Network Security

MAC continued

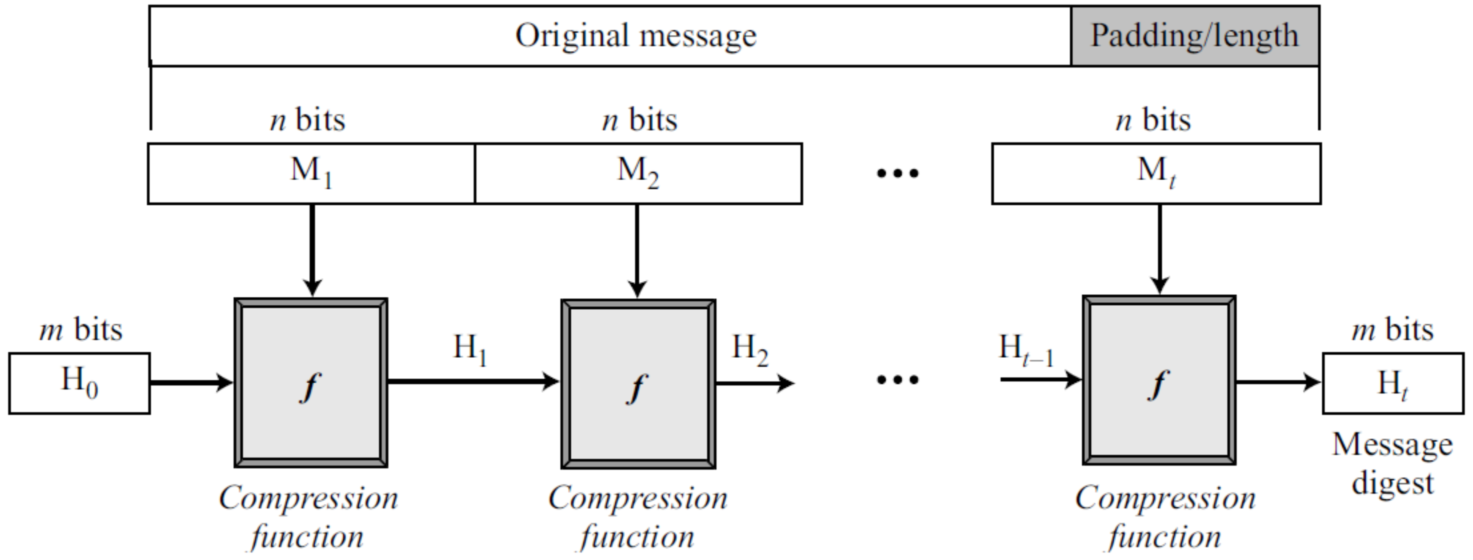
Kamalika Bhattacharjee

Cryptographic Hash Functions

- A **hash function** H accepts a variable-length block of data *M* as input and produces a fixed-size hash value *h* = H(*M*).
- A “good” hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random
- 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 value.
- **Iterated Hash Function**
    - All cryptographic hash functions need to create a fixed-size digest out of a variable-size message. This is best accomplished using iteration.
    - Instead of using a hash function with variable-size input, a function with fixed-size input is created and is used a necessary number of times.
    - The fixed-size input function is referred to as a ***compression function***. It compresses an n-bit string to create an m-bit string where n is normally greater than m.
    - The scheme is referred to as **an iterated cryptographic hash function**.

Merkle-Damgard Scheme

- Iterated Hash Function
- One basic requirement is that it should be computationally infeasible to find two distinct messages that hash to the same value.
- It is **collision resistant** if the compression function is collision resistant



Merkle-Damgard Scheme

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).

Two Groups of Compression Functions

- We can design a compression function that is collision resistant and insert it in the Merkle-Damgard scheme.
- Two Approaches:
  - First, the compression function is made from scratch: it is particularly designed for this purpose.
    - Message Digest (MD), Secure Hash Algorithm (SHA), RACE Integrity Primitives Evaluation Message Digest (RIPMED)
    - 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
  - Second, a symmetric-key block cipher serves as a compression function.
    - Rabin Scheme, Davies-Meyer Scheme, Matyas-Meyer-Oseas Scheme, Miyaguchi-Preneel Scheme

Message Digest (MD)

- Several hash algorithms designed by Ron Rivest.
- Referred to as MD2, MD4, and MD5, where MD stands for Message Digest.
- Last version, MD5, divides the message into blocks of 512 bits and creates a 128-bit digest.
- One basic requirement of any cryptographic hash function is that it should be computationally infeasible to find two distinct messages that hash to the same value.
- MD5 fails this requirement catastrophically; such collisions can be found in seconds on an ordinary home computer.
  - A message digest of size 128 bits is too small to resist collision attack.

Message Digest 5 (MD5)

- The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words); the message is padded so that its length is divisible by 512.
- Padding Technique
  - First, a single bit, 1, is appended to the end of the message.
  - This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512.
  - The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.

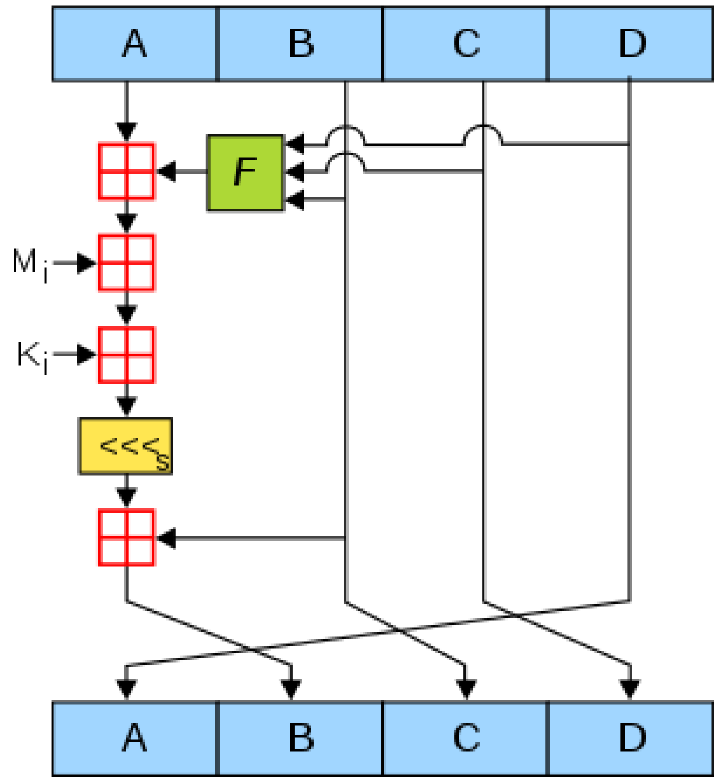
- Algorithm operates on a 128-bit state, divided into four 32-bit words, denoted *A*, *B*, *C*, and *D*. These are initialized to certain fixed constants.
- It uses each 512-bit message block in turn to modify the state.
- The processing of a message block consists of four similar stages (*rounds).*

Message Digest 5 (MD5)

- MD5 consists of 64 of these operations, grouped in four rounds of 16 operations.
- *F* is a nonlinear function; one function is used in each round.

$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$

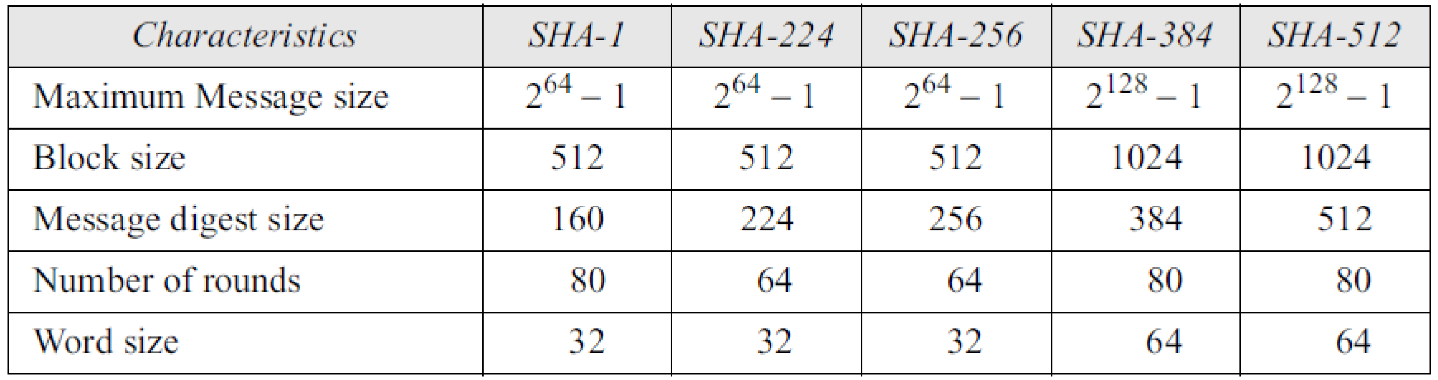
- M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation.
- $<<<_s$ denotes a left bit rotation by *s* places; *s* varies for each operation.
- ⊞ denotes addition modulo 2^32



One MD5 operation

Secure Hash Algorithm (SHA)

- Standard developed by the National Institute of Standards and Technology (NIST) and published as a Federal Information Processing standard (FIP 180).
- Sometimes referred to as Secure Hash Standard (SHS). The standard is mostly based on MD5.



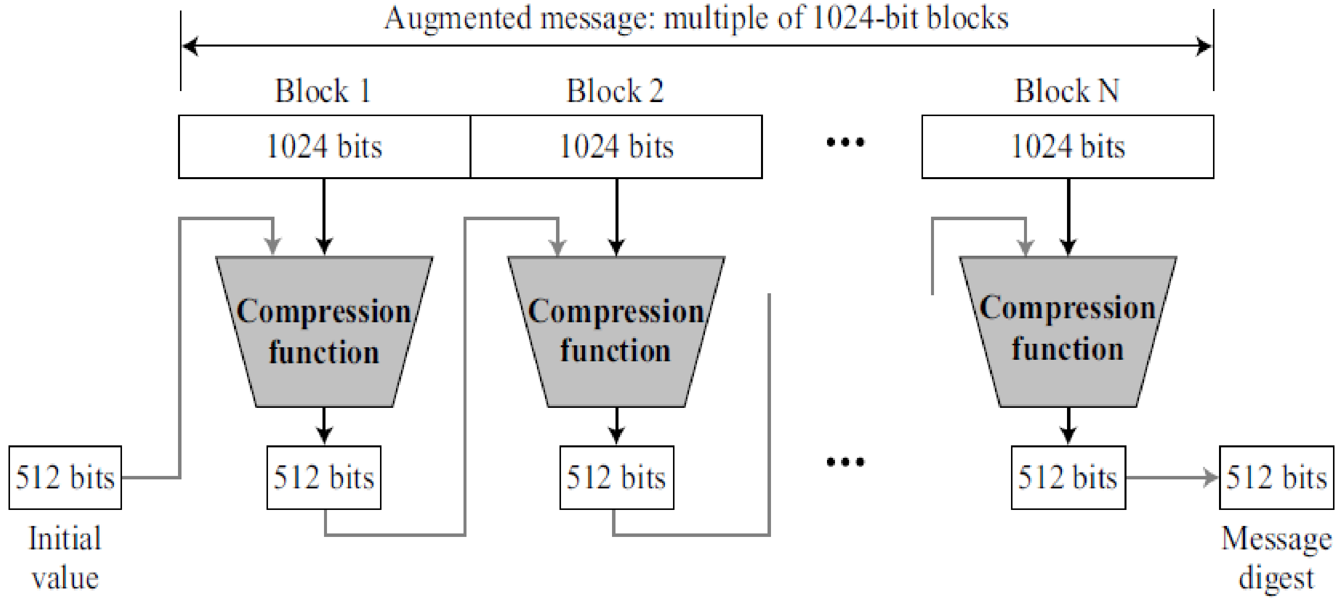
Characteristics	*SHA-1*	*SHA-224*	*SHA-256*	*SHA-384*	*SHA-512*
Maximum Message size	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

All of these versions have the same structure.

SHA-512

- SHA-512 creates a digest of 512 bits from a multiple-block message. Each block is 1024 bits in length.
- SHA-512 insists that the length of the original message be less than 2^128 bits.

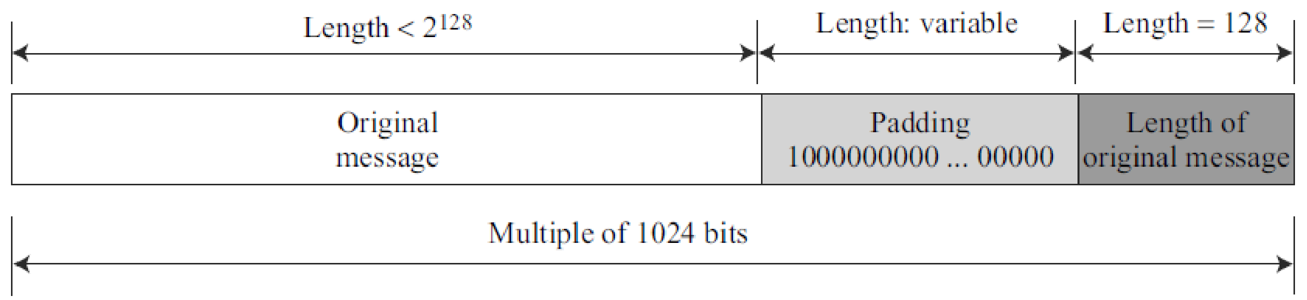
How many pages are occupied by a message of 2^128 bits?
Consider page size 2048 bits



Message digest creation SHA-512

SHA-512

- Length Field and Padding:
  - Before the message digest can be created, SHA-512 requires the addition of a 128-bit unsigned-integer length field to the message that defines the length of the original message in bits.



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

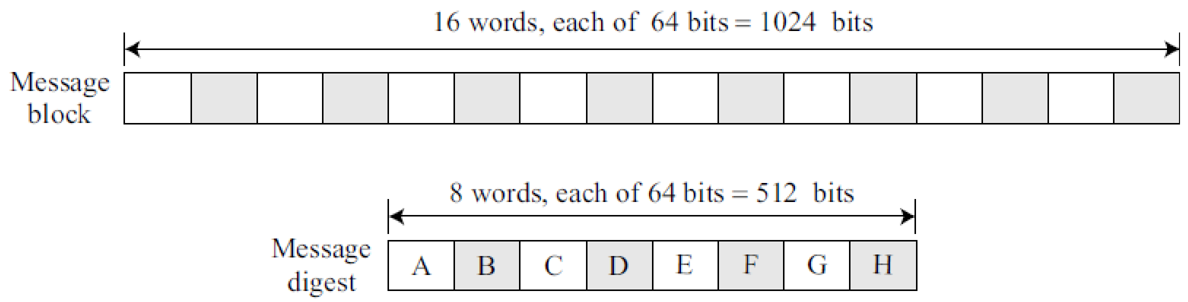
  - Before the addition of the length field, we need to pad the original message to make the length a multiple of 1024.
  - The length of the padding field can be calculated as follows. Let |M| be the length of the original message and |P| be the length of the padding field

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

  - The format of the padding is one 1 followed by the necessary number of 0s

SHA-512

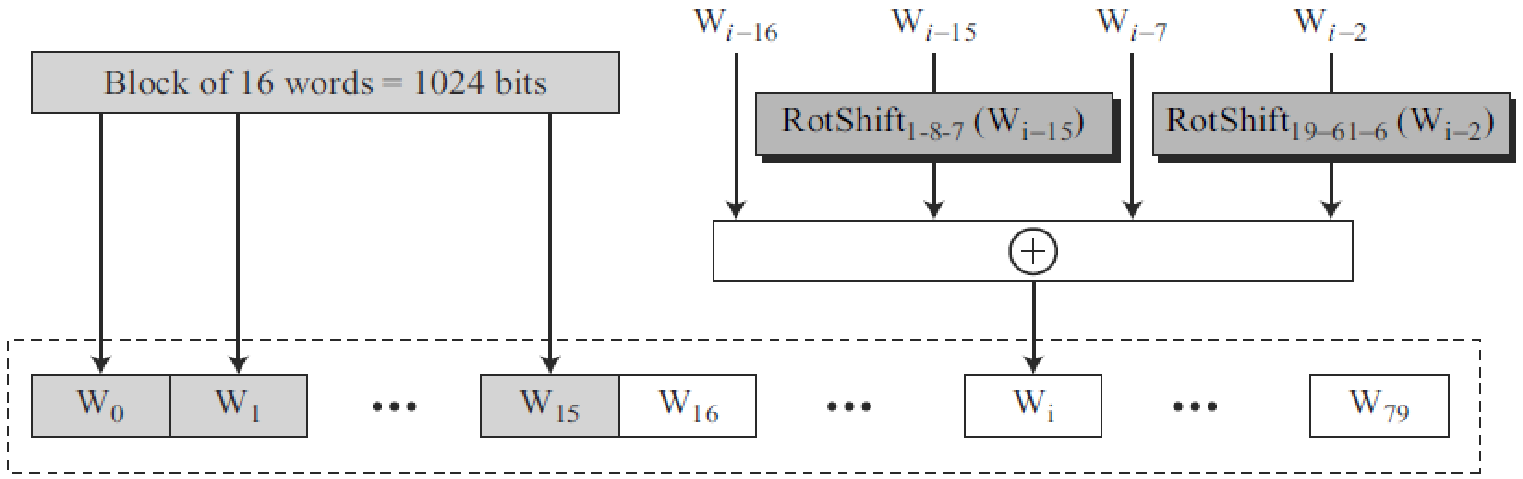
- SHA-252 is word-oriented. Each block is 16 words; the digest is only 8 words


16 words, each of 64 bits = 1024 bits
Message block
8 words, each of 64 bits = 512 bits
Message digest
A
B
C
D
E
F
G
H

- Before processing, each message block must be expanded.
- A block is made of 1024 bits, or sixteen 64-bit words. We need 80 words in the processing phase. So the 16-word block needs to be expanded to 80 words
- The 1024-bit block becomes the first 16 words; the rest of the words come from already-made words according to the operation

SHA-512

- Word expansion in SHA-512



W_{i-16}
W_{i-15}
W_{i-7}
W_{i-2}
Block of 16 words = 1024 bits
$\text{RotShift}_{1\text{-}8\text{-}7}(W_{i-15})$
$\text{RotShift}_{19\text{-}61\text{-}6}(W_{i-2})$
$\oplus$
W_0
W_1
W_{15}
W_{16}
W_i
W_{79}

RotRi(x) is actually a circular shiftright operation.

$\text{RotShift}_{l\text{-}m\text{-}n}(x)$: $\boldsymbol{RotR_l(x)} \oplus \boldsymbol{RotR_m(x)} \oplus \boldsymbol{ShL_n(x)}$

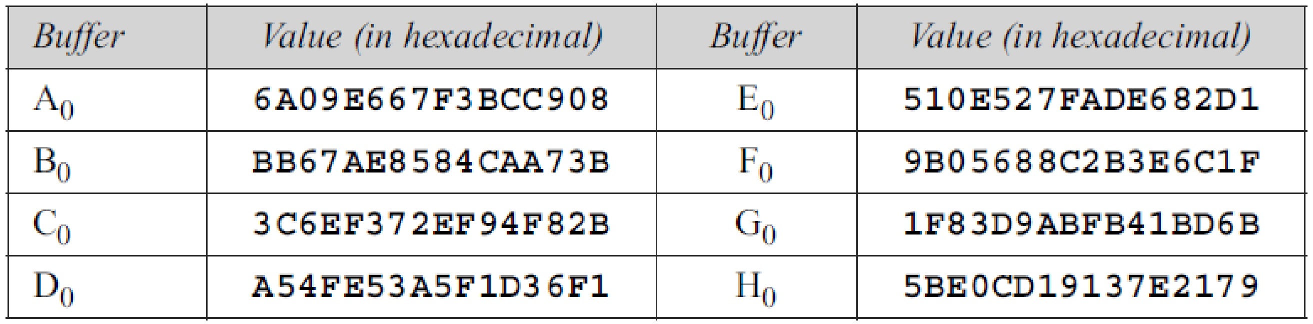
$\boldsymbol{RotR_i(x)}$: Right-rotation of the argument x by i bits
$\boldsymbol{ShL_i(x)}$: Shift-left of the argument x by i bits and padding the left by 0's.

Example:

$$W_{60} = W_{44} \oplus \text{RotShift}_{1\text{-}8\text{-}7}(W_{45}) \oplus W_{53} \oplus \text{RotShift}_{19\text{-}61\text{-}6}(W_{58})$$

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

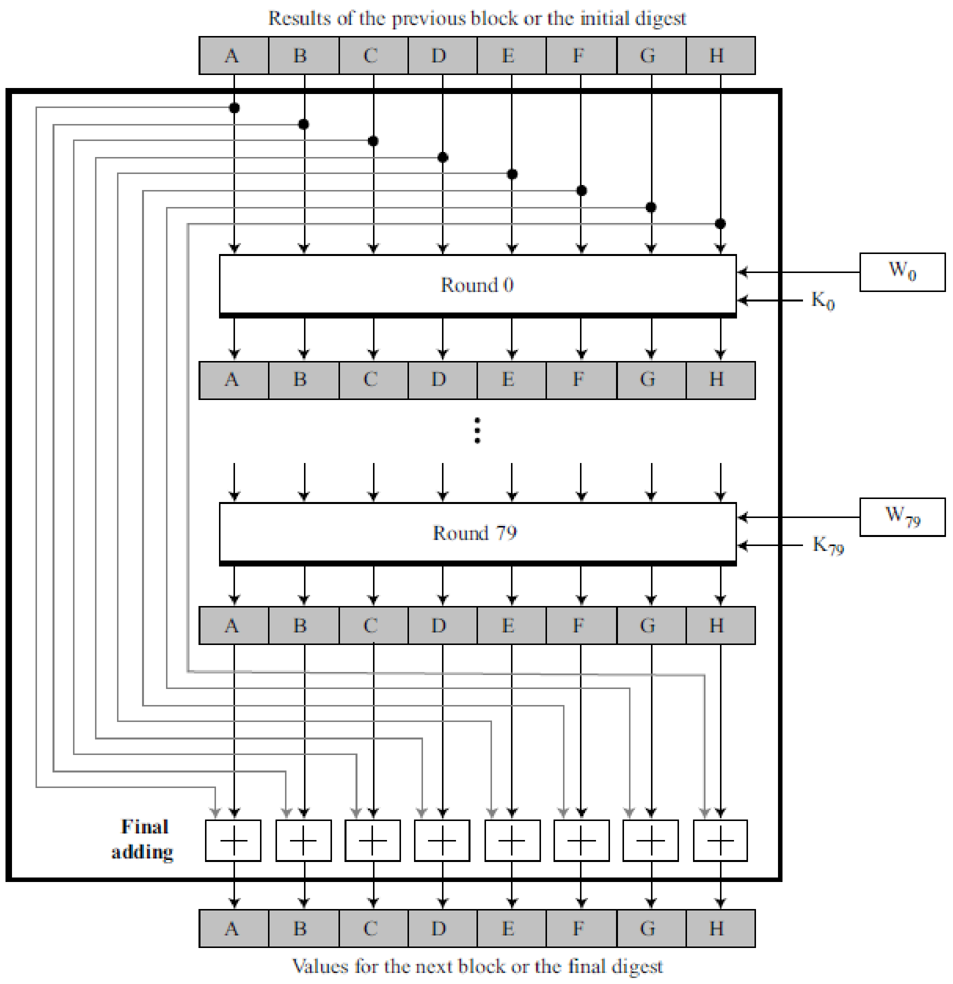
  - The values are calculated from the first eight prime numbers (2, 3, 5, 7, 11, 13, 17, and 19).
  - Each value is the fraction part of the square root of the corresponding prime number after converting to binary and keeping only the first 64 bits.
  - Example: Eighth prime is 19, with the square root => 4.35889894354. Converting the number to binary with only 64 bits in the fraction part, we get

$$(100.0101\ 1011\ 1110\ \ldots\ 1001)_2 \quad \rightarrow \quad (4.5BE0CD19137E2179)_{16}$$

    - SHA-512 keeps the fraction part as an unsigned integer

SHA-512

- Compression Function
    - creates a 512-bit (eight 64-bit words) message digest from a multiple-block message where each block is 1024 bits.
    - In each round, the contents of eight previous buffers, one word from the expanded block (W_i), and one 64-bit constant (K_i) are mixed together and then operated on to create a new set of eight buffers.
    - At the beginning of processing, the values of the eight buffers are saved into eight temporary variables.
    - At the end of the processing (after step 79), these values are added to the values created from step 79



SHA-512

- Majority function takes three corresponding bits in three buffers (A, B, and C) and calculates

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

  - The resulting bit is the majority of three bits. If two or three bits are 1's, the resulting bit is 1; otherwise it is 0.

- Conditional function takes three corresponding bits in three buffers (E, F, and G) and calculates

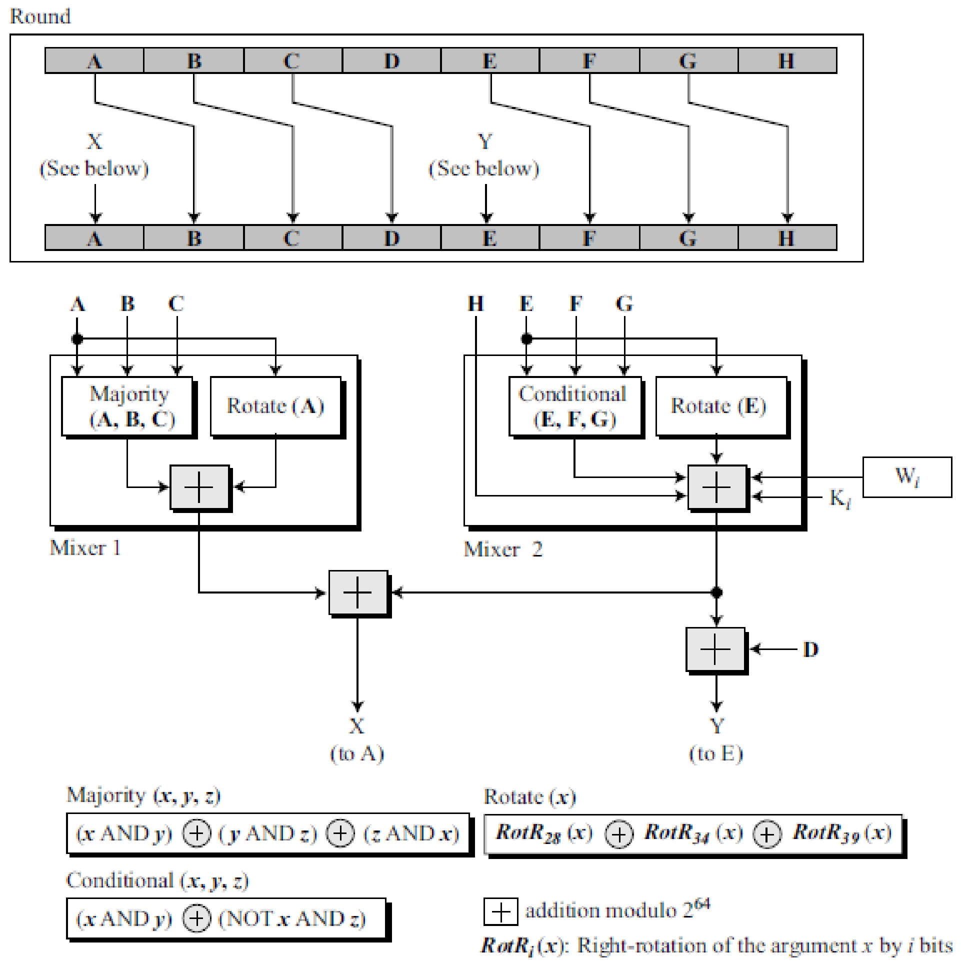
$$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$$

The resulting bit is the logic "If E_j then F_j; else G_j".

- Rotate function

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

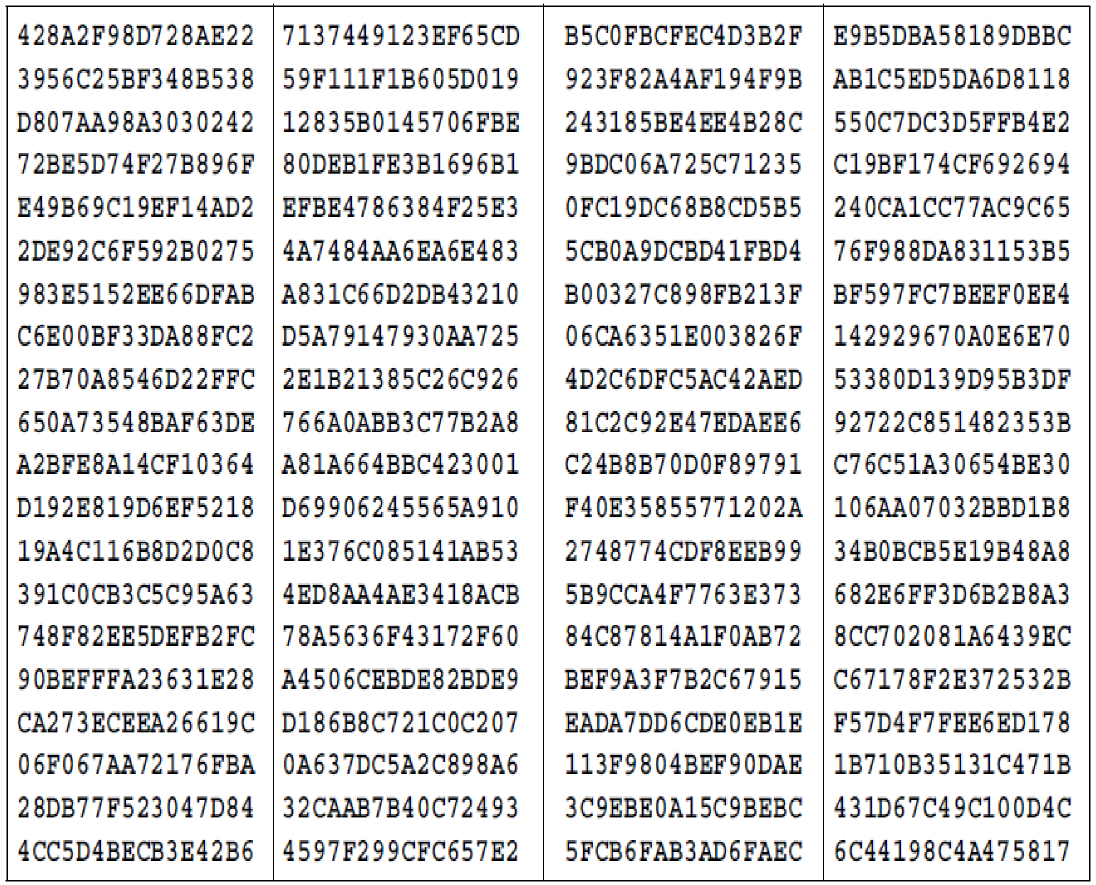
$$\text{Rotate (E): } RotR_{28}(E) \oplus RotR_{34}(E) \oplus RotR_{29}(E)$$



SHA-512

- The values of 80 constants are calculated from the first 80 prime numbers (2, 3,…, 409).
- Each value is the fraction part of the cubic root of the corresponding prime number after converting it to binary and keeping only the first 64 bits

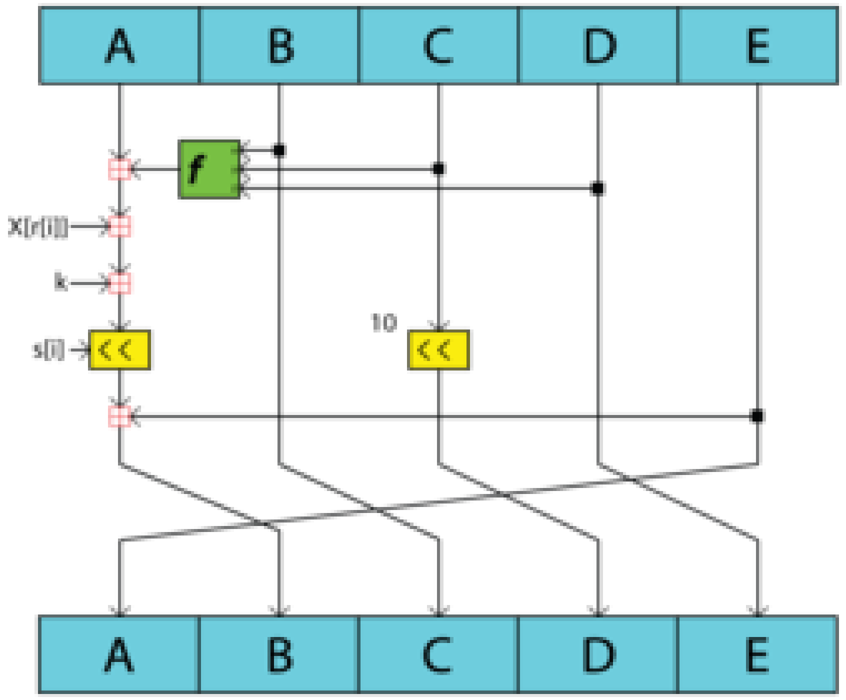
- With a message digest of 512 bits, SHA-512 expected to be resistant to all attacks, including collision attacks.
- It has been claimed that this version's improved design makes it more efficient and more secure than the previous versions. However, more research and testing are needed to confirm this claim



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

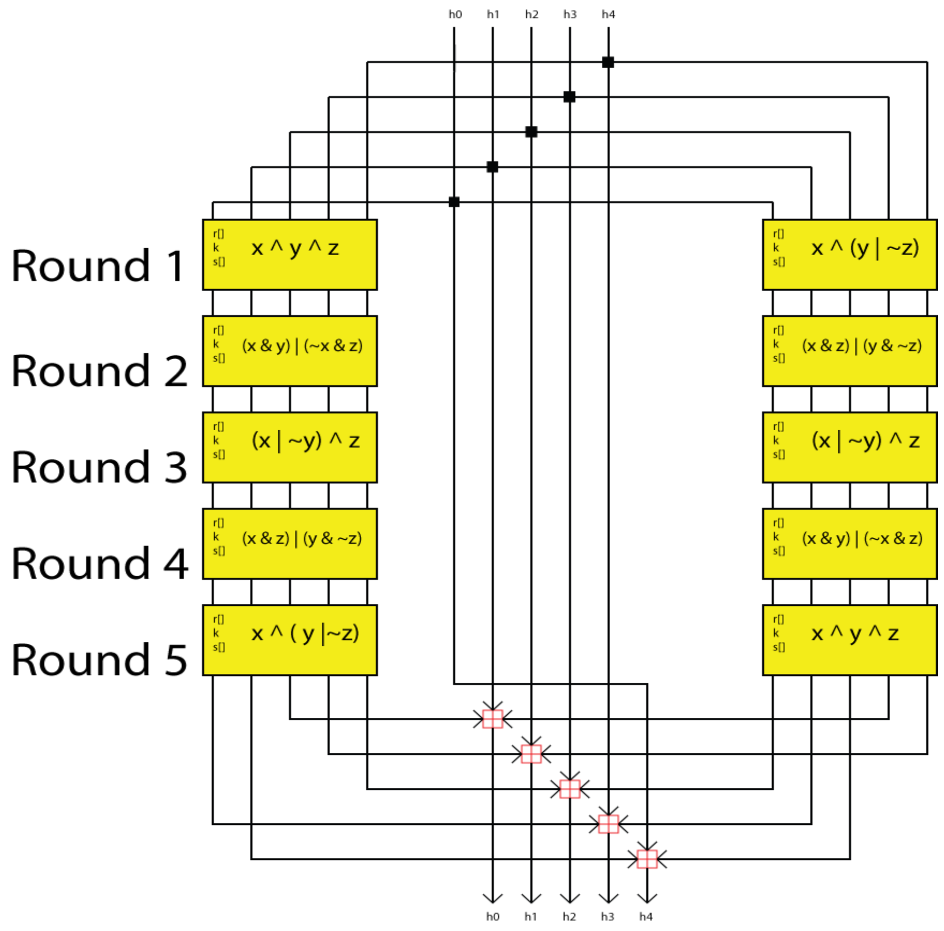
RIPEMD

- RACE Integrity Primitives Evaluation Message Digest (RIPMED)
- RIPEMD-160
  - uses the same structure as MD5 but uses two parallel lines of execution
  - used in the Bitcoin standard.
  - It is a strengthened version of the RIPEMD algorithm which produces a 128 bit hash digest while the RIPEMD-160 algorithm produces a 160-bit output
  - The compression function consists of 80 stages made up of 5 blocks that run 16 times each.
  - This pattern runs twice with the results being combined at the bottom using modulo 32 addition

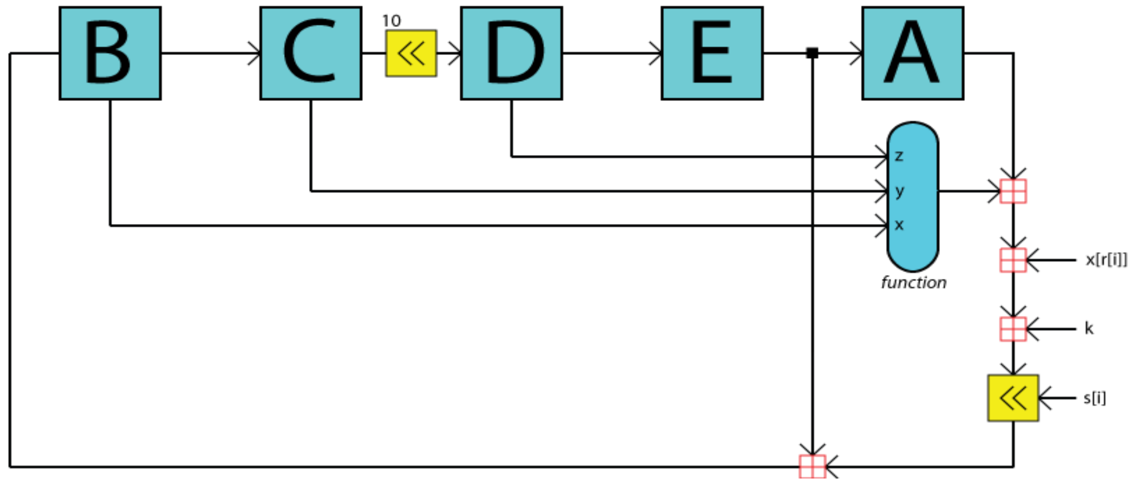


A sub-block from the compression function of the RIPEMD-160 hash algorithm

RIPEMD-160



The full compression function



A sub-block from the compression function

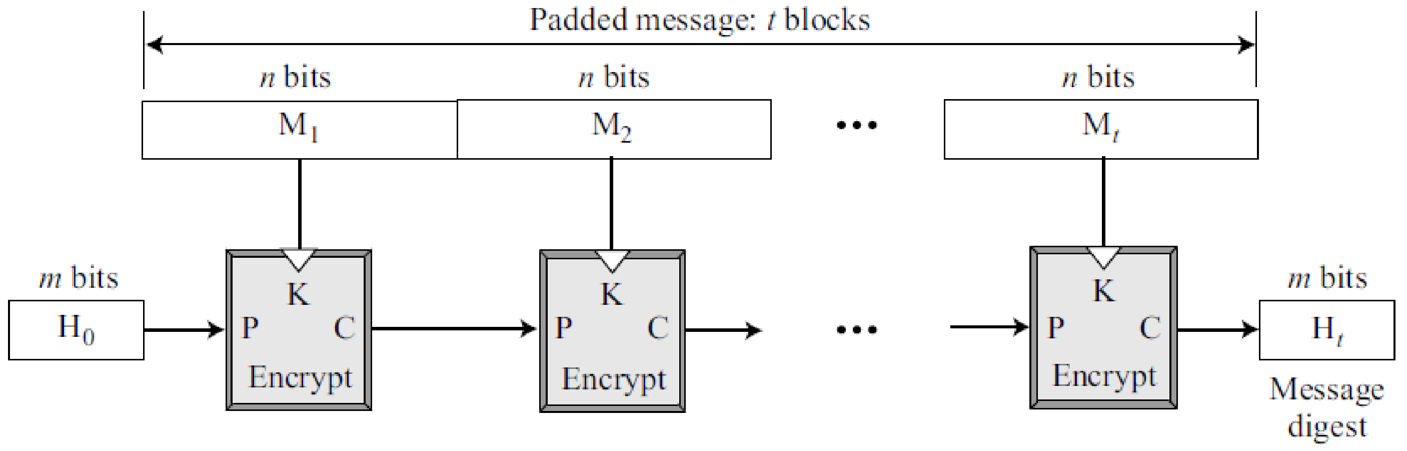
k values for left:	k values for right:
1.0x00000000	1.0x50A28BE6
2.0x5A827999	2.0x5C4DD124
3.0x6ED9EBA1	3.0x6D703EF3
4.0X8F1BBCDC	4.0x7A6D76E9
5.0XA953FD4E	5.0x00000000

Hash Functions Based on Block Ciphers

- An iterated cryptographic hash function can use a symmetric-key block cipher as a compression function.
- Idea is to use from the several secure symmetric-key block ciphers, such as triple DES or AES, to make a one-way function instead of creating a new compression function
- The block cipher in this case only performs encryption
- Example: Whirlpool

Rabin Scheme

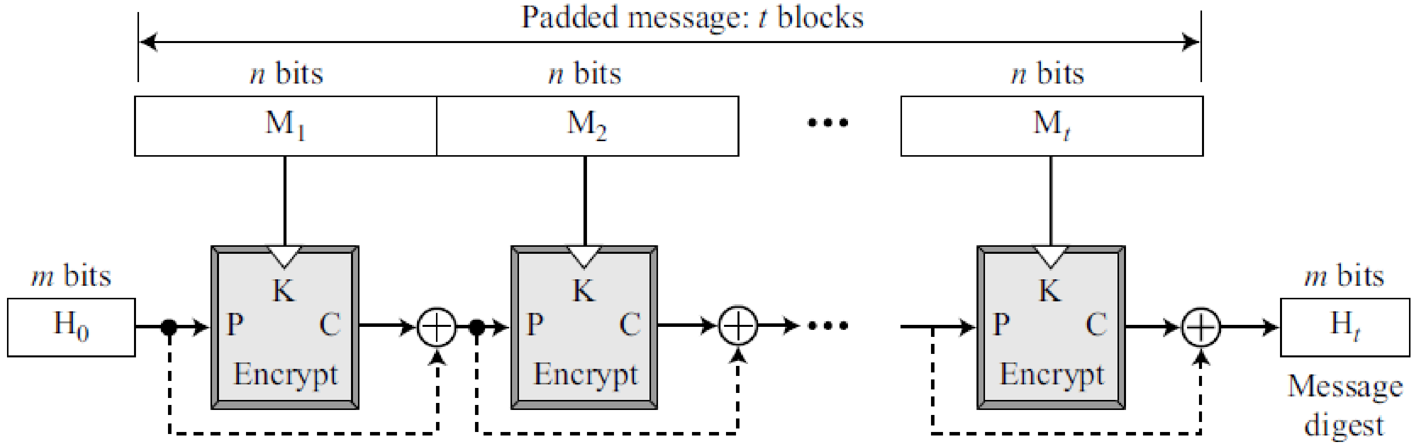
- based on the Merkle-Damgard scheme.
- The compression function is replaced by any encrypting cipher.
- Message block is used as the key; the previously created digest is used as the plaintext. The ciphertext is the new message digest.
- Size of the digest is the size of data block cipher in the underlying cryptosystem. For example, if DES is used as the block cipher, the size of the digest is only 64 bits.



Although the scheme is very simple, it is subject to a meet-in-the-middle attack, because the adversary can use the decryption algorithm of the cryptosystem.

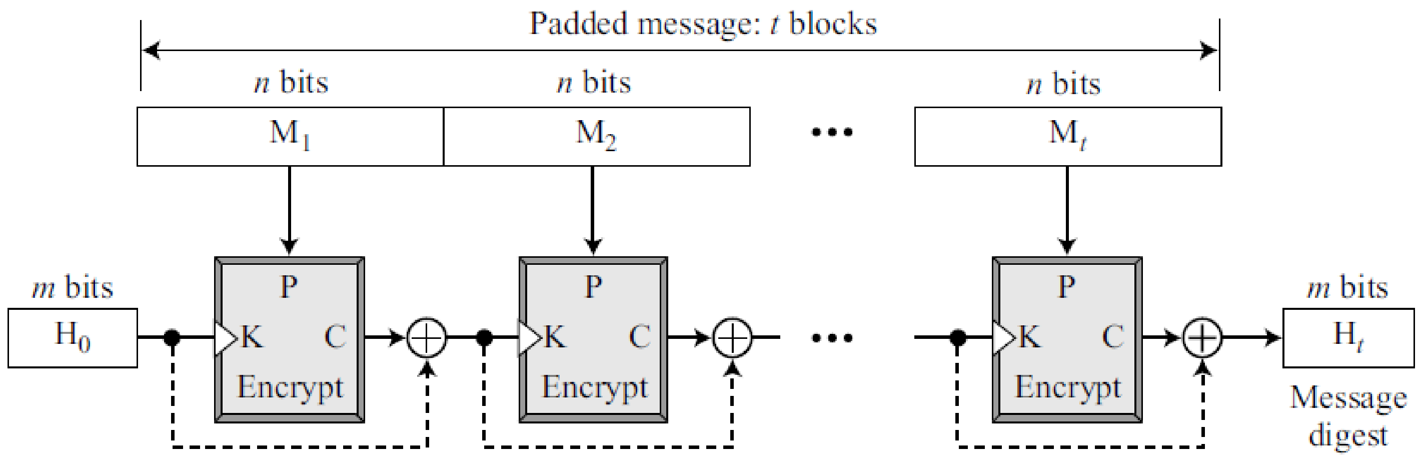
Davies-Meyer Scheme

- Same as the Rabin scheme except that it uses forward feed to protect against meet-in-the-middle attack



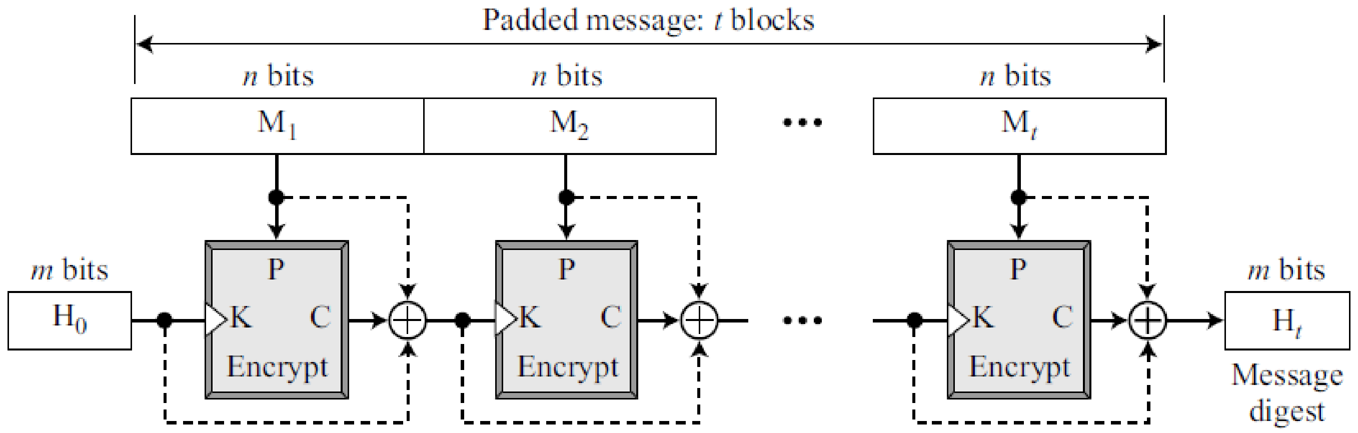
Matyas-Meyer-Oseas Scheme

- It is a dual version of the Davies-Meyer scheme: the digest block is used as the key to the cryptosystem.
- The scheme can be used if the data block and the cipher key are the same size. AES is a good candidate.



Miyaguchi-Preneel Scheme

- It 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



MACs BASED ON HASH FUNCTIONS

- In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function. The motivations for this interest are
  - Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers such as DES.
  - Library code for cryptographic hash functions is widely available
- A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose, because it does not rely on a secret key.
- There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. Most supported approach is HMAC.
  - HMAC has been issued as **RFC 2104**, has been chosen as the mandatory-to-implement MAC for **IP security**, and is used in other Internet protocols, such as **SSL**.
  - HMAC has also been issued as a NIST standard (FIPS 198).

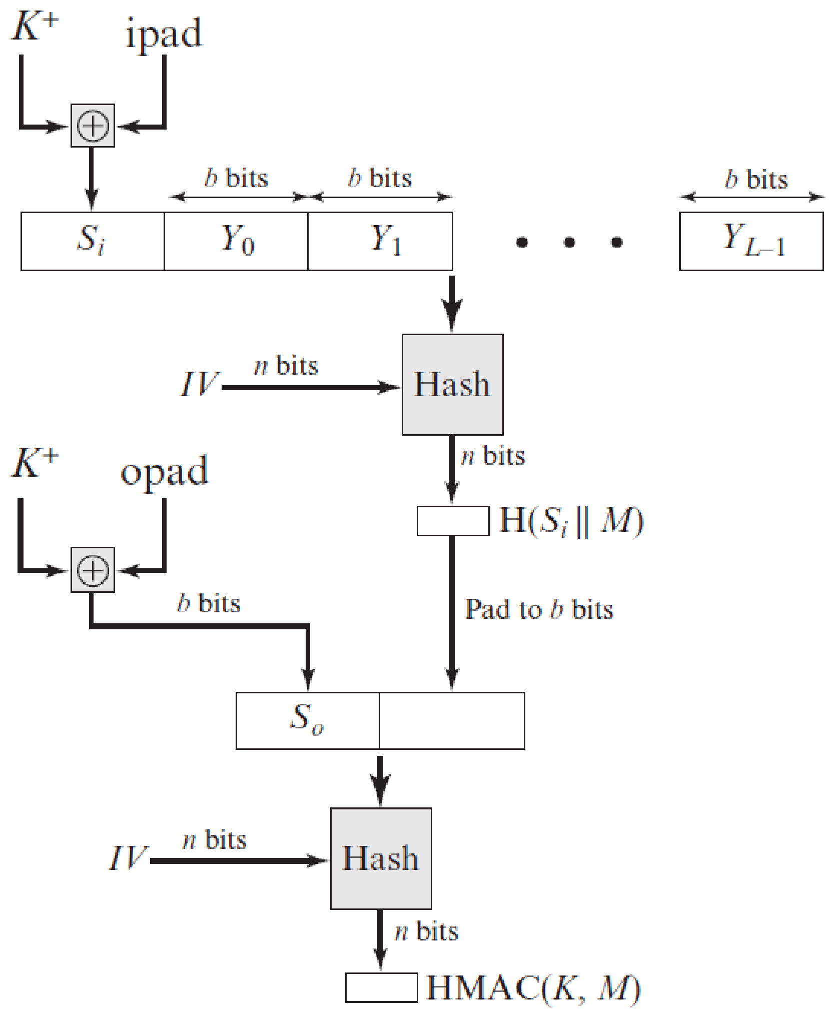
HMAC: Design Objectives

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
  - ➢ HMAC treats the hash function as a “black box”
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.
  - ➢ This is the main advantage of HMAC. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths

HMAC: Algorithm

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

H	=	embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
IV	=	initial value input to hash function
M	=	message input to HMAC (including the padding specified in the embedded hash function)
Y_i	=	i th block of M, $0 \le i \le (L - 1)$
L	=	number of blocks in M
b	=	number of bits in a block
n	=	length of hash code produced by embedded hash function
K	=	secret key; recommended length is $\ge n$; if key length is greater than b, the key is input to the hash function to produce an n-bit key
K^+	=	K padded with zeros on the left so that the result is b bits in length
ipad	=	00110110 (36 in hexadecimal) repeated $b/8$ times
opad	=	01011100 (5C in hexadecimal) repeated $b/8$ times

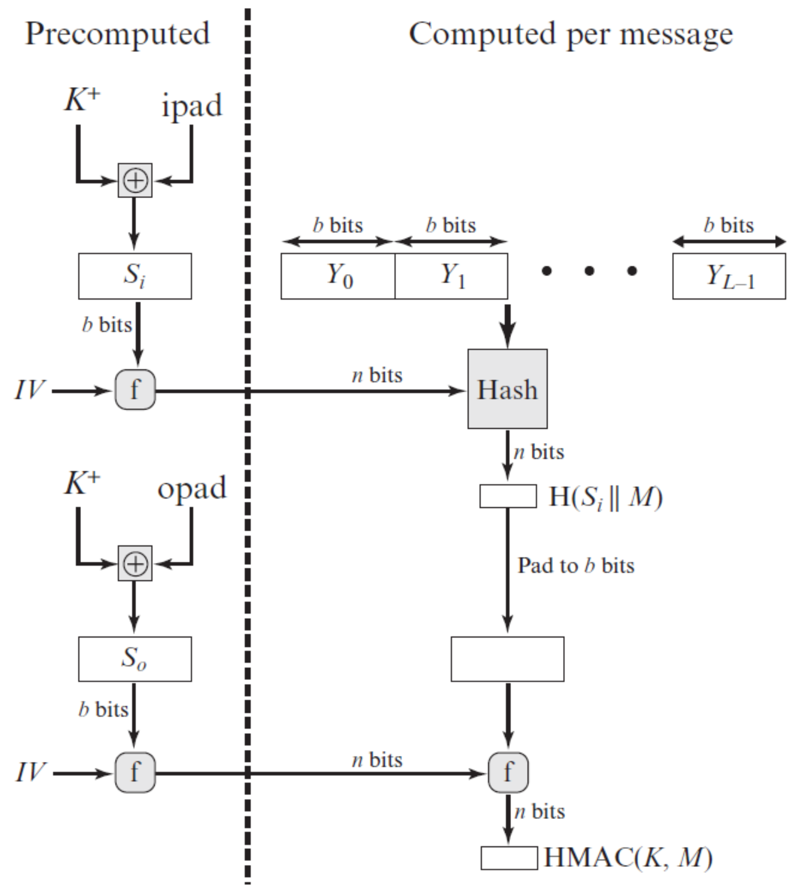


HMAC: Algorithm

$$\mathrm{HMAC}(K, M) = \mathrm{H}[(K^{+} \oplus \mathrm{opad}) \| \mathrm{H}[(K^{+} \oplus \mathrm{ipad}) \| M]]$$

- XOR with ipad and opad result in flipping one-half of the bits of *K*.
- By passing *Si* and *So* through the compression function of the hash algorithm, pseudorandomly generate two keys from *K*.
- f(cv, block) is the compression function for the hash function, which takes as arguments a chaining variable of *n* bits and a block of *b* bits and produces a chaining variable of *n* bits.
- These quantities only need to be computed initially and every time the key changes.

➢ only one additional instance of the compression function is added



Efficient Implementation of HMAC

HMAC: Security

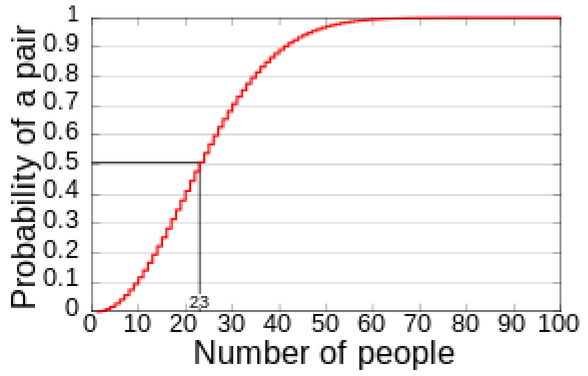
- Depends in some way on the cryptographic strength of the underlying hash function.
- The security is expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-tag pairs created with the same key.
- Proved: for a given level of effort (time, message–tag pairs) on messages generated by a legitimate user and seen by the attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function.
    - The attacker is able to compute an output of the compression function even with an *IV* that is random, secret, and unknown to the attacker.
    - The attacker finds collisions in the hash function even when the *IV* is random and secret.

HMAC: Security

- First Case:
  - The attacker is able to compute an output of the compression function even with an *IV* that is random, secret, and unknown to the attacker.
  - Compression function as equivalent to the hash function applied to a message consisting of a single *b*-bit block.
  - For this attack, the *IV* of the hash function is replaced by a secret, random value of *n* bits.
  - An attack on this hash function requires either a brute-force attack on the key, which is a level of effort on the order of 2^n, or a *birthday attack*

HMAC: Security

- Second Case: The attacker finds collisions in the hash function even when the *IV* is random and secret.
  - Attacker is looking for two messages *M* and *M′* that produce the same hash: H(*M*) = H(*M′*), known as ***Birthday Attack.***
  - The effort required is explained by a mathematical result referred to as the **birthday paradox**.
  - Counterintuitively, the probability of a shared birthday exceeds 50% in a group of only 23 people.
  - If we choose random variables from a uniform distribution in the range 0 through *n* - 1, then the probability that a repeated element is encountered exceeds 0.5 after $\sqrt{n}$ choices have been made.
  - For an *m*-bit hash value, if we pick data blocks at random, we can expect to find two data blocks with the same hash value within $\sqrt{2^m} = 2^{m/2}$ $_2$attempts.



Counter-intuitively, the probability that at least one student has the same birthday as *any* other student on any day is

$$1 - \frac{365!}{(365-n)! \cdot 365^n}$$

That is around 70% for n=30

HMAC: Security

- Second Case:
  - On this basis, does this mean that a 128-bit hash function such as MD5 is unsuitable for HMAC?
  - No!
  - To attack MD5, the attacker can choose any set of messages and work on these off line on a dedicated computing facility to find a collision.
  - Attacker knows the hash algorithm and the default *IV*, the attacker can generate the hash code for each of the messages that the attacker generates.
  - To attack HMAC, the attacker cannot generate message/ code pairs off line because the attacker does not know *K*.
  - Therefore, the attacker must observe a sequence of messages generated by HMAC under the same key and perform the attack on these known messages.
  - For a hash code length of 128 bits, this requires to observe 2^72 bits generated using the same key.
  - On a 1-Gbps link, a continuous stream of messages with no change in key for about 150,000 years!