Module 4 : Integrity, Authentication and Digital Certificates

A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value h = H(M).

Module 3 : Hashes, Message Digests and Digital Certificates

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.

Module 3 : Hashes, Message Digests and Digital Certificates

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.

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.

The kind of hash function needed for security applications is referred to as a cryptographic hash function.

The kind of hash function needed 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) a data object that maps to a pre-specified hash result (the one-way property)

or

The kind of hash function needed 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) a data object that maps to a pre-specified hash result (the one-way property)

or

(b) 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.

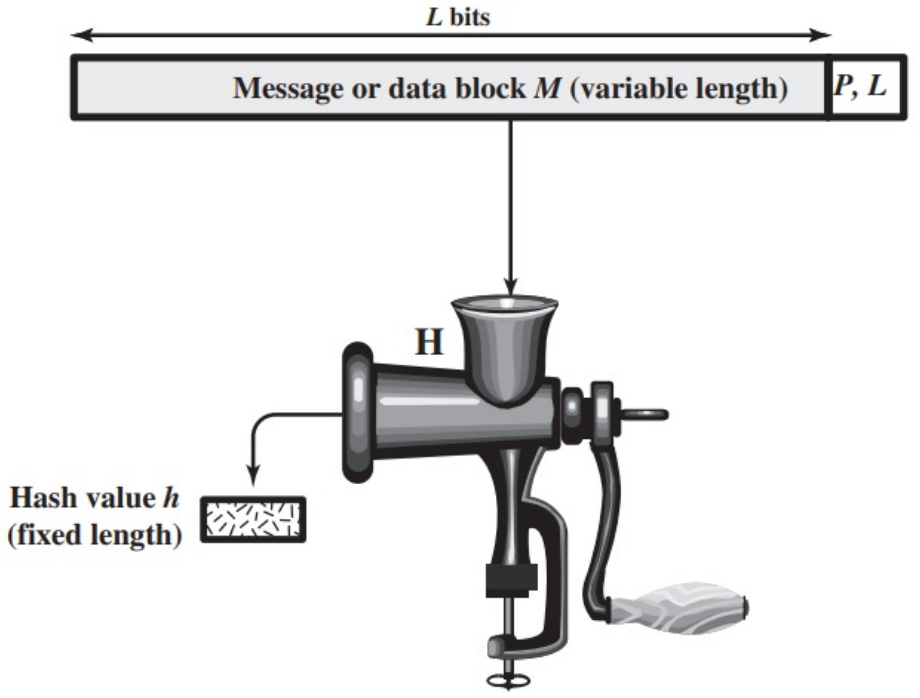


Figure depicts the general operation of a cryptographic hash function. Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original mes

Applications of Cryptographic Hash Functions

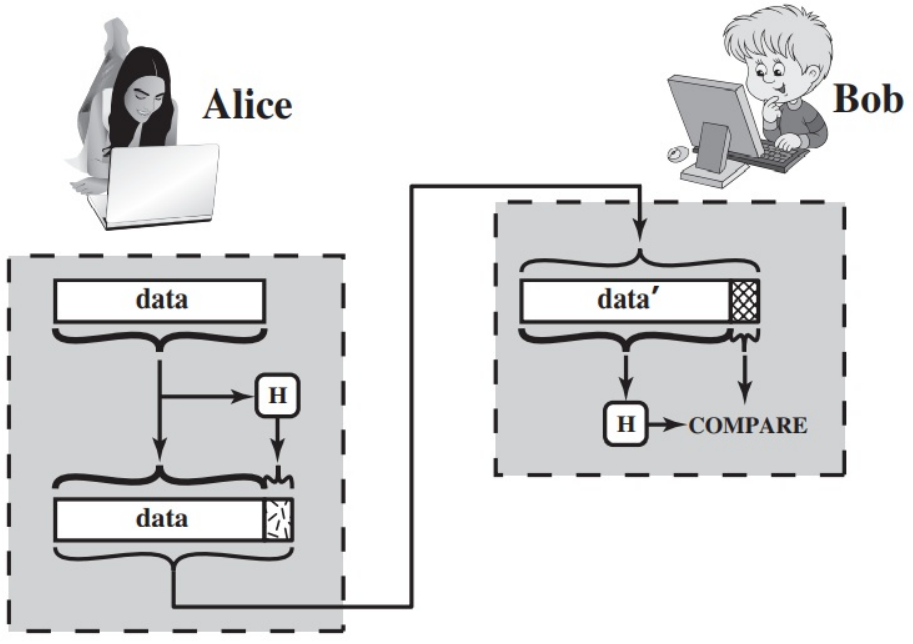
To better understand some of the requirements and security implications for cryptographic hash functions, it is useful to look at the range of applications in which it is employed.

1. 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 (i.e., contain no modification, insertion, deletion, or replay).

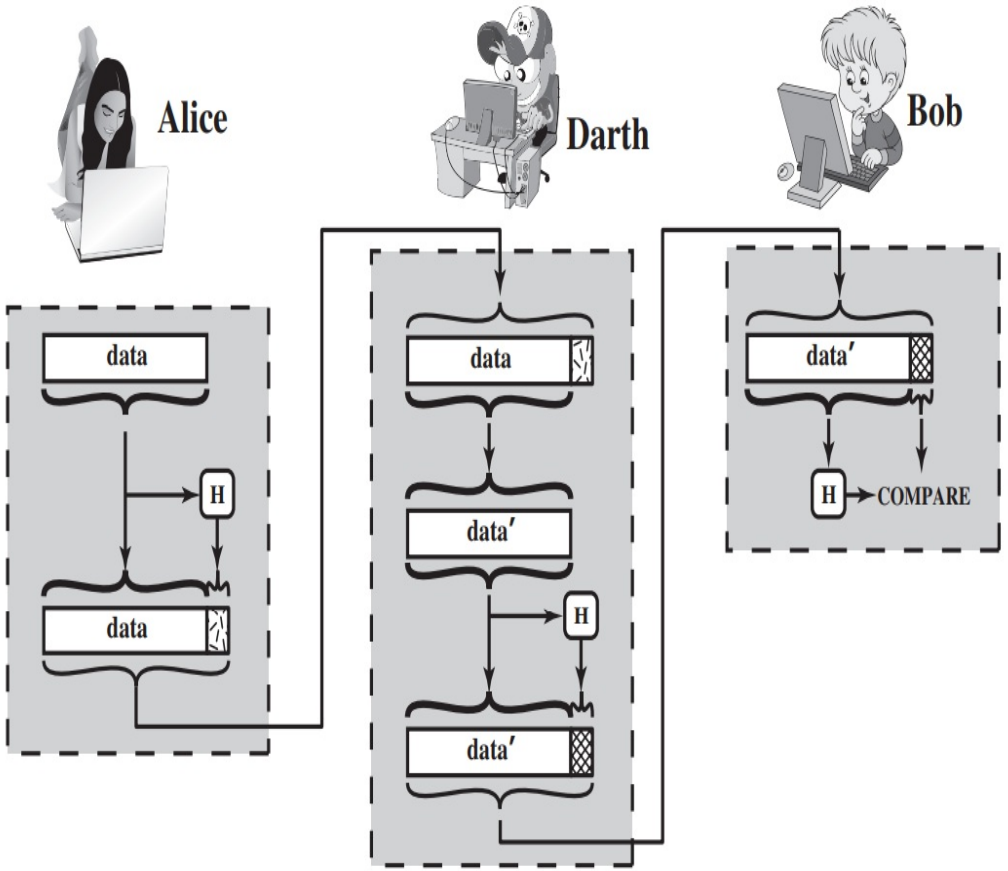
In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid. When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.

A Common Scenario



The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value. If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered .

An Attack Scenario (MiTM)

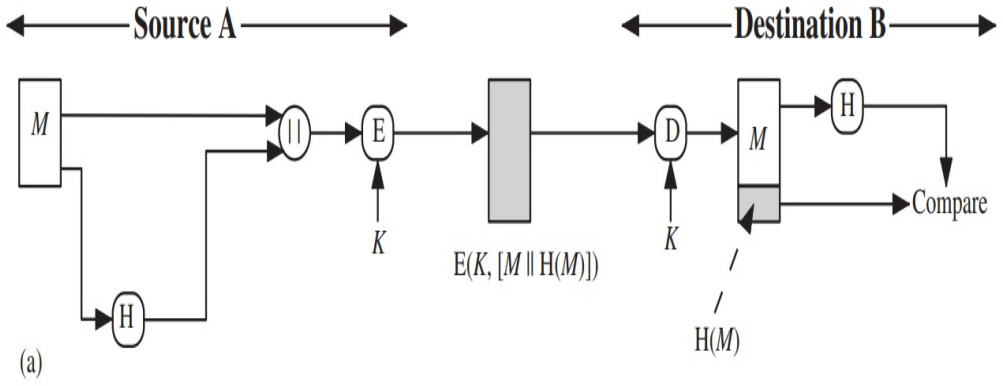


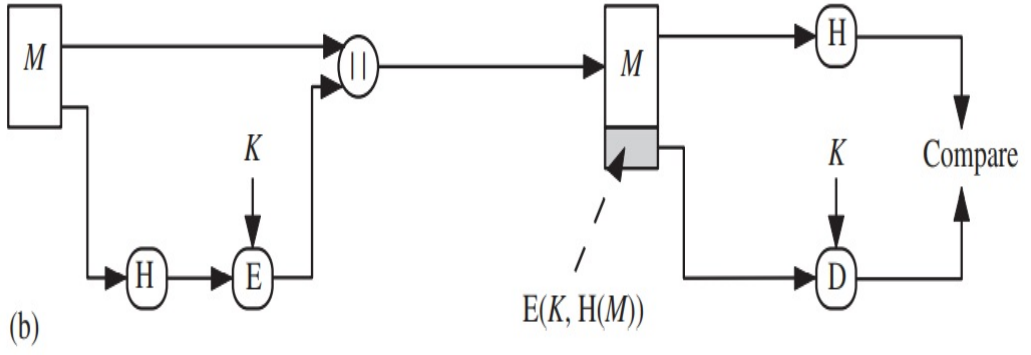
The hash function must be transmitted in a secure fashion. That is, the hash function must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver.

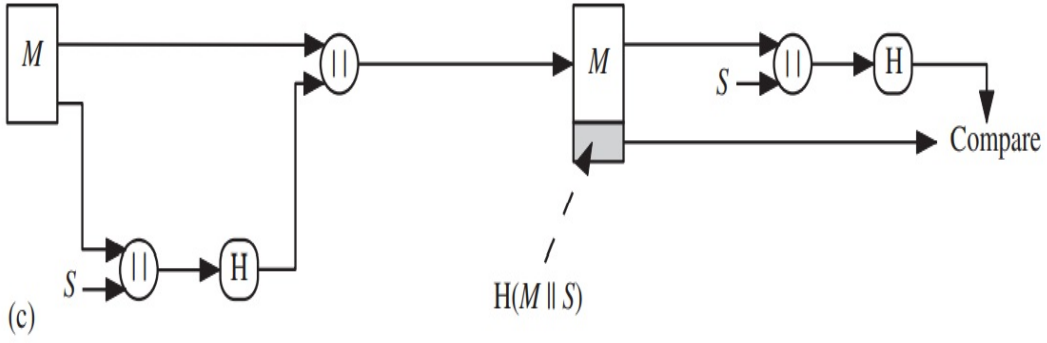
This type of attack is shown in the figure . In this example, Alice transmits a data block and attaches a hash value. Darth intercepts the message, alters or replaces the data block, and calculates and attaches a new hash value.

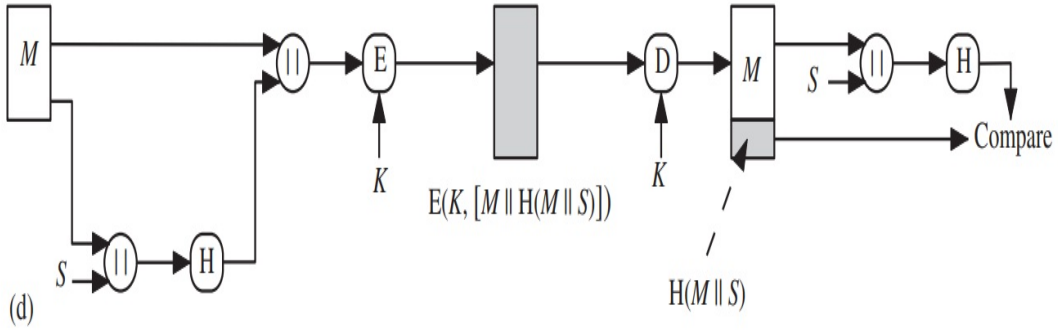
Bob receives the altered data with the new hash value and does not detect the change. To prevent this attack, the hash value generated by Alice must be protected

How to Protect Hash Values from Attacks

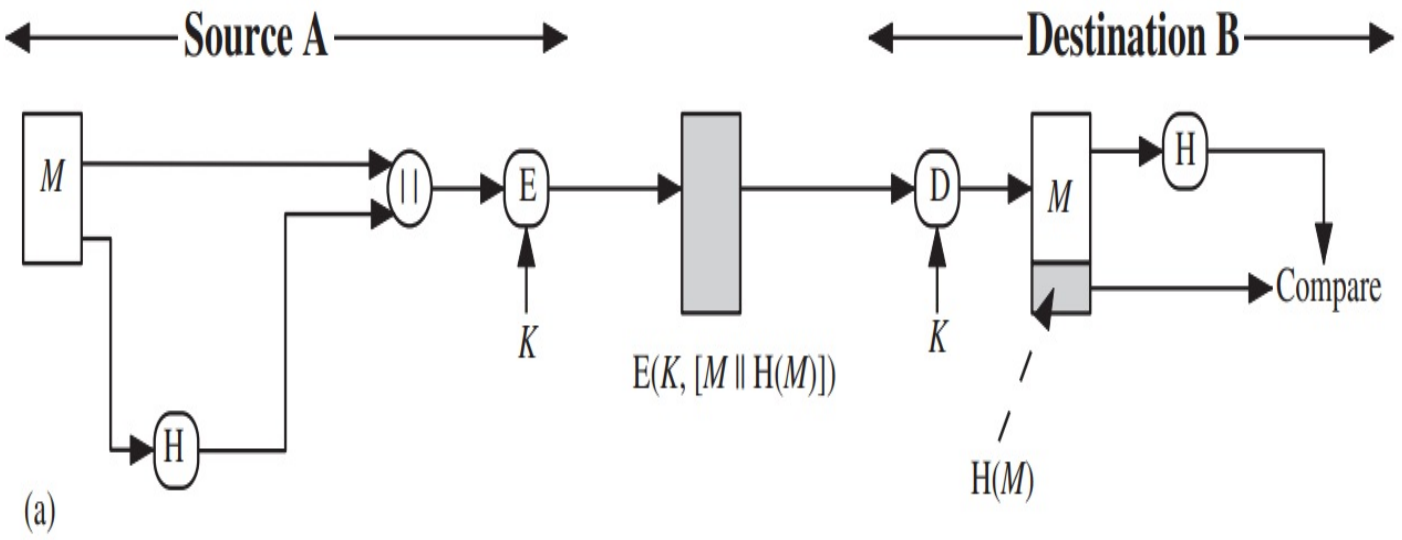






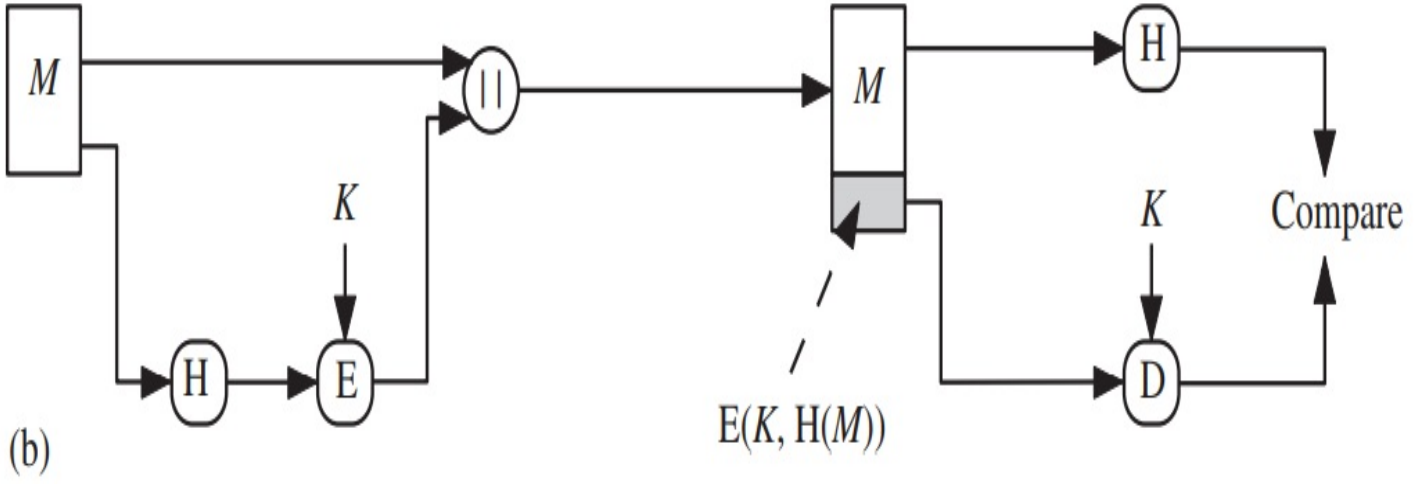


How to Protect Hash Values from Attacks



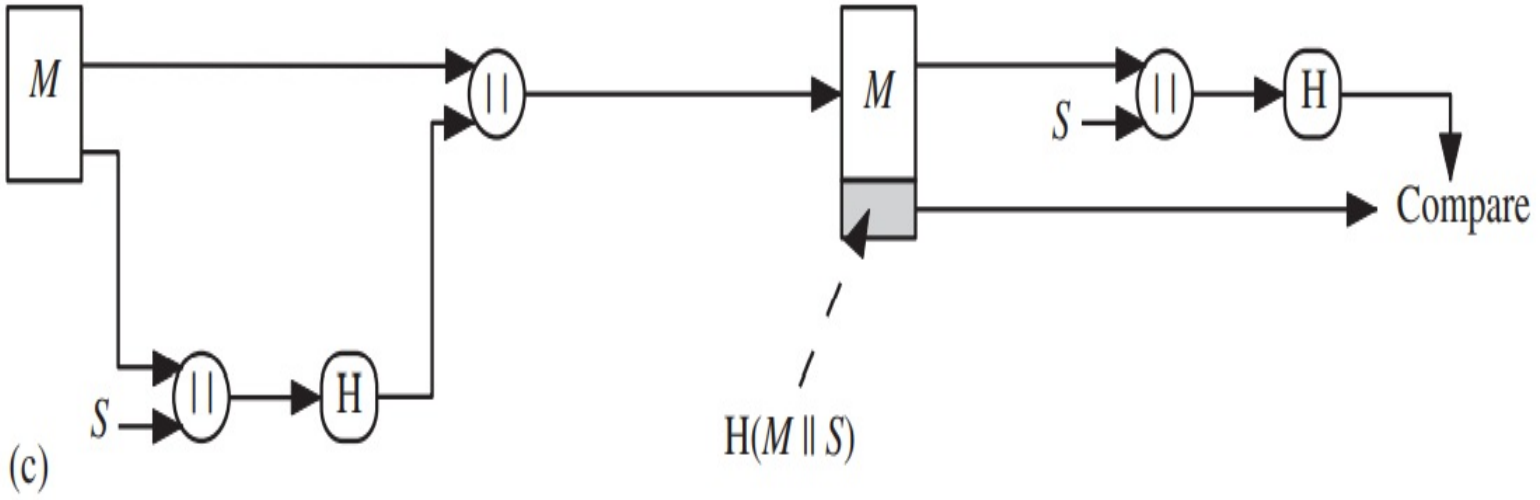
a. The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied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

How to Protect Hash Values from Attacks



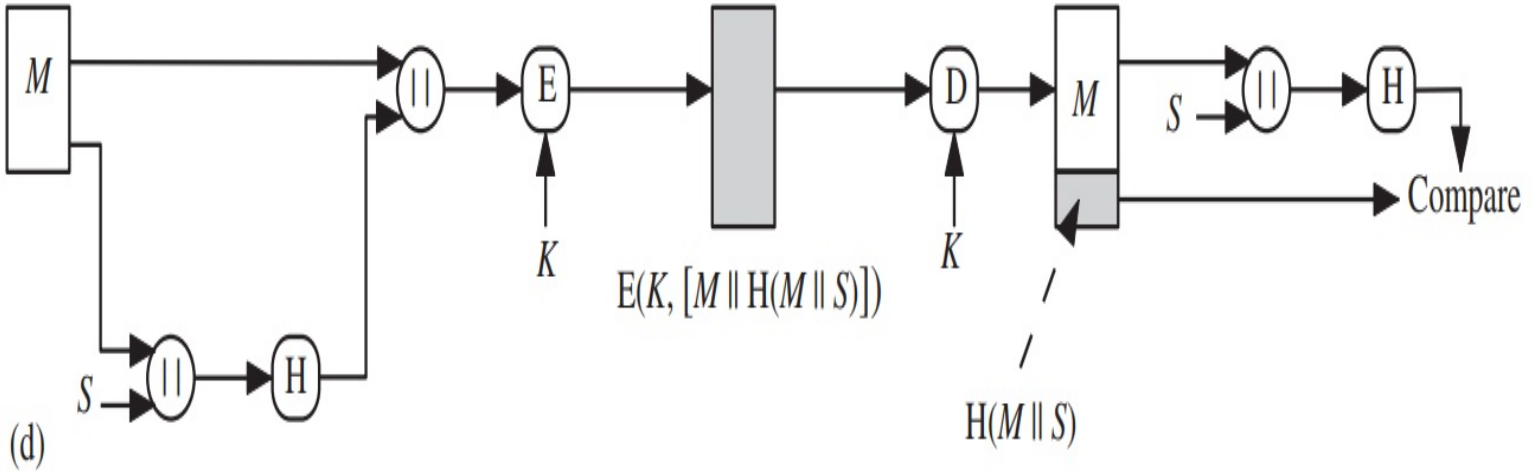
b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

How to Protect Hash Values from Attacks



c. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the 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 S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

How to Protect Hash Values from Attacks



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

Message Authentication is achieved using a Message Authentication Code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

Message Authentication is achieved using a Message Authentication Code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.

Message Authentication is achieved using a Message Authentication Code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.

If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value.

Message Authentication is achieved using a Message Authentication Code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.

If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value.

An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key. Note that the verifying party also knows who the sending party is because no one else knows the secret key.

Message Authentication is achieved using a Message Authentication Code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.

If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value.

An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key. Note that the verifying party also knows who the sending party is because no one else knows the secret key.

Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC. That is, E(K, H(M)) is a function of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key. In practice, specific MAC algorithms are designed that are generally more efficient than an encryption algorithm.

2. 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.

2. 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.

2. 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.

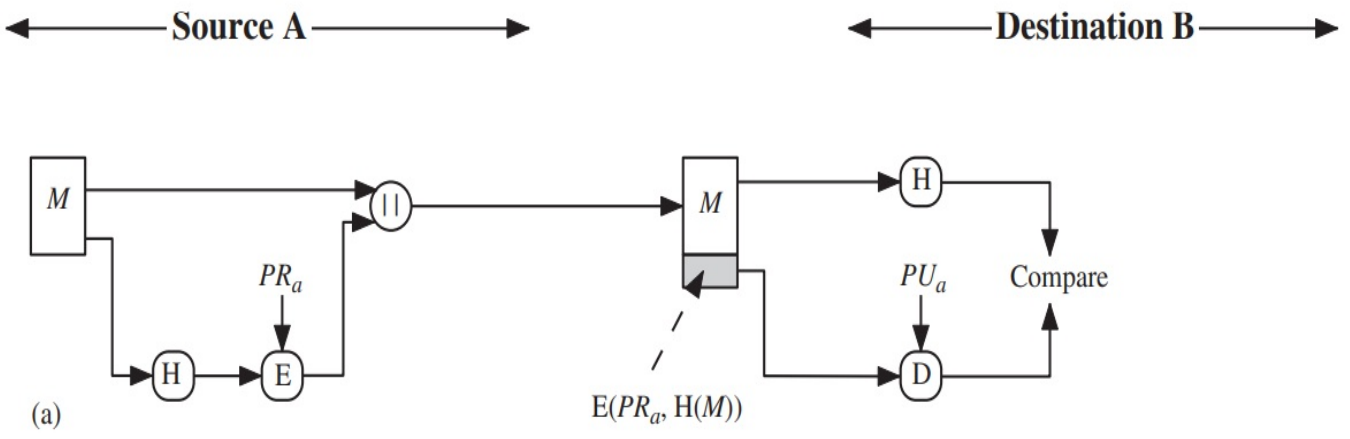
2. DIGITAL SIGNATURES

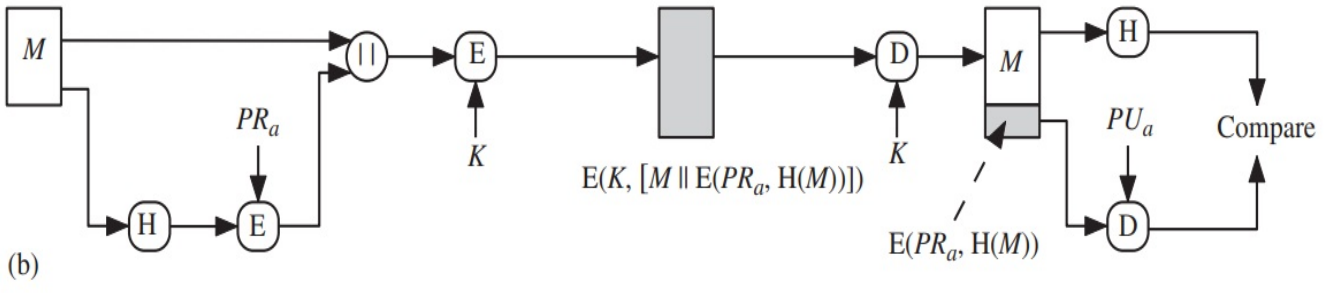
Another important application, which is similar to the message authentication application, is the digital signature. The operation of the digital signature is similar to that of the MAC.

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 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. In fact, this is the essence of the digital signature technique.

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. This is a common technique.

Other Applications

Hash functions are commonly used to create a one-way password file. In this scheme a hash of a password is stored by an operating system rather than the password itself. Thus, the actual password is not retrievable by a hacker who gains access to the password file. In simple terms, when a user enters a password, the hash of that password is compared to the stored hash value for verification. This approach to password protection is used by most operating systems.

Hash functions can be used for intrusion detection and virus detection. Store H(F) for each file on a system and secure the hash values (e.g., on a CD-R that is kept secure). One can later determine if a file has been modified by recomputing H(F). An intruder would need to change F without changing H(F).

Properties of Hash Functions

There are 4 main properties of hash functions :-

1) It is quick to compute the hash value of any given number.

2) It is infeasible to generate a message from its hash value except by trying all the possible combinations. (One Way Function)

3) A small change to a message should change the hash value so extensively that all the new hash values appear uncorrelated to the old hash value. (Avalanche Effect)

4) It is infeasible to find two different messages with the same hash value. (Collision Resistance)

Message Digest 5 (MD 5)

MD 5 is quite fast and produces a 128 bit message digest.

Message Digest 5 (MD 5)

MD 5 is quite fast and produces a 128 bit message digest.

The input text is processed in 512 bit blocks (which are further divided into 16 32-bit sub blocks).

Message Digest 5 (MD 5)

MD 5 is quite fast and produces a 128 bit message digest.

The input text is processed in 512 bit blocks (which are further divided into 16 32-bit sub blocks).

The output of the algorithm is a set of four 32- bit blocks, which make up the 128 bit message digest.

Message Digest 5 (MD 5)

512 x 1 ⇒ 512
512 x 2 ⇒ 1024
512 x 3 ⇒ 1536
1472 - 1000 ⇒ 472

1536 - 64
⇒ 1472

Working of MD 5

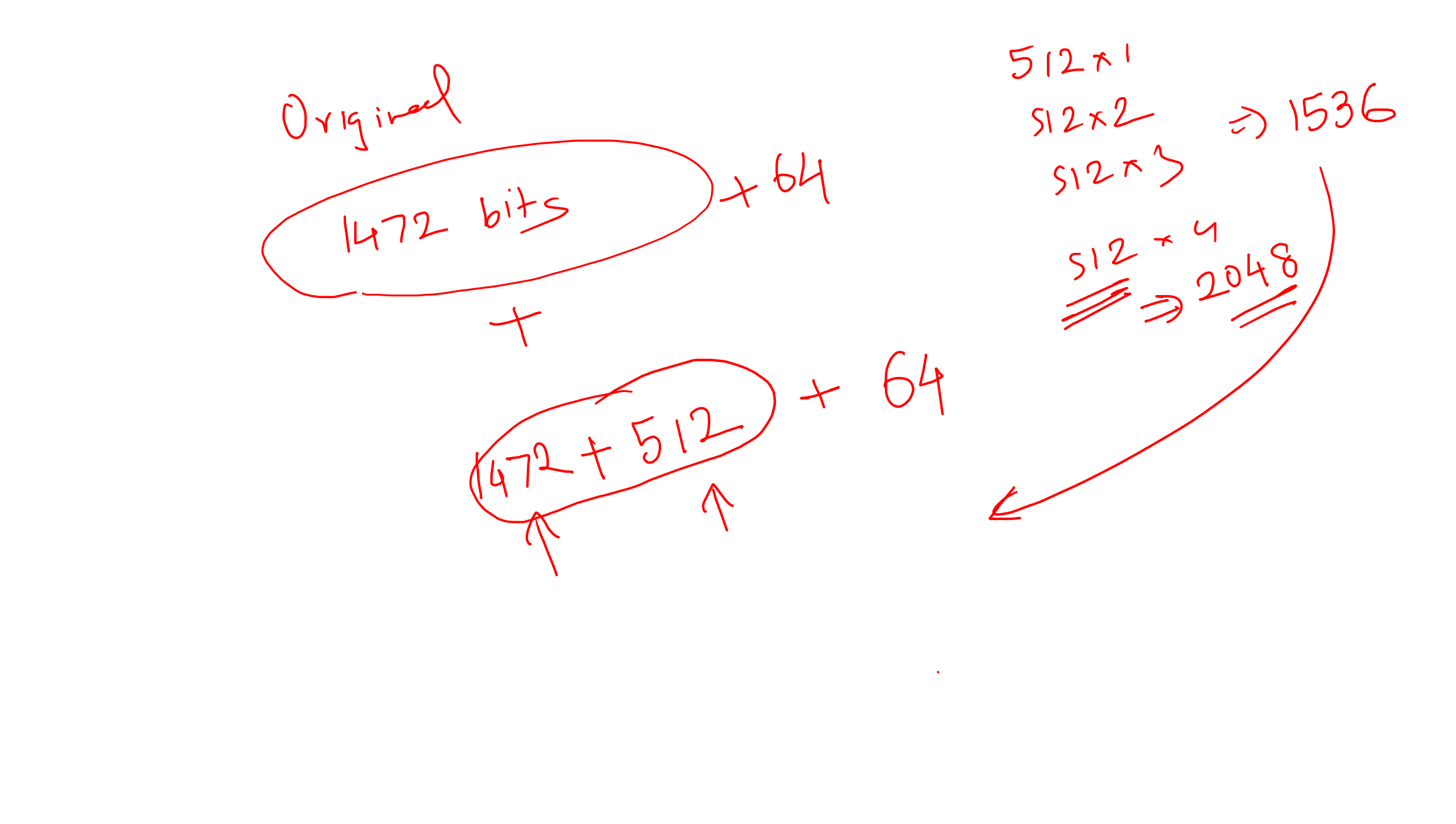
Step 1: Padding

The aim of this step is to make the length of the original message equal to a value which is 64 bits less than an exact multiple of 512 bits.

For ex, if the length of the original message is 1000 bits, we add a padding of 472 bits to make the lenght if the message 1472 bits, because 1536= 512 x 3. Also 1536 - 64 = 1472.

The padding consists of a single 1 bit followed by all 0 bits, as required.

 # Padding bits are always added even if the length is already 64 bits less than an exact multiple of 512.

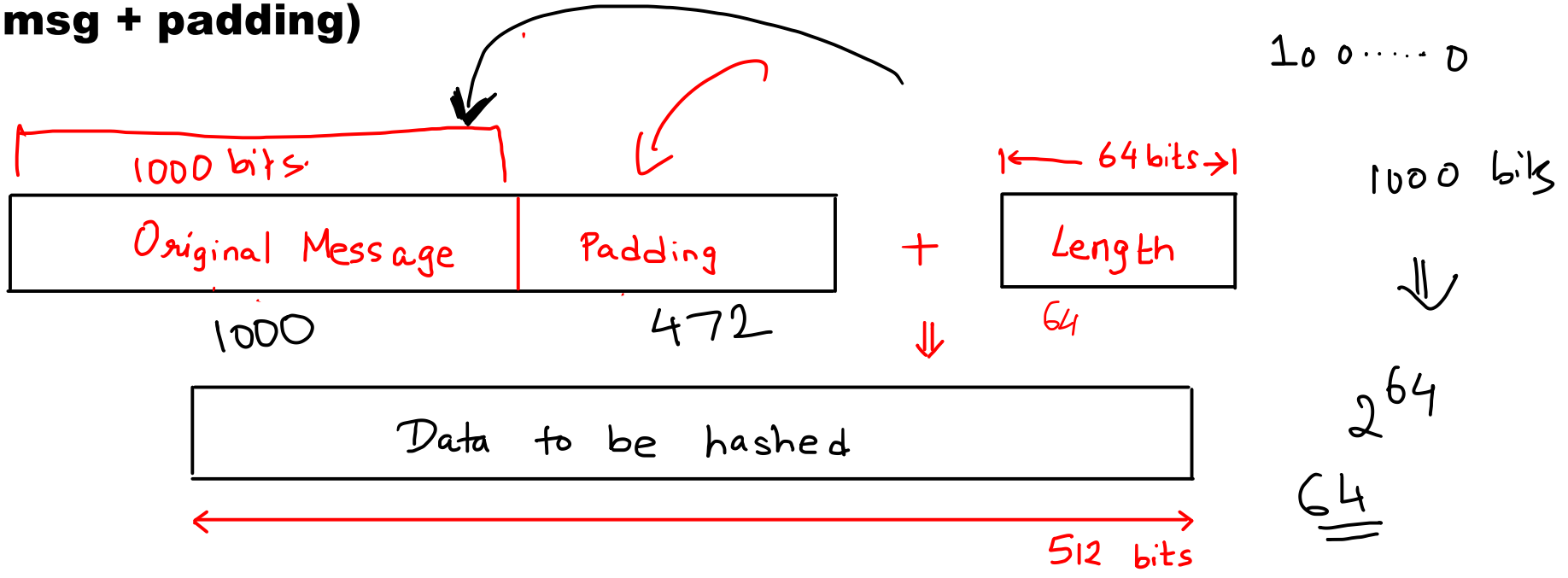

Original
1472 bits
+ 64
+
1472 + 512
+ 64
512 * 1
512 * 2
512 * 3
512 * 4
⇒ 2048
⇒ 1536

Message Digest 5 (MD 5)

Working of MD 5

Step 2: Append Length

After padding bits are added, the next step is to calculate the length of the original message in terms of 64 bits (2^64), which is then appended at the end of the (original msg + padding)



Message Digest 5 (MD 5)

Working of MD 5

Step 3: Divide the input into 512 bit blocks

Step 4: Initialise the chaining variables:

In this step, four variables (called chaining variables) are initialized. They are called A,B,C and D.

Each of these is a 32 bit number. The initial hexadecimal values of the chaining variables are as follows:

A : 01 23 45 67
B : 89 AB CD EF
C : FE DC BA 98
D : 76 54 32 10

Message Digest 5 (MD 5)

Working of MD 5

Step 5: Process Blocks:

After all the initializations, the real algorithm begins. There is a loop that runs for as many 512-bit blocks as are follows:

5.1: Copy the 4 chaining variables into four corresponding variables a,b,c and d
A=a , B=b, C=c, D=d.

5.2: Divide the current 512 - bit blocks into 16 sub blocks. Thus, each sub-block contains 32 bits.

5.3: We have 4 ROUNDS. In each round, we process all the 16 sub-blocks belonging to a block.
The inputs to each round are

a) All the 16 sub blocks
b) The variables a,b,c,d
c) Some constant, designated as K.

Internal Operations of each Round

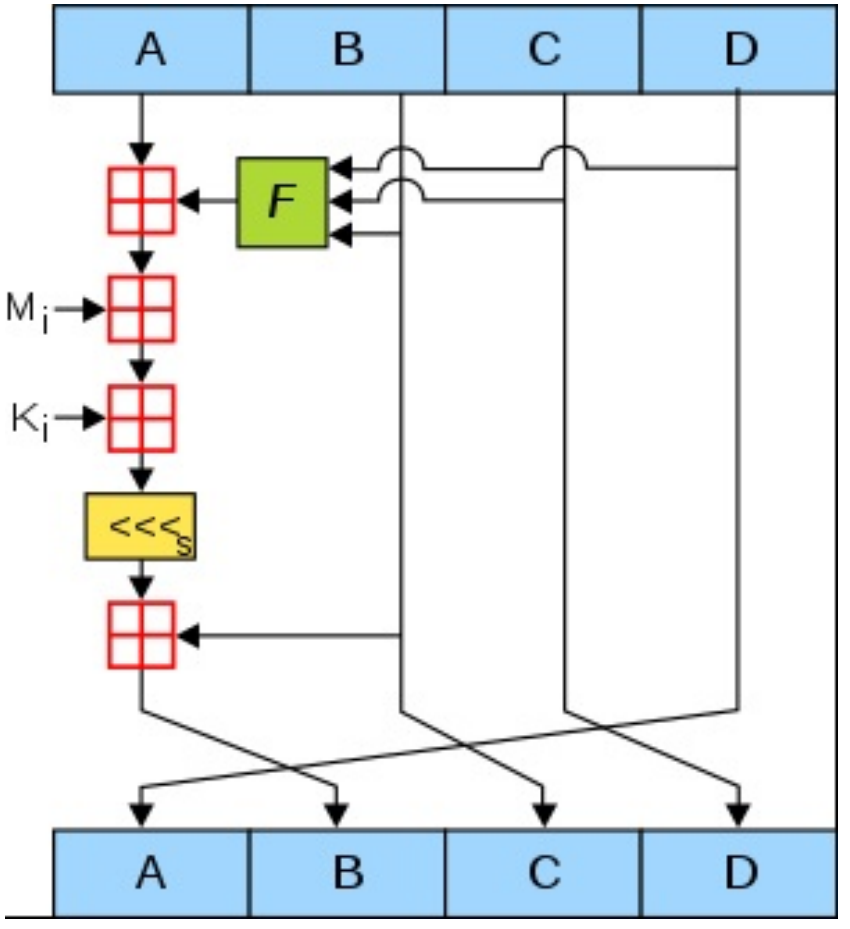
All the four rounds vary in one major way:

Step 1 of the four rounds has different processing. The other steps an all the four rounds are the same.

In each round we have 16 input sub blocks named M[0], M[1]...M[15].

Also T is an array of constants. it contains 64 elements, with each element consisting of 32 bits. We denote it as K[0], K[1], ... K[63].

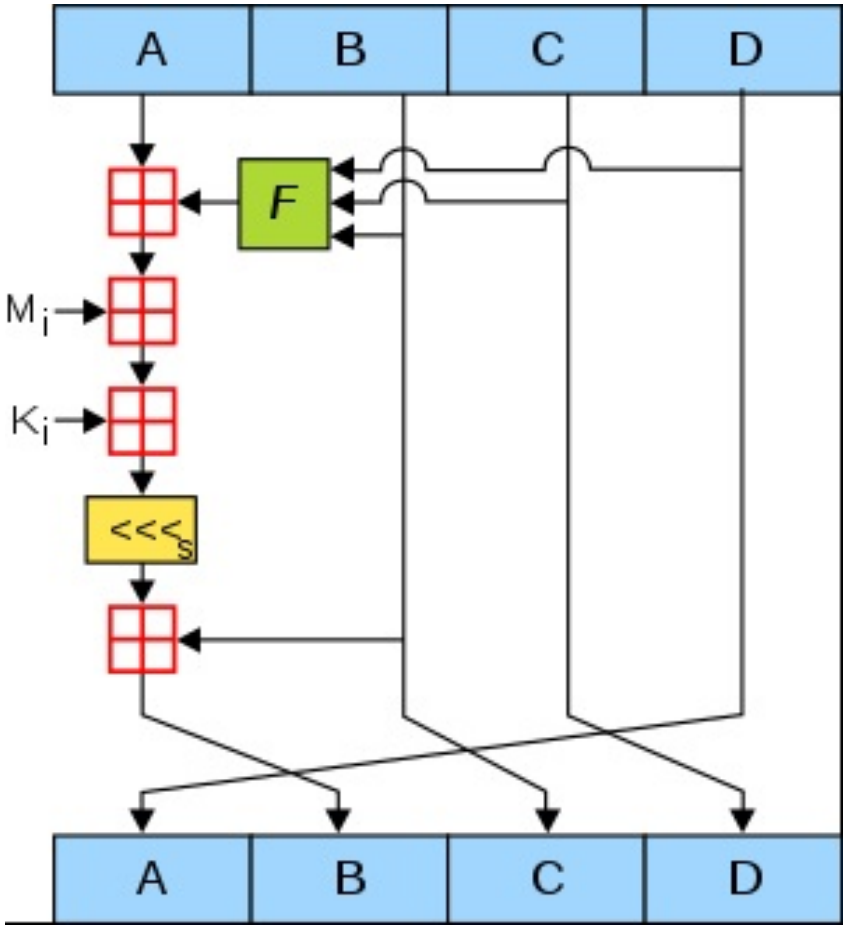
Since there are 4 rounds, we use 16 values out of K in each round.



F(X,Y,Z) = (X and Y) or (not(X) and Z)
G(X,Y,Z) = (X and Z) or (Y and not(Z))
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X or not(Z))

Internal Operations of each Round

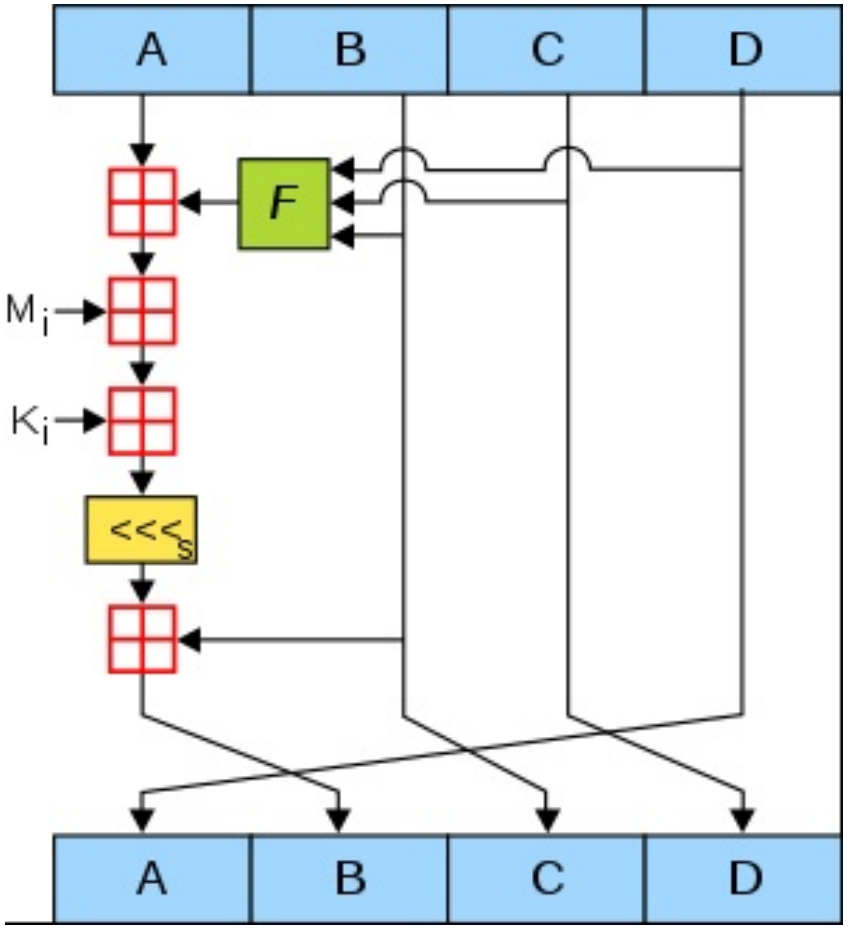
1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.



Internal Operations of each Round

1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

2. The Variable a is added to the output of the process F.

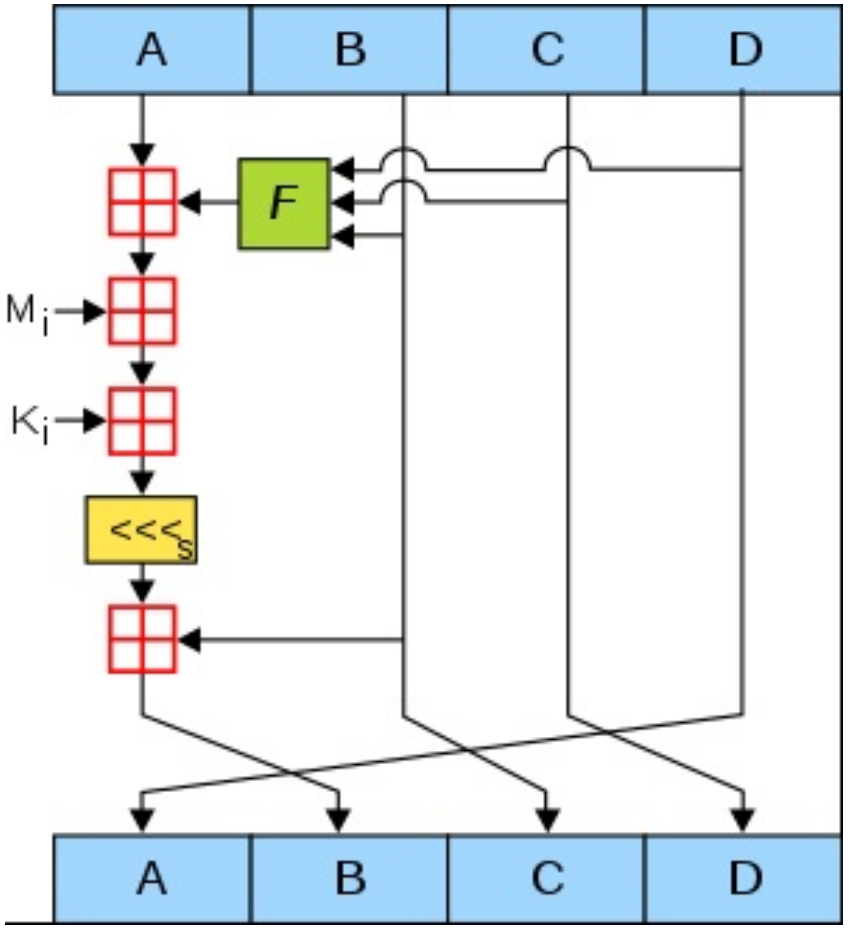


Internal Operations of each Round

1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

2. The Variable a is added to the output of the process F.

3.The message sub-block M[i] is added to the output of step 2.



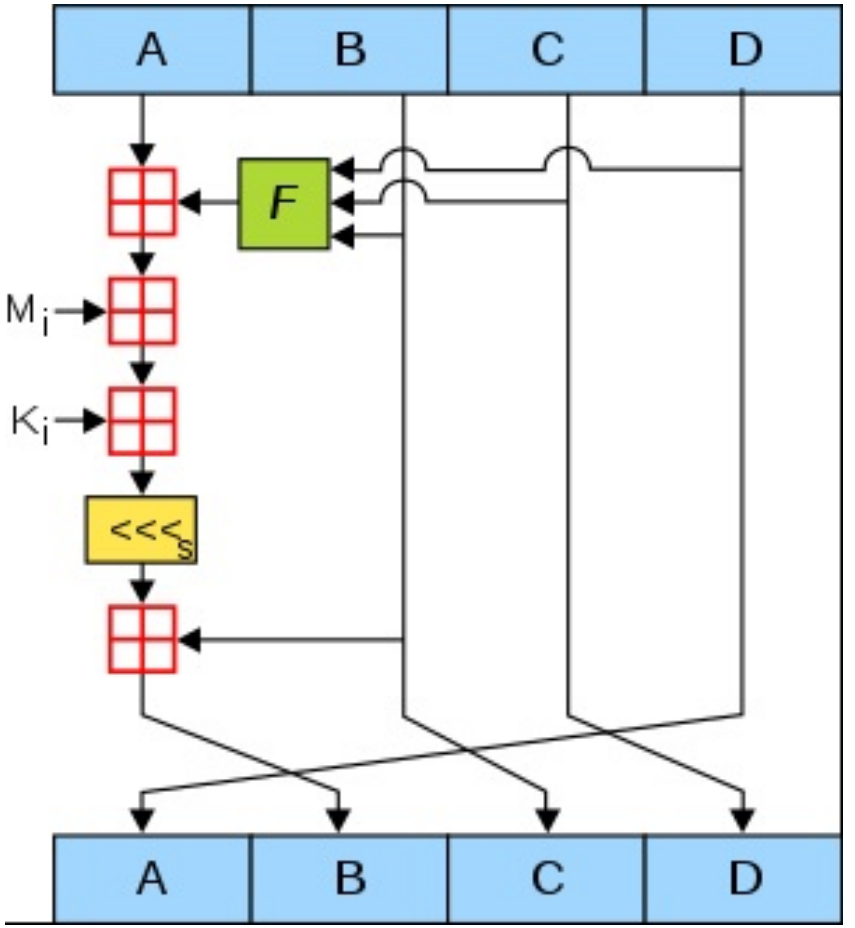
Internal Operations of each Round

1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

2. The Variable a is added to the output of the process F.

3.The message sub-block M[i] is added to the output of step 2.

4. The constant K[i] is added to the output of step 3.



Internal Operations of each Round

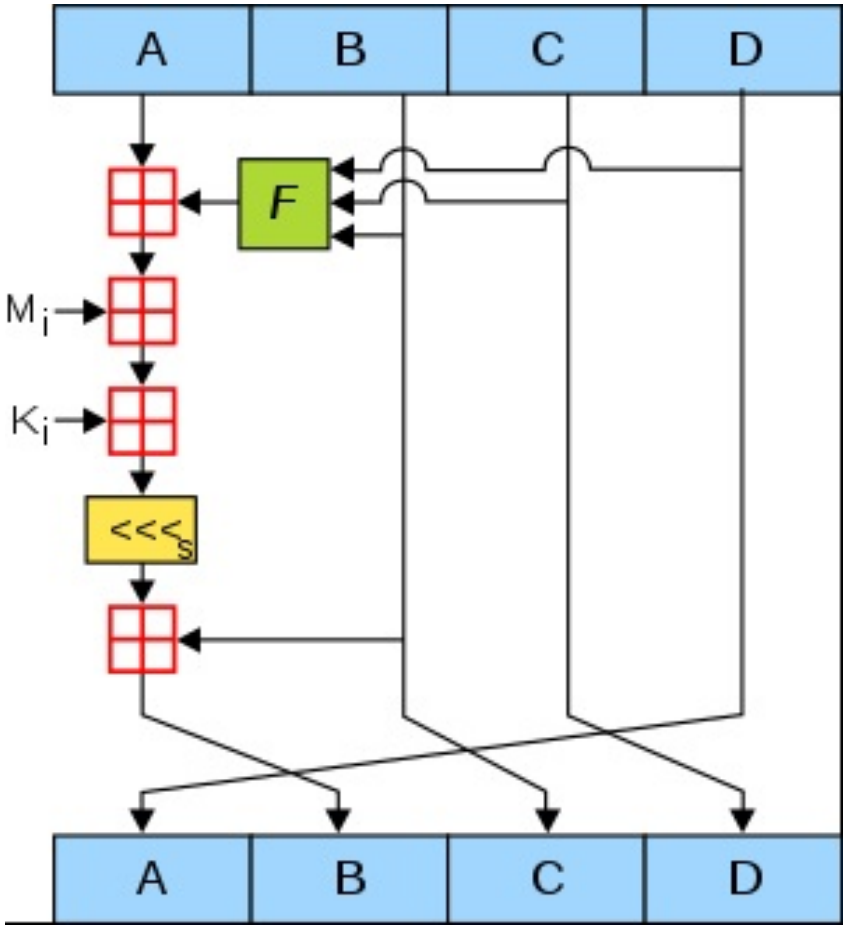
1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

2. The Variable a is added to the output of the process F.

3.The message sub-block M[i] is added to the output of step 2.

4. The constant K[i] is added to the output of step 3.

5. The output of step 4 is circularly left shifted by s bits.



Internal Operations of each Round

1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

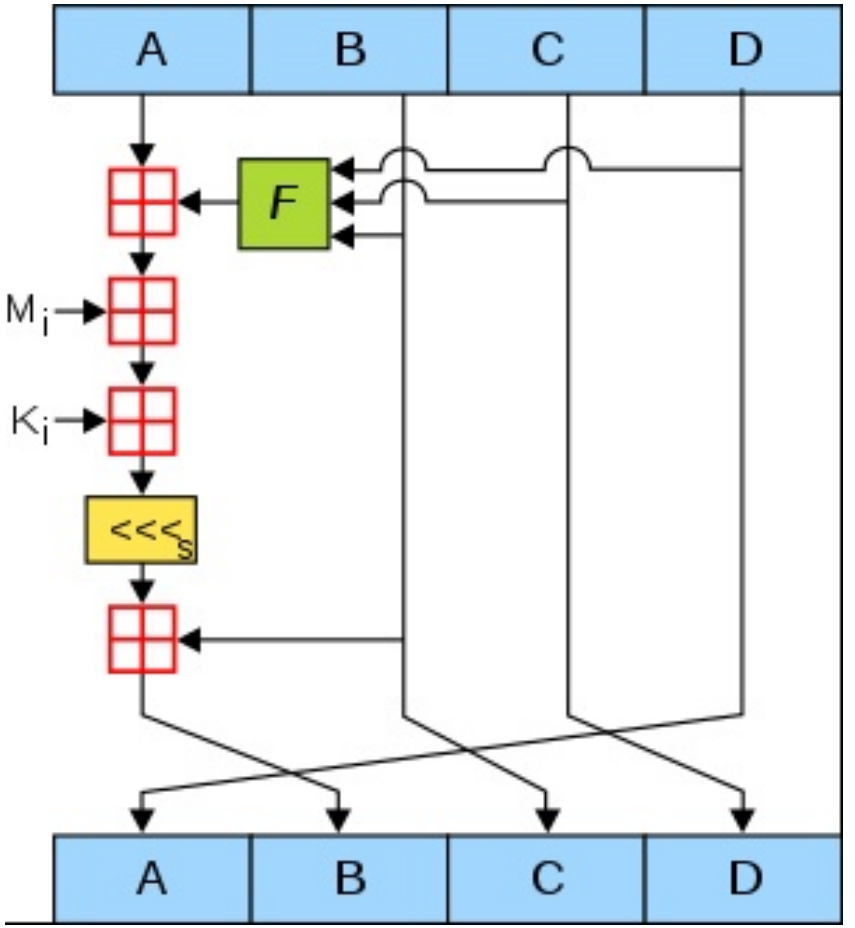
2. The Variable a is added to the output of the process F.

3.The message sub-block M[i] is added to the output of step 2.

4. The constant K[i] is added to the output of step 3.

5. The output of step 4 is circularly left shifted by s bits.

6. The variable b is added to the output of step 5.



F(B,C,D) = (B and C) or (not(B) and D)

F(B,C,D) =(B and D) or (C and not(D))

F(B,C,D)= B xor C xor D

F(B,C,D)= C xor (B or not(D))

Internal Operations of each Round

1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

2. The Variable a is added to the output of the process F.

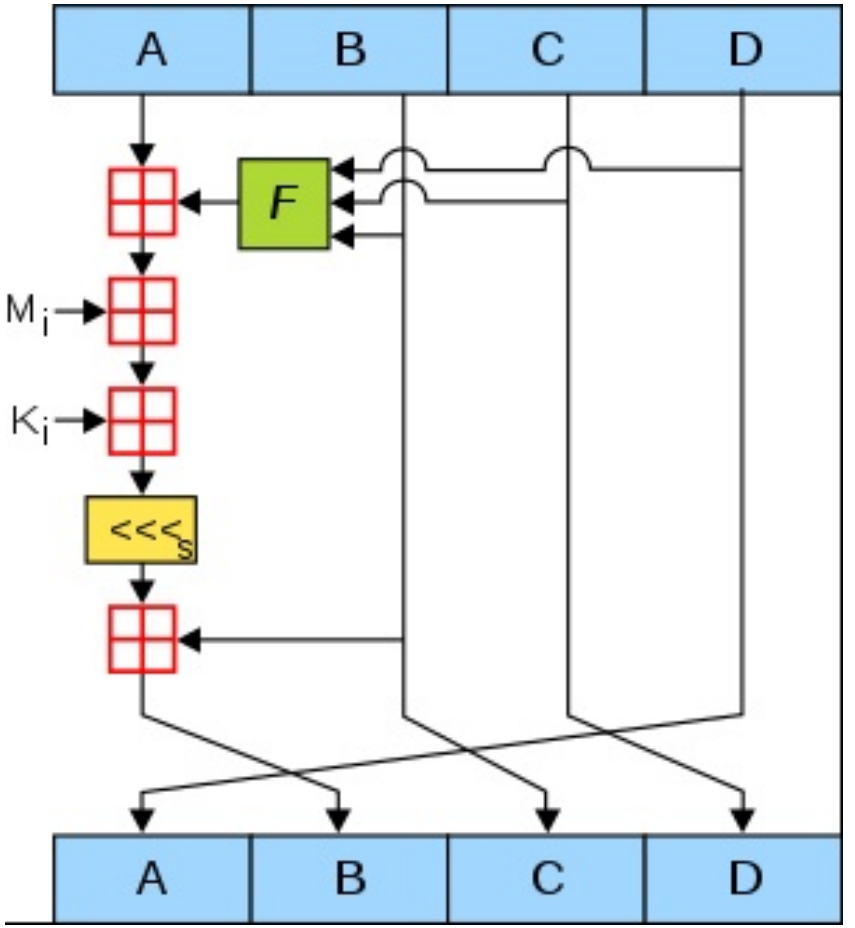
3.The message sub-block M[i] is added to the output of step 2.

4. The constant K[i] is added to the output of step 3.

5. The output of step 4 is circularly left shifted by s bits.

6. The variable b is added to the output of step 5.

7. The output of step 6 becomes the new b.



Internal Operations of each Round

1. A Process F is first performed on b,c,d. This process is different in all the 4 rounds.

2. The Variable a is added to the output of the process F.

3.The message sub-block M[i] is added to the output of step 2.

4. The constant K[i] is added to the output of step 3.

5. The output of step 4 is circularly left shifted by s bits.

6. The variable b is added to the output of step 5.

7. The output of step 6 becomes the new b.

8. all other variables are right shifted by 1 position.

c=b, d=c, a=d.