HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

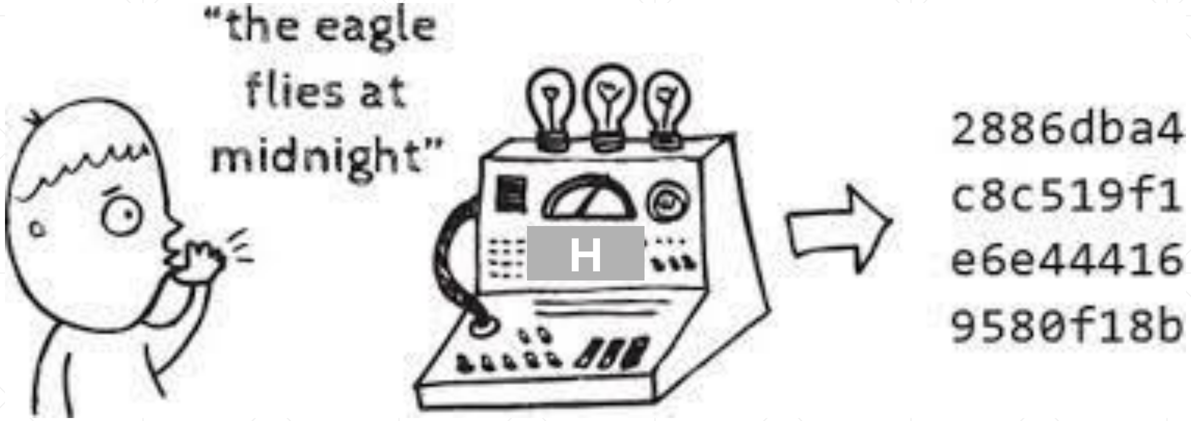
LÝ THUYẾT MẬT MÃ

Cryptography Theory

ET3310

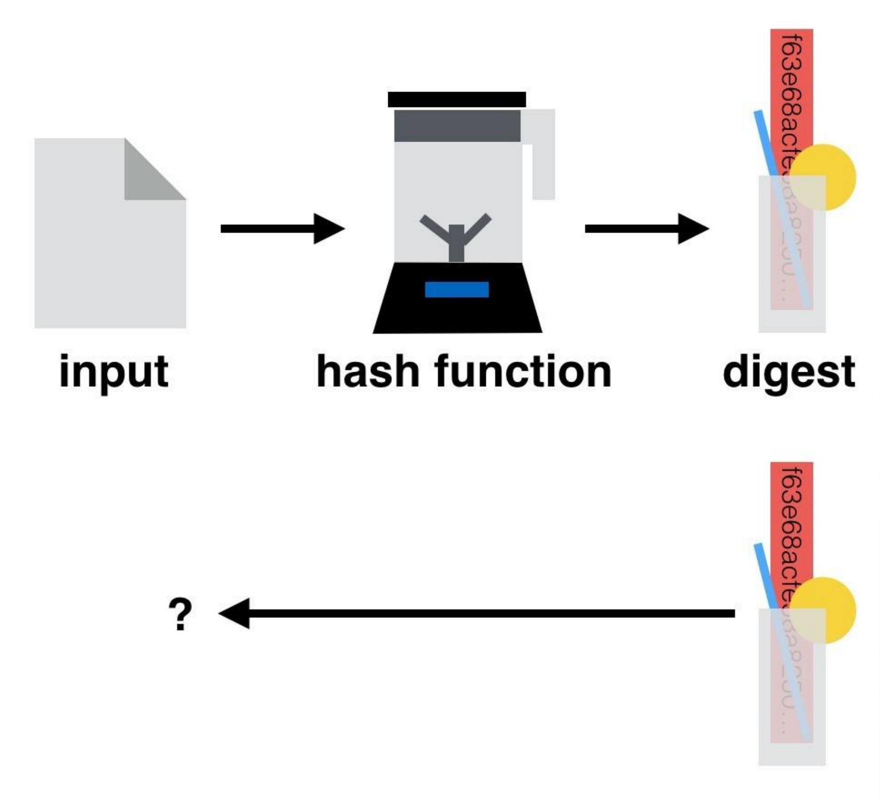
PGS. TS. Đỗ Trọng Tuấn
Trường Điện - Điện tử * Đại học Bách Khoa Hà Nội

ONE LOVE. ONE FUTURE.



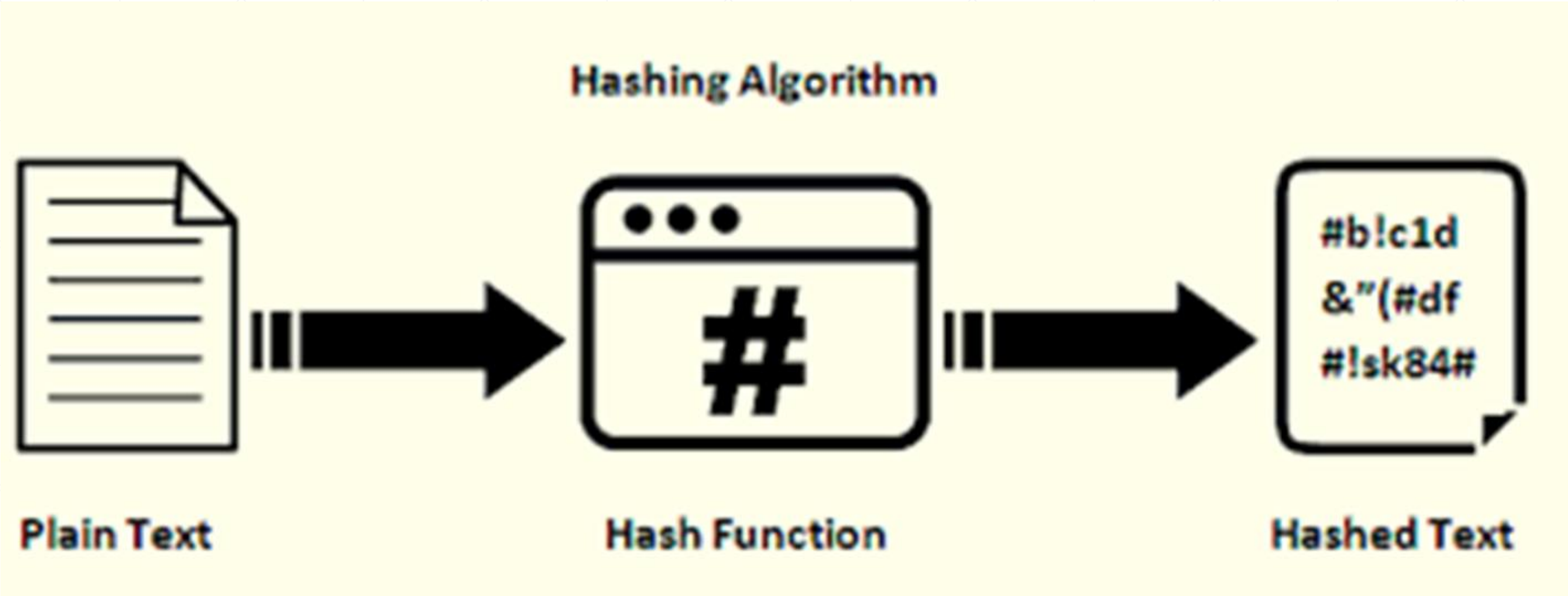
Cryptographic Hash Function

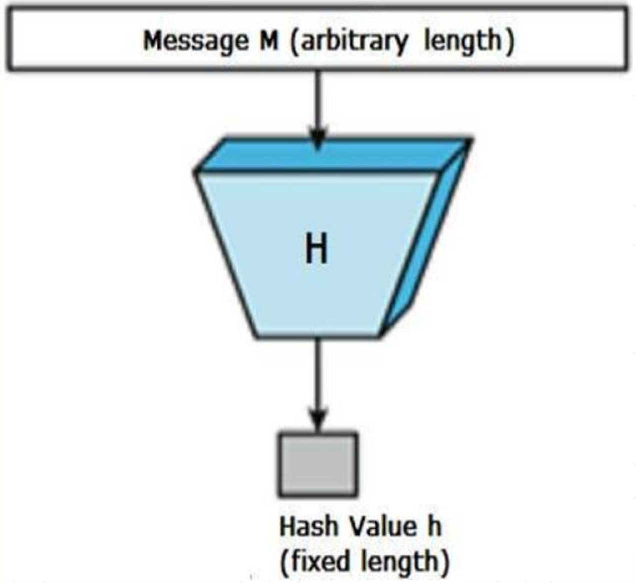
Contents



1. Introduction
2. Hash Function
3. SHA – 512
4. Hash Demonstrations

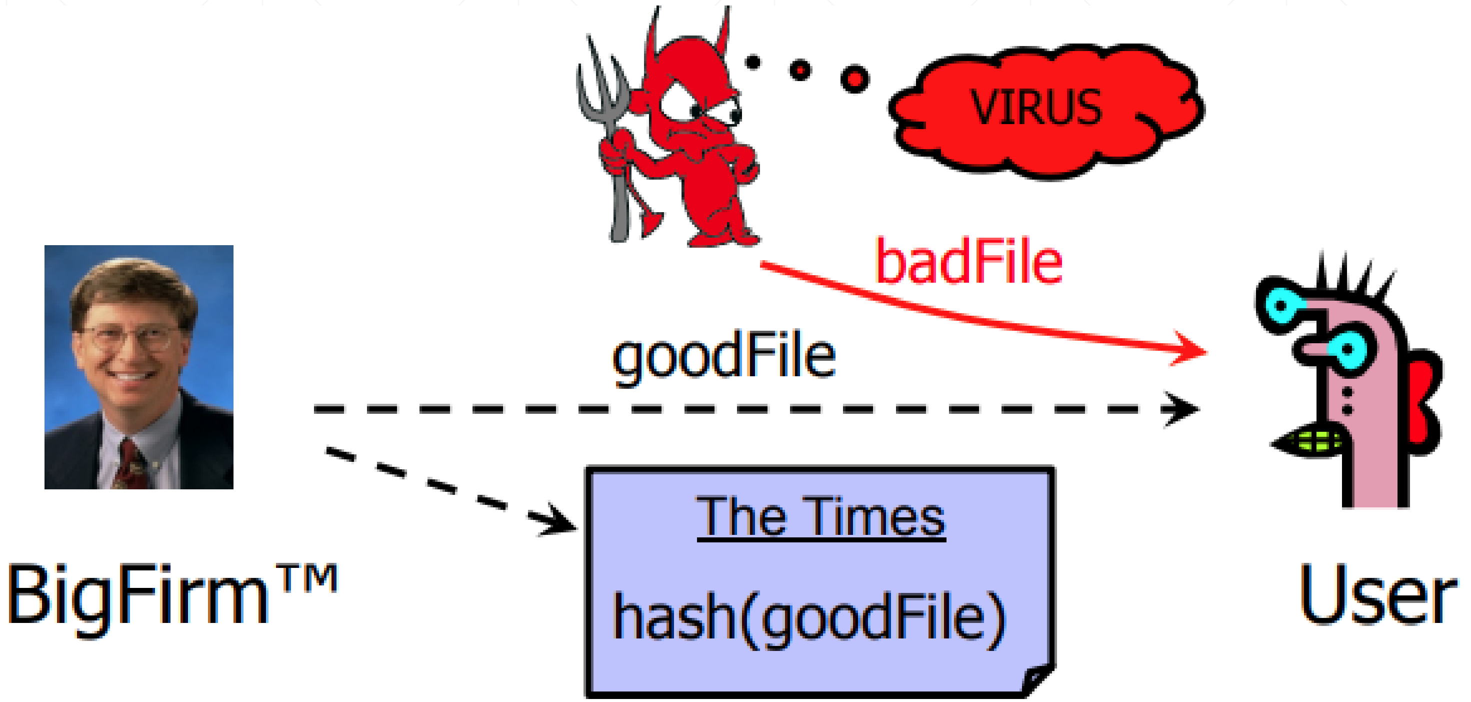
1 Introduction

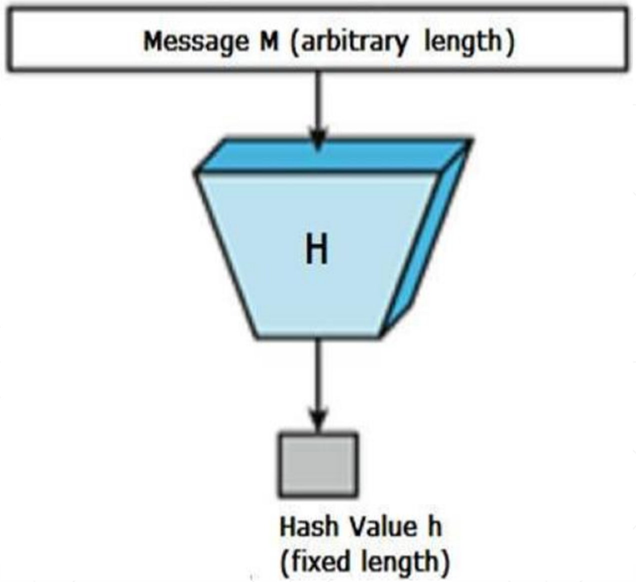




- *A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length.*
- *The ultimate goal of this chapter is to discuss the details of the two most promising cryptographic hash algorithms: SHA-512 and Whirlpool.*

1 Introduction



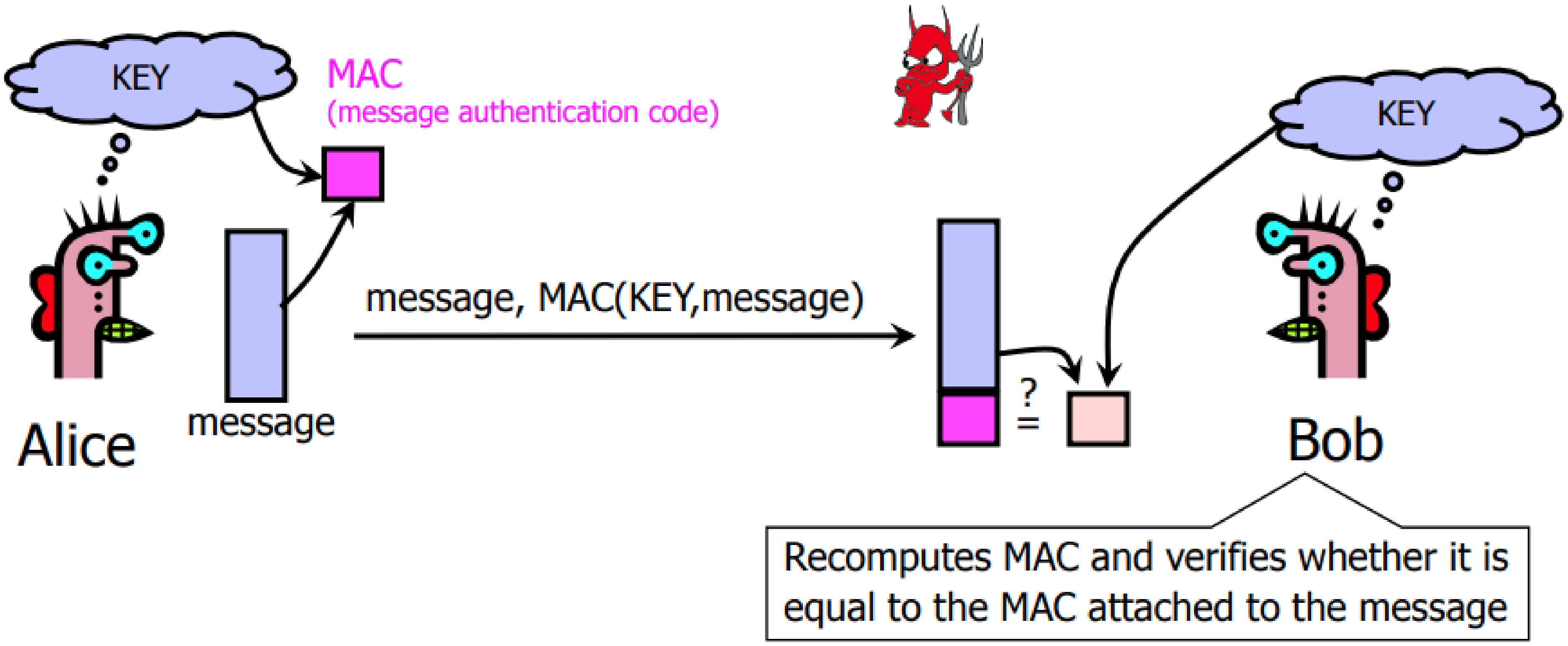


https://www.cs.columbia.edu/~suman/security_arch/crypto_summary.pdf

- *A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length.*
- *The ultimate goal of this chapter is to discuss the details of the two most promising cryptographic hash algorithms: SHA-512 and Whirlpool.*

1 Introduction

Integrity and Authentication



https://www.cs.columbia.edu/~suman/security_arch/crypto_summary.pdf

Two Groups of Compression Functions

1. The compression function is made from scratch.

Message Digest (MD)

2. A symmetric-key block cipher serves as a compression function.

Whirlpool

1 Introduction

The compression function is made from scratch >|< A symmetric-key block cipher serves as a compression function

A set of cryptographic hash functions uses compression functions that are made from scratch. These compression functions are specifically designed for the purposes they serve.

Message Digest (MD) Several hash algorithms were designed by Ron Rivest. These are referred to as **MD2, MD4,** and **MD5,** where MD stands for Message Digest. The last version, MD5, is a strengthened version of MD4 that divides the message into blocks of 512 bits and creates a 128-bit digest. It turned out that a message digest of size 128 bits is too small to resist collision attack.

Secure Hash Algorithm (SHA) The **Secure Hash Algorithm (SHA)** is a standard that was developed by the National Institute of Standards and Technology (NIST) and published as a Federal Information Processing standard (FIP 180). It is sometimes referred to as **Secure Hash Standard (SHS).** The standard is mostly based on MD5. The standard was revised in 1995 under FIP 180-1, which includes **SHA-1**. It was revised later under FIP 180-2, which defines four new versions: **SHA-224, SHA-256, SHA-384,** and **SHA-512.** Table 12.1 lists some of the characteristics of these versions.

1 Introduction

Other Algorithms **RACE Integrity Primitives Evaluation Message Digest (RIPMED)** has several versions. **RIPEMD-160** is a hash algorithm with a 160-bit message digest. RIPEMD-160 uses the same structure as MD5 but uses two parallel lines of execution. **HAVAL** is a variable-length hashing algorithm with a message digest of size 128, 160, 192, 224, and 256.The block size is 1024 bits.

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	$2^{64} - 1$ bits
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

1 Introduction

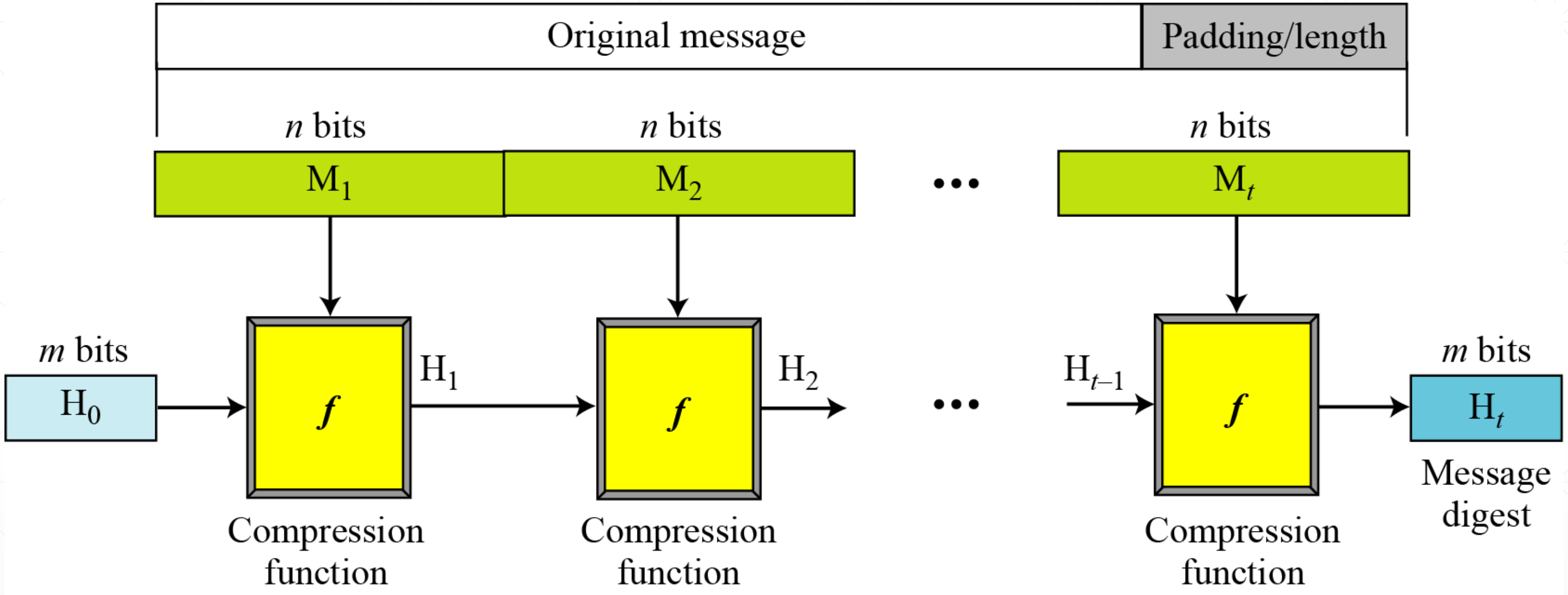
Hash Functions Based on Block Ciphers

An iterated cryptographic hash function can use a symmetric-key block cipher as a compression function. The whole idea is that there are several secure symmetric-key block ciphers, such as triple DES or AES, that can be used to make a one-way function instead of creating a new compression function. The block cipher in this case only

Characteristics	*SHA-1*	*SHA-224*	*SHA-256*	*SHA-384*	*SHA-512*
Maximum Message size	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{128}-1$	$2^{128}-1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

2 Hash Function

Merkle-Damgard Scheme

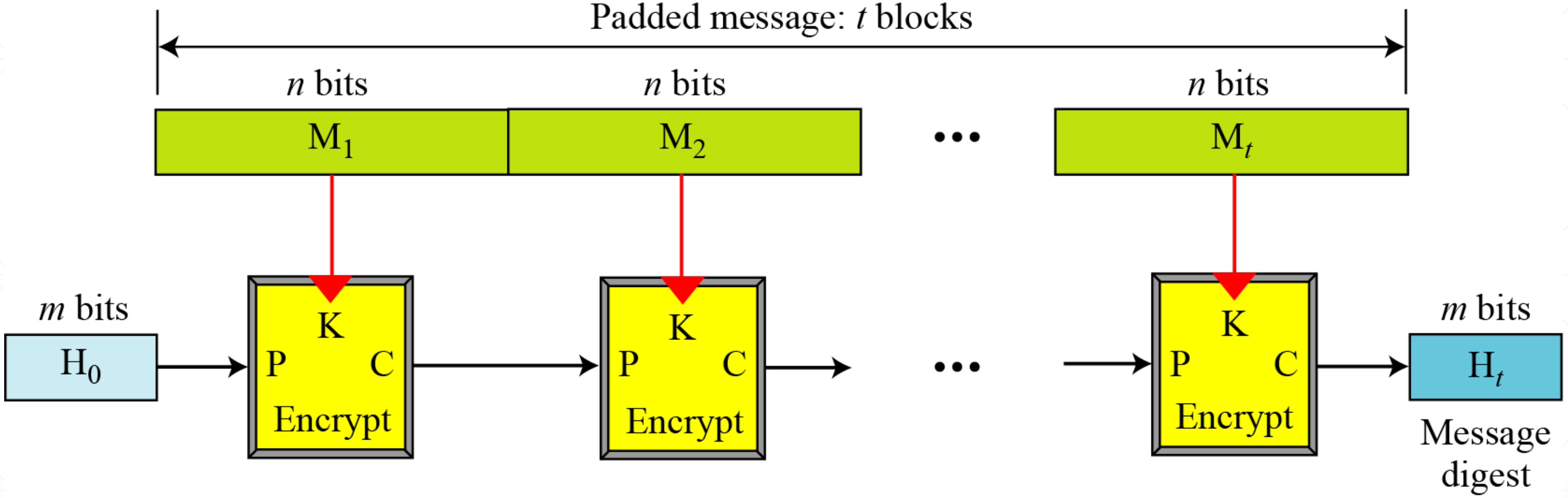


The scheme uses the following steps:

1. The message length and padding are appended to the message to create an augmented message that can be evenly divided into blocks of n bits, where n is the size of the block to be processed by the compression function.
2. The message is then considered as t blocks, each of n bits. We call each block M_1, $M_2, \ldots, M_t$. We call the digest created at t iterations $H_1, H_2, \ldots, H_t$.
3. Before starting the iteration, the digest H_0 is set to a fixed value, normally called IV (initial value or initial vector).
4. The compression function at each iteration operates on H_{i-1} and M_i to create a new H_i. In other words, we have $H_i = f(H_{i-1}, M_i)$, where f is the compression function.
5. H_t is the cryptographic hash function of the original message, that is, h(M).

2 Hash Function

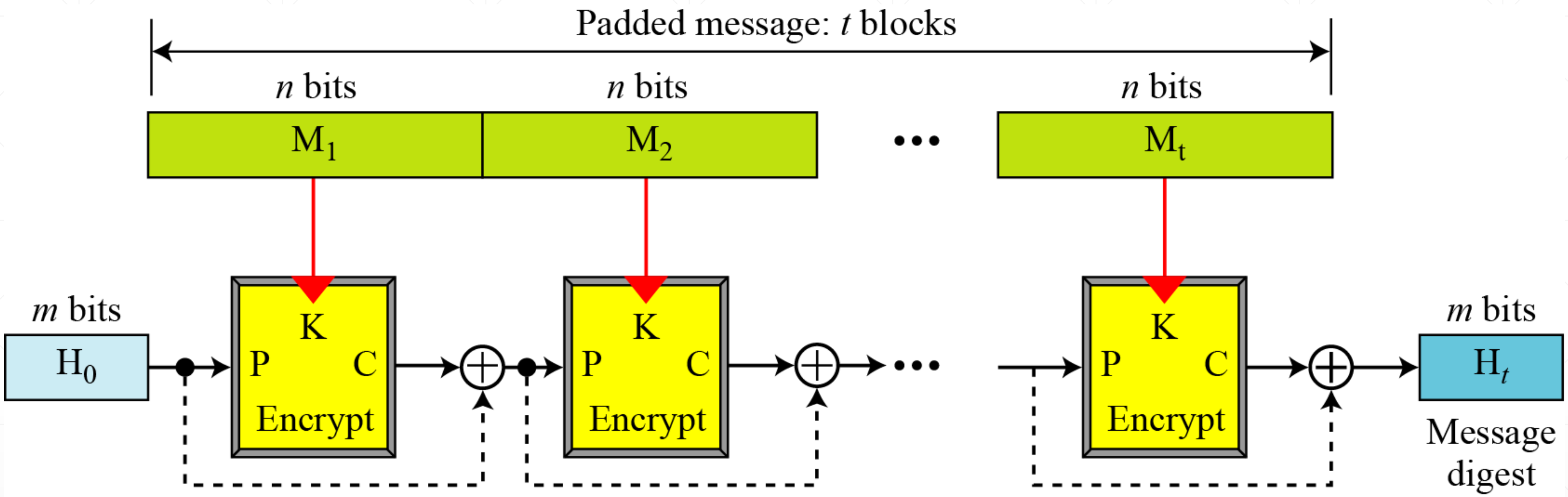
Rabin Scheme



Rabin Scheme The iterated hash function proposed by Rabin is very simple. The **Rabin scheme** is based on the Merkle-Damgard scheme. The compression function is replaced by any encrypting cipher. The message block is used as the key; the previously created digest is used as the plaintext. The ciphertext is the new message digest. Note that the size of the digest is the size of data block cipher in the underlying cryptosystem. For example, if DES is used as the block cipher, the size of the digest is only 64 bits.

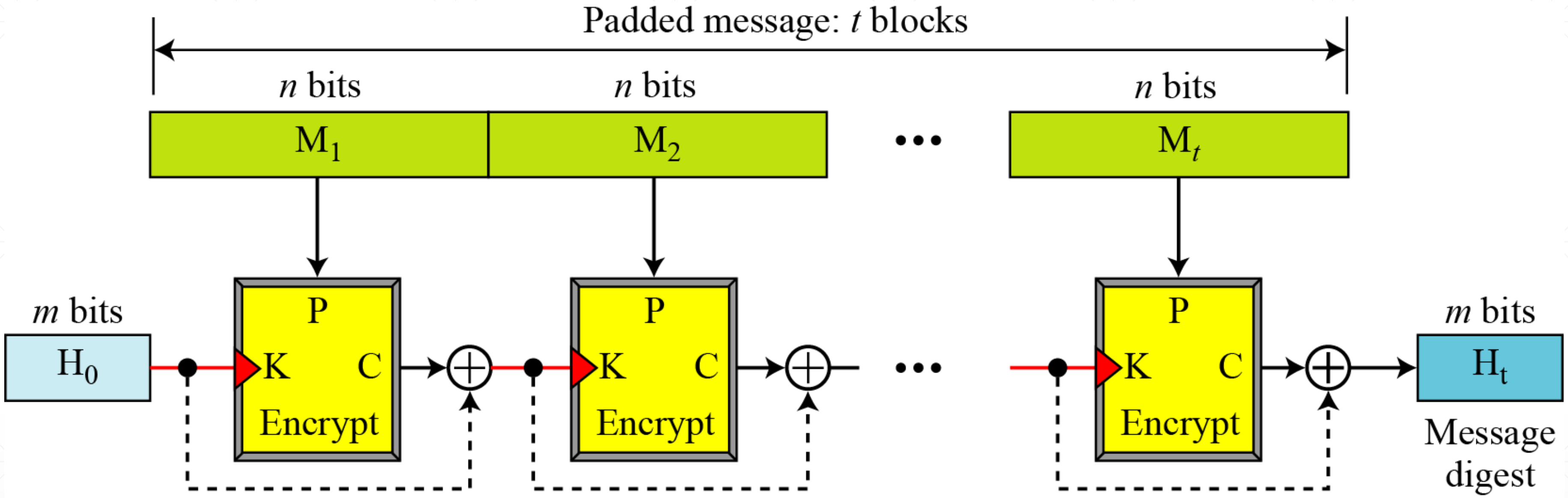
Hash Function

Davies-Meyer Scheme



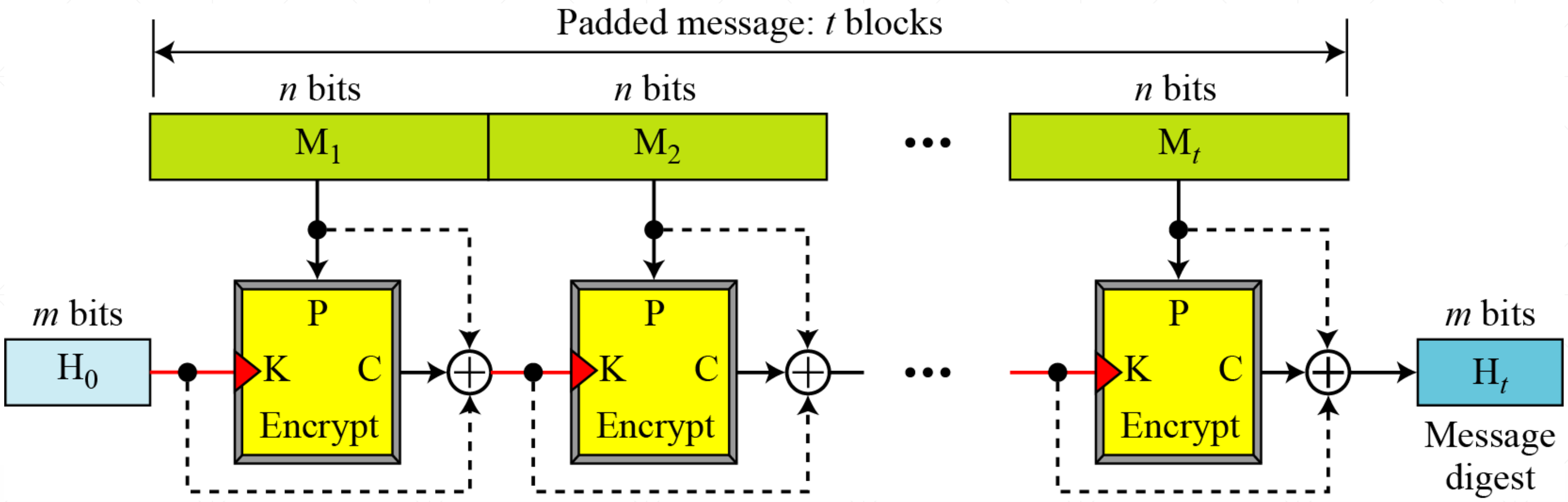
Davies-Meyer Scheme The **Davies-Meyer scheme** is basically the same as the Rabin scheme except that it uses forward feed to protect against meet-in-the-middle attack. Figure 12.3 shows the Davies-Meyer scheme.

Matyas-Meyer-Oseas Scheme



Matyas-Meyer-Oseas Scheme The **Matyas-Meyer-Oseas scheme** is a dual version of the Davies-Meyer scheme: the message block is used as the key to the cryptosystem.

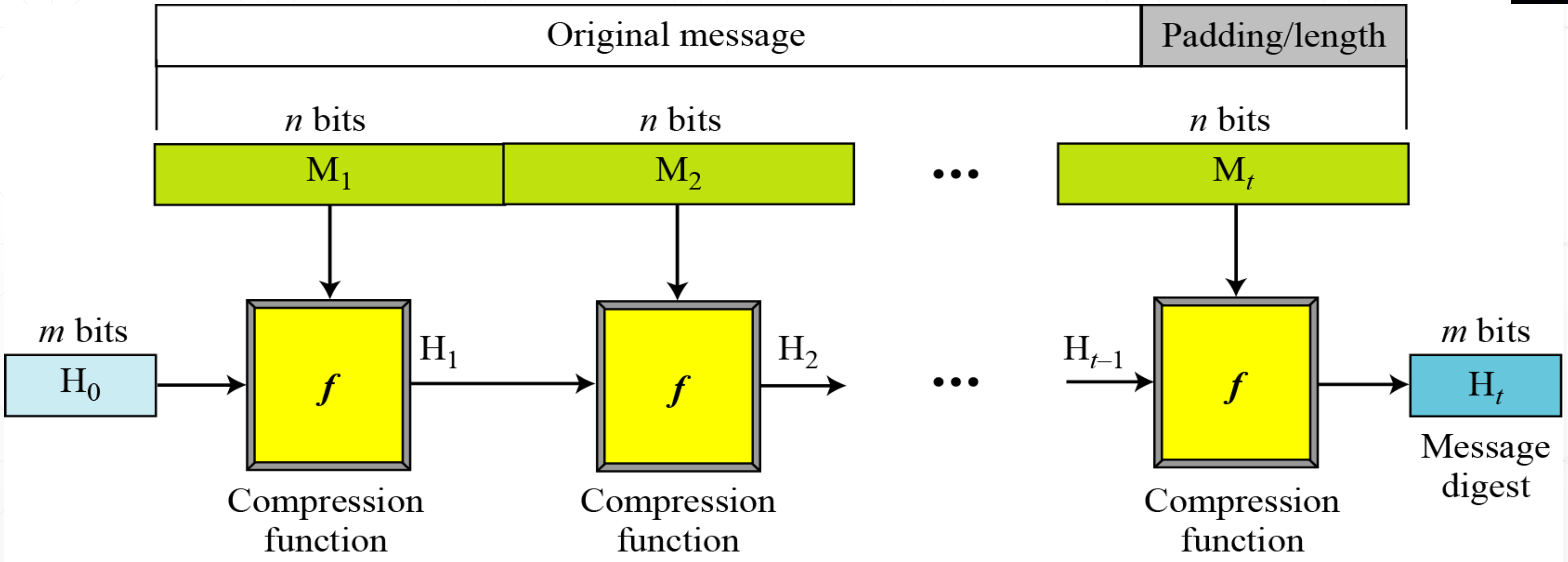
Miyaguchi-Preneel Scheme



Miyaguchi-Preneel Scheme The **Miyaguchi-Preneel scheme** is an extended version of Matyas-Meyer-Oseas. To make the algorithm stronger against attack, the plaintext, the cipher key, and the ciphertext are all exclusive-ored together to create the new digest. This is the scheme used by the Whirlpool hash function.

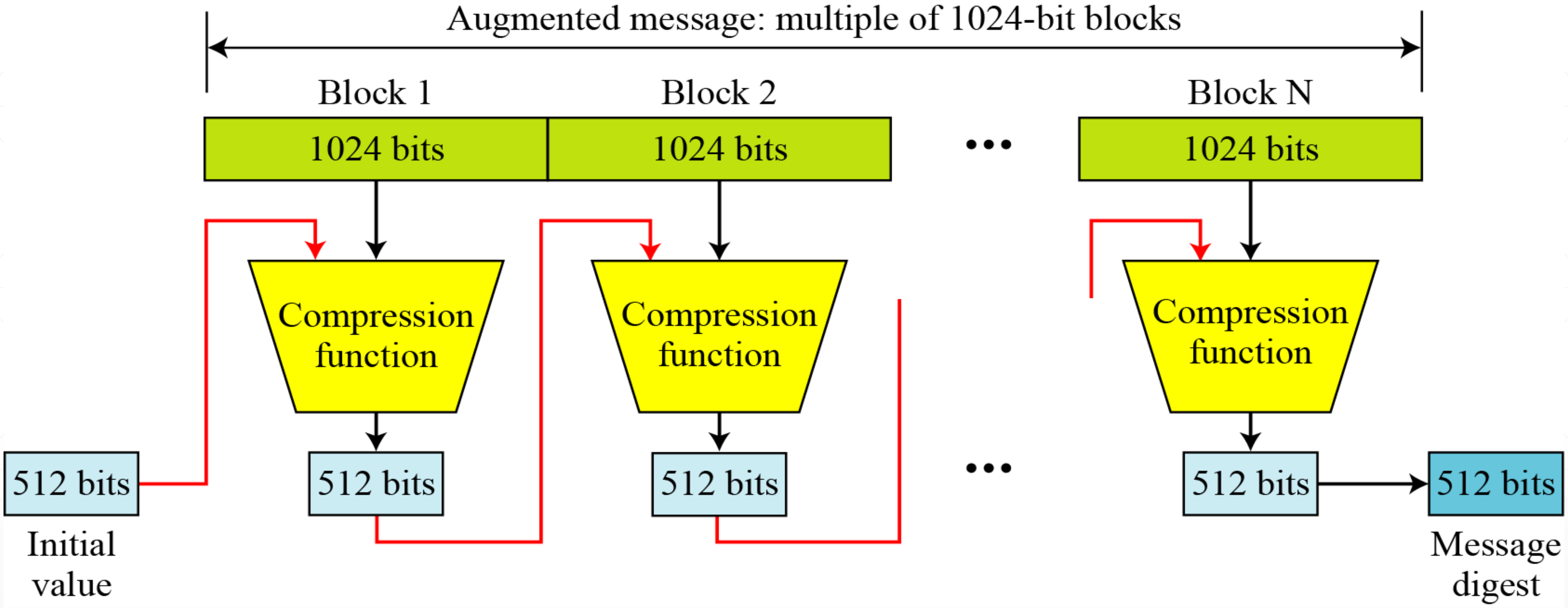
3 SHA – 512

SHA-512 is the version of SHA with a 512-bit message digest. This version, like the others in the SHA family of algorithms, is based on the Merkle-Damgard scheme.



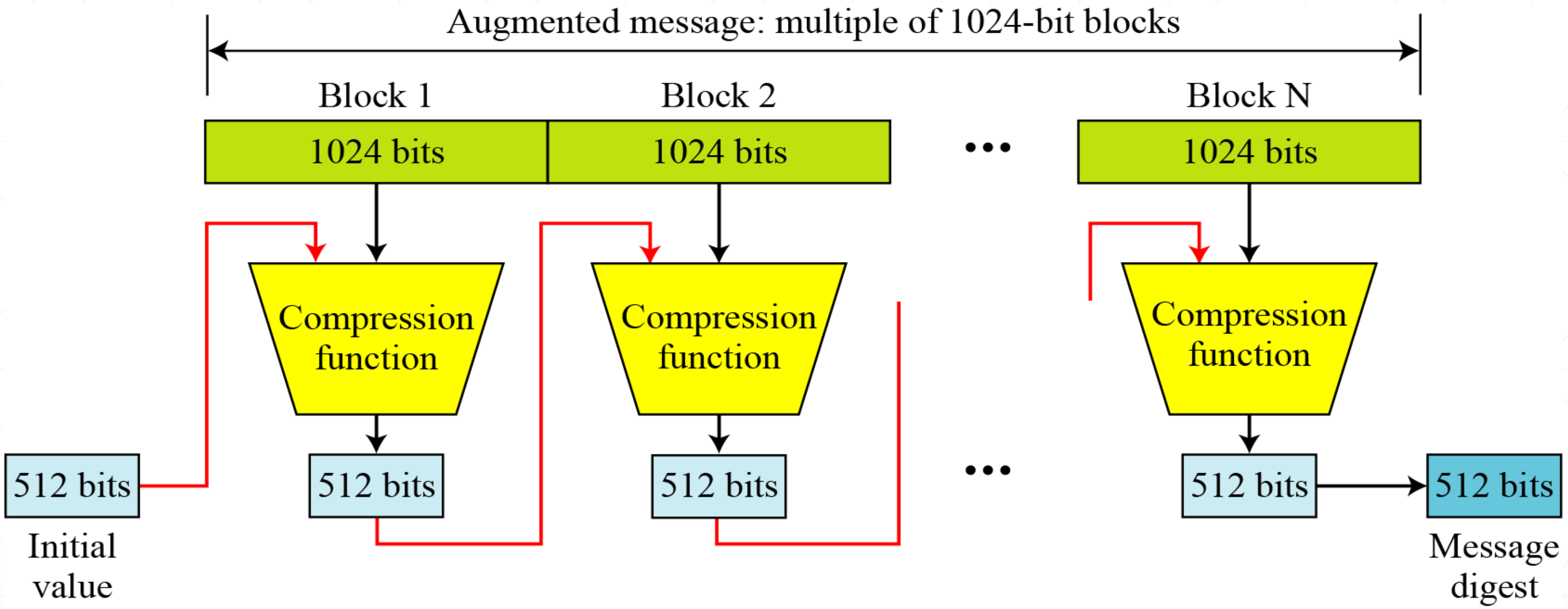
Merkle-Damgard Scheme

SHA – 512



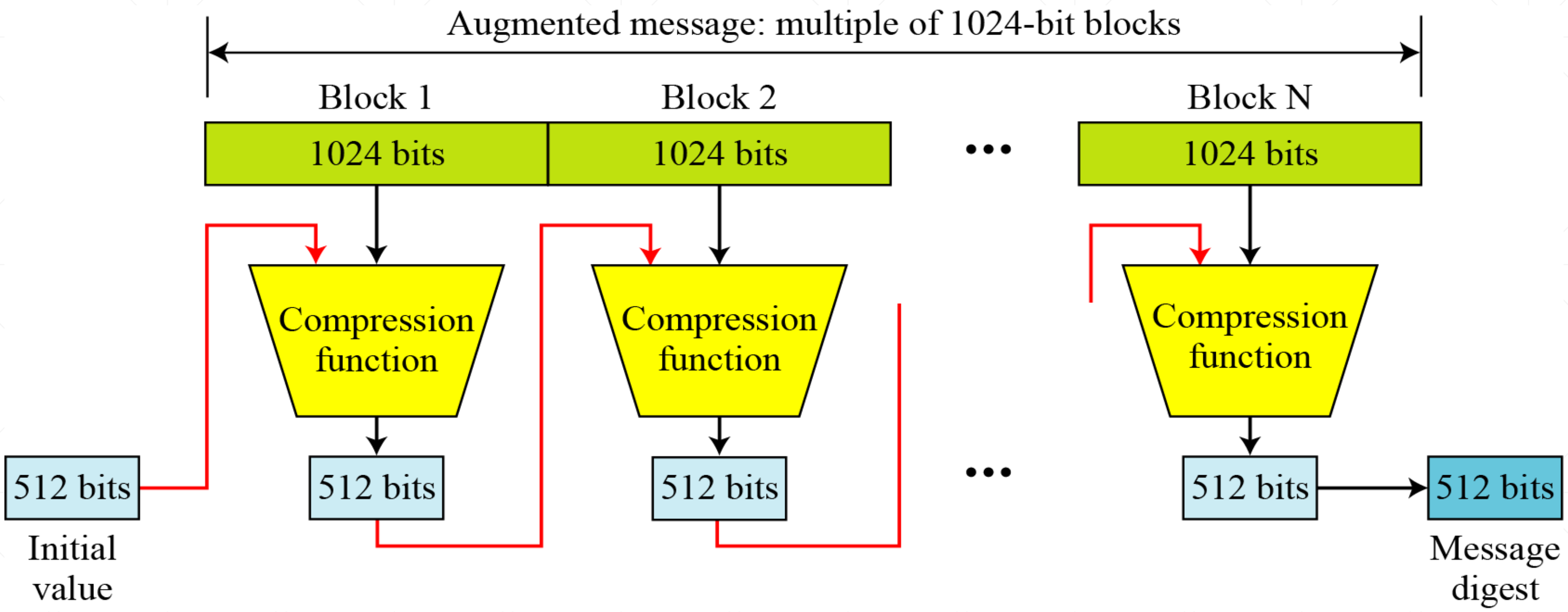
SHA-512 creates a digest of 512 bits from a multiple-block message. Each block is 1024 bits in length, as shown in Figure 12.6.

3 SHA – 512



The digest is initialized to a predetermined value of 512 bits. The algorithm mixes this initial value with the first block of the message to create the first intermediate message digest of 512 bits. This digest is then mixed with the second block to create the second intermediate digest. Finally, the $(N - 1)$th digest is mixed with the Nth block to create the Nth digest. When the last block is processed, the resulting digest is the message digest for the entire message.

3 SHA – 512

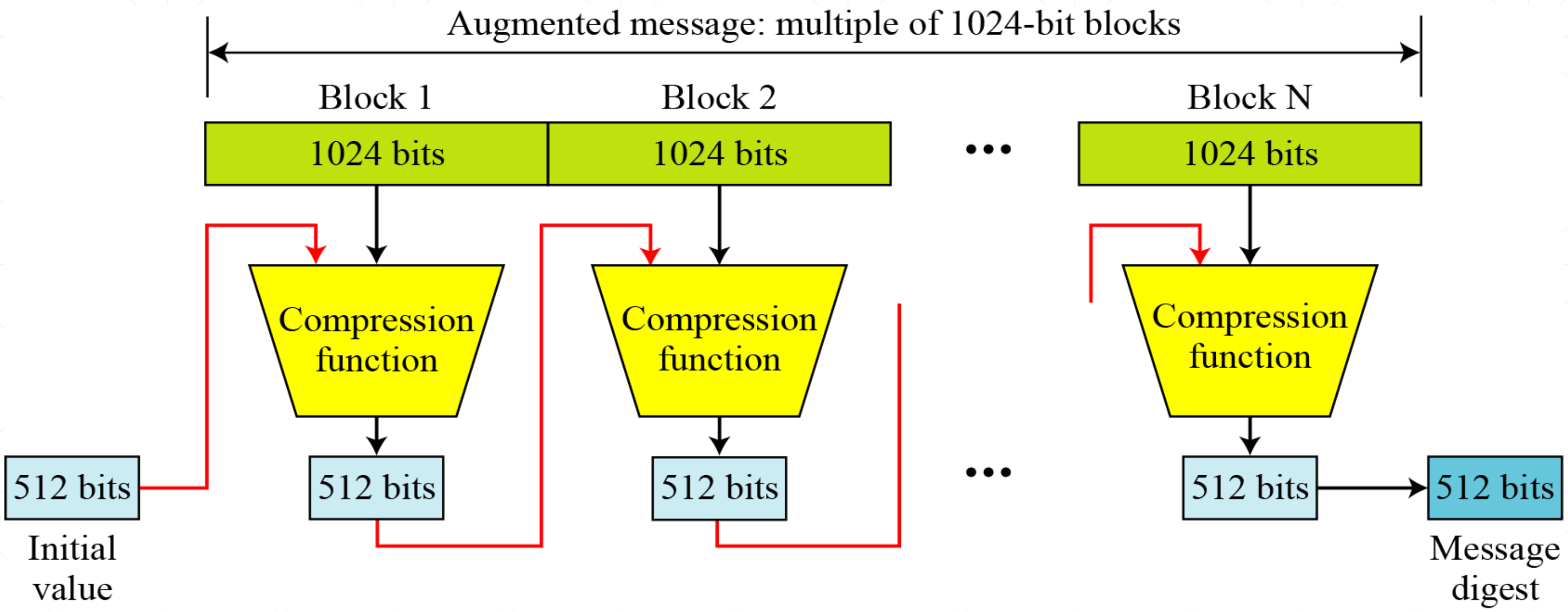


Message Preparation

SHA-512 insists that the length of the original message be less than 2^{128} bits.

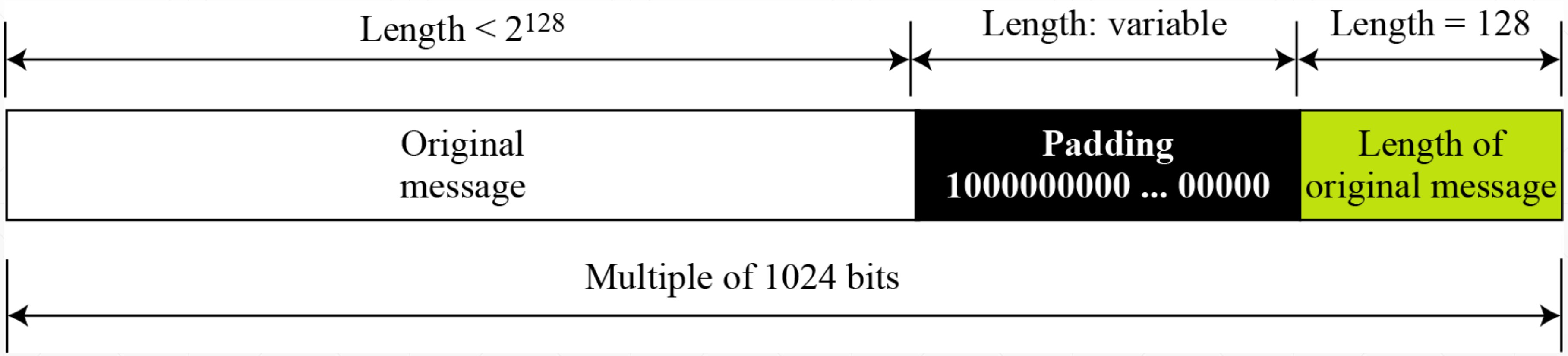
SHA-512 creates a 512-bit message digest out of a message less than 2^{128}.

3 SHA – 512



Message Preparation

Padding and length field in SHA-512

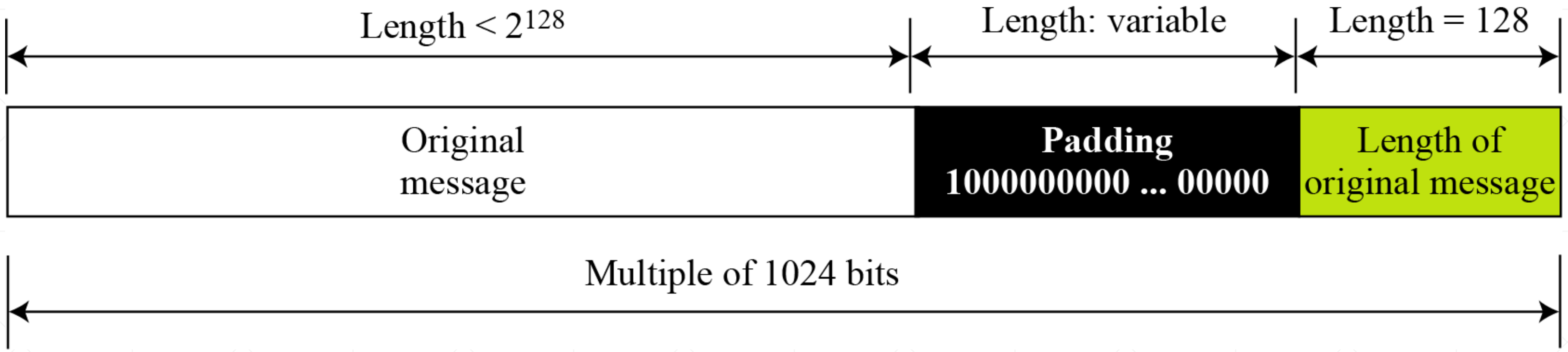


$$(|M| + |P| + 128) = 0 \bmod 1024 \quad \rightarrow \quad |P| = (-|M| - 128) \bmod 1024$$

3 SHA – 512

Message Preparation

$$(|M| + |P| + 128) = 0 \bmod 1024 \rightarrow |P| = (-|M| - 128) \bmod 1024$$



Example:

What is the number of padding bits if the length of the original message is 2590 bits?

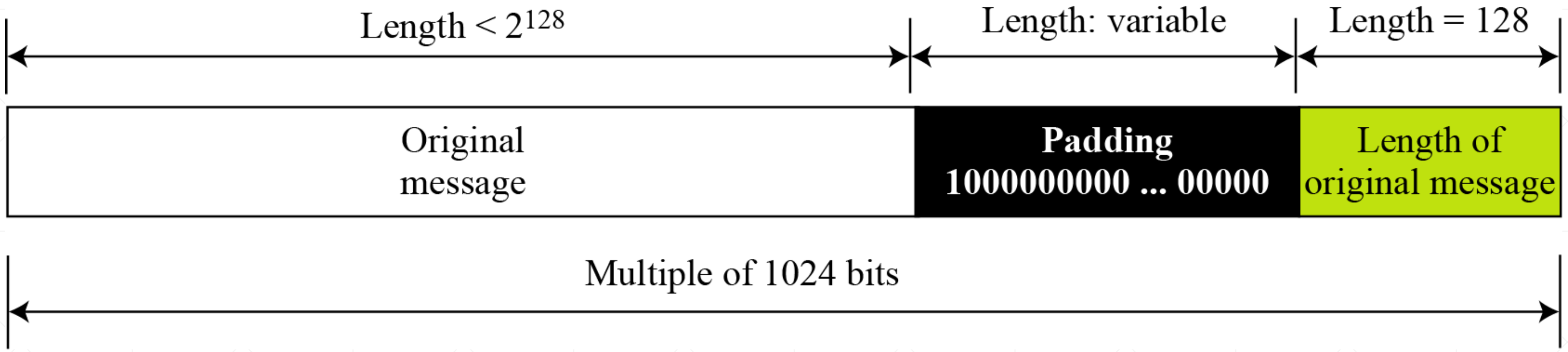
$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

The padding consists of one 1 followed by 353 0's.

3 SHA – 512

Message Preparation

$(|M| + |P| + 128) = 0 \bmod 1024 \rightarrow |P| = (-|M| - 128) \bmod 1024$



Example:

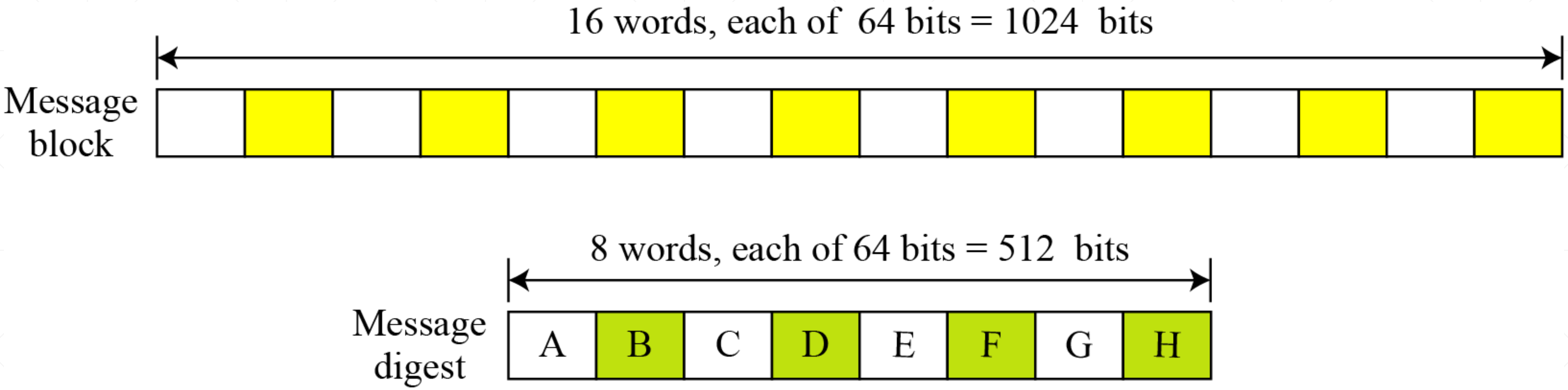
Do we need padding if the length of the original message is already a multiple of 1024 bits?

Solution

Yes, because we need to add the length field. So padding is needed to make the new block a multiple of 1024 bits.

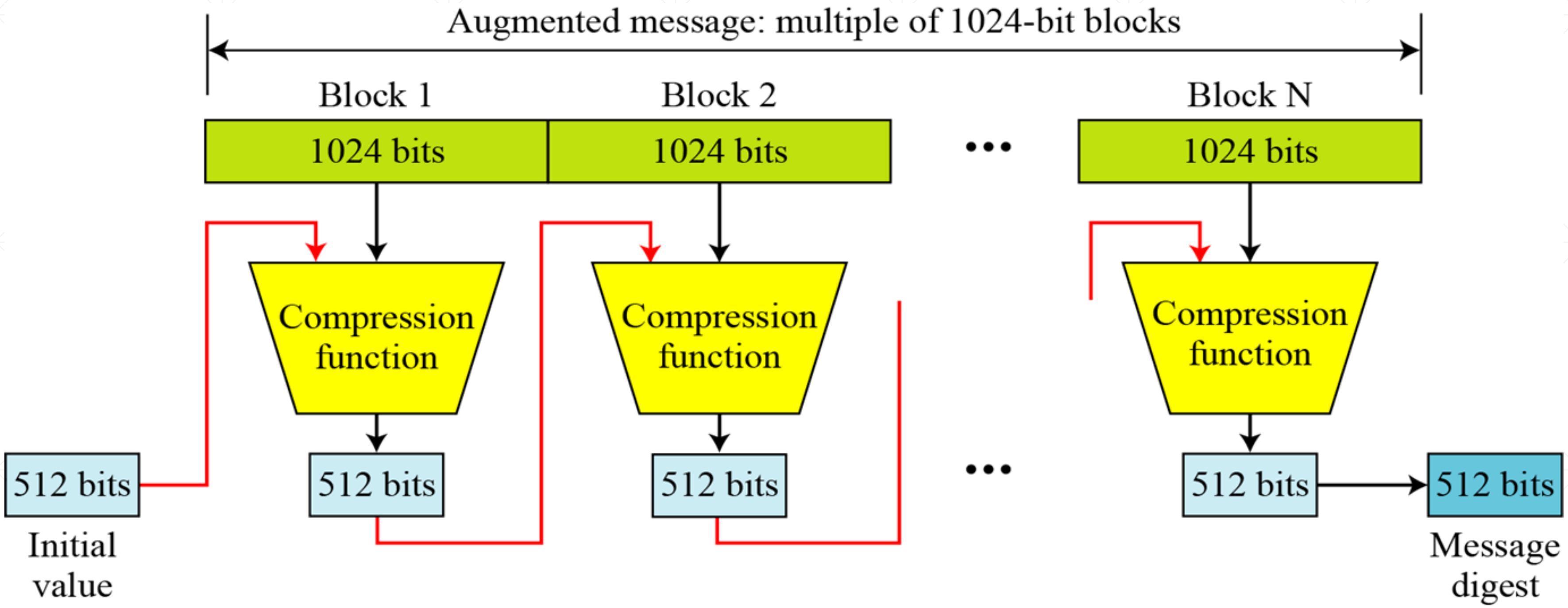
SHA – 512

Words



SHA-512 operates on words; it is **word oriented.** A word is defined as 64 bits. This means that, after the padding and the length field are added to the message, each block of the message consists of sixteen 64-bit words. The message digest is also made of 64-bit words, but the message digest is only eight words and the words are named A, B, C, D, E, F, G, and H.

SHA – 512

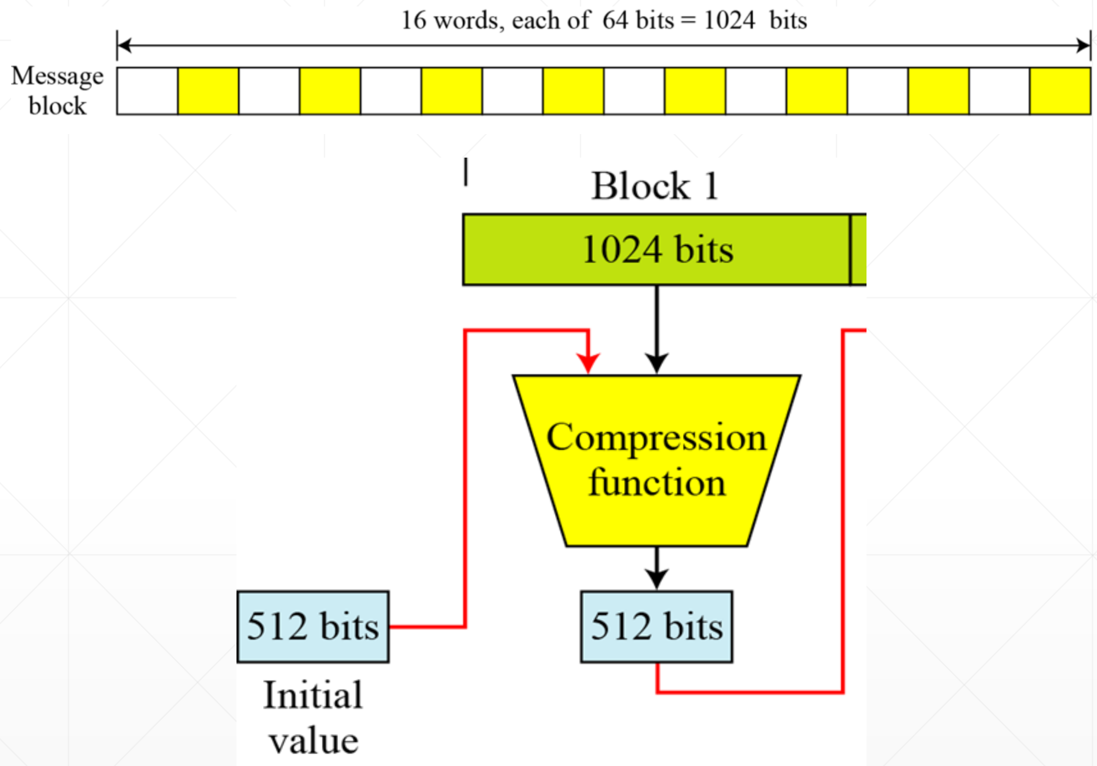


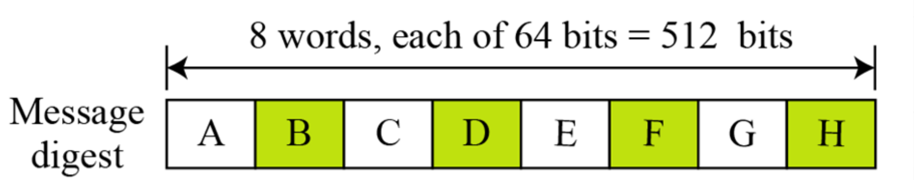
Message Digest Initialization

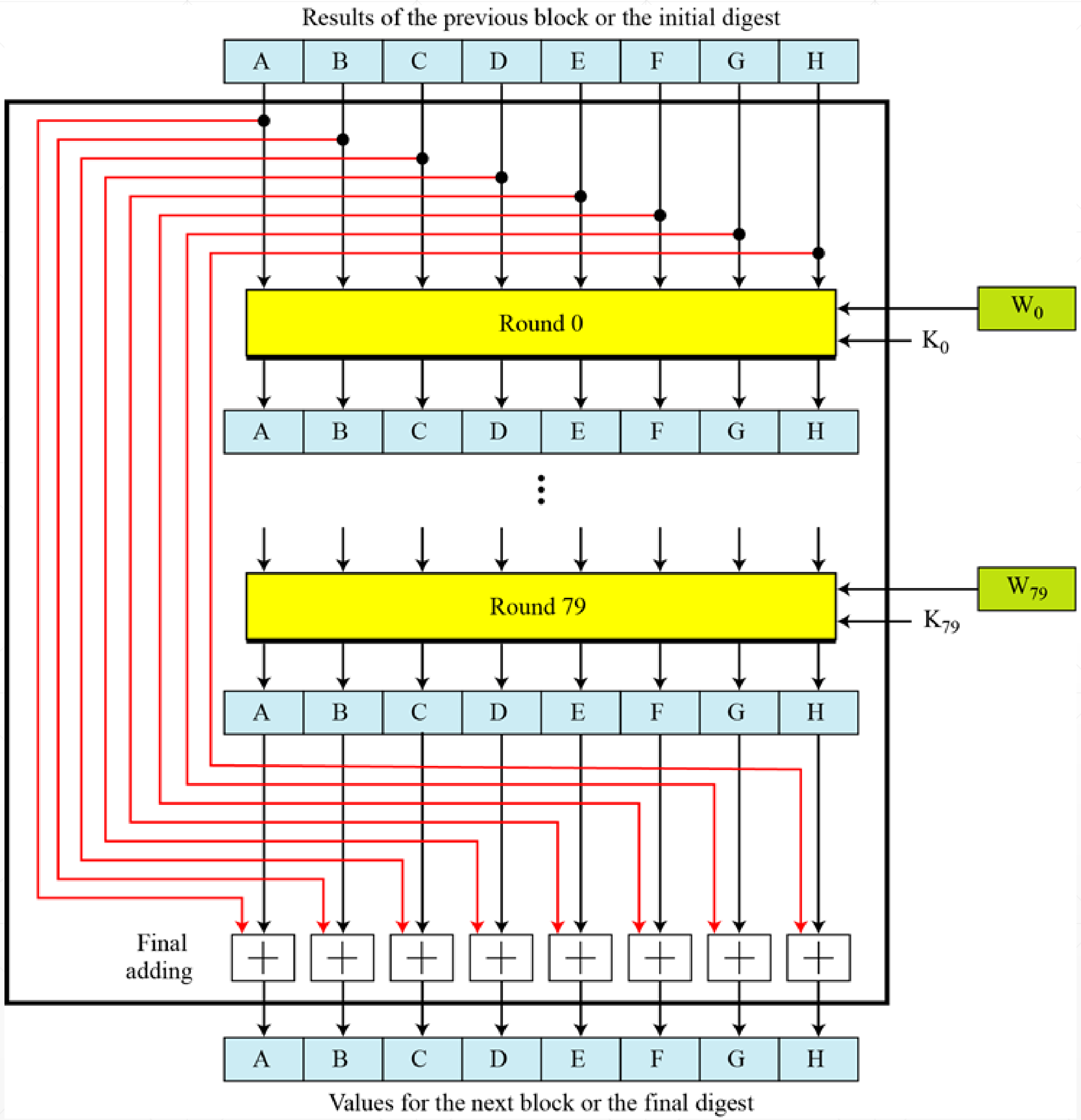
Buffer	*Value (in hexadecimal)*	*Buffer*	*Value (in hexadecimal)*
A_0	**6A09E667F3BCC908**	E_0	**510E527FADE682D1**
B_0	**BB67AE8584CAA73B**	F_0	**9B05688C2B3E6C1F**
C_0	**3C6EF372FE94F82B**	G_0	**1F83D9ABFB41BD6B**
D_0	**A54FF53A5F1D36F1**	H_0	**5BE0CD19137E2179**

3 SHA – 512

Compression function in SHA-512

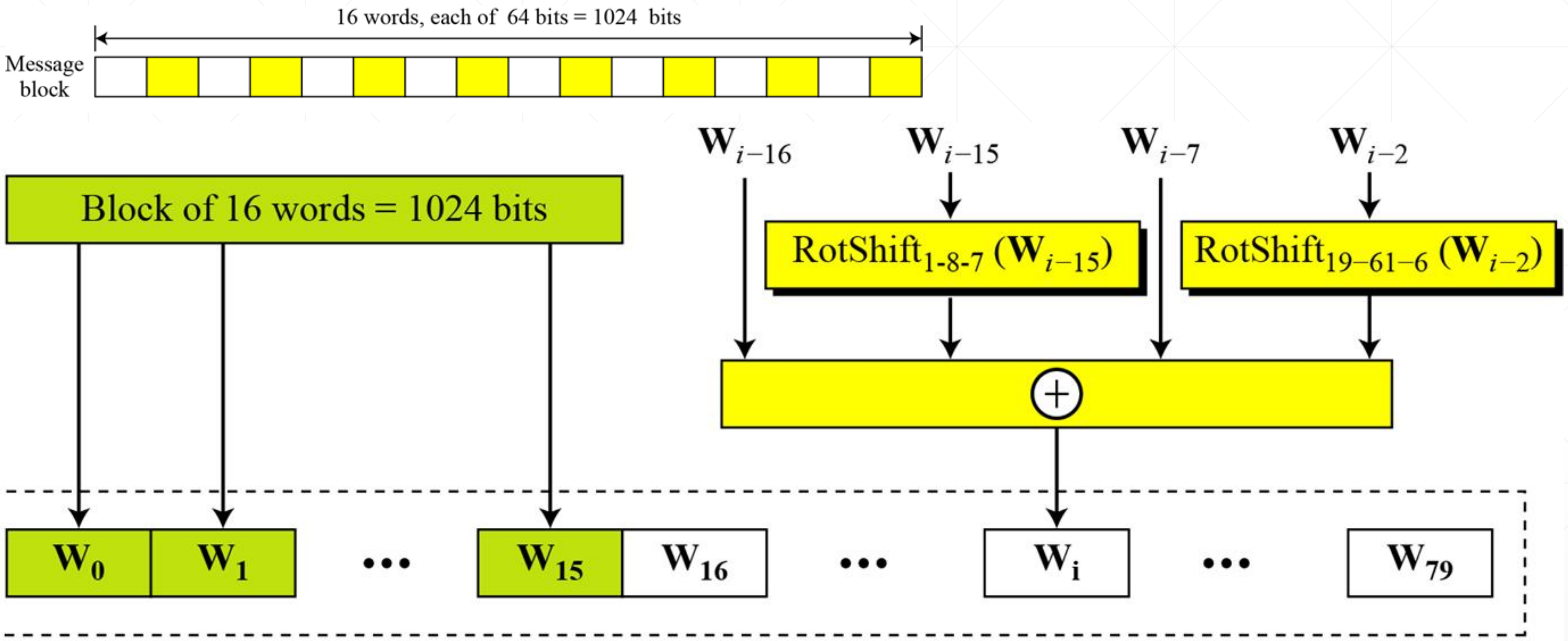


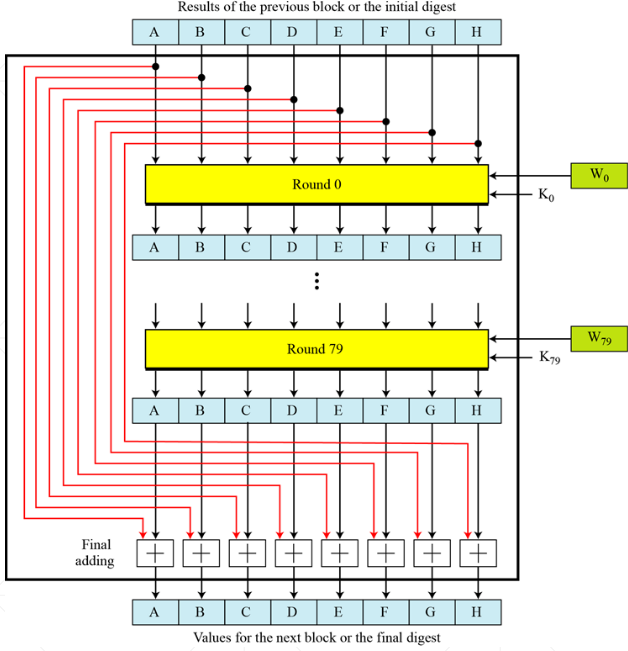




SHA – 512

Word Expansion





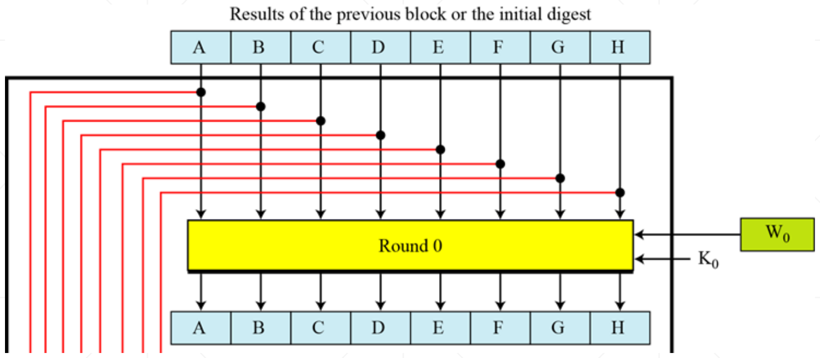
$\text{RotShift}_{l\text{-}m\text{-}n}(x)$: $\text{RotR}_l(x) \oplus \text{RotR}_m(x) \oplus \text{ShL}_n(x)$

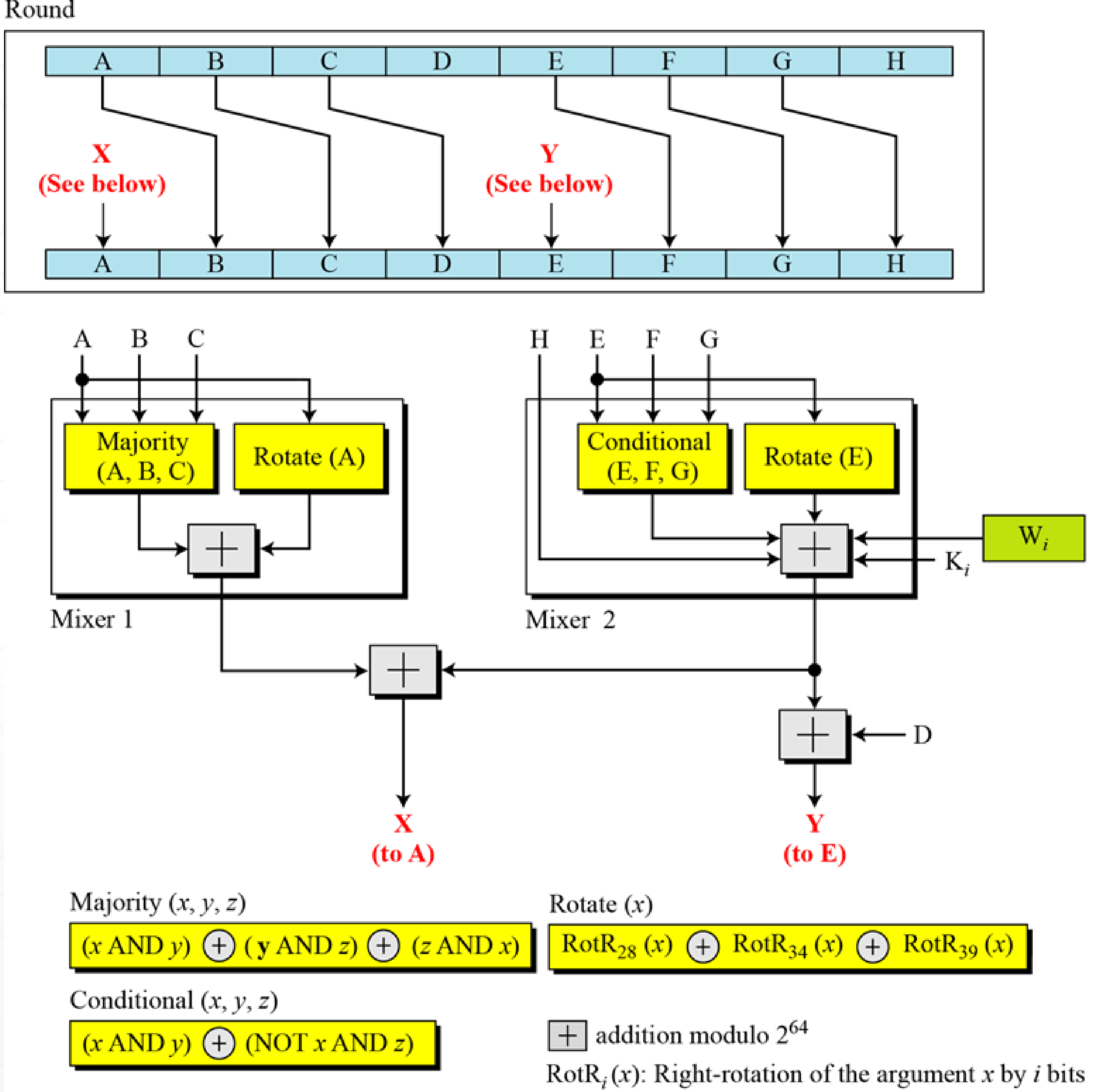
$\text{RotR}_i(x)$: Right-rotation of the argument x by i bits

$\text{ShL}_i(x)$: Shift-left of the argument x by i bits and padding the left by 0's.

3 SHA – 512

Structure of each round





Majority Function

$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$

Conditional Function

$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$

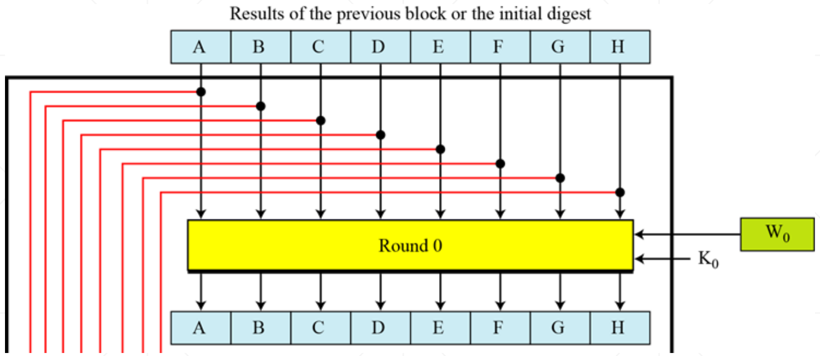
Rotate Functions

Rotate (A): $RotR_{28}(A) \oplus RotR_{34}(A) \oplus RotR_{29}(A)$

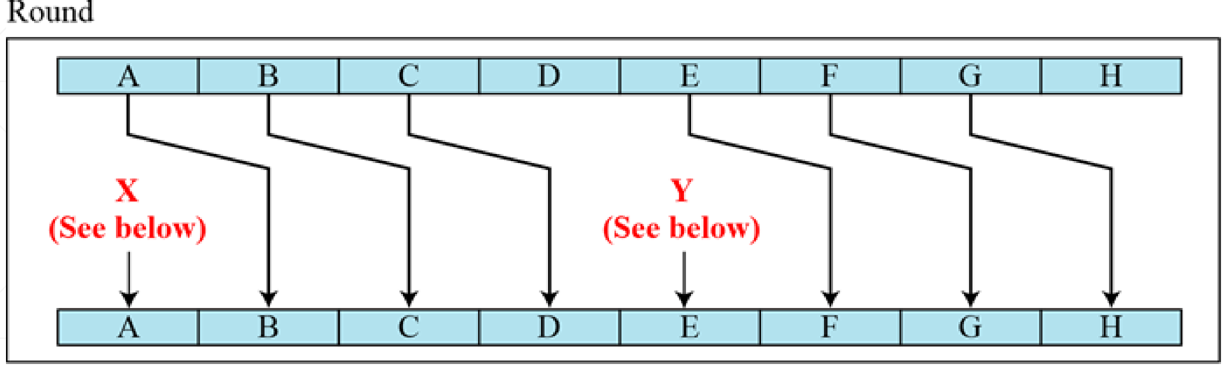
Rotate (E): $RotR_{28}(E) \oplus RotR_{34}(E) \oplus RotR_{29}(E)$

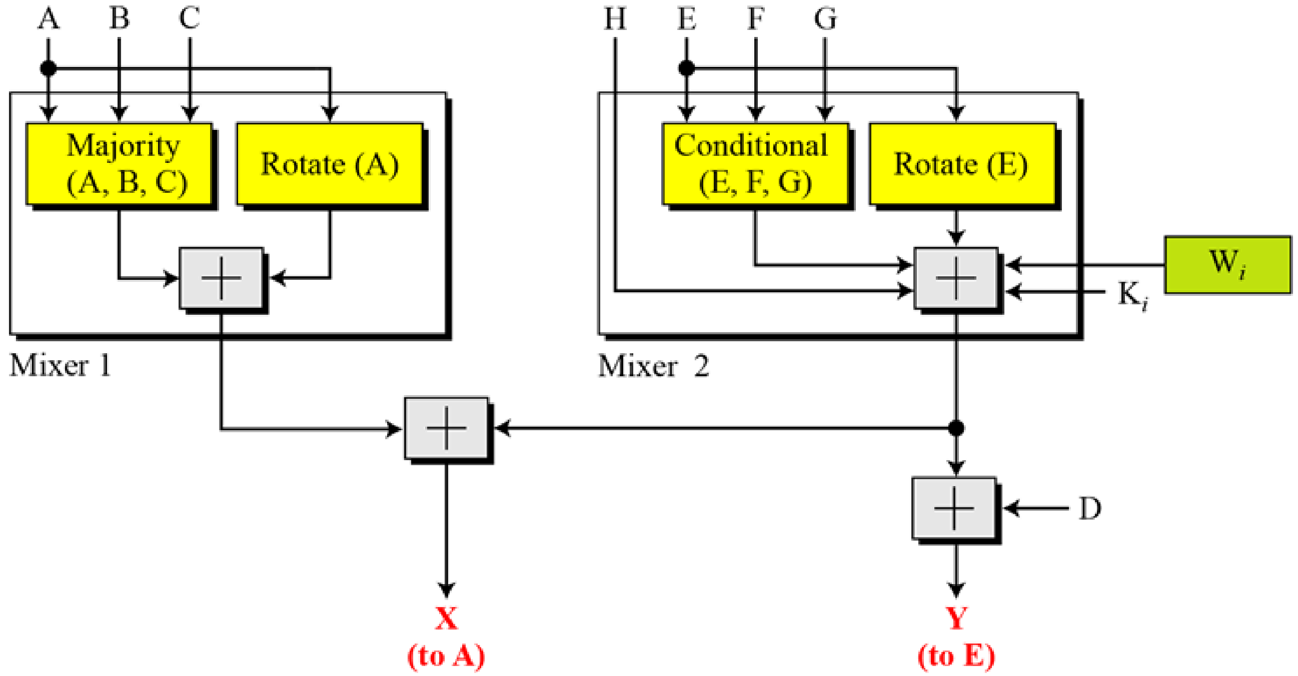
3 SHA – 512

Structure of each round



428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817





Majority (x, y, z)

$(x \text{ AND } y) \oplus (y \text{ AND } z) \oplus (z \text{ AND } x)$

Rotate (x)

$\text{RotR}_{28}(x) \oplus \text{RotR}_{34}(x) \oplus \text{RotR}_{39}(x)$

Conditional (x, y, z)

$(x \text{ AND } y) \oplus (\text{NOT } x \text{ AND } z)$

$\boxed{+}$ addition modulo 2^{64}

$\text{RotR}_i(x)$: Right-rotation of the argument x by i bits

4 Hash Demonstrations

In [1]:

```
from hashlib import sha512
print(sha512('hello'.encode()).hexdigest())
```

```
9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca72323c3d99ba5c11d7c7acc6e14b8c5da0c466
3475c2e5c3adef46f73bcdec043
```

In [2]:

```
import hashlib
m = hashlib.sha512()
m.update(b"Message") #change message to what you want to encode
#If you want to use a variable use the other version of m.update()
variable = "Message"
m.update(variable.encode("utf8"))
hashedMessage = m.digest()
print(hashedMessage)
```

```
b'\xb5\xc3\xe4J\xda\xfe"\xad\xbc\xfd9\x10[\xcc*q3@;\x0ct(\x84KH\x862^.\xcd`\x9e~\xfbE\xc6\x81\x11\xa3
\x04\xa2\xb3]\xcbx\xdf\x0eg\x1eo\x1d\xf5\xc0\xad8\x13s _j<\xd8<@'
```

4 Hash Demonstrations

In [7]:
```
from IPython.display import Image, display
from PIL import Image as Img
import imagehash

display(Image(filename='hanoi.png'))
hash = imagehash.average_hash(Img.open('hanoi.png'))
print(hash)

display(Image(filename='hanoi2.png'))
hash = imagehash.average_hash(Img.open('hanoi2.png'))
print(hash)
```

HUST

THANK YOU !

hust.edu.vn   fb.com/dhbkhn