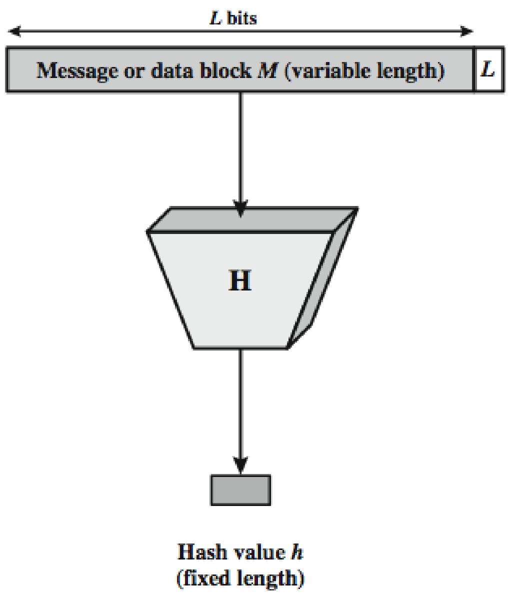
Hash Function

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



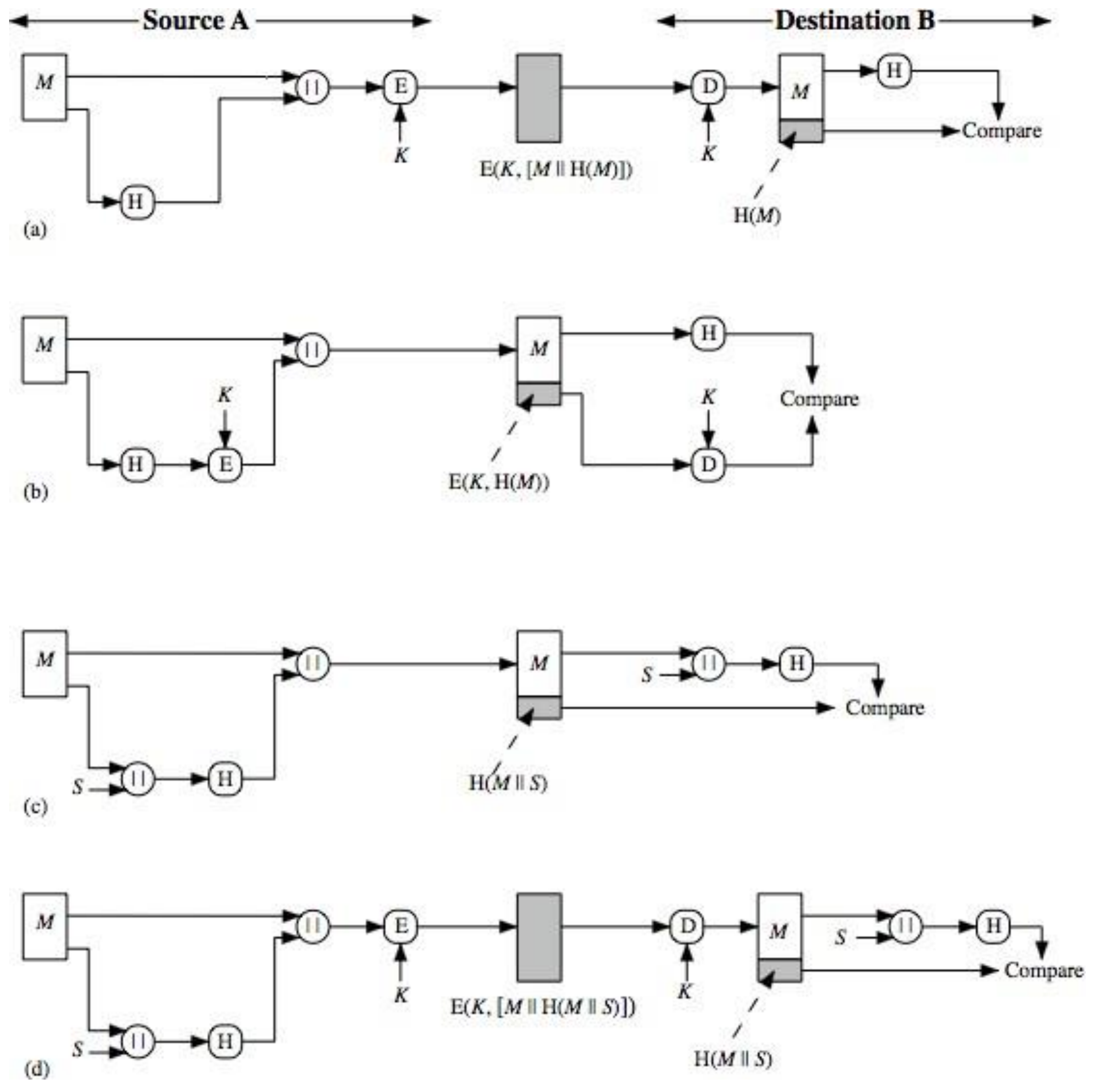
- Hash function used for security applications is referred to as a **cryptographic hash function**.
- A cryptographic hash function is an algorithm for which it is computationally infeasible to find either
  - A data object that maps to a pre-specified hash result (the one-way property) or
  - Two data 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.
- Figure depicts the general operation of a cryptographic hash function.

Applications of Cryptographic Hash Functions

The range of applications in which it is employed.

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.
- Hash function is used to provide message authentication.
- The hash function value is often referred to as a **message digest**.
- Variety of ways in which a hash code can be used to provide message authentication, as follows.



a) The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Confidentiality is also provided.

b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

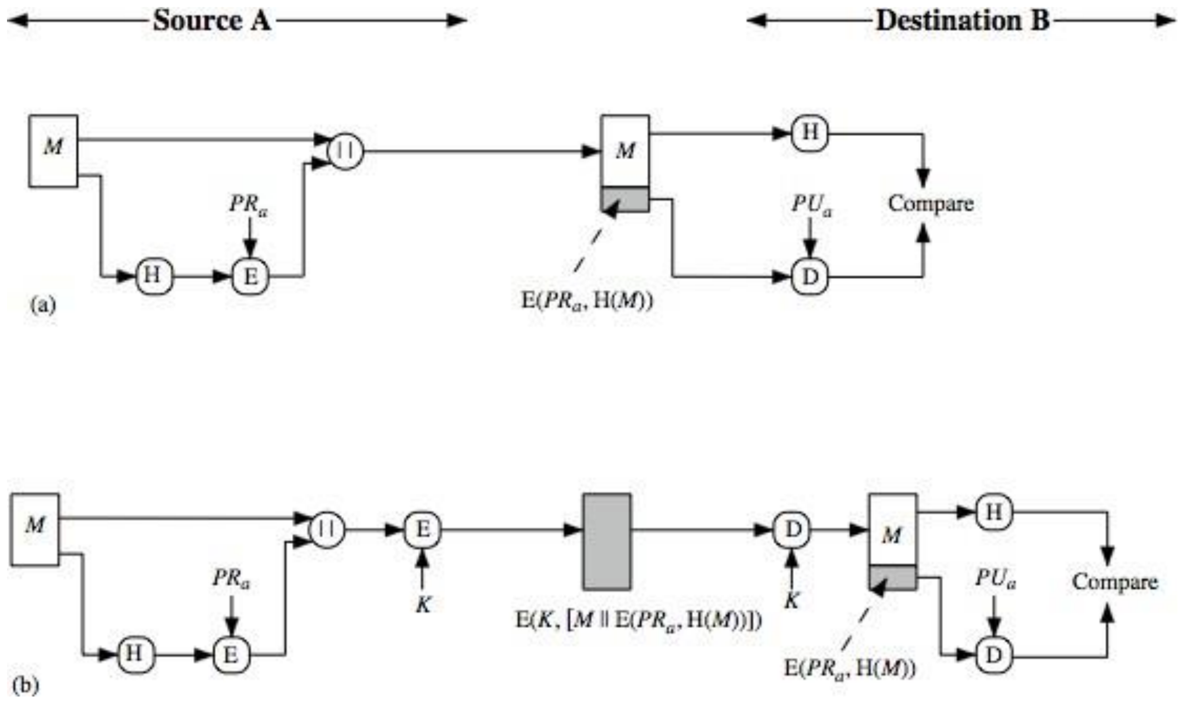
c) It is possible to use a hash function but no encryption for message authentication. Two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses, it can recomputed the hash value to verify. Opponent cannot generate a false message.

d) Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

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.



a) The hash code is encrypted, using public-key encryption with the sender's private key. This provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code.

b) If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key.

Other Applications

- Hash functions are commonly used to create a **one-way password file**.
- Hash functions can be used for **intrusion detection** and **virus detection**.
- A cryptographic hash function can be used to construct a **pseudorandom function (PRF)** or a **pseudorandom number generator (PRNG)**.

Simple Hash Functions

- Two simple, insecure hash functions are shown here.
- All hash functions operate using the following general principles.
  - The input (message, file, etc.) is viewed as a sequence of ***n***-bit blocks.
  - The input is processed one block at a time in an iterative fashion to produce an -bit hash function**.**

1. First Function

- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block.
- This can be expressed as

  $C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$

  Where

$C_i = i$th bit of the hash code, $1 \leq i \leq$ n
$m =$ number of n − bit blocks in the input
$b_{ij} = i$th bit in jth block
$\oplus =$ XOR operation

- This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check.
- It is reasonably effective for random data as a data integrity check.
- Each n-bit hash value is equally likely.
- Thus, the probability that a data error will result in an unchanged hash value is 2^{-n}.

2. Second Function

- A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed.
- The procedure can be summarized as follows.
  1) Initially set the n-bit hash value to zero.
  2) Process each successive n-bit block of data as follows:
     a. Rotate the current hash value to the left by one bit.
     b. XOR the block into the hash value.
- This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input.
- Although the second procedure provides a good measure of data integrity, it is virtually useless for data security.
- When an encrypted hash code is used with a plaintext message, it is an easy matter to produce a new message that yields that hash code.
- Simply prepare the desired alternate message and then append an n-bit block that forces the new message plus block to yield the desired hash code.

Security Requirements for Cryptographic Hash Functions

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that H(y) = h.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x).
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Security attack on Cryptographic Hash Function

Brute-Force Attacks

- A brute-force attack does not depend on the specific algorithm but depends only on bit length.

- In the case of a hash function, a brute-force attack depends only on the bit length of the hash value.
- A cryptanalysis in contrast, is an attack based on weaknesses in a particular cryptographic algorithm.
- We look first at brute-force attacks.

Preimage and Second Preimage Attacks

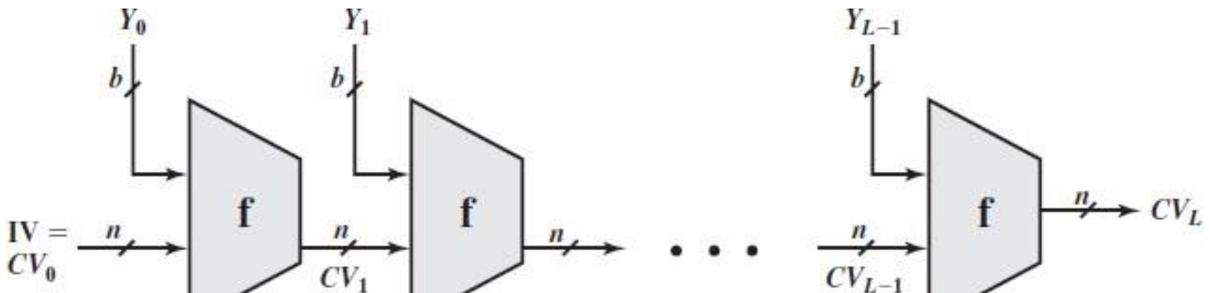
- Adversary wishes to find a value such that H(y) is equal to a given hash value h.
- The brute-force method is to pick values of y at random and try each value until a collision occurs.
- For an m-bit hash value, the level of effort is proportional to 2^m.
- Specifically, the adversary would have to try, on average, 2^{m-1} values of y to find one that generates a given hash value h.

Collision Resistant Attacks

- Adversary wishes to find two messages or data blocks, x and y, that yield the same hash function: H(x)=H(y).
- This turns out to require considerably less effort than a preimage or second preimage attack.
- The effort required is explained by a mathematical result referred to as the birthday paradox.
- Thus, 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 $\sqrt{2^m} = 2^{m/2}$ attempts.
- Yuval proposed the following strategy to exploit the **birthday paradox** in a collision resistant attack
  1. The source, A, is prepared to sign a legitimate message x by appending the appropriate m-bit hash code and encrypting that hash code with A's private key.
  2. The opponent generates $2^{m/2}$ variations x' of x, all of which convey essentially the same meaning, and stores the messages and their hash values.
  3. The opponent prepares a fraudulent message y for which A's signature is desired.
  4. The opponent generates minor variations y' of y, all of which convey essentially the same meaning. For each y', the opponent computes H(y'), checks for matches with any of the H(x') values, and continues until a match is found.
  5. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient.

Cryptanalysis

- As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search.
- The way to measure the resistance of a hash algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack.
- In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions.
- To understand these, we need to look at the overall structure of a typical secure hash function, indicated in Figure.



IV = Initial value	L = Number of input blocks
CV_i = Chaining variable	n = Length of hash code
Y_i = ith input block	b = Length of input block
f = Compression algorithm	

- Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f.
- Once that is done, the attack must take into account the fixed value of IV.
- The attack on f depends on exploiting its internal structure.
- Typically, as with symmetric block ciphers, f consists of a series of rounds of processing, so that the attack involves analysis of the pattern of bit changes from round to round.

Hash Functions Based on Cipher Block Chaining

- A number of proposals have been made for hash functions based on using a cipher block chaining technique, but without using the secret key. One of the first such proposals was that of Rabin.
- Divide a message M into fixed-size blocks M1, M2, … , M_N and use a symmetric encryption system such as DES to compute the hash code G as

$$H_0 = initial\ value$$
$$H_i = E(M_i, H_{i-1})$$
$$G = H_N$$

- This is similar to the CBC technique, but in this case, there is no secret key.
- As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.
- Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings.
- Here is the scenario: We assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is bits long.
  1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code G.
  2. Construct any desired message in the form Q_1, Q_2, … , Q_{N-2}.
  3. Compute $H_i = E(Q_i, H_{i-1})$ for $1 \leq i \leq (N-2)$.
  4. Generate $2^{m/2}$ random blocks; for each block X, compute $E(X,\ H_{N-2})$. Generate an additional $2^{m/2}$ random blocks; for each block Y, compute D (Y, G), where D is the decryption function corresponding to E.
  5. Based on the birthday paradox, with high probability there will be an X and Y such that $E(X, H_{N-2}) = D(Y, G)$.
  6. Form the message Q_1, Q_2, … , Q_{N-2}, X, Y, This message has the hash code G and therefore can be used with the intercepted encrypted signature.
- This form of attack is known as a **meet-in-the-middle-attack**.
- A number of researchers have proposed refinements intended to strengthen the basic block chaining approach. For example, Davies and Price describe the variation:

$$H_i = E(M_i, H_{i-1}) \oplus H_{i-1}$$

- Another variation, proposed is

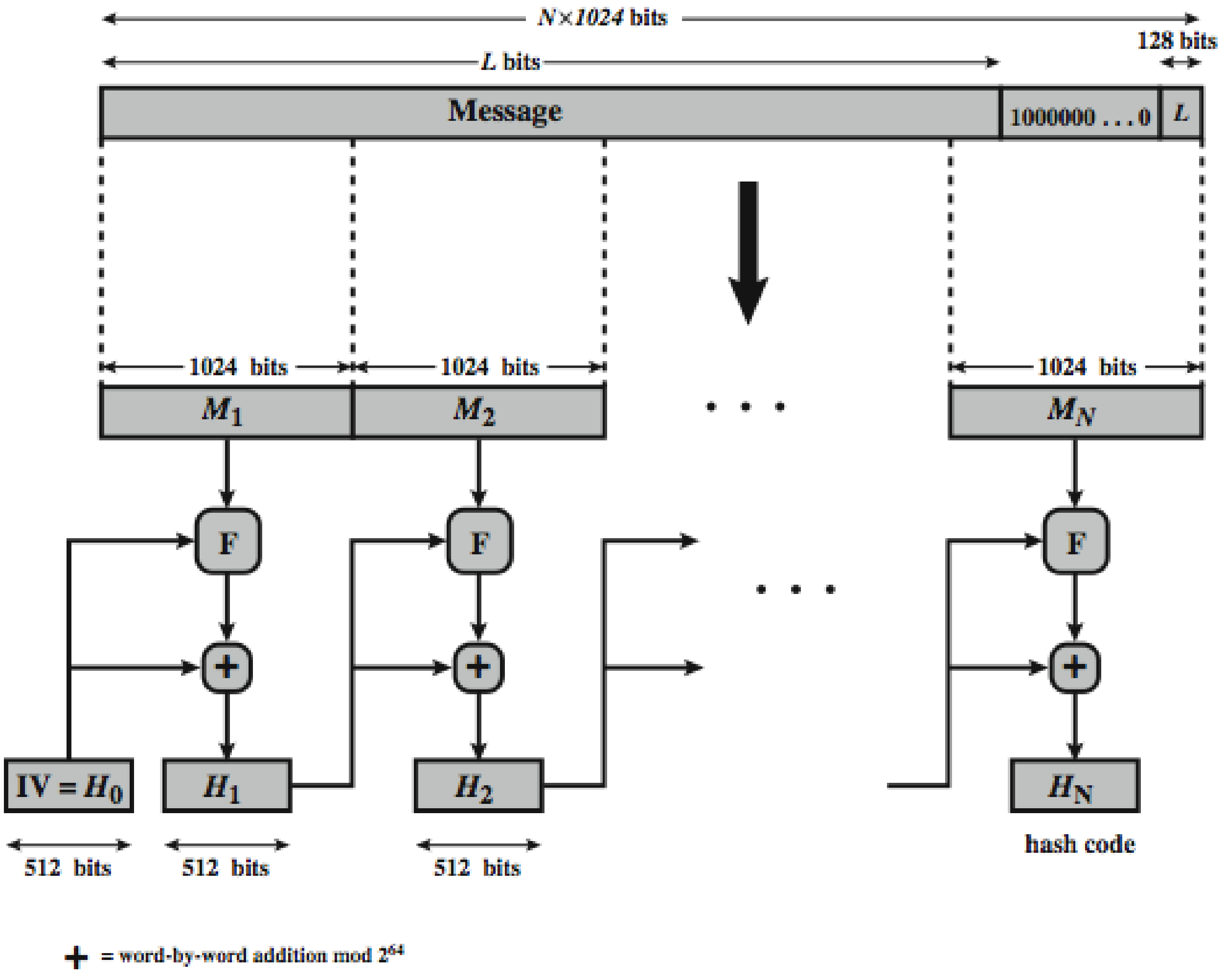
$$H_i = E(H_{i-1}, M_i) \oplus M_i$$

- However, both of these schemes have been shown to be vulnerable to a variety of attacks.

Secure Hash Algorithm (SHA)

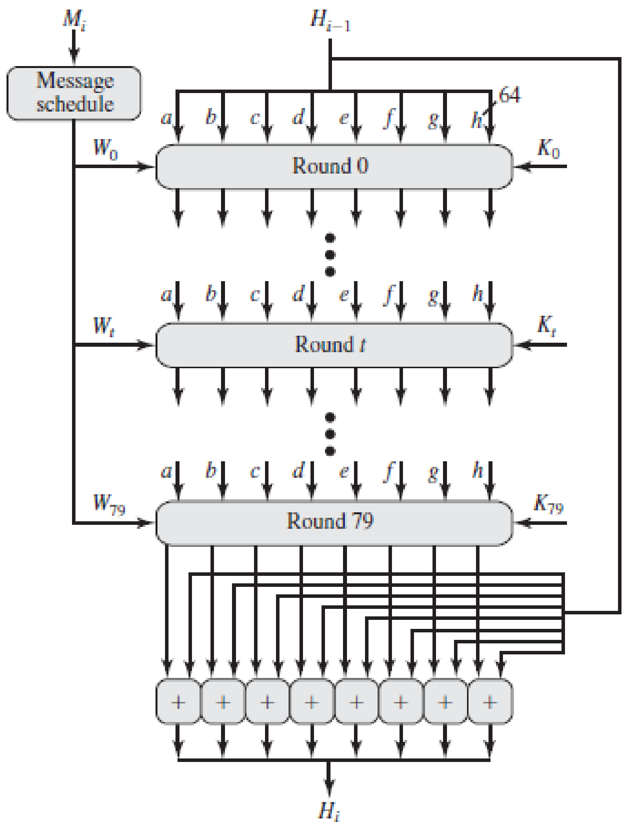
- SHA is based on the hash function MD4.
- The algorithm takes as input a message of maximum length of less than 2128bitsand produces a 512-bit message digest.
- The input is processed in 1024-bit blocks.

- The processing consists of the following steps:
  1. **Append padding bits.**
     - ✓ The message is padded so that its length is congruent to 896 modulo 1024.
     - ✓ The padding consists of a single 1-bit followed by the necessary number of 0-bits.
  2. **Append length.**
     - ✓ A block of 128 bits is appended to the message. This block contains the length of the original message (before the padding).
     - ✓ The message is now an integer multiple of 1024 bits in length.
- In the figure below, expanded message is represented as the sequence of 1024-bit blocks M1, M2,..., MN and the total length of the expanded message is N x 1024 bits.



  3. **Initialize hash buffer.**
     - ✓ A 512-bit buffer is used to hold intermediate and final results of the hash function.
     - ✓ The buffercan be represented as eight 64-bit registers (a, b, c, d, e, f, g, h).
     - ✓ These registers are initialized to the 64-bit integers(hexadecimal values) obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eightprime numbers.
  4. **Process message in 1024-bit (128-word) blocks.**
     - ✓ The heart of the algorithm is a module **F**that consists of 80 rounds.
- SHA has 80 rounds.
- Each round takes as input:
  - 512-bit buffer value (**H_{i-1}**)
  - 64-bit words **W_t**obtained from the current data block by message schedule.

- Additive constant K_t which represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers.
- The contents of the buffer are updated after every round.



SHA algorithm

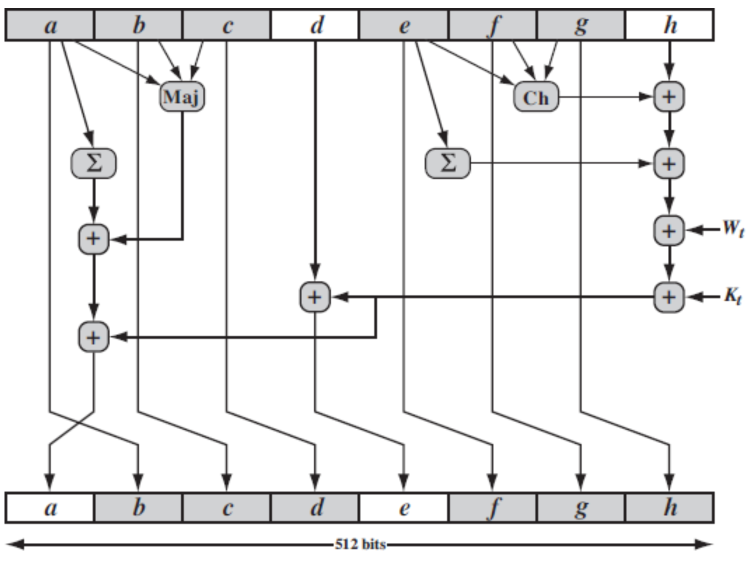
- The output of the eightieth round is added modulo 264to the input to the first round (Hi-1) to produce Hi.

5. Output.

✓ After all *N* 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest.

SHA-512 Round Function

- Each round updates the buffer in the following way:



$$
\begin{aligned}
T_1 &= h + \mathrm{Ch}(e, f, g) + \left(\textstyle\sum_1^{512} e\right) + W_t + K_t \\
T_2 &= \left(\textstyle\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

Where

$$
\begin{aligned}
t &= \text{step number}; 0 \leq t \leq 79 \\
\mathrm{Ch}(e, f, g) &= (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g) \\
&\quad \textit{the conditional function: If e then f else g}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Maj}(a, b, c) &= (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c) \\
&\quad \textit{the function is true only of the majority (two or three) of the} \\
&\quad \textit{arguments are true} \\
\left(\textstyle\sum_0^{512} a\right) &= \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a) \\
\left(\textstyle\sum_1^{512} e\right) &= \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e) \\
\mathrm{ROTR}^{n}(x) &= \text{circular right shift (rotation) of the 64-bit argument } x \text{ by } n \text{ bits} \\
W_t &= \text{a 64-bit word derived from the current 512-bit input block} \\
K_t &= \text{a 64-bit additive constant} \\
+ &= \text{addition modulo } 2^{64}
\end{aligned}
$$

Message Schedule

- The 64-bit word values Wtare derived from the 1024-bit message.
- The first16 values ofWt are taken directly from the 16 words of the current block. The remaining values are defined as follows:

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

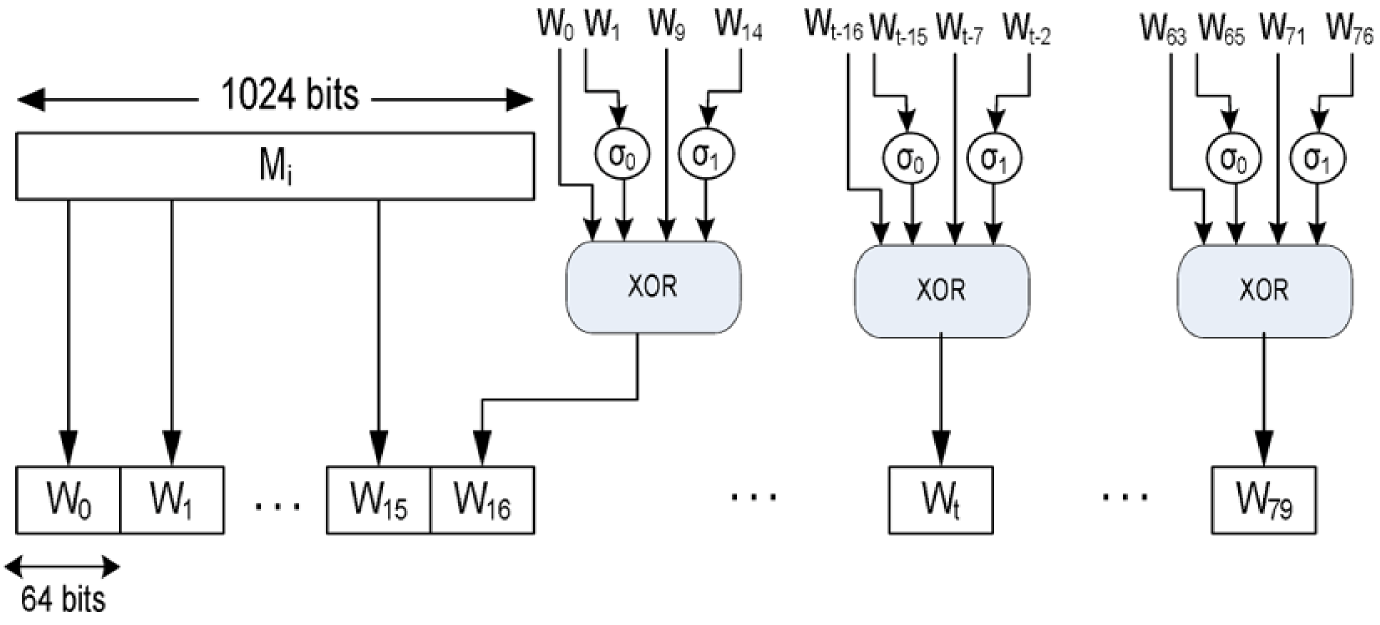
where

$$\sigma_0^{512}(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

$\mathrm{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits with padding by zeros on the right

$+$ = addition modulo 2^{64}



Message schedule

- The message schedule introduces a great deal of redundancy and interdependence into the message blocksthat are compressed, which complicates the task of finding a different message block that maps to the same compression function output.