

Introduction to Cryptography

Lecture 7: Cryptographic Hash Functions

CYB 650

Introduction to Cryptography

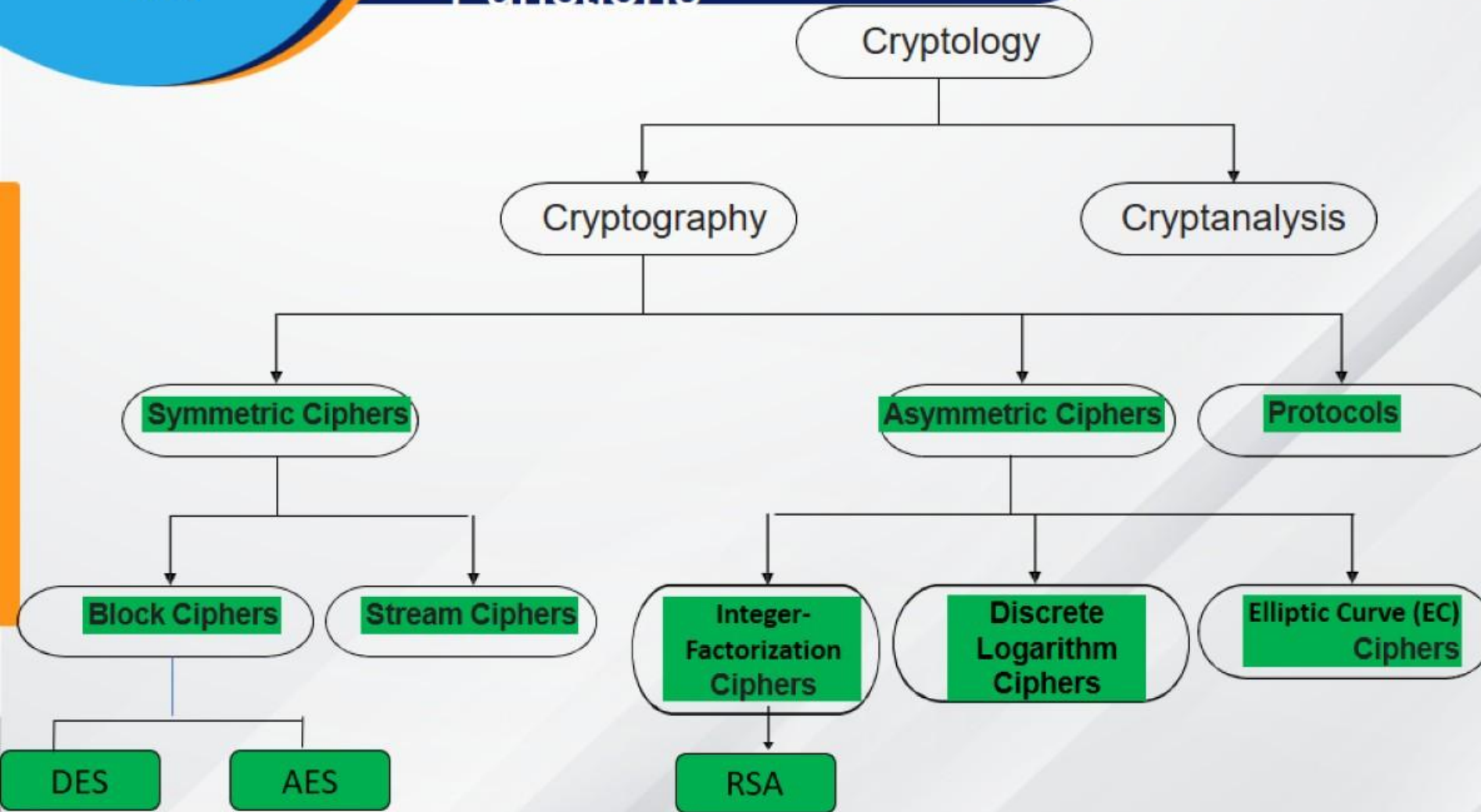
Presented by

Dr. Mohammad Alkhatib

- Introduction to Hash Functions
- Hash Functions Properties
- Security Requirements for Cryptographic Hash Functions
- Overview of Hash Functions
- The Secure Hash Function Algorithm(SHA)

Module 03

Cryptography Hash Functions



- ❑ Hash functions are an important cryptographic primitive and are widely used in protocols.
- ❑ Hash functions are an essential part of digital signature schemes, message authentication codes, key derivation, ... etc.

- ❑ A hash function H accepts a variable-length block of data as input and produces a fixed-size hash value. $h = H(M)$.
- ❑ In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in (M) results, with high probability, in a change to the hash code

❑ A **cryptographic hash function** is an algorithm for which it is computationally infeasible to find either:

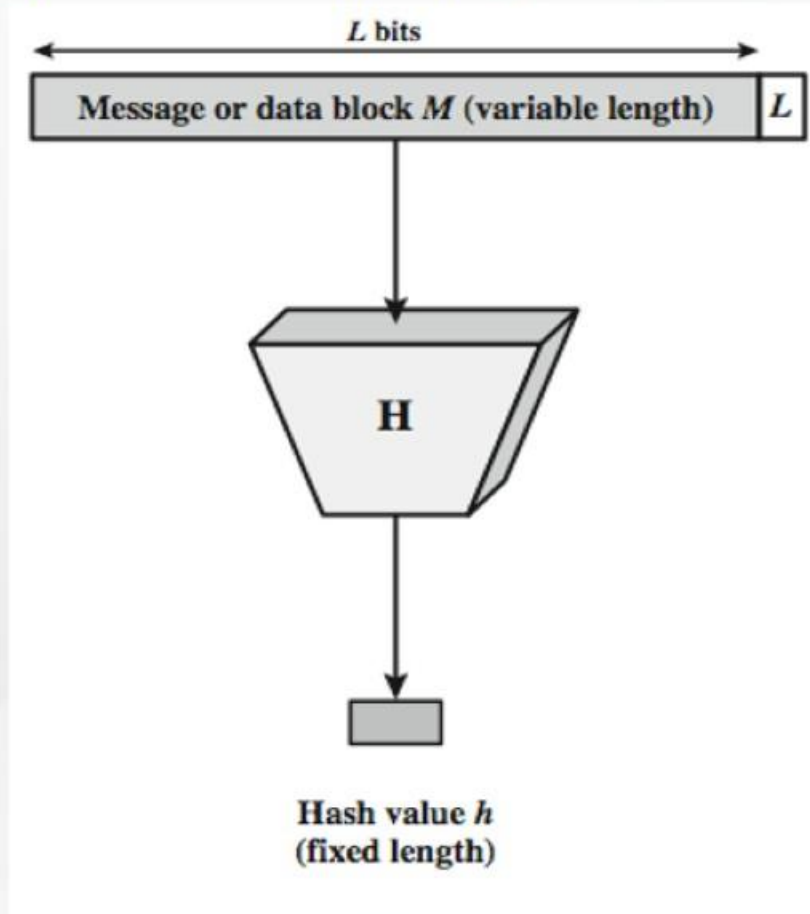
- i. a data object that maps to a pre-specified hash result (the one-way property) or
- ii. two data objects that map to the same hash result (the collision-free property).

❑ Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Module 03

Introduction to Hash Functions

- ❑ The general operation of a cryptographic hash function.

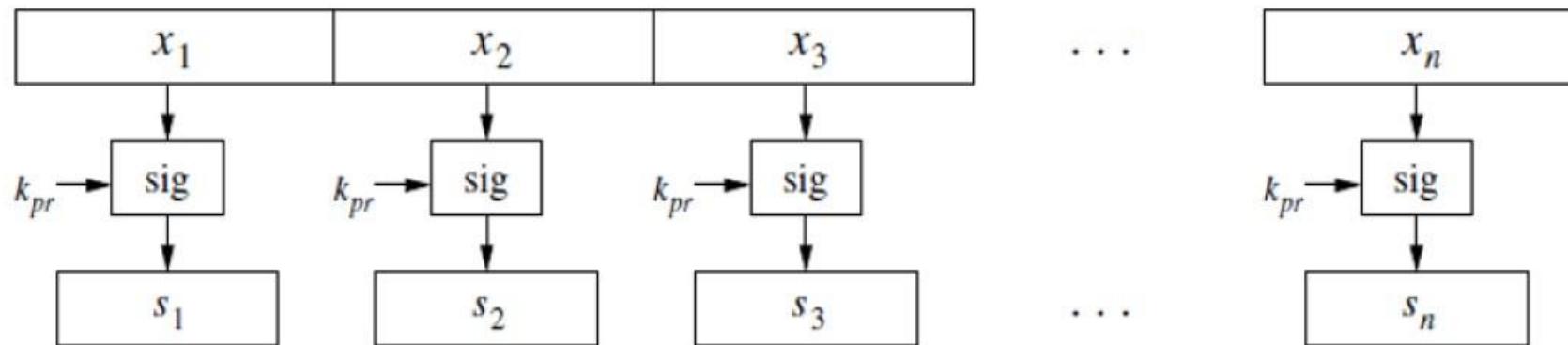


- ❑ Hash functions play important role in the practical use of digital signatures
- ❑ For all schemes, the length of the plaintext is limited. For instance, in the case of RSA, the message cannot be larger than the modulus, which is in practice often between 1034 and 3072-bits long. Remember this translates into only 128–384 bytes; most emails are longer than that.

❑ **Question:** How are we going to efficiently compute signatures of large messages?

❑ **An intuitive approach:** Utilizing the block ciphers approach: Divide the message x into blocks x_i of size less than the allowed input size of the signature algorithm, and sign each block separately, as depicted in the following

Figure



➤ Problem 1: High Computational Load

Digital signatures are based on computationally intensive asymmetric operations such as modular exponentiations of large integers. Even if a single operation consumes a small amount of time and energy, the signatures of large messages, e.g., email attachments or multimedia files, would take too long on current computers.

Furthermore, not only does the signer have to compute the signature, but the verifier also has to spend a similar amount of time and energy to verify the signature.

➤ Problem 2: Message Overhead

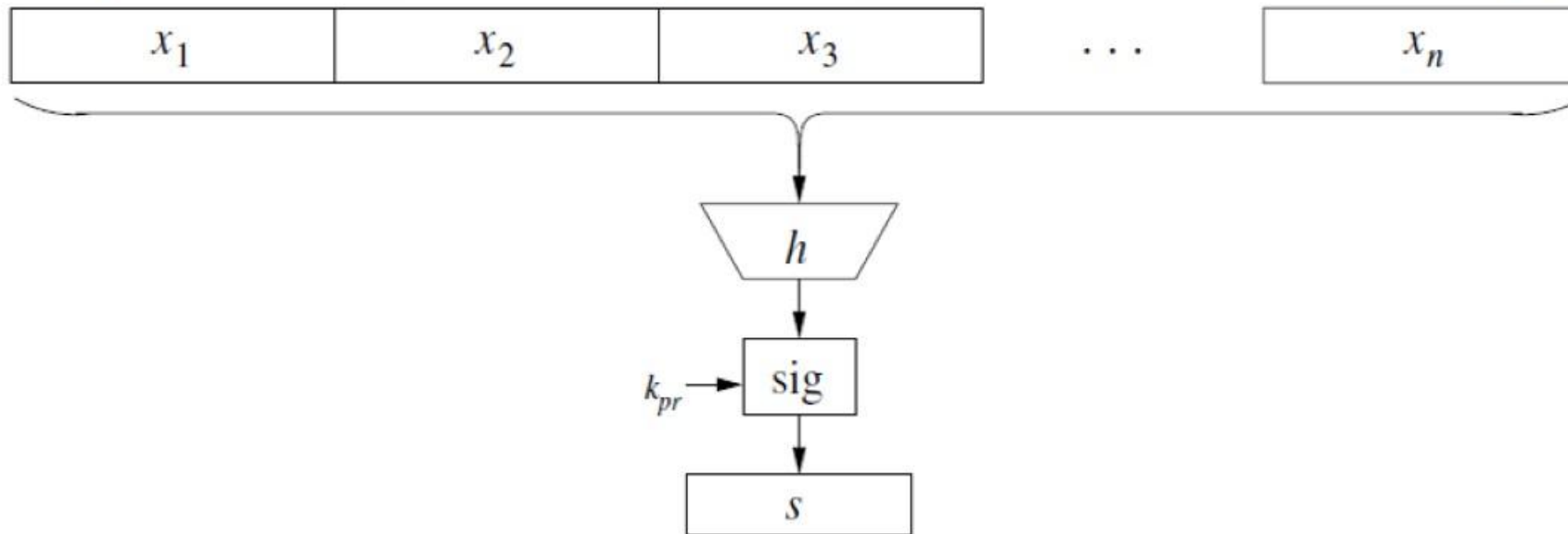
This approach doubles the message overhead because not only must the message be sent but also the signature, which is of the same length in this case. For instance, a 1-MB file must yield an RSA signature of length 1 MB, so that a total of 2 MB must be transmitted.

Problem 3: Security Limitations

This is the most serious problem if we attempt to sign a long message by signing a sequence of message blocks individually. The approach shown in the previous figure leads immediately to new attacks: For instance, Oscar could remove individual messages and the corresponding signatures, or he could reorder messages and signatures. Even though an attacker cannot perform manipulations within an individual block, we do not have protection for the whole message.

Solution: Hash Functions

For performance as well as for security reasons we would like to have one short signature for a message of arbitrary length. The solution to this problem is hash functions. If we had a hash function that somehow computes a fingerprint of the message x , we could perform the signature operation as shown in the following figure



Basic Digital Signature Protocol

Alice

Bob

← $k_{pub,B}$

generate $k_{pr,B}, k_{pub,B}$

publish public key

sign message:

$$s = \text{sig}_{k_{pr}}(x)$$

← (x, s)

send message + signature

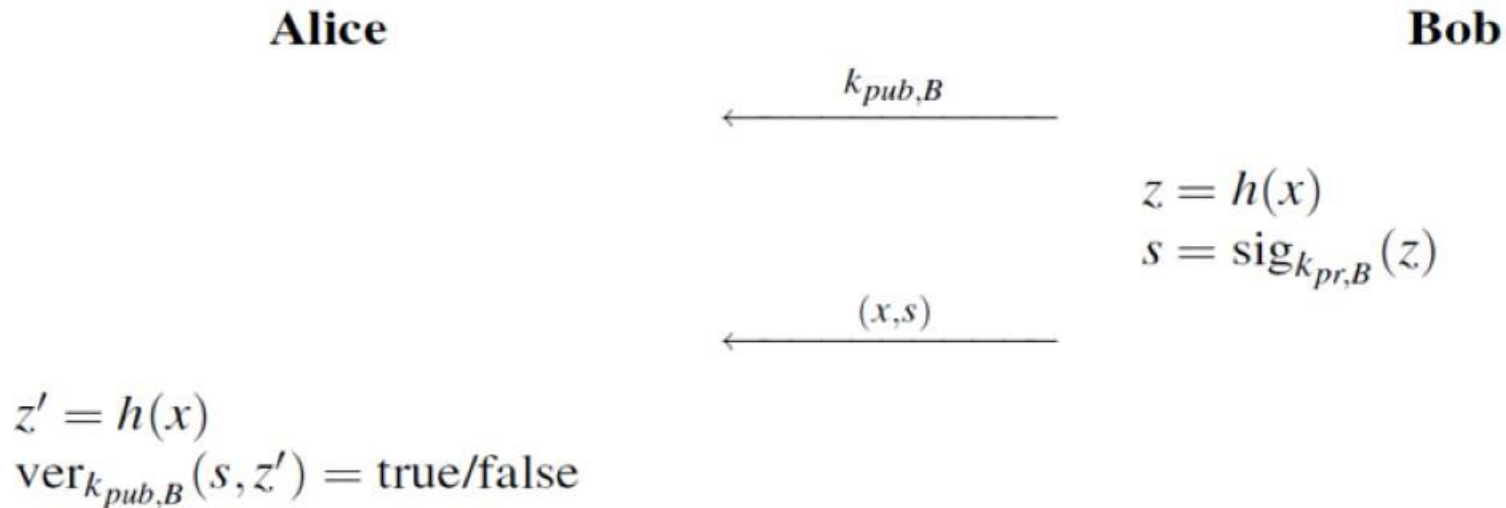
verify signature:

$$\text{ver}_{k_{pr,B}}(x, s) = \text{true/false}$$

Module 03

Basic Protocol for Digital Signatures with Hash Functions

□ Assuming we possess such a hash function, we now describe a basic protocol for a digital signature scheme with a hash function. Bob wants to send a digitally signed message to Alice



Module 03

Basic Protocol for Digital Signatures with Hash Functions

- ❑ Bob computes the hash of the message x and signs the hash value z with his private key $k_{pr,B}$.
- ❑ On the receiving side, Alice computes the hash value z of the received message x .
- ❑ Then, she verifies the signature s with Bob's public key $k_{pub,B}$.

- ❑ Note that both the signature generation and the verification operate on the hash value z rather than on the message itself. Hence, the hash value represents the message.
- ❑ The hash is sometimes referred to as the message digest or the **fingerprint** of the message.

- ❑ The function h should be computationally efficient. Even if we hash large messages in the range of, hundreds of megabytes, it should be relatively fast to compute.
- ❑ The output of a hash function should be of fixed length and independent of the input length. Practical hash functions have output lengths between 128–512 bits.
- ❑ Finally, the computed fingerprint should be highly sensitive to all input bits. That means even if we make minor modifications to the input x , the fingerprint should look very different.

message

message digest

Alice was beginning to get very tired of
sitting by her sister on the bank,
and having nothing to do.

h

DFDC349A

I am not a crook.

h

FB93E283

I am not a cook.

h

A3F4439B

Hash functions properties are illustrated in the above figure

Properties of Hash Functions

1. **Arbitrary message size** $h(x)$ can be applied to messages x of any size.
2. **Fixed output length** $h(x)$ produces a hash value z of fixed length.
3. **Efficiency** $h(x)$ is relatively easy to compute.
4. **Preimage resistance** For a given output z , it is impossible to find any input x such that $h(x) = z$, i.e, $h(x)$ is one-way.
5. **Second preimage resistance** Given x_1 , and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$.
6. **Collision resistance** It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.

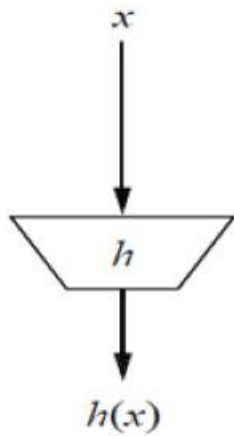
□ Note that the first three are practical requirements, whereas the last three relate to the security of hash functions

Module 03

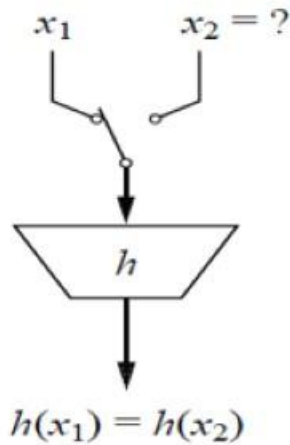
Security Requirements for Cryptographic Hash Functions

There are three central properties which hash functions need to possess in order to be secure:

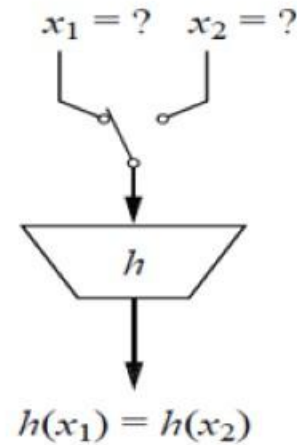
1. preimage resistance (or one-wayness)
2. second preimage resistance (or weak collision resistance)
3. collision resistance (or strong collision resistance)



preimage resistance



second preimage
resistance



collision resistance

❑ Hash functions need to be one-way: Given a hash output z it must be computationally infeasible to find an input message x such that $z = h(x)$. In other words, given a fingerprint, we cannot derive a matching message.

❑ Assumes that Bob transmits the pair: $(e_k(x), \text{sig}_{k_{pr,B}}(z))$.

❑ Here, $ek()$ is a symmetric cipher, e.g., AES, with some symmetric key shared by Alice and Bob. Let's assume Bob uses an RSA digital signature, where the signature is computed as: $s = \text{sig}_{k_{pr,B}}(z) \equiv z^d \pmod{n}$

❑ The attacker Oscar can use Bob's public key to compute the signature z as follows:

$$s^e \equiv z \pmod{n}$$

❑ If the hash function is not one-way, Oscar can now compute the message x from $h^{-1}(z) = x$. Thus, the symmetric encryption of x is circumvented by the signature, which leaks the plaintext. For this reason, $h(x)$ should be a one-way function

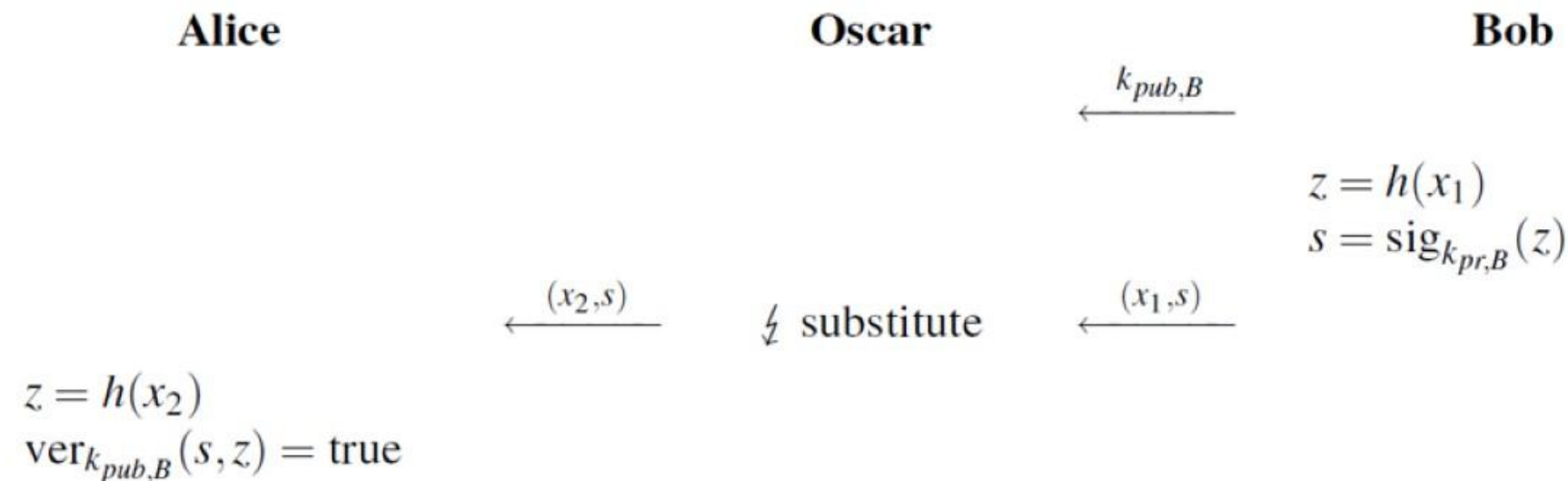
Module 03

Second Preimage Resistance

❑ For digital signatures with hash it is essential that two different messages do not hash to the same value.

❑ This means it should be computationally infeasible to create two different messages $x_1 \neq x_2$ with equal hash values $z_1 = h(x_1) = h(x_2) = z_2$.

Assume Bob hashes and signs a message x_1 . If Oscar is capable of finding a second message x_2 such that: $h(x_1) = h(x_2)$, he can run the following substitution attack:



- ❑ As we can see, Alice would accept x_2 as a correct message since the verification gives her the statement “true”. How can this happen?
- ❑ From a more abstract viewpoint, the attack (illustrated in previous slide) is possible because both signing (by Bob) and verifying (by Alice) do not happen with the actual message itself, but rather with the hashed version of it.
- ❑ Hence, if an attacker manages to find a second message with the same fingerprint (i.e., hash output), signing and verifying are the same for this second message.

❑ We call a hash function collision resistant or strong collision resistant if it is computationally infeasible to find two different inputs $x_1 \neq x_2$ with $h(x_1) = h(x_2)$.

❑ This property is harder to achieve than weak collision resistance since an attacker has two degrees of freedom: Both messages can be altered to achieve similar hash values.

❑ We show now how Oscar could turn his ability to find collisions into an attack. He starts with two messages, for instance:

$x_1 = \text{Transfer \$10 into Oscar's account}$

$x_2 = \text{Transfer \$10,000 into Oscar's account}$

❑ Oscar continues altering the messages until the condition $h(x_1) = h(x_2)$ is fulfilled.

Module 03

Collision Resistance

This attack assumes that Oscar can trick Bob into signing the message x_1 . This is, of course, not possible in every situation, but one can imagine scenarios where Oscar can pose as an innocent party, e.g., an e-commerce vendor on the Internet, and x_1 is the purchase order that is generated by Oscar

Alice

Oscar

Bob

$\xleftarrow{k_{pub,B}}$

$\xrightarrow{x_1}$

$$z = h(x_1)$$
$$s = \text{sig}_{k_{pr,B}}(z)$$

$\xleftarrow{(x_2,s)}$

$\nleftrightarrow$ substitute

$\xleftarrow{(x_1,s)}$

$$z = h(x_2)$$
$$\text{ver}_{k_{pub,B}}(s, z) = \text{true}$$

- Based on design of MD4 with key differences
- Produces 160-bit hash values
- In 2003:
 - 3 additional versions of SHA (known as SHA-2)
 - SHA-256, SHA-384, SHA-512
- Recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

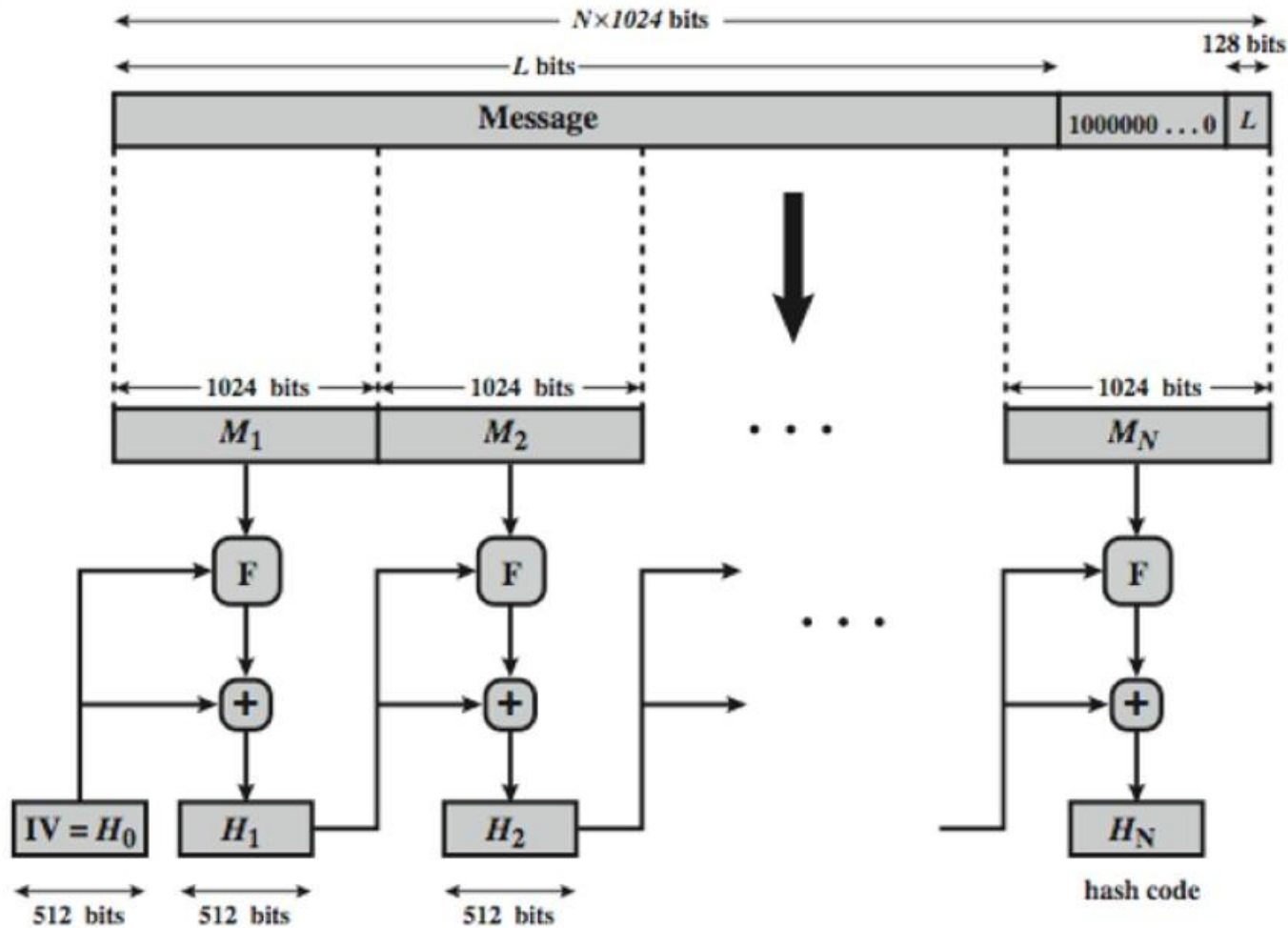
Module 03

SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

Module 03

SHA-512 Overview



$+$ = word-by-word addition mod 2^{64}

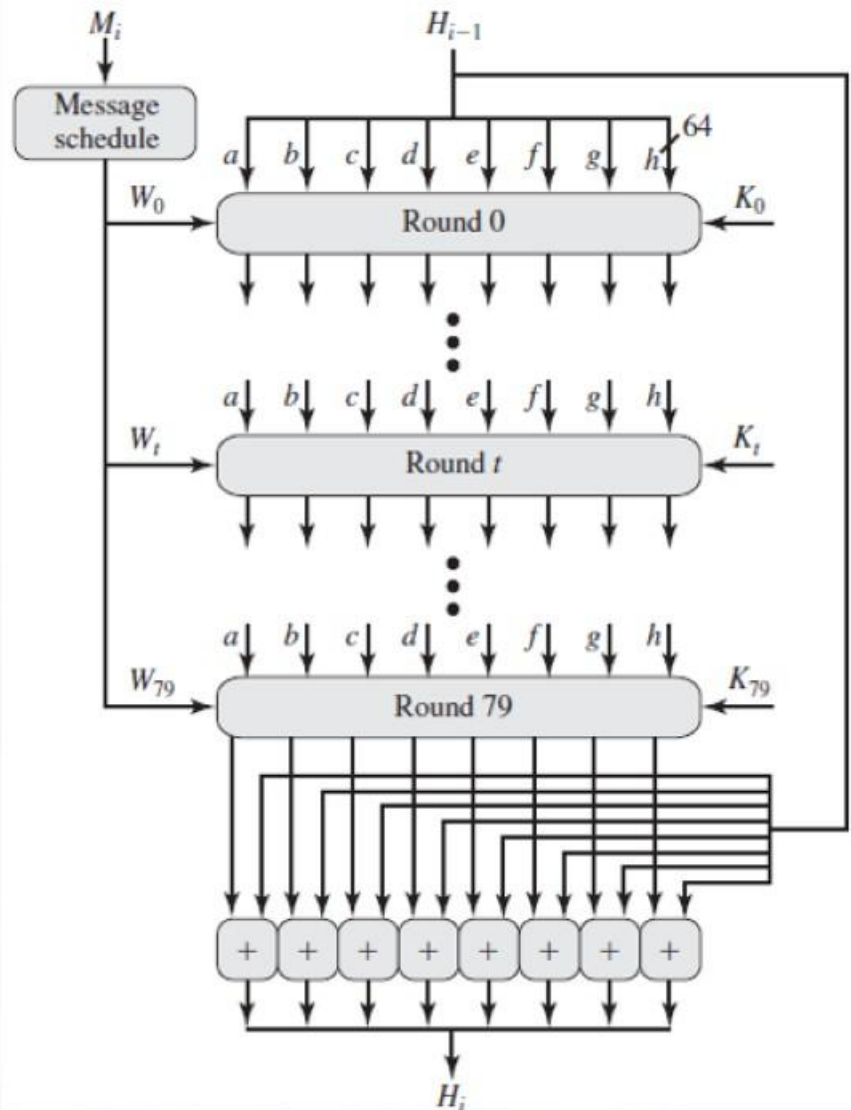
Heart of the algorithm

Processing message in 1024-bit blocks

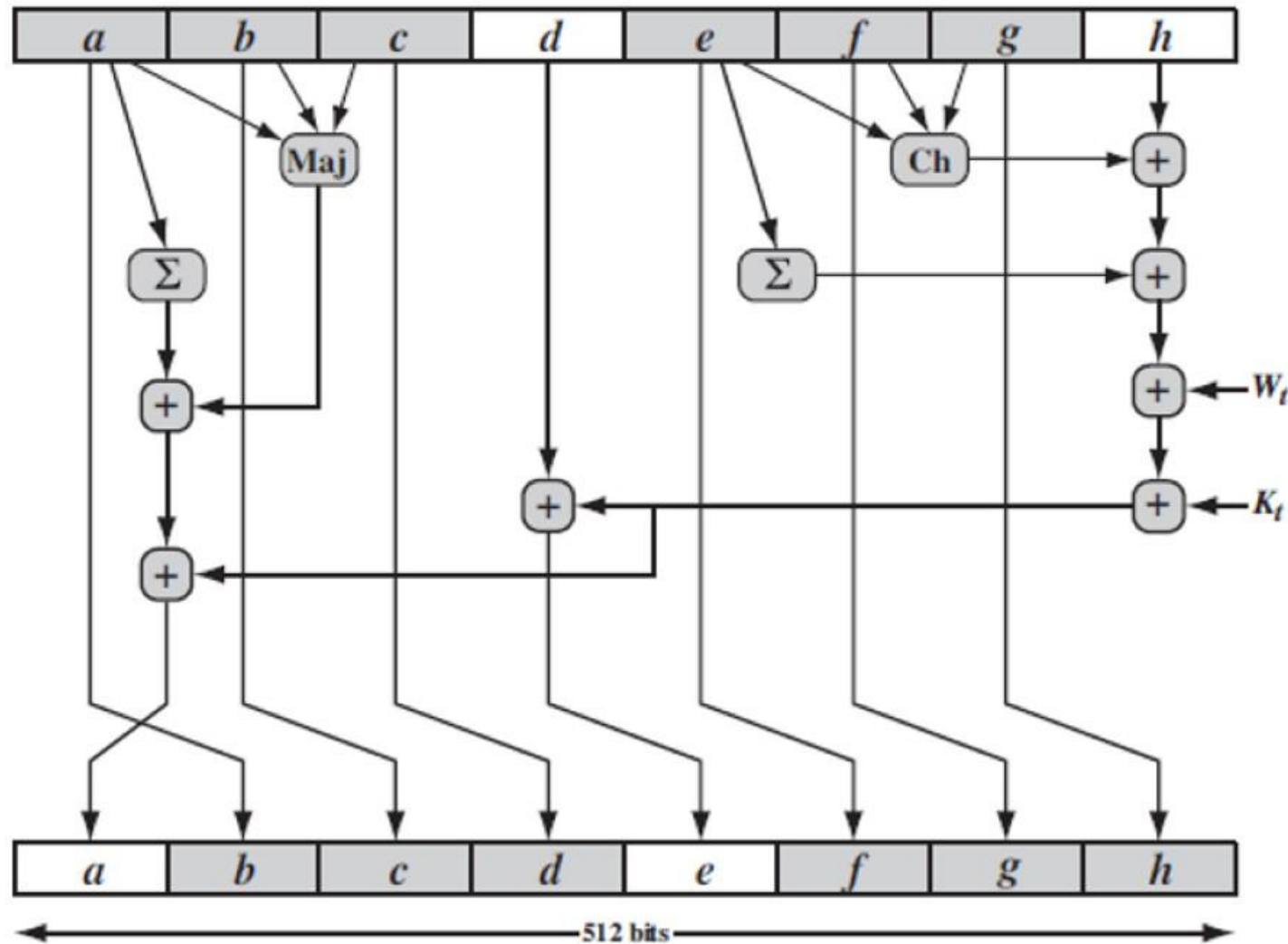
- 80 rounds
- Updating a 512-bit buffer
- Using a 64-bit value W_t derived from the current message block
- A round constant based on cube root of first 80 prime numbers

Module 03

SHA-512 Compression Function

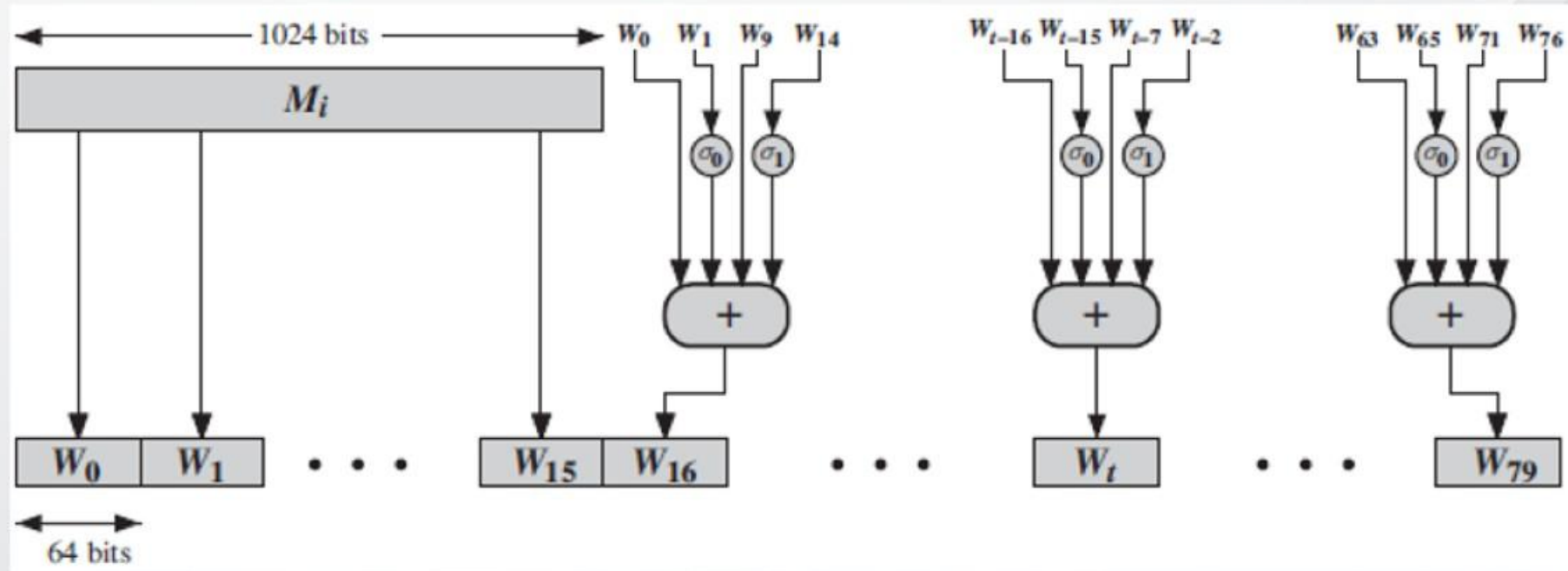


SHA-512 Round Function



Module 03

SHA-512 Word Creation



- SHA-1 not yet "broken"
 - But similar to broken MD5 & SHA-0, so considered insecure
- SHA-2 (esp. SHA-512) seems secure
 - Shares same structure and mathematical operations as predecessors so have concern
- NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function
 - To have in place by 2012 but not fixed

- Replace SHA-2 with SHA-3 in any use
 - Use same hash sizes
- Preserve the online nature of SHA-2
 - Must process small blocks (512 / 1034 bits)
- Evaluation criteria
 - Security close to theoretical max for hash sizes
 - Cost in time & memory
 - Characteristics: such as flexibility & simplicity



Thank You