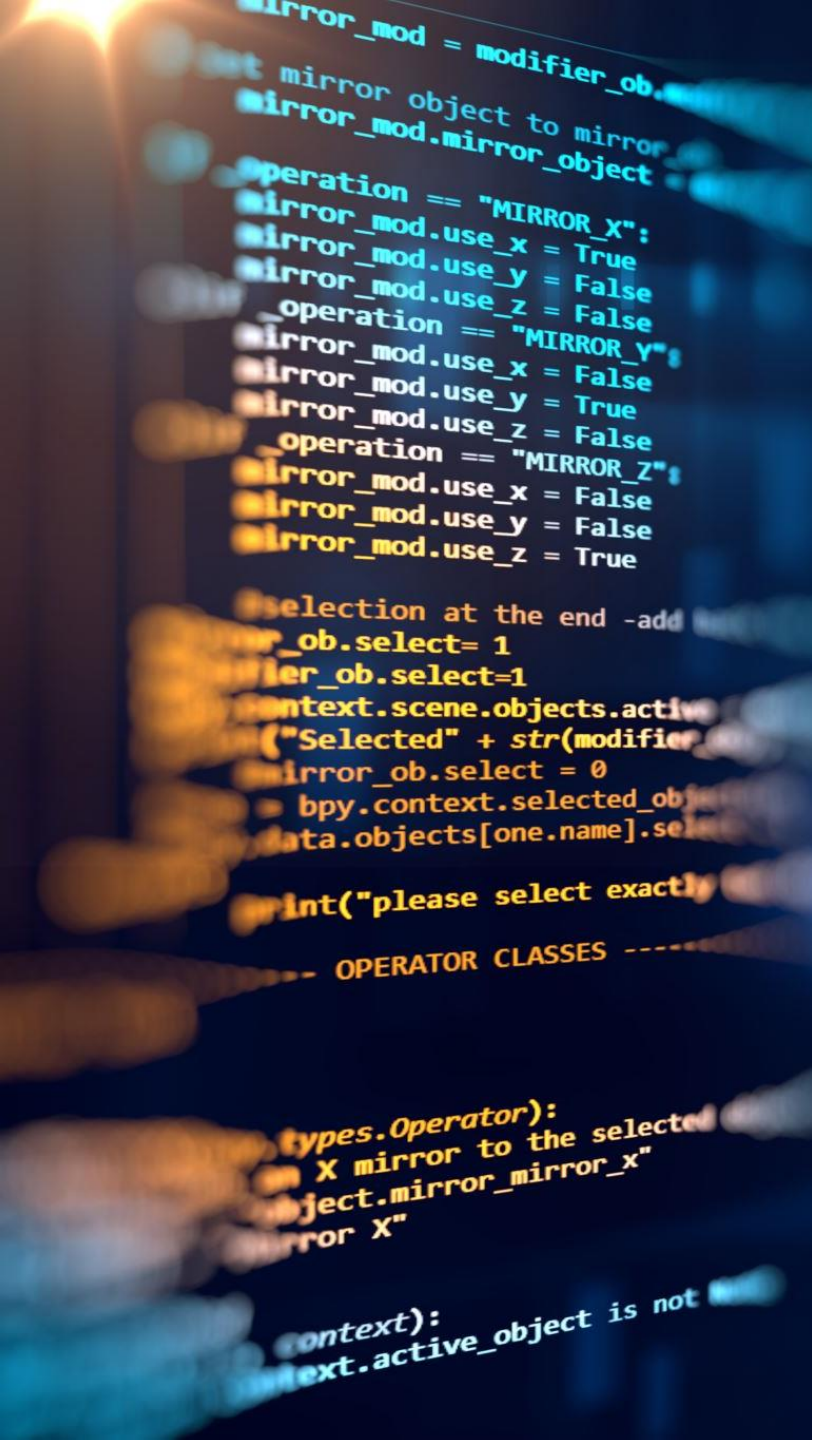
Computer Security

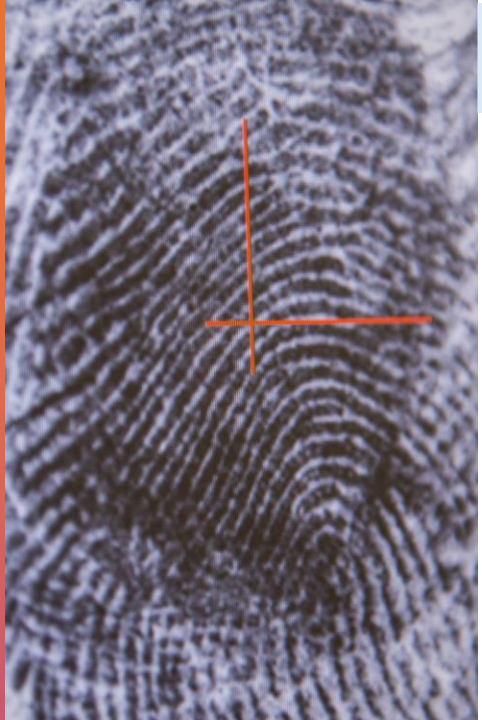
Chapter 08
Hash Functions



Outline

- Hash Functions
- Uses of Hash functions:
    - Modification Detection Code (MDC)
    - Storing passwords in database
    - Secure software updates
- Properties of Hash functions
- Message Authentication Code (MAC)
- Digital Signature
- Entity authentication

Motivation for Hash Algorithms



Confidentiality:
Protect transmission and storage of information from unauthorized access.

Integrity:
Protect transmission and storage of information from unauthorized change.

Availability:
Information (i.e., Transmitted & Stored Info.) is available when it is needed.

- A **hash function** is any function that can be used to map data of arbitrary size to fixed-size values.

- Also, the **cryptographic systems** provide secrecy, or confidentiality, but not **integrity**.
- There are occasions where we may not even need **secrecy** but instead must have **integrity**.
- For **example**, if we have a will, we want to be sure that it was not altered.
- If Alice needs to be sure that the contents of her document will not be changed, she can generate a **fingerprint of the document** (called **message digest** - i.e., message summary).

- **Document Fingerprint means Message Digest**

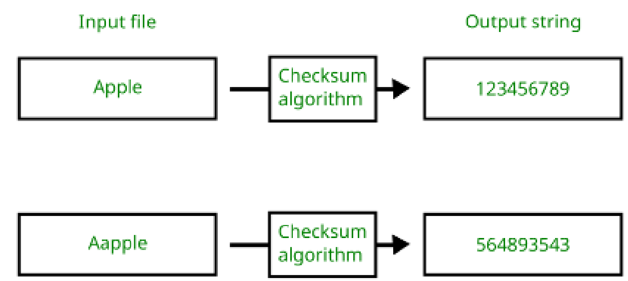
Hash Function

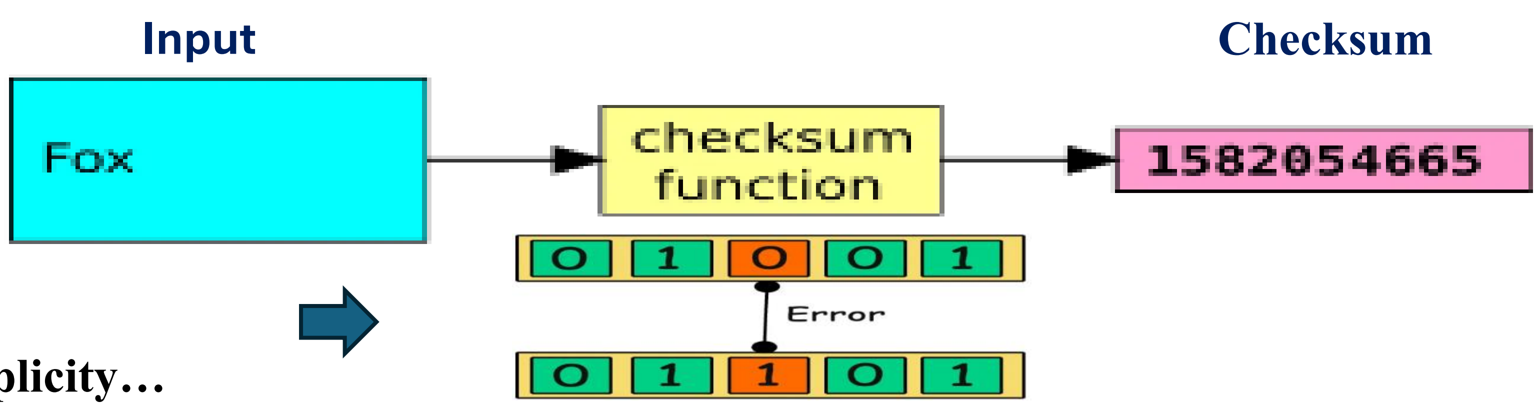
❑ Checksum…

- A **checksum** is a unique fingerprint attached to the data before it's transmitted.
- When the data arrives at the recipient's end, the fingerprint is recalculated to ensure it matches the original one.

- If the **checksum** of a piece of data matches the expected value, you can be confident that the data hasn't been modified or damaged.

- **A checksum** is a value that represents the number of bits in a transmission message.

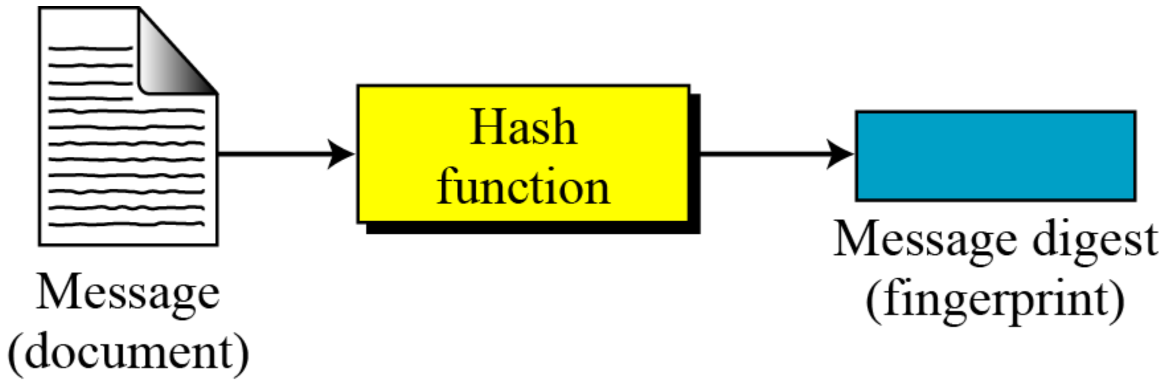




- **For simplicity…**

Message and Message Digest

- The **digest** is generated by passing the message through an algorithm called a **cryptographic hash function**.
- The function creates a **compressed image** of the message that can be used like a fingerprint.
- The digest is normally called a **modification detection code (MDC)**.



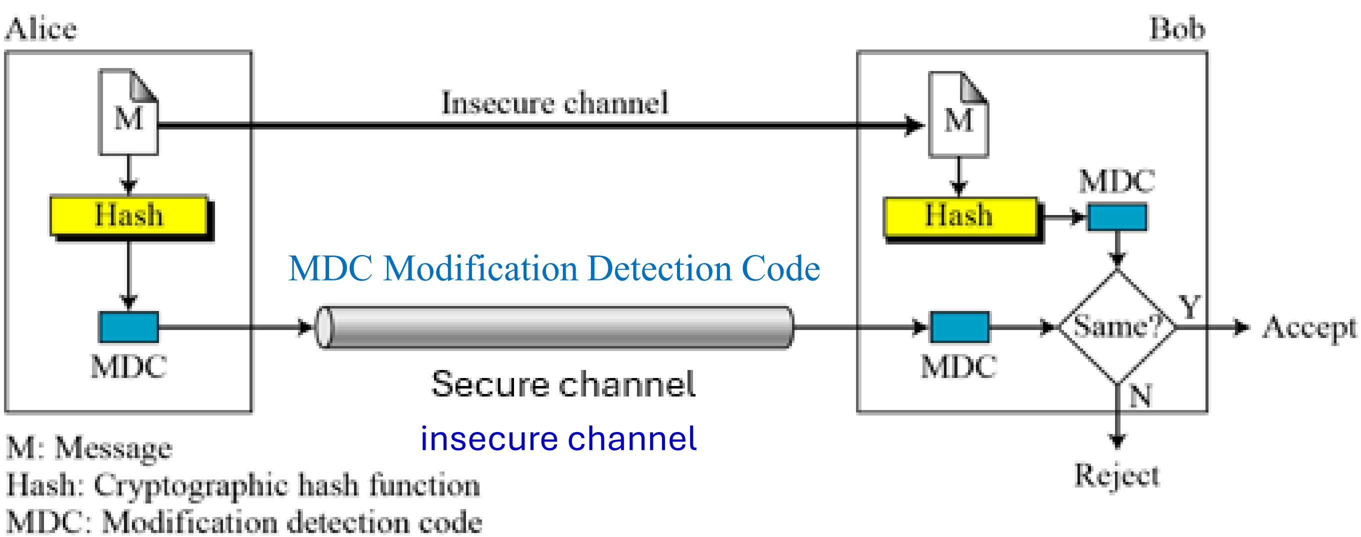
Message and digest

- **Applying an algorithm called cryptographic hash function on the message → O/P: Digest.**
- **Digest is called a modification detection code (MDC).**

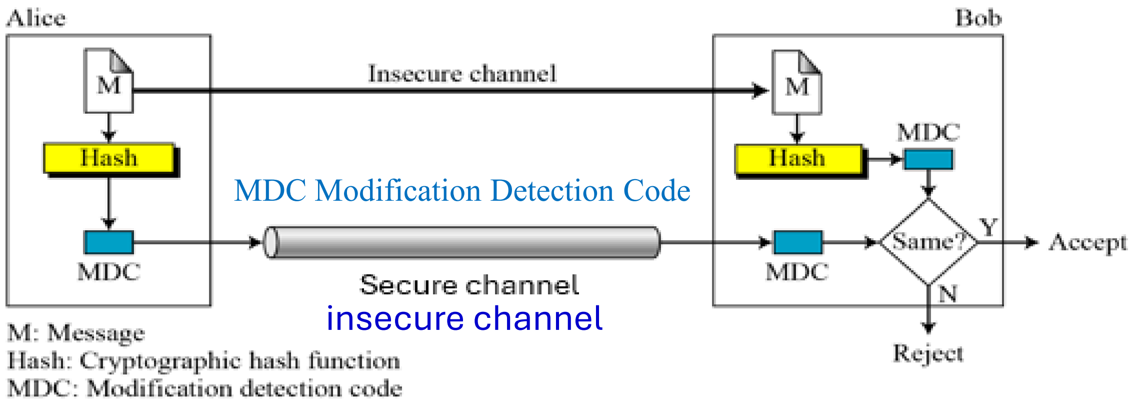
Modification Detection Code (MDC)

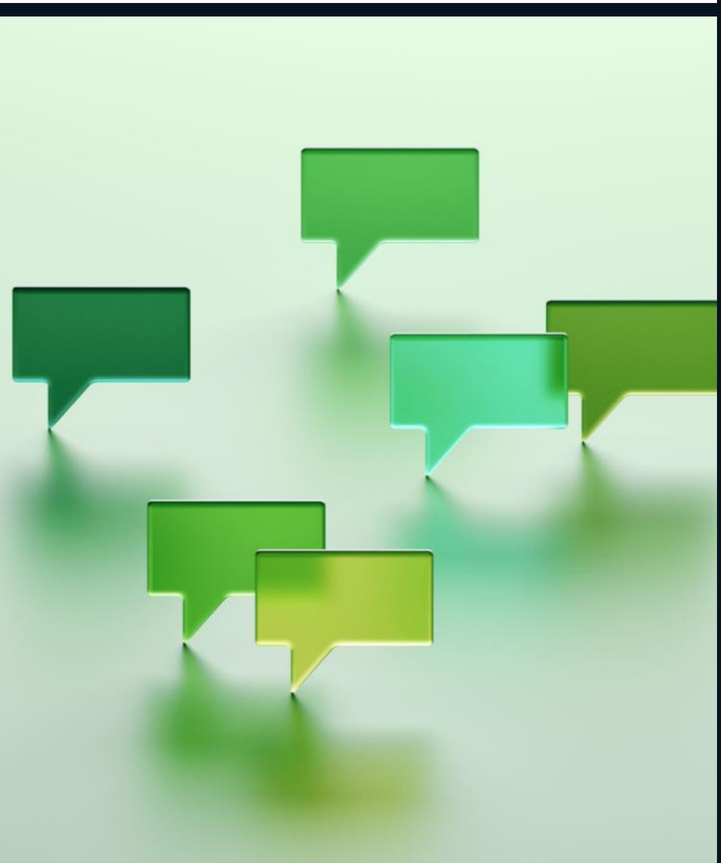
- If **Alice** needs to send a message to **Bob** and be sure that the message will not change during transmission, **Alice** sends **both** the message and the MDC to Bob.
  Modification Detection Code

- **Bob** can create a new MDC from the message and compare the received MDC and the new MDC. If they are the same, the message has not been changed.



Modification Detection Code (MDC)

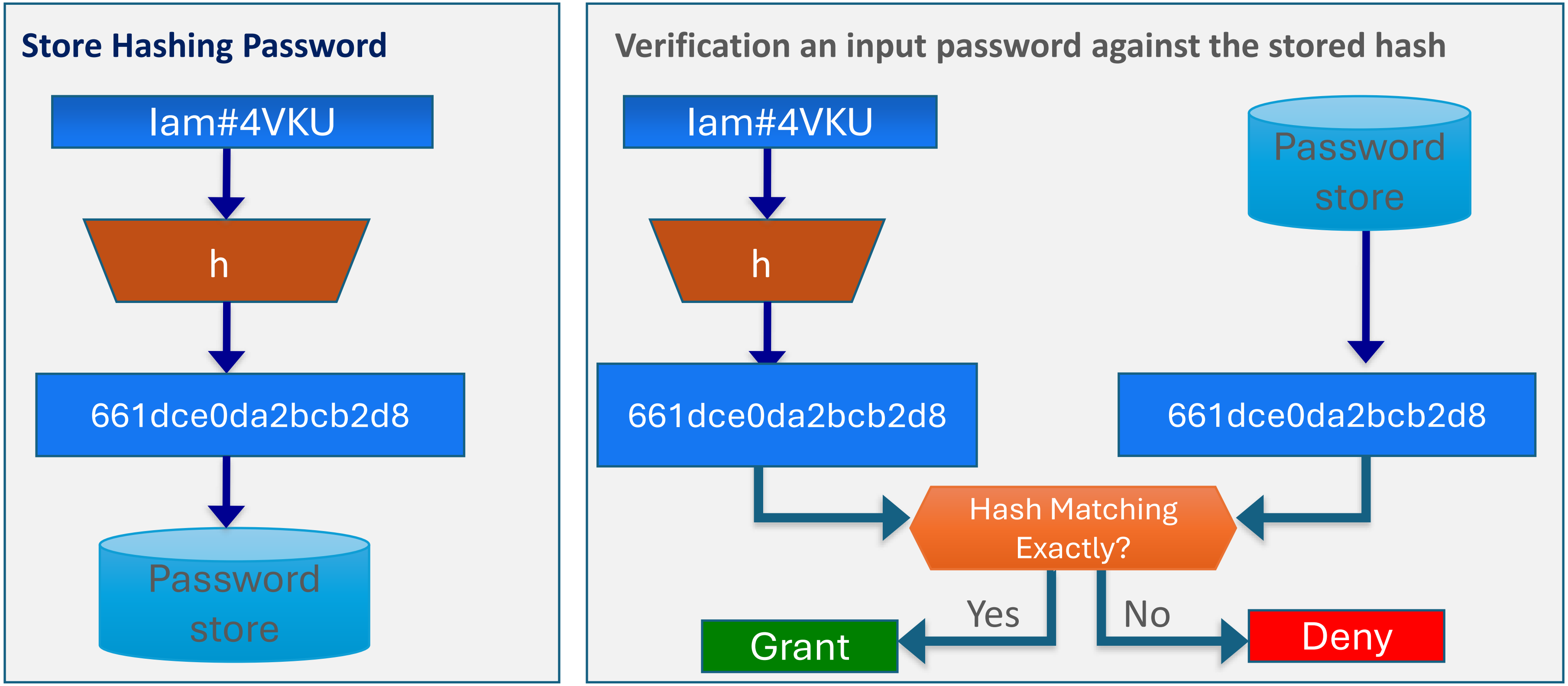


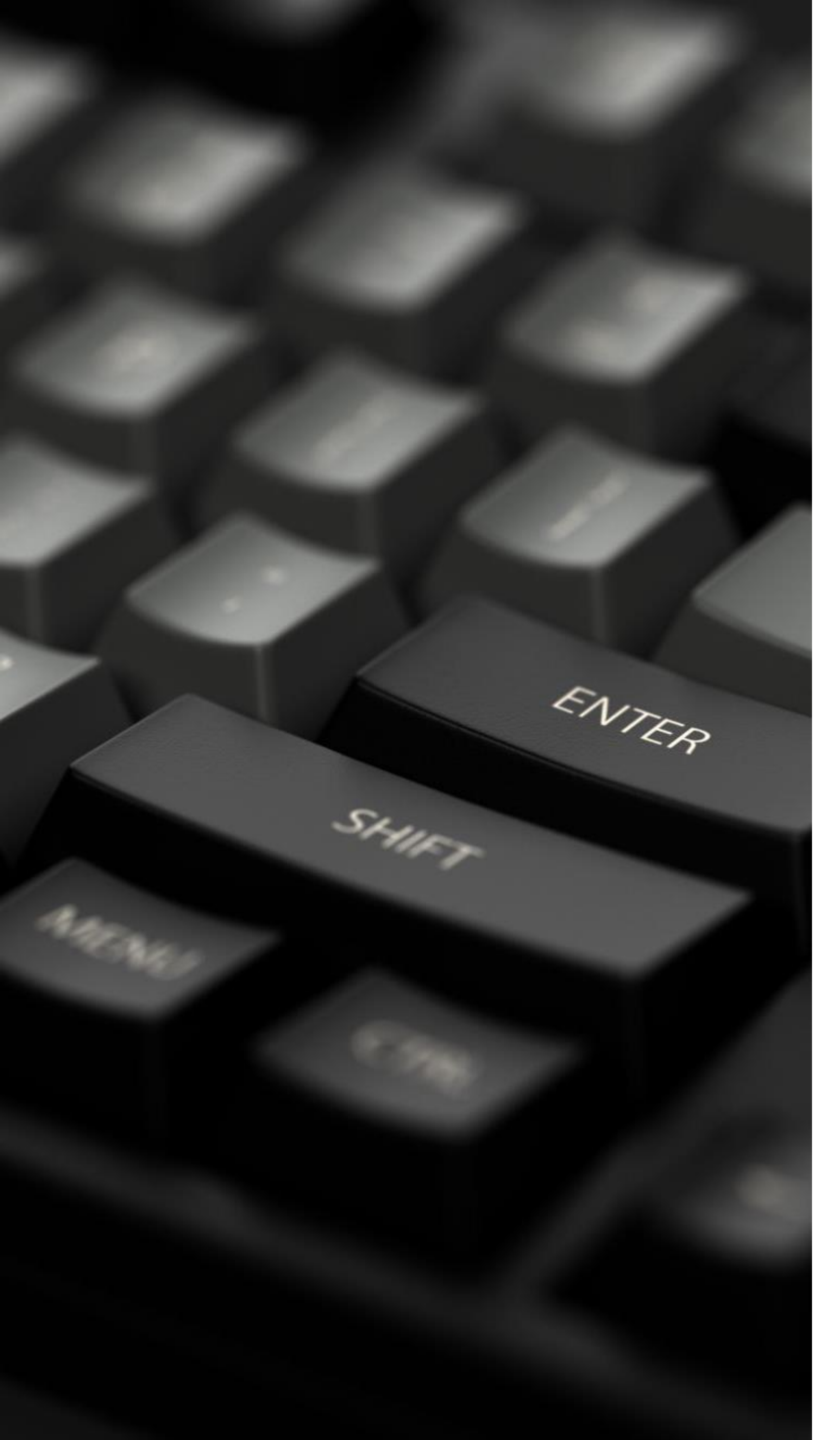


- If <u>both</u> the <u>message</u> and the <u>MDC</u> are <u>sent</u> through the <u>insecure channel</u>, **Eve** can <u>intercept</u> the <u>message</u>, <u>change it</u>, create a **new MDC** from the <u>message</u>, and <u>send both</u> to **Bob**. → **Bob** <u>never knows</u> that the message has come from **Eve**.
- Note that a safe or <u>secure channel</u> can mean a **trusted party**.
- The <u>MDC scheme</u> provide **integrity** but <u>not confidentiality</u> because the <u>message is unencrypted</u>.

Hash function for storing password

- If the hash value of a password is stored instead of the actual password, someone having access to the database will not be able to obtain the password.





Digest vs. Checksum

- The message **digest** is like the **checksum** used in computer networks for <u>error detection</u>.
  - However, it is <u>very possible to construct several messages that match the checksum</u>
- The goal with hash functions is that <u>each message</u> has a **unique digest** or <u>hash value</u>.
- Hence, an accidental or intentional change to the message will change the **<u>hash value</u>**
  - So, we can detect that the <u>message has been changed</u>.

Examples: Digest and Checksum

Digest (Hash) Example

A **digest** (or cryptographic hash) is designed to be unique, irreversible, and secure.

Example → Data: "hello"
Digest using SHA-256
(The SHA 256 algorithm, is one of the most widely used hash algorithms):
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Change even one letter (e.g., "Hello") and you get a completely different digest:
185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969

Checksum Example

A **checksum** is a simple method to detect errors in data by performing basic arithmetic.

Example →Data: 10, 20, 30, 40
Checksum (simple sum):
10 + 20 + 30 + 40 = 100

- If someone receives the data and recalculates the checksum as 100, they assume the data hasn't changed.
- But it's **not secure** – an attacker can change values to 5, 15, 35, 45 and still get 100.

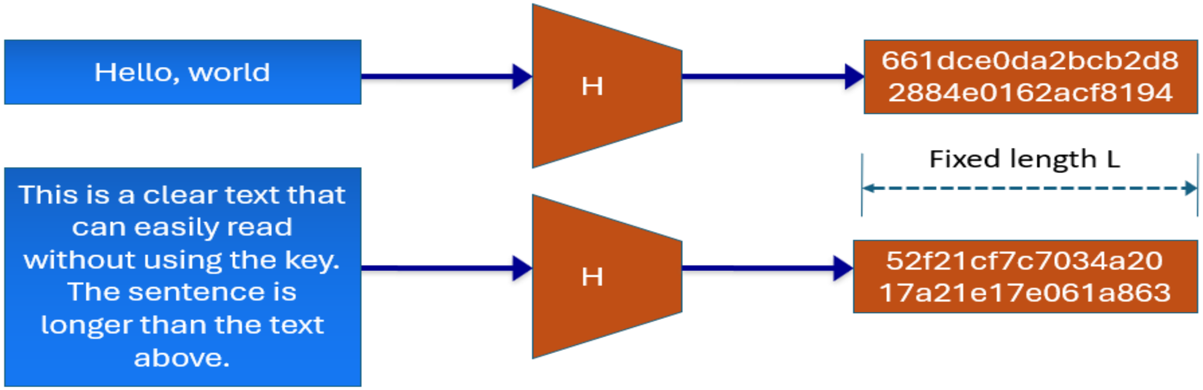
Digest and Checksum

Feature	Digest (Hash Value)	Checksum
Purpose	Data integrity, and security	Error detection
Method	Complex algorithms (e.g., SHA-256, MD5)	Simple arithmetic
Security	High, and hard to reverse or find collisions	Low, and easy to spoof
Output Length	Fixed length (e.g., 256 bits for SHA-256)	Short, and varies
Example Usage	Password storage, digital signatures, data integrity	File transfer error checking

Property 1: Fixed length

Properties of a cryptographic hash function

1. Arbitrary-length message to fixed-length digest
2. Preimage resistant (One-way property)
3. Second preimage resistant (Weak collision resistant)



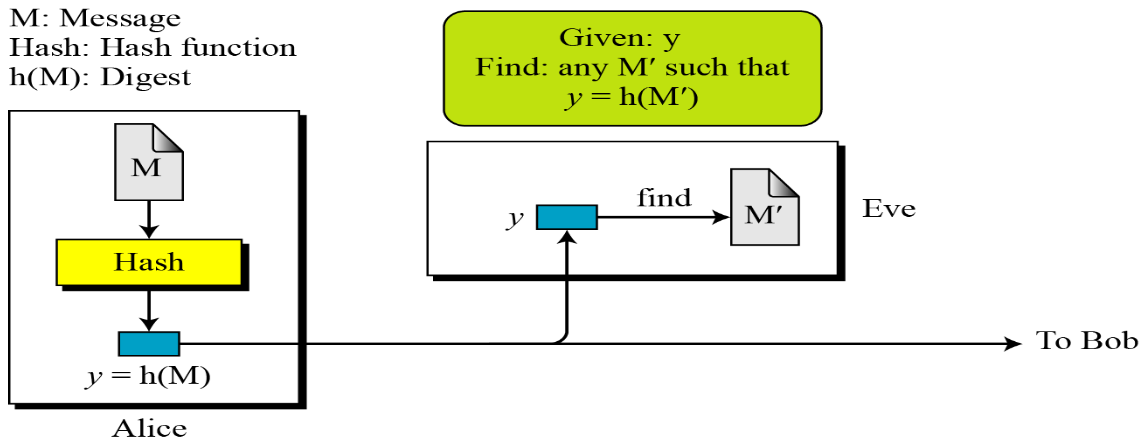
- **A cryptographic hash function takes an input of any size and produces a fixed-size output (digest).**

- **Input 1** (short): "A"
  SHA-256:
  559aead08264d5795d3a03b6...
  (64 hex characters = 256 bits)
- **Input 2** (long):
  "A" * 1,000 (1000 A's)
  SHA-256:
  41edece42d5ab0aa747f2d89... (also 256 bits)

✓ **No matter how long the input is, the digest is always the same length.**

Property 2: Preimage Resistance

- Given only a message digest, it must be very difficult to find the message or any message (or preimage) that generates that digest.
- That is, the **hash function** must be one-way.

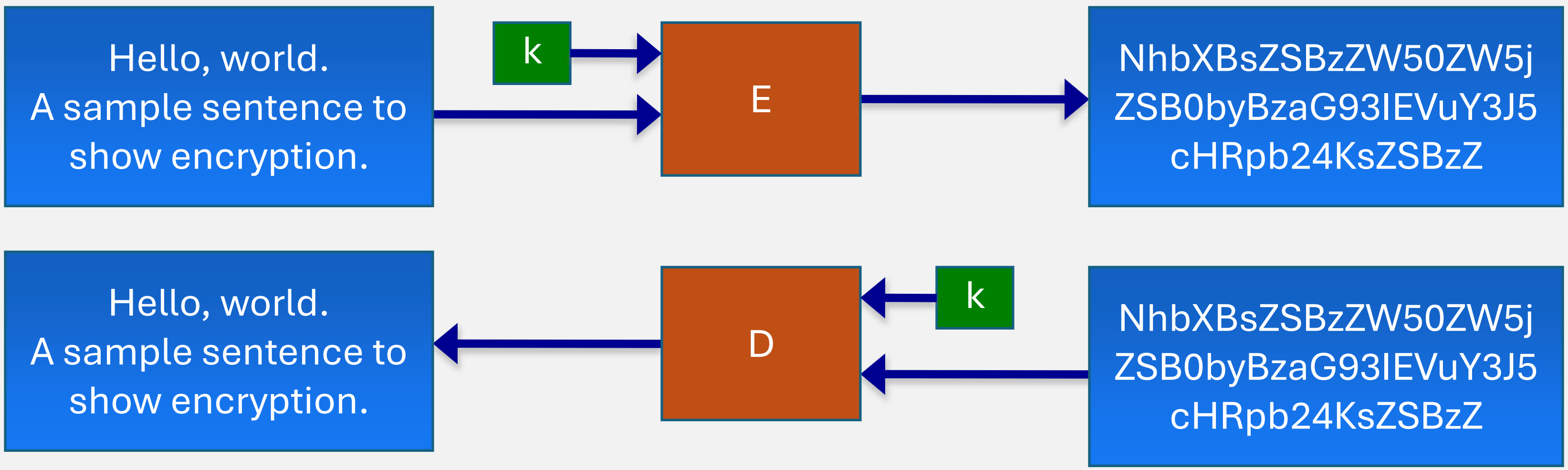


❑ **Example**

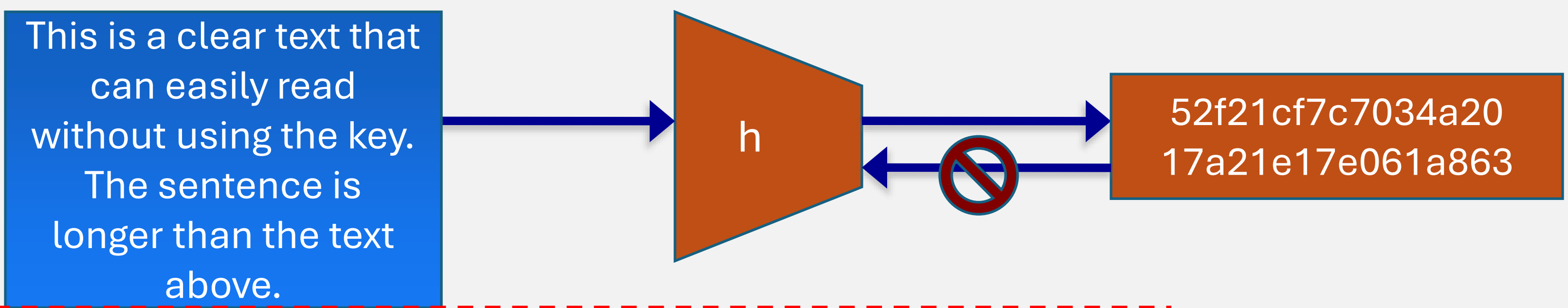
- Given a hash **H**, it should be **computationally infeasible** to find any message **M** such that: **Hash(M) = H**
- SHA-256("hello") = 2cf24dba5fb0a30e26e83b2a...
- You have the hash, but **you can't reverse it** to find that the input was "hello" without brute-force guessing.

✓ You **can't go backwards** from the the digest to find the original message.

Hashing vs. Encryption



- Encryption is two way, and requires a key to encrypt/decrypt

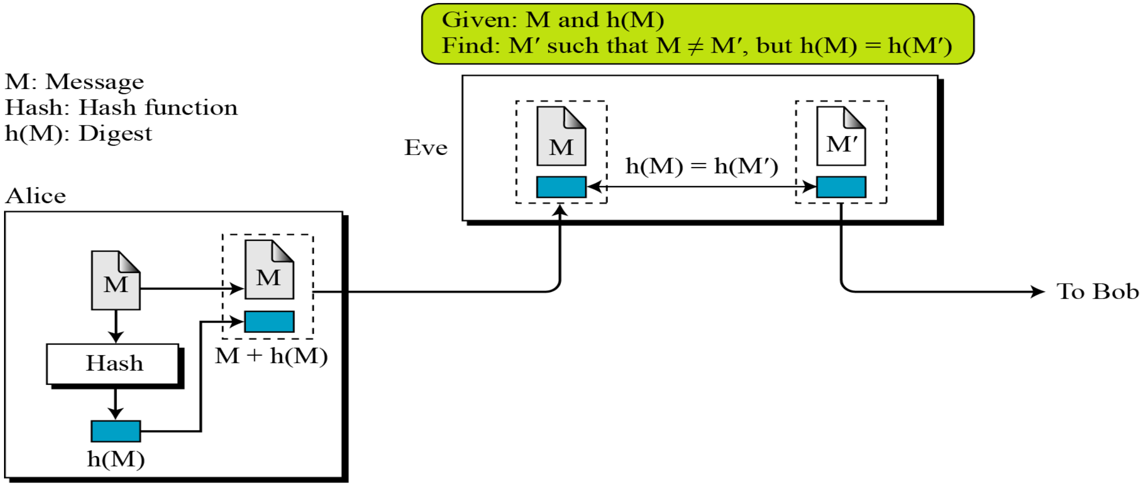


- Hashing is one-way. There is no 'de-hashing'

Property 3: Second Preimage Resistance

- Given one message and its digest, it must be very difficult to find another message (**preimage**) that has the same message digest.

- If the hash function is not second preimage resistant,
  It would be easy for Eve to forge (i.e., copy) new message and claim it is the one sent by Alice.



- 2nd preimage resistance property of a hash function:
  - It is computationally infeasible to find any second input that has the same output as a given input.

- Given an input M1, it should be hard to find a **different input** $M2 \neq M1$ such that:

$$\text{Hash}(M1) = \text{Hash}(M2)$$

- **Example**

- M1 = "Raya is an excellent engineer"
  SHA-256: 414fa339a6e2804a...

✓ Even though you know M1, you can't find M2 that matches its hash.

- Now try to find a **different string** M2 (e.g., "Rana is an excellent engineer") such that it produces **the exact same hash**. This should be practically impossible.

Example

- Assume that
  - the messages in a **hash function** are 6 bits long, and 2^6
  - the digests are only 4 bits long. 2^4
- Then the possible
  - number of digests is $2^4 = 16$, and the
  - possible number of messages is $2^6 = 64$.
- This means that at least **one digest** corresponds to **four** messages
  - (i.e., 1 digest $\rightarrow \frac{Message}{Digest} \rightarrow \frac{16}{4} \rightarrow 4\ messages$)
  - $\longrightarrow$ Corresponding to

- **1 digest → 4 messages**

Con. To Example

- In practice, a good **hash** tries to avoid **collisions**.
- But if the **input space** is bigger than the **output space** → some I/Ps must share the same O/P.
- **Example:**
- **Assume:**
  - Messages are 6-bit numbers (from 0 to 63)
  - Digests are 4-bit numbers (from 0 to 15)
  - If we have a simple **fake hash function** that works like this:
    - Hash(message) = message MOD 16
    - This function reduces any 6-bit message to a 4-bit digest.

❑ **Calculation of e a few hashes**

Message (6-bit)	Hash = msg MOD 16
0	0
1	1
2	2
...	...
15	15
16	0 ← collision with 0
17	1 ← collision with 1
18	2 ← collision with 2
...	...
31	15 ← collision with 15
32	0 ← another collision with 0
48	0 ← again!

Con. To Example

Message (6-bit)	Hash = msg MOD 16
0	0
1	1
2	2
...	...
15	15
16	0 ← collision with 0
17	1 ← collision with 1
18	2 ← collision with 2
...	...
31	15 ← collision with 15
32	0 ← another collision with 0
48	0 ← again!

Message **Digest**

- $0 \rightarrow 0$
- $16 \rightarrow 0$
- $32 \rightarrow 0$
- $48 \rightarrow 0$

- **These messages all produce the same digest.**

✓ **The digest 0 maps to 4 different messages!**

- **Each message** always <u>produces</u> **one digest.** The hash is deterministic.
- **But different messages** can <u>produce</u> the **same digest** (a collision).
  - That's the nature of **hash functions** with **small digest sizes**.
  - That's don't notice it with big hashes like SHA-256 because the chance of collision is extremely low.

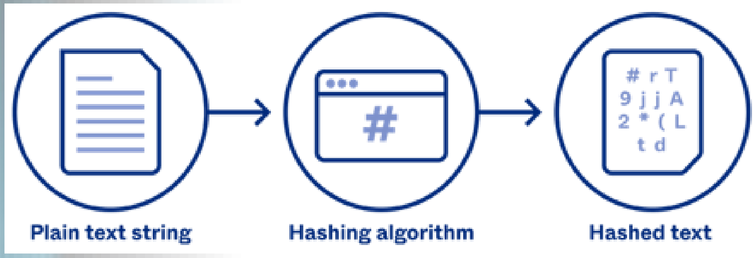
Conclusion…

- A **message** always maps to **one digest**.
- But a **digest** can map to **many messages**.
  - IN the above example, **64 messages** are squeezed into **16 digests**, so, **some digest values must be shared by multiple messages**.

Hash function Schemes

- There are several well-known methods to build cryptographic hash function.

- How Hashing Algorithm work.



Versions of Hash Functions

- A **cryptographic hash function** takes a message of arbitrary length and creates a **message digest of fixed length**.

	MD5	SHA-1	SHA-256	SHA-384	SHA-512
Digest size (bits)	128	160	256	384	**512**
Message size (bits)	2^{64}-1	2^{64}-1	2^{64}-1	2^{128}-1	**2^{128}-1**
Block size (bits)	512	512	512	1024	**1024**
Year	1992	1995	2001	2001	**2001**

Insecure
Full collision found

→ more secure

Hash Functions Family

MD (Message Digest)

Designed by **Ron Rivest**

Family: MD2, MD4, MD5

SHA (Secure Hash Algorithm)

Designed by **NIST**

Family: SHA-0, SHA-1, and SHA-2

- SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
- SHA-3: New standard in competition

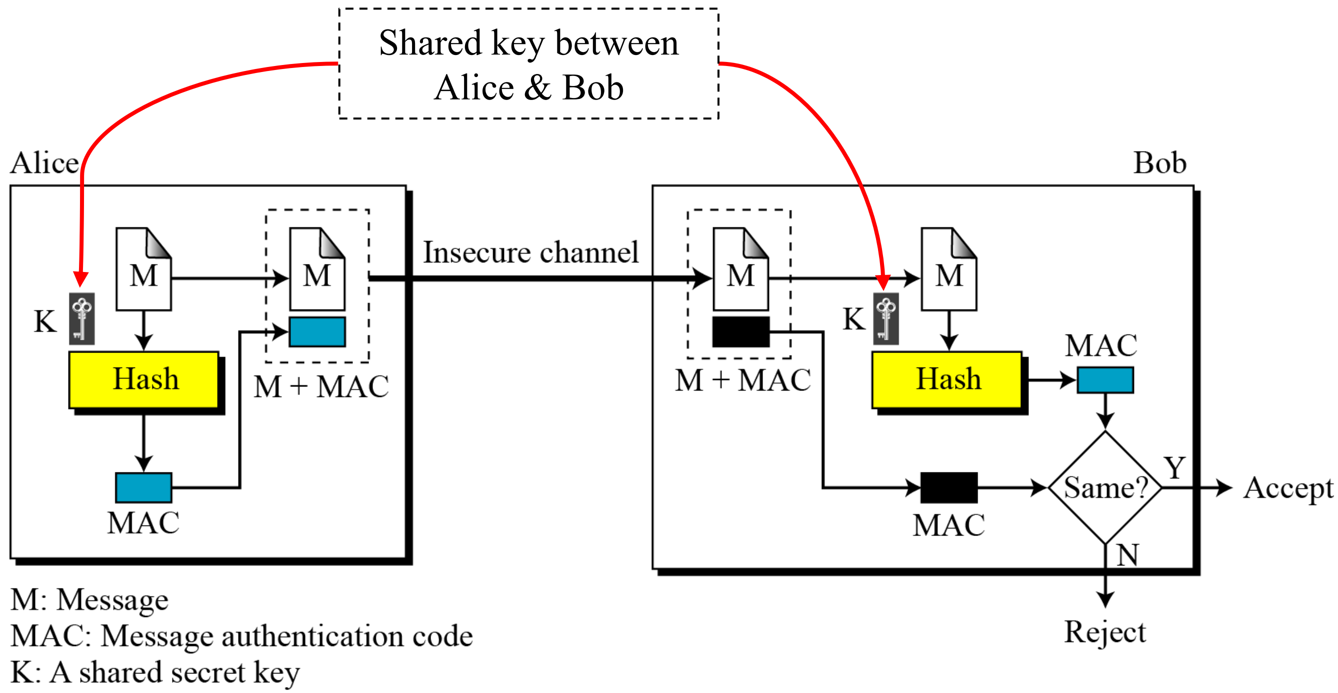
❑ **Please try this hashing tools**
https://www.digitalvolcano.co.uk/hash.html

Message Authentication Code (MAC)

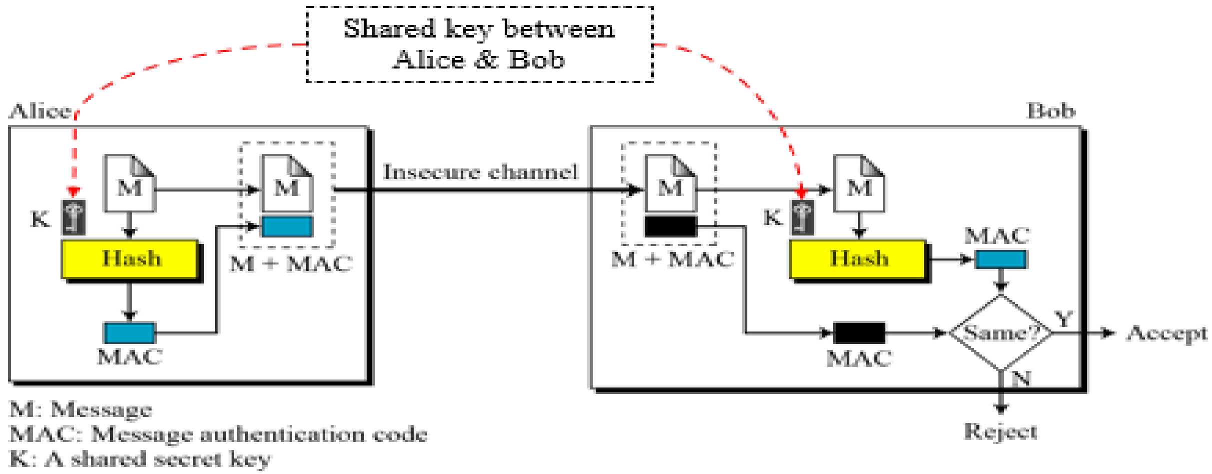
MDC Modification Detection Code

- MDC does not **authenticate** the sender of the message → **Yes** → **Just integrity Message**
- To ensure both message **integrity** → and data origin **authentication**, The **MDC** is modified to message authentication code (MAC), where the **secret key** is used between **Alice** and **Bob** and which **Eve** does not know.
- **In this case, Alice** uses a hash function to create a **MAC** from the concatenation of the **key** and the **message**, h (K|M). She sends the message and the MAC to **Bob** over the insecure channel.

- Remind You!
  - Applying cryptographic hash function algorithm on the message → O/P: Digest, which is called a modification detection code (MDC).



Message Authentication Code (MAC)



Message Authentication Code

- Bob separates the **message** from the **MAC**.
  →He then makes a new MAC from the concatenation of the message and the secret key.
- If the **two MACs match**, the message is authentic and has not been modified by an adversary.
- **Note:** in this case, both message and the MAC can be sent on the same insecure channel.
  - **Eve** can see the message, but she cannot forge a new message to replace it because **Eve** does not possess the secret key between **Alice** and **Bob**.
  - Eve is unable to create the same MAC as Alice did.

MAC doesn't provide Non-Repudiation

- **MAC** technique does not provide a non-repudiation service.

⇩

- The **sender** could deny having sent the message and claim that the receiver forged it, ▶
  as it is **impossible** to determine which of the two parties computed the MAC because both have the shared secret key.

⇩

- **This limitation** can be overcome by using the **public key** based **digital signature**.

Digital Signature

- **Digital signatures** are one of the most important cryptographic tools widely used today.

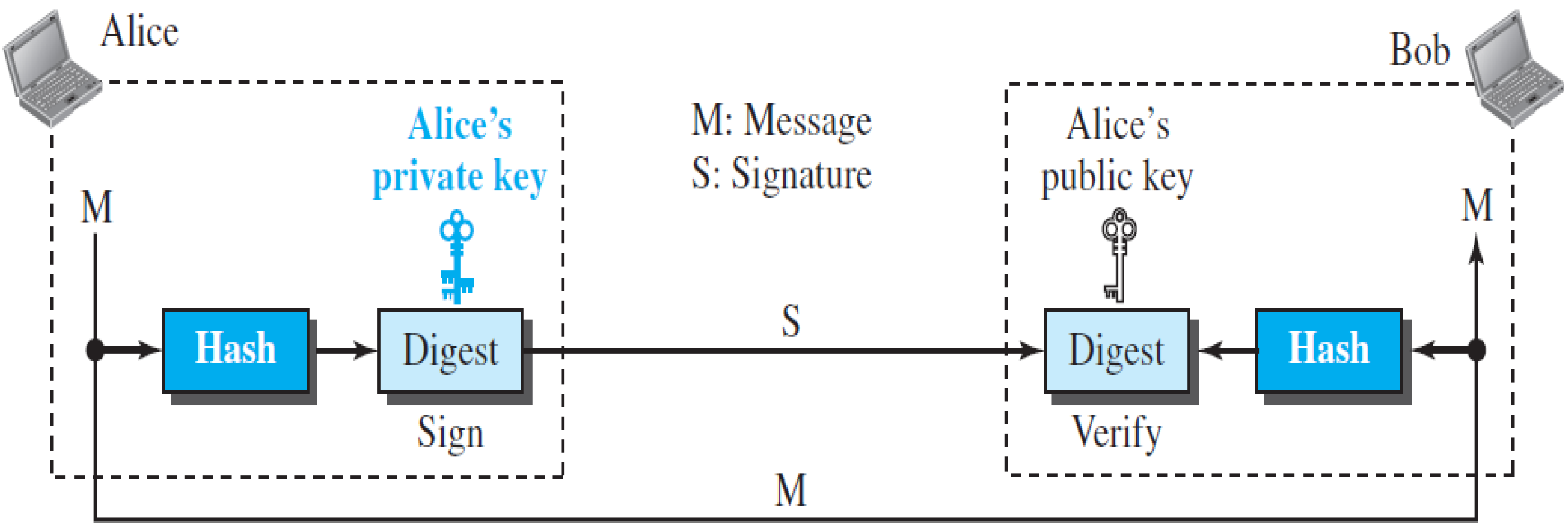
- Applications for digital signatures:
  - digital certificates for secure e-commerce
  - legal signing of contracts

- **Digital signature** with key establishment over insecure channels, **they form the most two important instances for public-key cryptography.**

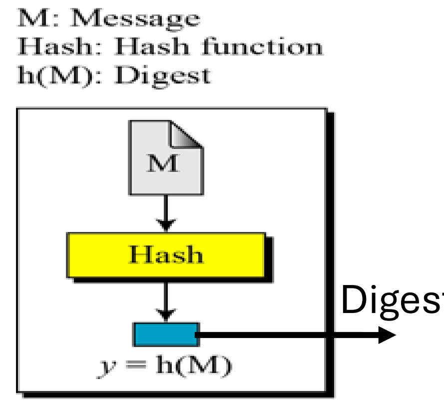
Digital Signature

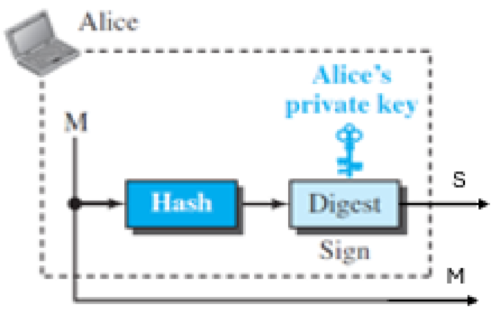
- When **Alice** sends a message to **Bob**, Bob needs to verify the:
  - Integrity of the message
  - Authenticity of the sender
    (i.e., he needs to be sure that the message comes from Alice and not Eve).

- **A digital signature achieves both goals.**

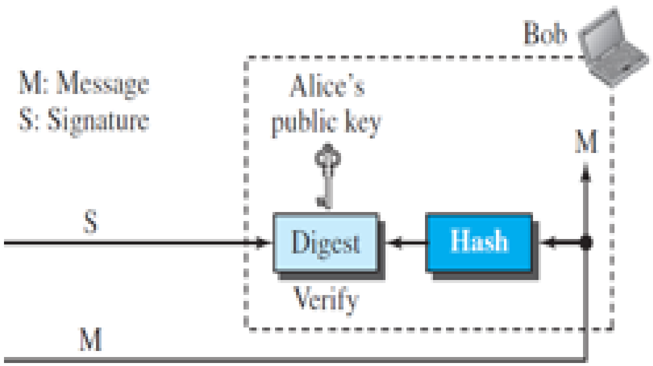


- **At Alice's site:**
  - Message is hashed to creates the digest.
  - Alice signs (encrypt) the digest using her private key (digital signature). No one else has this signature.
  - The signed digest is sent along with the message itself.





- **At Bob's site:**
  - The **same public hash function** is used to create the **digest of received message**.
  - The received signed digest is decrypted (verified) using Alice's public key.
  - If both **digests** are the same, the message is not modified, its sender is authenticated, and so, it is **accepted**; otherwise, it is **rejected**.
  - **Note:**
    When a document is signed, anyone, including Bob, can verify it because everyone has access to Alice's public key.



Notes

Message Authentication Code

- Digital Signature is the **public-key** equivalent of **MAC** which uses secret-key cryptography. Both achieve message authentication.

- In **digital signature**, the signer signs with her **private key**; the receiver verifies with the **signer's public key**.

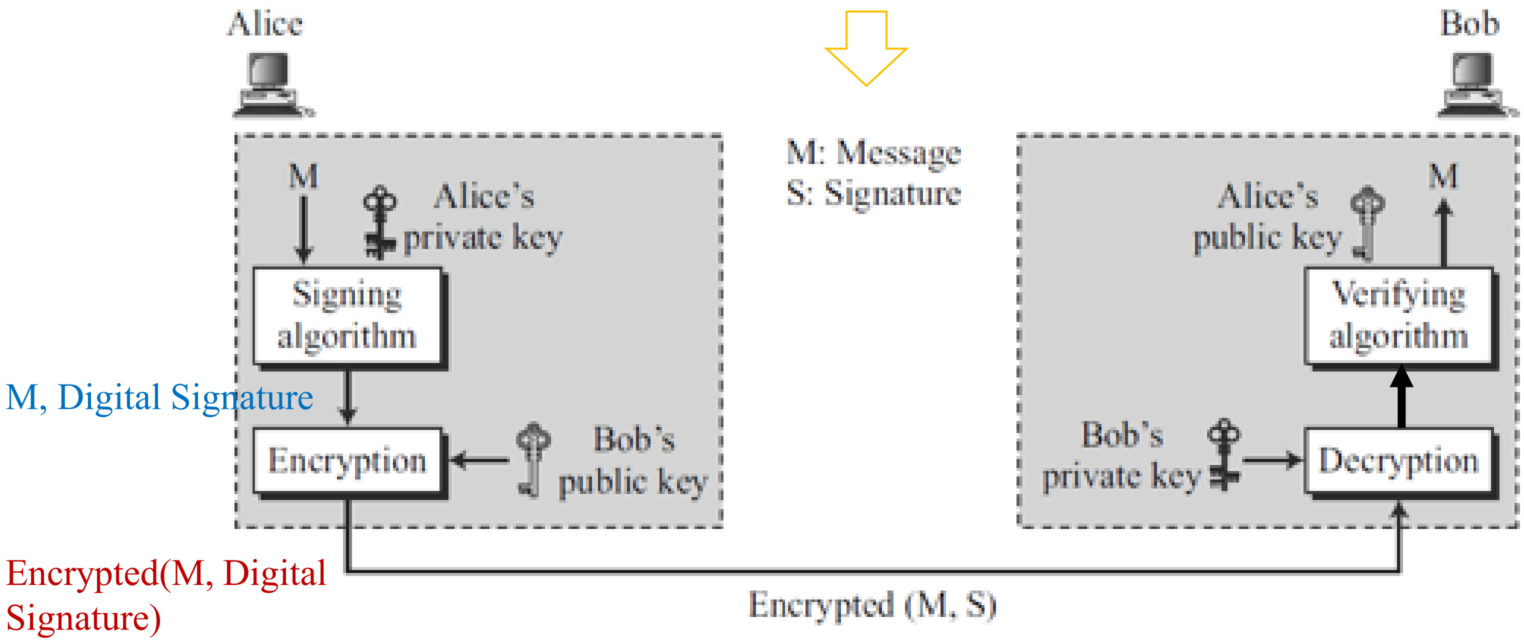
- A **cryptosystem** uses the private and public keys of the **receiver**;
- a **digital signature** uses the private and public keys of the **sender**.

- **Signing the digest** is much **more efficient** than **signing the mess**age because the digest is much shorter than the message.

Adding confidentiality to digital signature

- A **digital signature** provides message **authentication** and **integrity**, but not confidentiality.

- To achieve **confidentiality**, the message and the **signature** can be encrypted using Bob's public key as shown below.



- **Adding confidentiality to a digital signature scheme.**

Entity Authentication

- Message authentication (or data-origin authentication) is required when, e.g., an **e-mail** is sent from Alice → to Bob.
  - When Bob authenticates the message, **Alice** may or may not be online.

- On the other hand, **entity authentication** lets one party prove the identity of another party in real time.
  - ► An **entity** can be a person, a client, or a server.
  - This is required,
    - e.g., when **Alice** gets cash from an automatic teller machine ATM.
    - ► **Alice needs to be online**.

- **Techniques** for **entity authentication** include:
  - **Passwords**
  - **Challenge Réponse Questions**

Any Questions!