrivB)→ [Signed M] –Verify(PubA)→ M
```

✅ Confidentiality + ✅ Authentication + ✅ Integrity

2) MAC (Message Authentication Code)

Key idea: fixed-length tag produced using a **secret key** and the message.

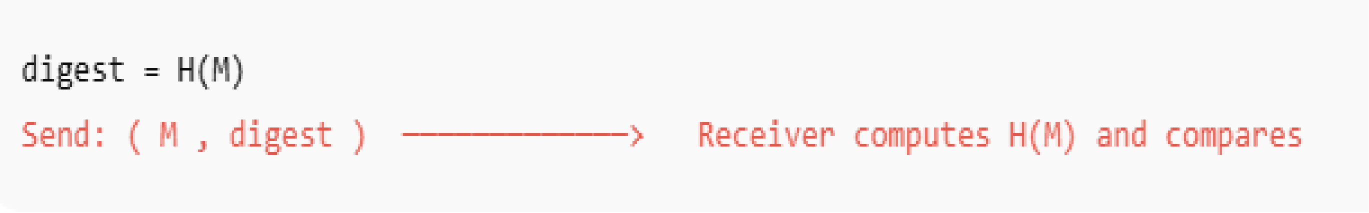


```
Sender A                                Receiver B
tag = MAC(K, M)
Send:  ( M , tag )  ————————————————>   Compute tag' = MAC(K, M)
                                       Accept if tag' == tag
```

✅ Integrity
✅ Authentication
❌ Confidentiality
❌ Non-repudiation (shared key)

3) Hash Function (H) / Message Digest

Key idea: fixed-length digest, **no key**.

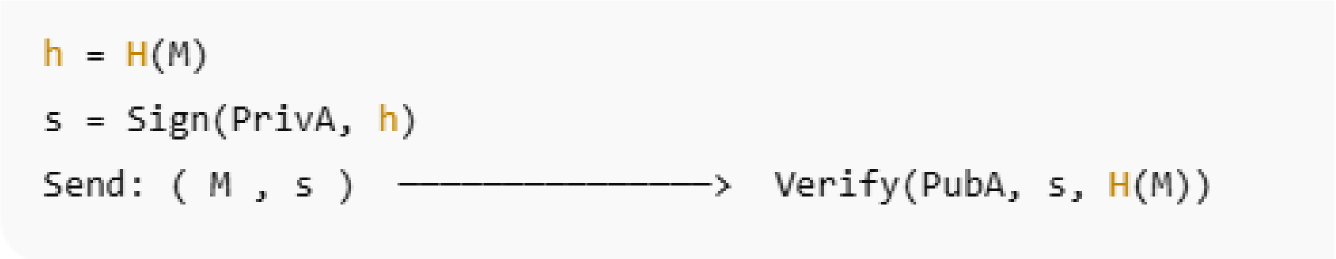


```
digest = H(M)
Send: ( M , digest )  ————————————>  Receiver computes H(M) and compares
```

✅ Detects change (integrity check)

⚠ Authentication ❌ (attacker can change M and recompute digest)

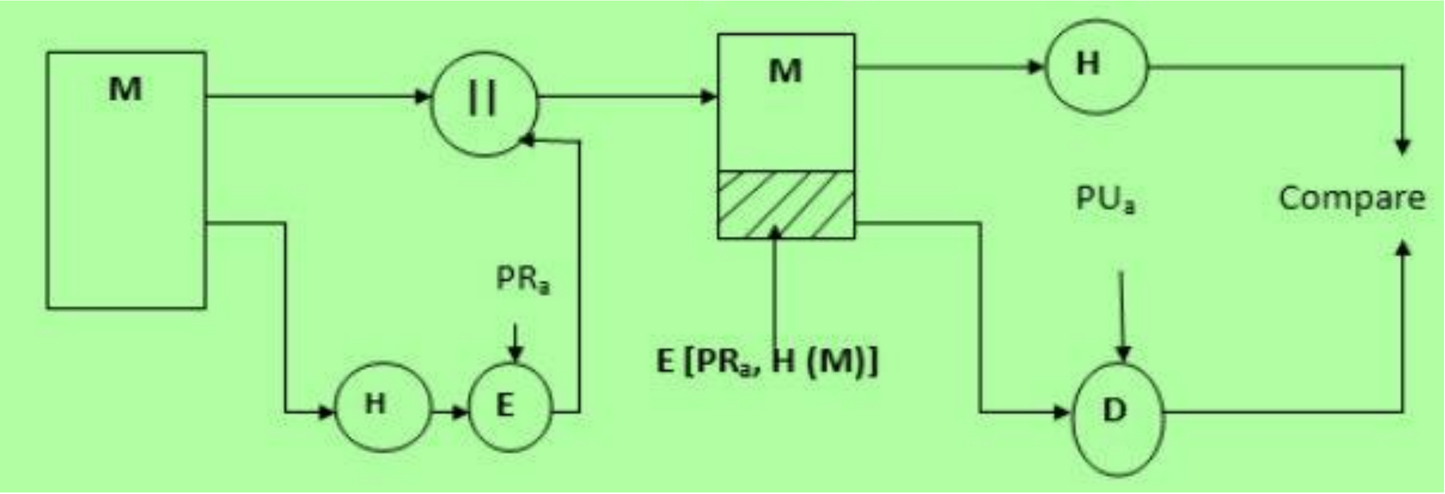
Hash + Signature (real authentication using hash)



```
h = H(M)
s = Sign(PrivA, h)
Send: ( M , s )  ————————————————>  Verify(PubA, s, H(M))
```

✅ Integrity + ✅ Authentication + ✅ Non-repudiation

Numerical RSA Setup



Choose small primes:

- $p = 11,\ q = 17$
- $n = pq = 11 \times 17 = 187$
- $\phi(n) = (p-1)(q-1) = 10 \times 16 = 160$

Pick public key exponent $e = 7$(coprime with 160)

Find private key $d$such that:

$$e \cdot d \equiv 1(\mathrm{mod}160)$$

Since $7 \times 23 = 161 \equiv 1(\mathrm{mod}160)$, so:

- $d = 23$

✅ Public key $PU_a = (e, n) = (7,187)$
✅ Private key $PR_a = (d, n) = (23,187)$

Step 1: Message and Hash

Let message be:

- $M = 123$

Use a simple toy hash (not real crypto hash):

$$H(M) = M\,mod\,100$$

So:

$$h = H(123) = 23$$

Step 2: Sign the Hash (E[PR$_a$, H(M)])

Signature:

$$S = h^d\,mod\,n = 23^{23}\,mod\,187$$

This evaluates to:

$$S = 133$$

Sender transmits:

$$(M \mid\mid S) = (123,\ 133)$$

Step 3: Verification at Receiver (using PU$_a$)

Receiver computes hash again:

$$h_1 = H(M) = H(123) = 23$$

Receiver “decrypts” signature using public key:

$$h_2 = S^e\,mod\,n = 133^7\,mod\,187$$

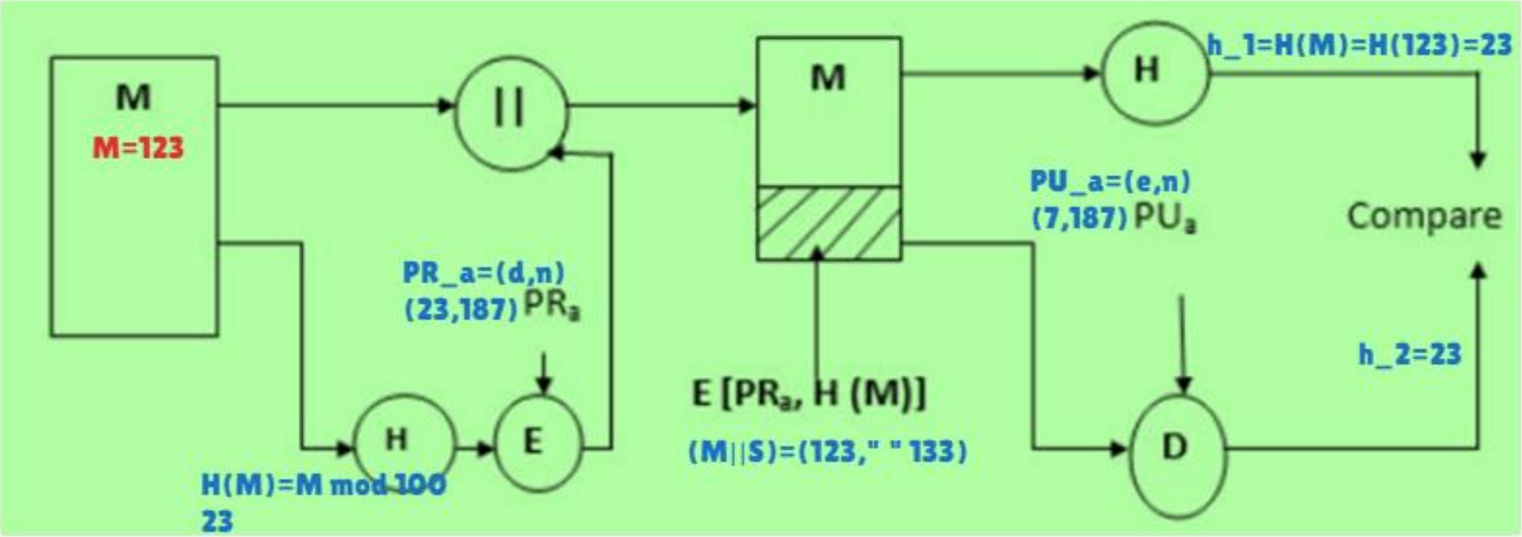
This evaluates to:

$$h_2 = 23$$

Step 4: Compare

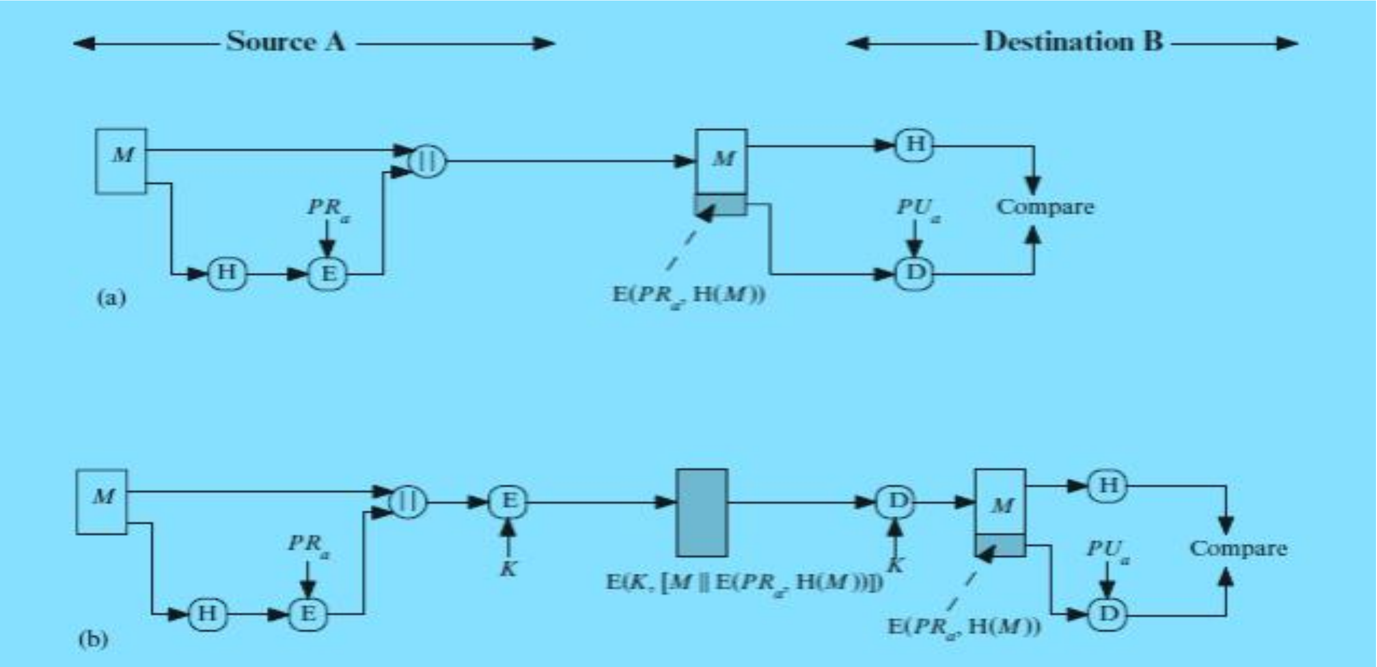
$h_1 =?\quad h_2 \ \Rightarrow\ 23 = 23$
$\Rightarrow$ Signature V ALID

✅ Integrity: message wasn’t changed
✅ Authentication: only A’s private key could create S
✅ Non-repudiation: A can’t deny (in real systems)



Digital Signatures:

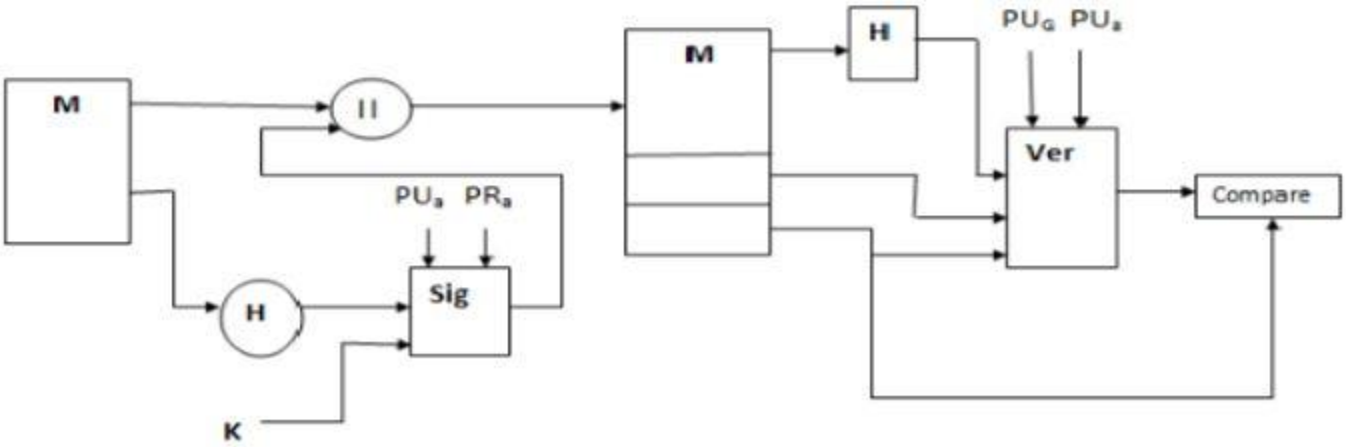
- Another important application, which is similar to the message authentication application, is the **digital signature**.
- The operation of the digital signature is similar to that of the MAC.
- In the case of the digital signature, the hash value of a message is encrypted with a user's private key.
- Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature.
- In this case, an attacker who wishes to alter the message would need to know the user's private key.
- Following figures illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.
    - The hash code is encrypted, using public-key encryption with the sender's private key. As with Figure b, this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.
    - If confidentiality as well as a digital signature is desired, then the message plus the private-keyencrypted hash code can be encrypted using a symmetric secret key. This is a common technique.



Digital Signature Standard: The DSS Approach, Digital Signature Algorithm

Idea

- DSS uses **Digital Signature Algorithm (DSA)**: sign the **hash (digest)** of the message, not the whole message.
- Signature has **two values: (r, s)**.



Keys/parameters

- Sender has **private key (PR$_a$)** and **public key (PU$_a$)**.
- DSA also uses public domain parameters (often treated as a “global public key / public parameters”).
- Uses a fresh random **k** for each signature (must never repeat).

Working (steps)

1. Compute digest: h = H(M)
2. Pick random k
3. Generate signature (r, s) using PR$_a$, k, and parameters.
4. Send (M, r, s)
5. Receiver computes H(M) and runs verification using sender’s PU$_a$ + public parameters.
6. Output confirms validity (verification result corresponds to signature component **r** when valid).

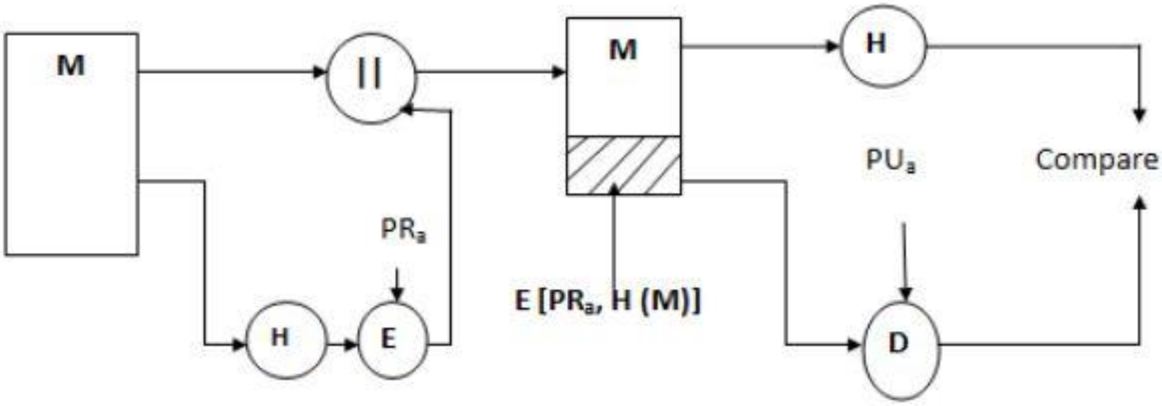
What it provides

- ✅ Integrity, ✅ Authentication, ✅ Non-repudiation (only sender with private key can create valid signature).

Digital Signature Standard: The RSA Approach

Idea

- RSA signatures: **sign the message hash** using RSA private key, verify using RSA public key.
- Common conceptual form:
  - **Sign:** S = (H(M))^d mod n
  - **Verify:** check H(M) == S^e mod n



Working (steps)

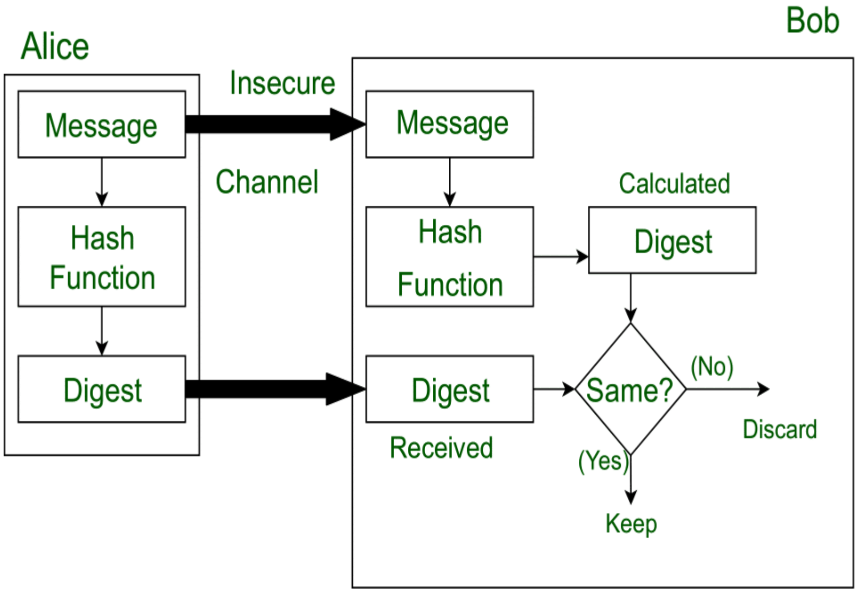
1. Compute digest: h = H(M)
2. Sender signs digest with **private key (d, n)** → signature S
3. Send (M, S)
4. Receiver computes h = H(M) and verifies using **public key (e, n)**

Message Digests: MD4 and MD5

A **message digest** is the **fixed-length hash value** produced from a message.

Used for **integrity checking** (if message changes, digest changes).

Message Digest is used to ensure the integrity of a message transmitted over an insecure channel (where the content of the message can be changed). It refers to a fixed-size numerical representation (hash value) of a message or data, created by a hash function.



It's a core concept in ensuring data integrity. The message is passed through a Cryptographic hash function. This function creates a compressed image of the message called Digest.

Message Digests Characteristics

- **Purpose**: To verify that data has not been altered.
- **Security Use**: Common in digital signatures, data integrity checks, and password storage.
- **Generated By**: Cryptographic hash functions like MD5, SHA-1, SHA-256.
- **Fixed Output**: No matter the size of the input data, the output (digest) is of fixed length.
- **Non-reversible**: You cannot retrieve the original data from its message digest (one-way function).
- **Deterministic**: The same input will always produce the same output.
- **Collision-resistant**: It should be hard to find two different inputs that produce the same digest.

MD4

MD5

Secure Hash Algorithms: SHA-1 and SHA-2

SHA-1

SHA-2

https://www.pvpsiddhartha.ac.in/dep_it/lecture%20notes/NTC/NTC-UNIT-5.pdf