UNIT III HASH FUNCTIONS AND DIGITAL SIGNATURES

Authentication requirement – Authentication function – MAC – Hash function – Security of hash function and MAC –MD5 - SHA - HMAC – CMAC - Digital signature and authentication protocols – DSS – EI Gamal – Schnorr.

PART – A

1. List the authentication requirements. (May/June'14)
Specify the requirements for message authentication. (May/June'12)

- Disclosure
- Traffic analysis
- Masquerade
- Content identification
- Sequence modification
- Timing modification
- Source repudiation
- Destination repudiation

2. Define hashing function. (Nov/Dec'13)

Hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M). A hash code does not use a key but is a function only of the input message. The hash code is also referred to as a message digest or hash value.

A hash value h is gene ated by a function H of the form

$$h = H(M)$$

where M is a variable-length message and H(M) is the fixed-length hash value.

3. What are the properties of hashing function in cryptography? (Nov/Dec'13 & May/June'11)

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to as the **one-way property**.
5. For any given block x, it is computationally infeasible to find y x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

4. What do you mean by one-way property in hash function? (April/May'11)

- H is one-way, for any given value h, it is computationally infeasible to find x such that H(x) = h.
- One-way property states that it is easy to generate a code given a message but virtually impossible to generate a message given a code.
- This property is essential for authentication.

5. Write down the purpose of hash function along with a simple hash function. (May/June'07)

- 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 it rative fashion to produce an n-bit hash function.
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as C = bi1 ⊕bi2 ⊕ g⊕bim
  where
  C_i = ith bit of the hash code, 1 … i … n m
  = number of n-bit blocks in the input
  b_{ij} = ith bit in jth block
  ⊕ = XOR operation
- This operation produces a simple parity for each bit position and is known as a longitudinal redund ncy check. It is reasonably effective for random data as a data integrity ch ck.

6. Define message digest. (April/May'10)

When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**. The hash code is a function of all the bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

7. Write any two differences between MD4 and SHA. (Nov/Dec'12)

MD4	SHA
Pad message so its length is 448 mod 512.	Pad message so its length is a multiple of 512 bits.
Initialize the 4-word (128 bit) buffer (A, B, C, D)	Initialize 5-bit (160 bit) buffer (A, B, C, D, E)
Process the message in 16-word chunks using 3 rounds of 16-bit operations each on chunk and buffer.	Process the message in 16-word chunks using 4 rounds of 20-bit operations.

8. What are the performance differences between MD5, SHA-512 and RIPEMD-160? (Nov/Dec'14)

1. MD5 produces a 128-bit hash value. SHA-512 produces 160-bit hash value.
2. Brute force attack harder. (160 like SHA-1 vs 128 bits for MD5)
3. Not vulnerable to known attacks, like SHA-1 though stronger (compared to MD4/5)
4. SHA-512 is slower than MD5 (more steps).
5. All designed as simple and compact.
6. SHA-1 optimized for big-endian CPUs vs RIPMED-160 and MD5 optimized for little-endian CPUs.

9. What are the two important key issues related to authenticated key exchange? (May/June'12)

Confidentiality

- Timeliness

10. What is digital signature? (April/May'15 & May/June'11)

A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator"s private key. The signature guarantees the source and integrity of the message.

11. What are the two approaches of digital signature? (Nov/Dec'12)

- Direct Digital Signature
- Arbitrated Digital Signature

12. List any two properties a digital signature should essentially have? (May/June'07)

- It must verify the author and the date and time of the signature.
- It must to authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

13. What are the requirements of digital signature?

1. The signature must be a bit pattern that depends on the message being signed.
2. The signature must use information unique to sender to prevent both forgery and denial.
3. It must be relatively easy to produce the digital sig ature.
4. It must be relatively easy to recognize and ve fy the digital signature.
5. It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
6. It must be practical to retain a copy of the digital signature in storage.

14. What are the security services provided by digital signature? (Nov/Dec'14)

- **Authentication -** The assurance that the communicating entity is the one that it claims to be.
- **Data Integrity -** The assu ance that data received are exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion, or replay).
- **Nonrepudiation -** Provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication

15. What are birthday attacks? (May/June'14)

A birthday attack is a name used to refer to class of brute force attacks. It gets its name from the surprising result that the probability that two or more people in a group of twenty three share the same birthday day is greater than ½ such a result is called birthday paradox.

MESSAGE AUTHENTICATION FUNCTIONS

a. Message Authentication Functions:

Message authentication or digital signature mechanism has two levels of functionality.

- At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message.
- This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

- Three classes of functions that may be used to produce an authenticator.
  - **Hash function:** A function that maps a message of any length into a fixed length hash value, which serves as the authenticator
  - **Message encryption:** The ciphertext of the entire message serves as its authenticator
  - **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator.

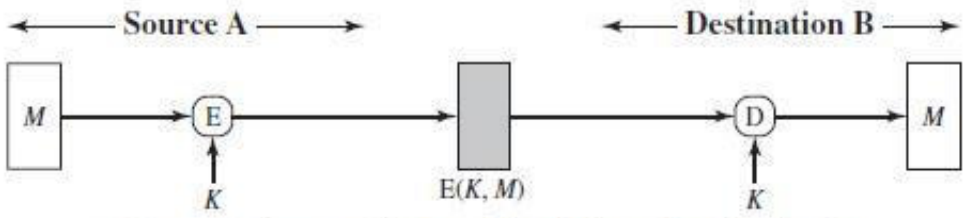
Message Encryption

- Conventional encryption can serve as authenticator
  - Conventional encryption provides *authentication* as well as *confidentiality*
  - Requires recognizable plaintext or other *structure* to distinguish between well-formed legitimate plaintext and meaningless random bits
    - e.g., ASCII text, an appended checksum, or use of layered protocols

Symmetric Encryption / Conventional Encryption

A message M transmitted from source A to destination B is encrypted using a secret key K shared by A and B.If no other party knows the key then confidentiality is provided:- No other party can recover the plaintext of the message.

- Given a decryption function D and a secret key K, the destination will accept any input Y and produce output X =Dk(Y)
- If Y is the ciphertext then X is some plaintext message M



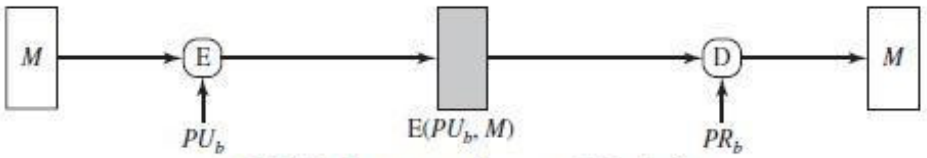
(a) Symmetric encryption: confidentiality and authentication

Public-key Encryption

(i) Public-key encryption: Confidentiality

- The source (A) uses the public key KU_b of the destination (B) to encrypt M. Because only B has the corresponding private k y KR_b, only B can decrypt the message.

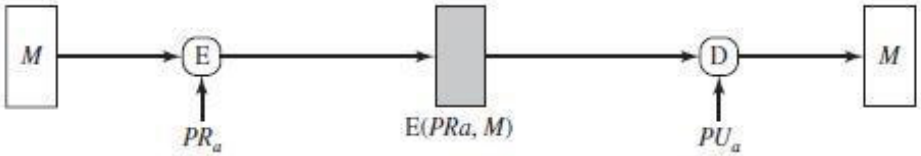
- This scheme provides no authentication because any opponent could also use B‟s public key to encrypt a message, claiming to be A.



(b) Public-key encryption: confidentiality

(ii) Public – key encryption: Authentication and Signature

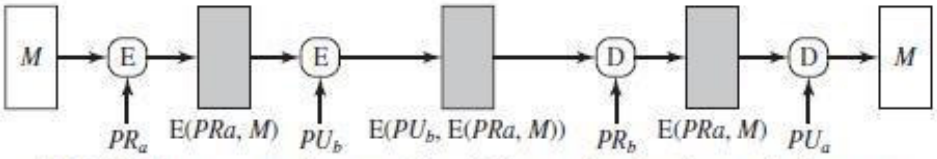
- A uses it private k y to encrypt the message, and B uses A‟s public key to decrypt.
- This provides authentication using the same type of reasoning as in the symmetric encryption case : -
  - The message must have come from A because A is the only party that possesses KR_a and therefore the only party with the information necessary to construct ciphertext that can be decrypted with KUa.
    - It also provides what is known as digital signature. Only A could have constructed the ciphertext because only A possesses KR_a. Not even B, the recipient, could have constructed the ciphertext.



(c) Public-key encryption: authentication and signature

(iii) Public – key encryption: Confidentiality, Authentication and Signature

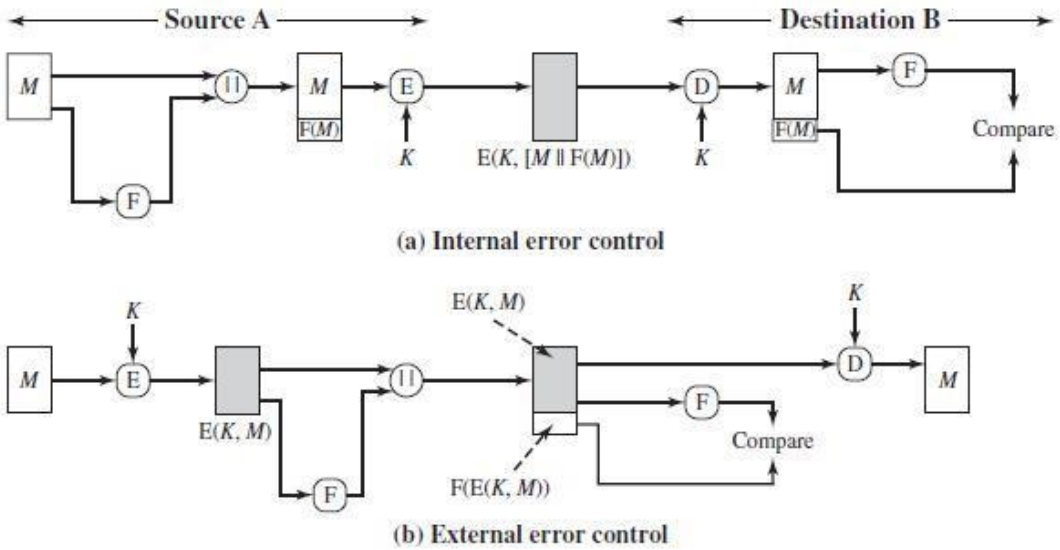
- To provide both confidentiality and authentication, A can encrypt M first using its private key, which provides the digital signature, and then using B‟s public key, which provides confidentiality.



(d) Public-key encryption: confidentiality, authentication, and signature

Internal and External Error control

- A prepares a plaintext message M and then provides this input to a function F that produces an FCS (Frame Check Sequence) or Checksum.
- The FCS is appended to M and the entire block is then encrypted.
- At the Destination B decrypts the incoming block and treats the results as a message with an appended FCS.
- B applies the same function F to attempt to produce the FCS.
- If the calculated FCS is equal to the incoming FCS then the message is considered authentic.



With internal error control, authentication is provided because an opponent would have difficulty generating ciphertext, when decrypted would have valid error control bits.If instead the FCS is the outer code an opponent can construct messages with valid error-control codes.Although the opponent cannot know what the decrypted plaintext will be, he or she can still hope to create confusion and disrupt operations.

TCP Segment:

An error contol code added to the transmitted message serves to strengthen the authentication capability.Such structure is provided by the use of a communications architecture consisting of layered protocols.

- The structured message is transmitted using TCP/IP protocol architecture.
- The TCP Header:-
  - Each pair of hosts shared a unique secret key so that all exchanges between a pair of hosts used the same key.
- Then one could simply encrypt all of the datagram except the IP Header.
- If an opponent substituted some arbitrary bit pattern for the encrypted TCP segment the resulting plaintext would not a meaningful Header.
- In this case the Header includes not only a check-sum but other useful information (such as the sequence number).

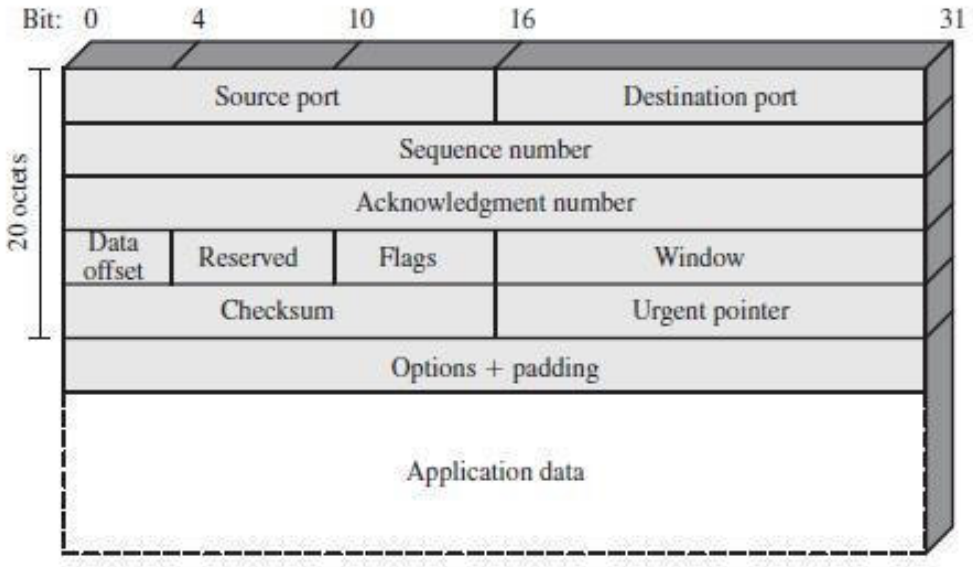


Figure: TCP Segment

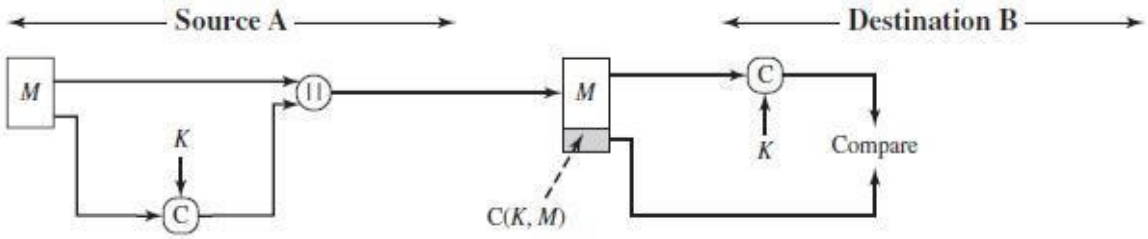
b. MESSAGE AUTHENTICATION CODE (MAC)

Message Authentication Code technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key *K*. When A has a message to send to B, it calculates the MAC as a function of the message and the key:

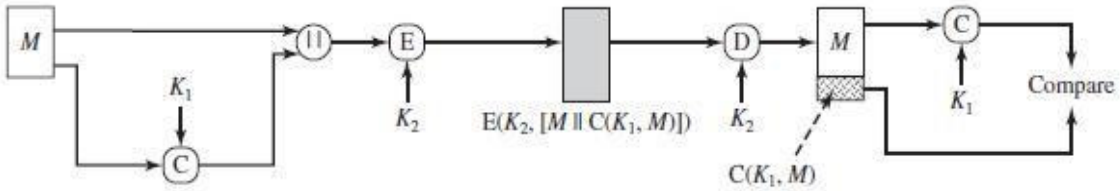
$$\mathbf{MAC = C(}\boldsymbol{K}\mathbf{,}\ \boldsymbol{M}\mathbf{)}$$

where

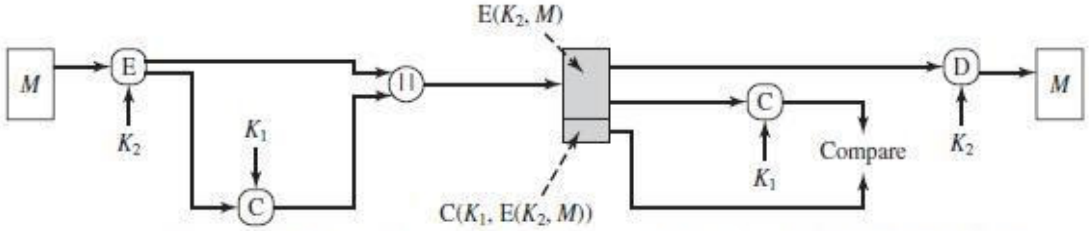
M = input message
C = MAC function
K = shared secret key
MAC = message authentication code



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC. If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver‟s calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.

2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

The following are three situations in which a message authentication code is used.

1. There are a number of applications in which the same message is broadcast to a number of destinations. It is cheaper and mo e reliable to have only one destination responsible for monitoring auth nticity. Thus, the message must be broadcast in plaintext with an associat d message authentication code. The responsible system has the secret key and performs authentication. If a violation occurs, the other destinat on systems are alerted by a general alarm.
2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, messages being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources.

SECURITY OF MACs

We can group attacks on MACs into two categories: brute-force attacks and cryptanalysis.

Brute-Force Attacks

- A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs.
- To attack a hash code, given a fixed message x with n-bit hash code $h = H(x)$, a brute-force method of finding a collision is to pick a random bit string y and check if $H(y) = H(x)$.
- The attacker can do this repeatedly off line. Whether an off-line attack can be used on a MAC algorithm depends on the relative size of the key and the tag.

The security property of a MAC algorithm can be expressed as follows.

Computation resistance: Given one or more text-MAC pairs $[x_i, MAC(K, x_i)]$, it is computationally infeasible to compute any text-MAC pair $[x, MAC(K, x)]$

for any new input $x \neq x_i$.

- In other words, the attacker would like to come up with the valid MAC code for a given message x. There are two lines of attack possible: attack the key space and attack the MAC value.
- If an attacker can determine the MAC key, then it is possible to generate a valid MAC value for any input x.
- Suppose the key size is k bits and that the attacker has one known text-tag pair. Then the attacker can compute the n-bit tag on the known text for all possible keys. At least one key is guaranteed to produce the correct tag, namely, the valid key that was initially used to produce the known text-tag pair. This phase of the

attack takes a level of effort proportional to $2k$ (that is, one operation for each of the $2k$ possible key values).

- MAC is a many-to-one mapping, there may be other keys that produce the correct value. Thus, if more than one key is found to produce the correct value, additional text-tag pairs must be tested. It can be shown that the level of effort drops off rapidly with each additional text-MAC pair and that the overall level of effort is roughly $2k$.
- An attacker can also work on the tag without attempting to recover the key. The objective is to generate a valid tag for a given message or to find a message that matches a given tag. In either case, the level of effort is comparable to that for attacking the one-way or weak collision-resistant property of a hash code, or 2 .
- In the case of the MAC, the attack cannot be conducted off line without further input; the attacker will require chosen text-tag pairs or knowledge of the key.

- The level of effort for brute-force attack on a MAC algorithm can be expressed as $\min(2^k, 2^n)$. The assessment of strength is similar to that for symmetric encryption algorithms. It would appear reasonable to require that the key length and tag length satisfy a relationship such as $\min(k, n) \geq N$, where N is perhaps in the range of 128 bits.

Cryptanalysis

- As ith encryption algorithms and hash functions, cryptanalytic attacks on MAC algorithms 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 MAC algorithm to cryptanalysis is to compare its strength to the effort required for a bruteforce attack. That is, an ideal MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.
- There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs.

c. HASH FUNCTION

Hash function accepts a variable size message M as input and produces a fixed-size output, referred to as hash code H(M). A hash code does not use a key but is a function only of the input message. Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a **message digest** or **hash value**.

The hash code is a function of all the bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

The below figure illustrates a 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. Because encryption is applied to the entire message plus hash code, 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. 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, it can recomputed 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.

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

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required.

A hash value h is generated by a function H of the form

$$\mathbf{h = H(M)}$$

where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value.

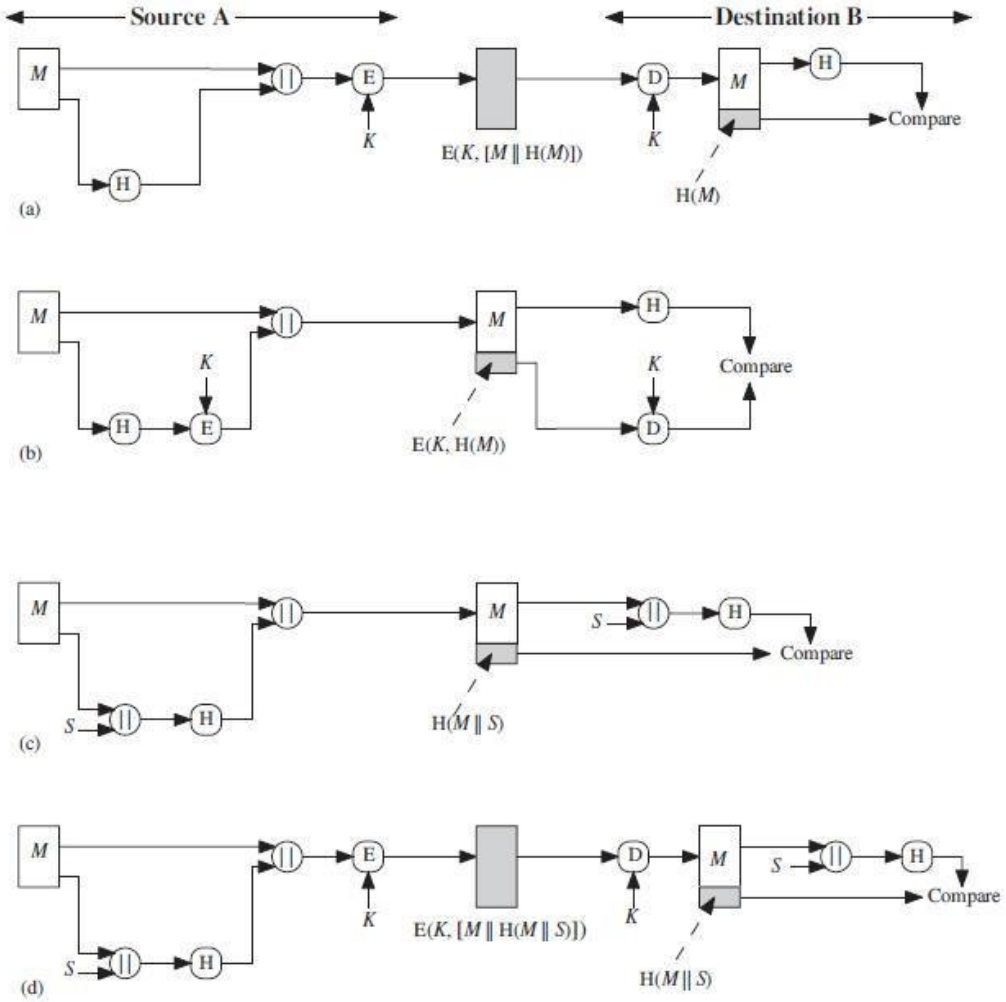


Figure: Simplified Examples of the Use of a Hash Function for Message Authentication

Requirements for a Hash Function

A hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementatio s practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to as the **one-way property**.
5. For any given block x, it is computationally infeasible to find y x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

The first three properties are requirements for the practical application of a hash function to message authentication.

- The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value. The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message M and the hash code $C = \mathrm{H}(S_{AB} \| M)$. The attacker then inverts the hash function to obtain $S_{AB} \| M = \mathrm{H}_1(C)$. Because the attacker now has both M and $S_{AB} \| M$, it is a trivial matter to recover S_{AB}.
- The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used.
- The sixth property refers to how resistant the hash function is to a type of attack known as the birthday attack.

Simple Hash Functions

All hash functions operate using the following general pri ciples. 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 n-bit hash function.

One of the simplest hash functions is the **bit-by-bit exclusive-OR (XOR)** of every block. This can be expressed as $C_i = b_{i1} \oplus b_{i2} \oplus \mathrm{g} \oplus b_{im}$

where

C_i = ith bit of the hash code, 1 … i … n

m = number of n-bit blocks in the input

b_{ij} = ith 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}. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used,

instead of an effectiveness of 2^{-128}, the hash function on this type of data has an effectiveness of 2^{-112}.

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 follo s.

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.

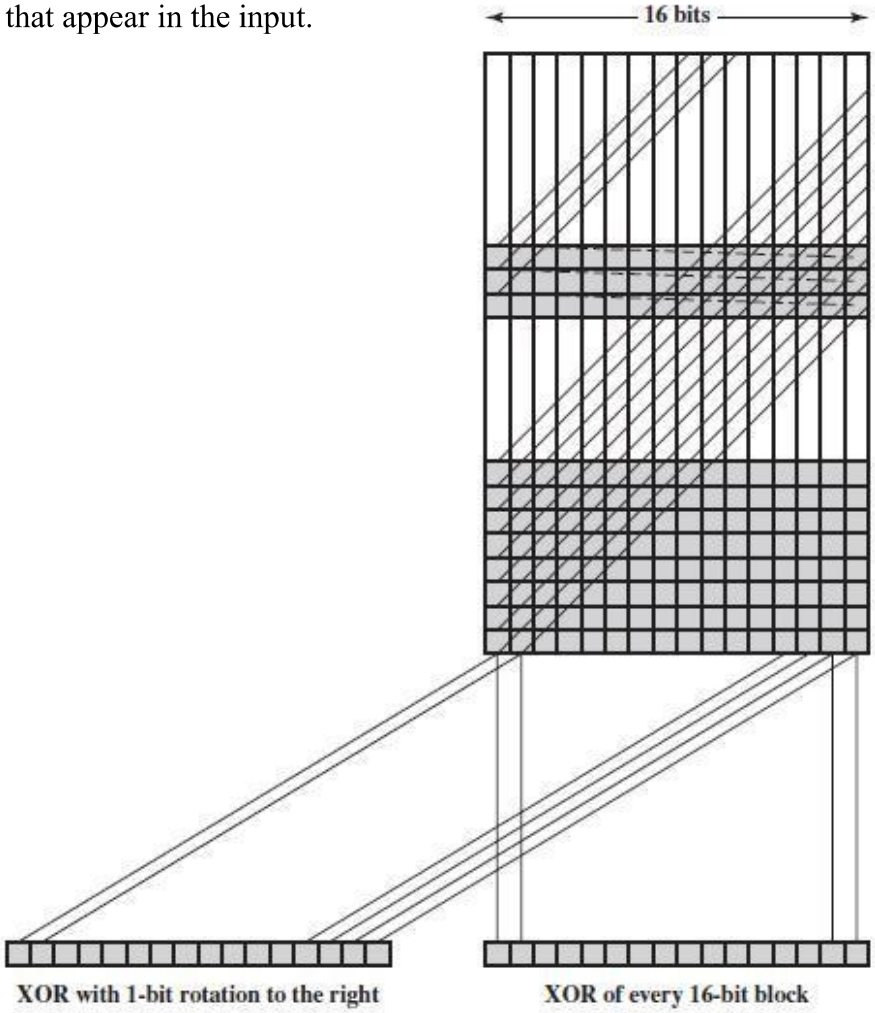


Figure: Two Simple Hash Functions

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. Given a 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.

Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted. A technique originally proposed by the National Bureau of Standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message that used the cipher block chaining (CBC) mode.

Given a message M consisting of a sequence of 64-bit blocks $X_1, X_2, \ldots, X_N$, define the hash code $h = H(M)$ as the block-by-block XOR of all blocks and append the hash code as the final block:

$$h = X_{N+1} = X_1 \oplus X_2 \oplus \ldots \oplus X_N$$

Next, encrypt the entire message plus hash code using CBC mode to produce the encrypted message Y1, Y2, …. , YN+1.

For example, by the definition of CBC, we have $X_1 = IV \oplus D(K, Y_1)$

$$X_i = Y_{i-1} \oplus D(K, Y_i)$$

$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

But X_{N+1} is the hash code:

$$X_{N+1} = X_1 \oplus X_2 \oplus \ldots \oplus X_N$$

$$= [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \ldots \oplus [Y_{N-1} \oplus D(K, Y_N)]$$

Because the terms in the preceding equation can be XORed in any order, it follows that the hash code would not change if the ciphertext blocks were permuted.

SECURITY OF HASH FUNCTION

We can group attacks on hash functions into two categories: brute-force attacks and cryptanalysis.

Brute-Force Attacks

The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. Recall from our discussion of hash functions that there are three desirable properties:

- **One-way:** For any given code h, it is computationally infeasible to find x such that $H(x) = h$.
- **Weak collision resistance:** For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
- **Strong collision resista ce:** It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.

For a hash code of length n, the level of effort required is as follows:

One way	2^n
Weak collision resistance	2^n
Strong collision resistance	$2^{n/2}$

Cryptanalyst

The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each. If necessary, the final block is padded to b bits. The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

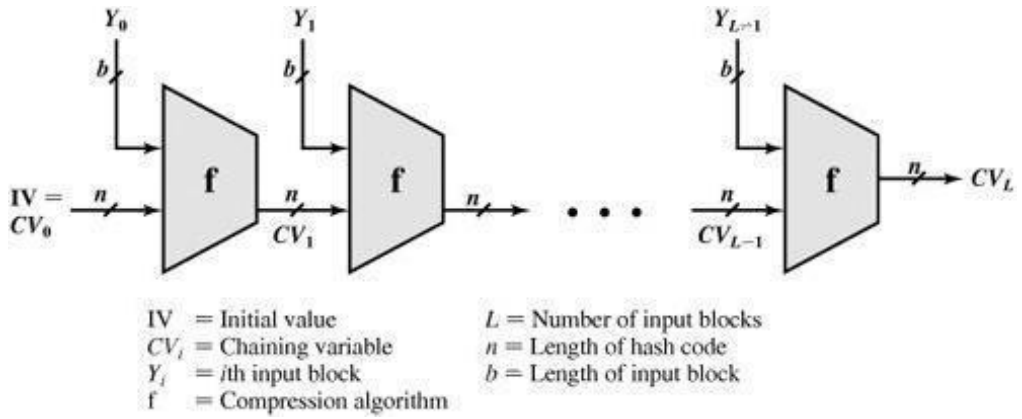


Figure: General Structure of secure hash code

The hash algorithm involves repeated use of a **compression function**, f, that takes two inputs (an *n*-bit input from the previous step, called the *chaining variable*, and a *b*-bit block) and produces an *n*-bit output. At the start of hashing, the chaining variable has an initial value that is specified as pa t of the algorithm. The final value of the chaining variable is the hash value. Often, $b > n$; hence the term *compression*. The hash function can be summarized as follows:

$$\mathbf{CV_0 = IV = Initial\ \ -bit\ value}$$
$$\mathbf{CV_i = f(CV_{i1}, Y_{i1})\ 1 \le i \le L}$$
$$\mathbf{H(M) = CV_L}$$

The motivation for this iterative structure is that if the compression function is collision resistant, then so is the resultant iterated hash function. Therefore, the structure can be used to produce a secure hash function to operate on a message of any length. The problem of designi g a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size.

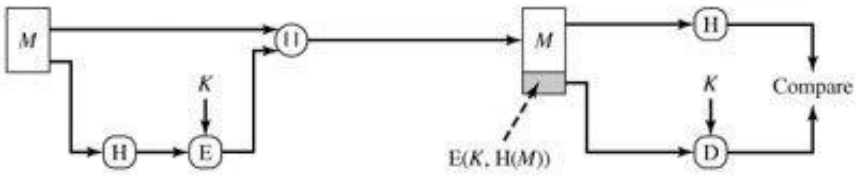
Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f. Once that is done, the attack must take into account the fixed value of IV. The attack on f depends on exploiting its internal structure.

Typically, as ith 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.

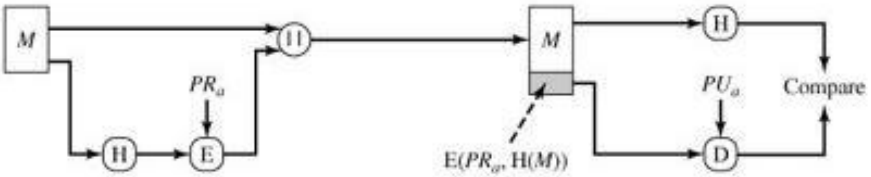
For any hash function there must exist collisions, because we are mapping a message of length at least equal to twice the block size *b* (because we must append a length field) into a hash code of length *n*, where $b \ge n$. What is required is that it is computationally infeasible to find collisions.

BIRTHDAY ATTACK

Birthday attack is used to find the collision in a cryptographic hash function. Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code C is transmitted with the corresponding unencrypted message *M*, then an opponent would need to find an *M*' such that H(*M*') = H(*M*) to substitute another message and fool the receiver. On average, the opponent would have to try about 2^{63} messages to find one that matches the hash code of the intercepted message.



In the above figure, only the hash code is encrypted, using symmetric encryption. This reduces the processi g burden for those applications that do not require confidentiality.



In above figure, only the hash code is encrypted, using public-key encryption and using the sender's private key. As with above, 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.

A different sort of attack is possible, based on the birthday paradox.

1. The source, A, is prepared to "sign" a message by appending the appropriate *m*-bit hash code and encrypting that hash code with A's private key.
2. The opponent generates 2m/2 variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations on the fraudulent message to be substituted for the real one.
3. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
4. 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. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of 2^{32}.

MESSAGE DIGEST 5: MD5

- Developed by Ron Rivest at MIT
- Input: a message of arbitrary length
- Output: 128-bit message digest
- 32-bit word units, 512-bit blocks

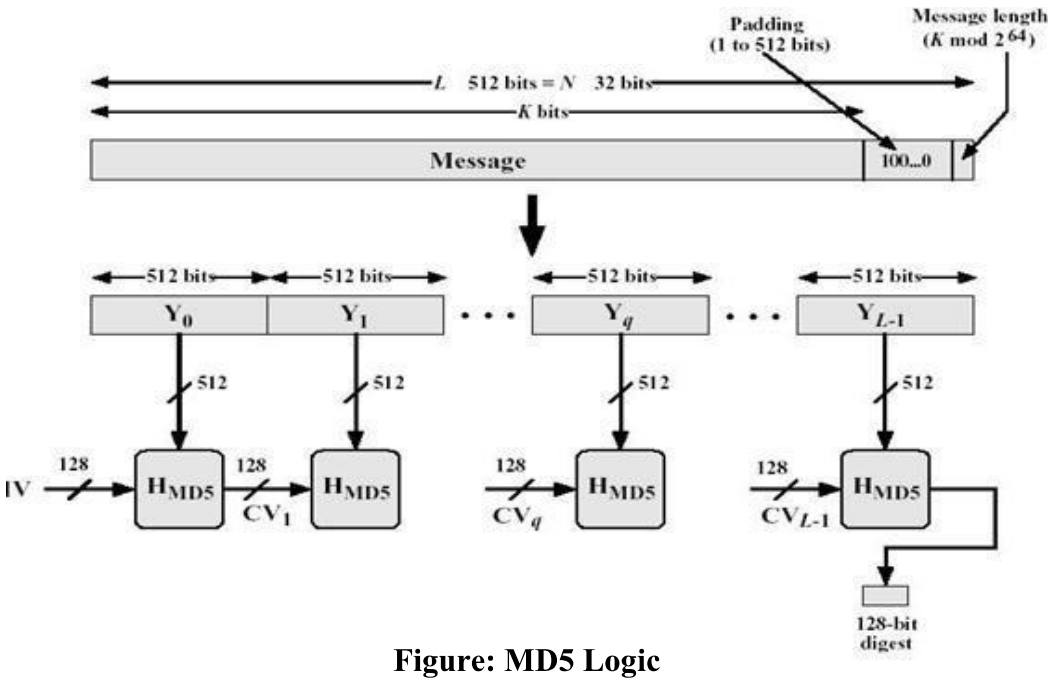


Figure: MD5 Logic

MD5 Logic:

Step 1: Append padding bits

- The message is Padded so that its bit length ≡ 448 mod 512 (i.e., the length of padded message is 64 bits less than an integer multiple of 512 bits)
- Padding is always added, even if the message is already of the desired length (1 to 512 bits)
- Padding bits: 1000….0 (a single 1-bit followed by the necessary number of 0-bits)

Step 2: Append length

- A 64-bit length: contains the length of the original message modulo 264
- The expanded message is Y0, Y1, …, YL-1; the total length is L × 512 bits
- The expanded message can be thought of as a multiple of 16 32-bit words
- Let M[0 … N-1] denote the word of the resulting message, where N = L × 16

Step 3: Initialize MD buffer

- 128-bit buffer (four 32-bit registers A,B,C,D) is used to hold intermediate and final results of the hash function
- A,B,C,D are initialized to the following values

A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476

✓ Stored in *little-endian* format (least significant byte of a word in the low-address byte position)

- E.g. word A : 01 23 45 67 (low address … high address)
- word B : 89 AB CD EF
- word C : FE DC BA 98
- word D : 76 54 32 10

Step 4: Process message in 512-bit (16-word) blocks

- Heart of the algorithm called a compression function Consists of 4 rounds
- The 4 rounds have a similar structure, but each uses a different primitive logical functions, referred to as F, G, H, and I
- Each round takes as input the current 512-bit block (Yq), 128-bit buffer value ABCD and updates the contents of the buffer
- Each round also uses the table T[1 … 64], constructed from the sine function;

  T[i] = 232 × abs(sin(i))
- The output of 4th round is added to the CVq to p oduce CVq+1

Step 5: Output

After all L 512-bit blocks have been proc ss d, the output from the Lth stage is the 128-bit message digest

$$\mathbf{CV_0 = IV}$$

$$\mathbf{CVq+1 = SUM32(CVq, RFI[Yq, RFH[Yq, RFG[Yq, RFF[Yq, CVq]]])}$$

$$\mathbf{MD = CV_L}$$

Where IV = initial value of the ABCD buffer, defined in step

3 Yq = the qth 512-bit block of the message

L = the number of blocks in the message (including padding and length fields) CVq = ch ining variable processed with the qth block of the message

RFx = round function using primitive logical function

x MD = final message digest value

SUM32 = addition modulo 232 performed separately on each word

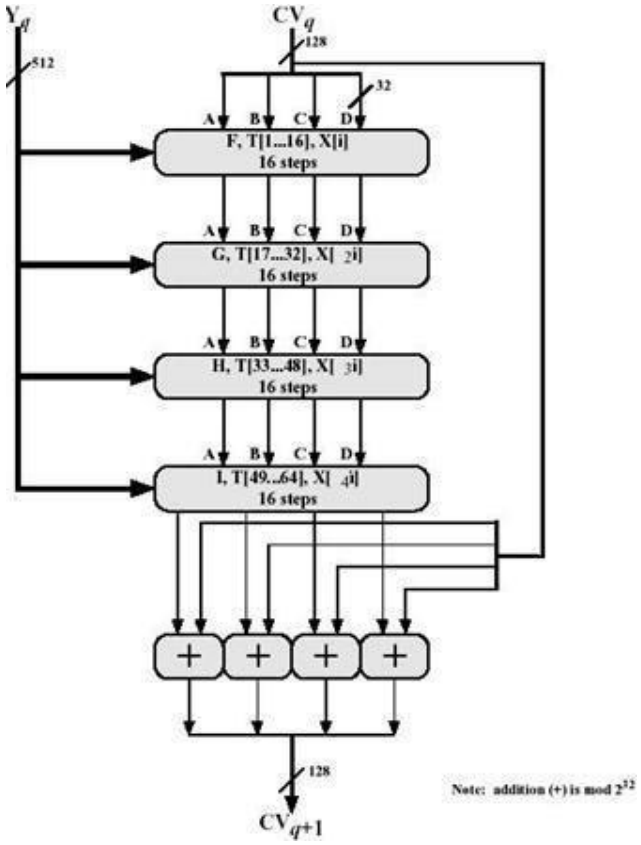


Figure: MD5 processing of a single 512-bit block (MD5 compression function)

MD5 Compression Function:

- Each round consists of a sequence of 16 steps operating on the buffer ABCD
- Each step is of the form

$$a \leftarrow b + ((\ a + g(b, c, d) + X[k] + T[i] <<< s\ )$$

where a,b,c,d = the 4 words of the buffer, in a specified order that varies across steps g = one of the primitive functions F, G, H, I

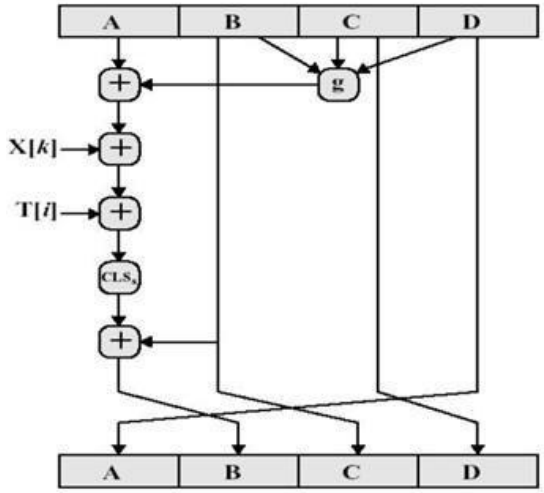
<<s = circular left shift (rotation) of the 32-bit arguments by s bits

X[k] = M[q × 16 + k] = the kth 32-bit word in the qth 512-bit block of the message

T[i] = the ith 32-bit word in table T

+ = addition modulo 232

MD5 Operation



One of the 4 primitive logical functions is used in each 4 rounds of the algorithm

- Each primitive function takes three 32-bit words as input and produces a 32-bit word output
- Each function performs a set of bitwise logical operations

Round	Primitive function g	g(b, c, d)
1	F(b, c, d)	(b ∧ c) ∨ (b‟ ∧ d)
2	G(b, c, d)	(b ∧ d) ∨ (c ∧ d‟)
3	H(b, c, d)	b ⊕ c ⊕ d
4	I(b c, d)	c ⊕ (b ∨ d‟)

b	C	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0

TRUTH TABLE

- The array of 32-bit words X[0..15] holds the value of current 512-bit input block being processed

Within a round, each of the 16 words of X[i] is used exactly once, during one step

- The order in which these words is used varies from round to round
- In the first round, the words are used in their original order
- For rounds 2 through 4, the following permutations are used
  - » $\rho 2(i) = (1 + 5i) \bmod 16$
  - » $\rho 3(i) = (5 + 3i) \bmod 16$
  - » $\rho 4(I) = 7i \bmod 16$

SECURE HASH ALGORITHM

- Developed by NIST (National Institute of Standards and Technology)
  - Published as a FIPS 180 in 1993
  - A revised version is issued as FIPS 180-1 IN 1995
  - Generally referred to as SHA-1

- SHA is based on the hash function MD4 and its design closely models MD4.
- SHA- 1 produces a hash value of 160 bits.
- Revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384 and 512 bits, known as SHA-256, SHA-384 and SHA-512.

SHA-512 Logic:

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks.

Step 1: Append padding bits

- The message is Padd d so that its bit length is congruent to 896 modulo 1024 [length K ≡ 896 mod 1024]

- Padding is always added, even if the message is already of the desired length.

- Thus, the number of padding bits is in the range of 1 to 1024.

- The padding consists of a single 1-bit followed by the necessary number of 0-bits.

Step 2: Append length

- A block of 128-bits is appended to the message.
- This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure, the expanded message is represented as the sequence of

1024-bit blocks M_1, M_2,.., M_n, so that the total length of the expanded message is N x 1024 bits.

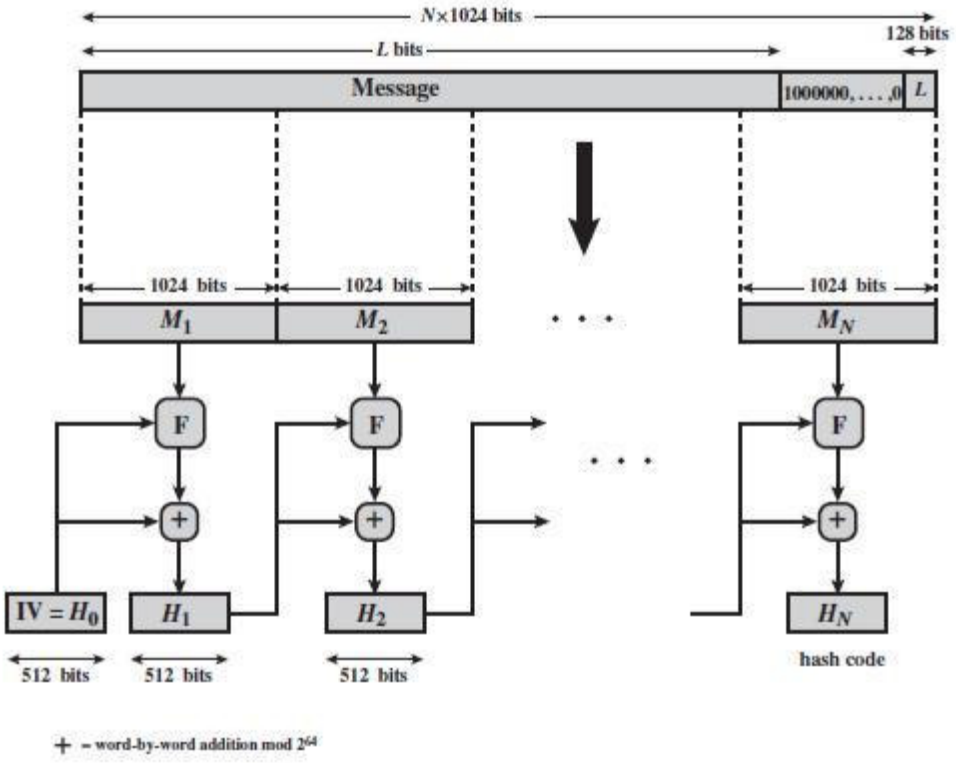


Figure: Message Digest Generation using SHA-512

Step 3: Initialize hash buffer

- A 512-bit buffer is used to hold intermediate and final results of the hash function.
- The buffer can be r pr sented as eight 64-bit registers (a, b, c, d, e, f, g and h).
- These registers are initialized to the following 64-bit integers (hexadecimal values):

```
a = 6A09E667F3BCC908    e = 510E527FADE682D1
b = BB67AE8584CAA73B    f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B    g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1    h = 5BE0CD19137E2179
```

- These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position.
- These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

Step 4: Process message in 1024-bit (128-word) blocks

The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in above figure.

- Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, Hi-1.
- Each round t makes use of a 64-bit value Wt, derived from the current 1024-bit block being processed (M_i).

- Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ indicates one of the 80 rounds.
- The output of the eightieth round is added to the input to the first round (H_{i-1}) to produce H_i. The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1}, using addition modulo 2^{64}.

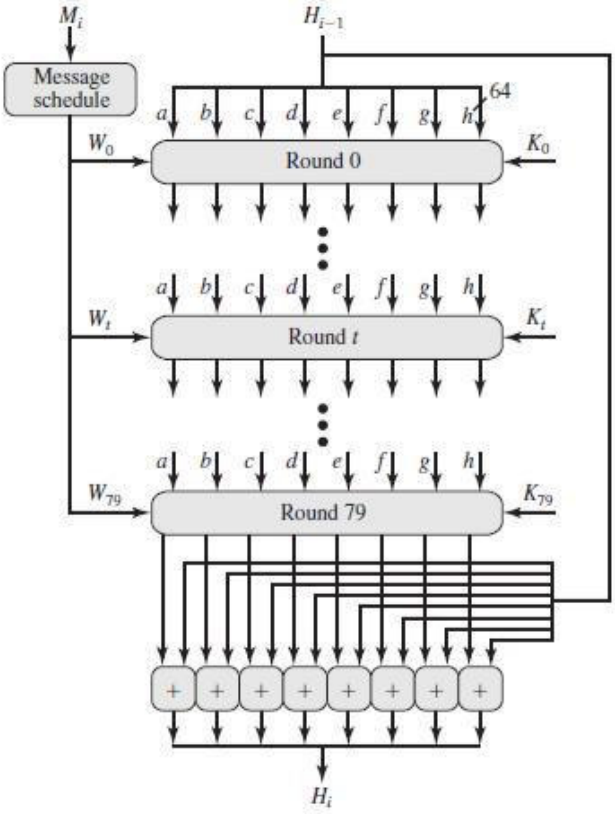


Figure: SHA-512 processing of a single 1024-bit block

Step 5: Output

After all *N* 1024-bit blocks have been processed, the output from the N^{th} stage is the 512-bit message digest.

$$\mathbf{H_0 = IV, H_i = SUM_{64}(H_{i-1}, abcdefgh_i)}$$

$$\mathbf{MD = H_N}$$

where,**IV** = initial value of the abcdefgh buffer, defined in step 3.

abcdefgh$_i$ = the output of the last round of processing of the i^{th} message block.

N = the number of blocks in the message (including padding and length fields).

SUM$_{64}$ = Addition modulo 2^{64} performed separately on each word of the pair of inputs.**MD** = final message digest value.

SHA-512 Round Functions:

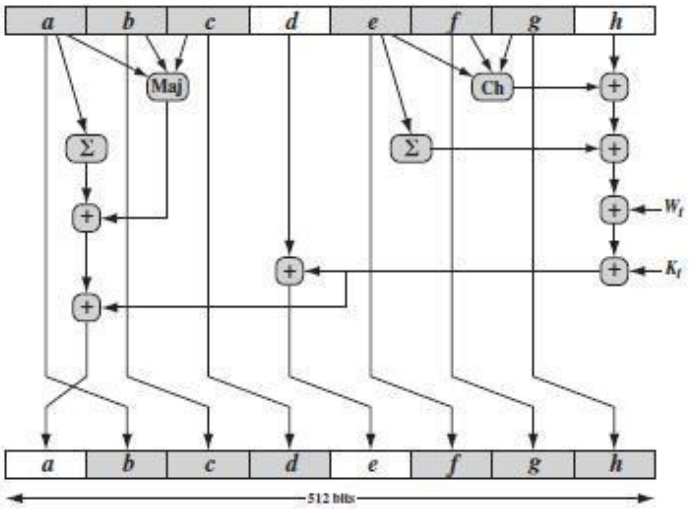


Figure: Elementary SHA-512 operation (single round)

Each round is defined by the following set of equations:

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

where

t = step number; $0 \le t \le 79$

$\text{Ch}(e, f, g)$ = $(e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$
the conditional function: If e then f else g

$\text{Maj}(a, b, c)$ = $(a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

$\left(\sum_0^{512} a\right)$ = $\text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$

$\left(\sum_1^{512} e\right)$ = $\text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$

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

W_t = a 64-bit word derived from the current 1024-bit input block

K_t = a 64-bit additive constant

$+$ = addition modulo 2^{64}

It remains to indic te how the 64-bit word values W_t are derived from the 1024-bit message. The first 16 values of W_t are taken directly from the 16 words of the current block. The remaining values are defined as:

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

where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$
$$\text{ROTR}^n(x) = \text{circular right shift (rotation) of the 64-bit argument } x \text{ by } n \text{ bits}$$
$$\text{SHR}^n(x) = \text{left shift of the 64-bit argument } x \text{ by } n \text{ bits with padding by zeros on the right}$$
$$+ = \text{addition modulo } 2^{64}$$

Thus, in the first 16 steps of processing, the value of W_t is equal to the corresponding word in the message block. For the remaining 64 steps, the value of W_t consists of the circular left shift by one bit of the XOR of four of the preceding values of W_t, with two of those values subjected to shift and rotate operations.

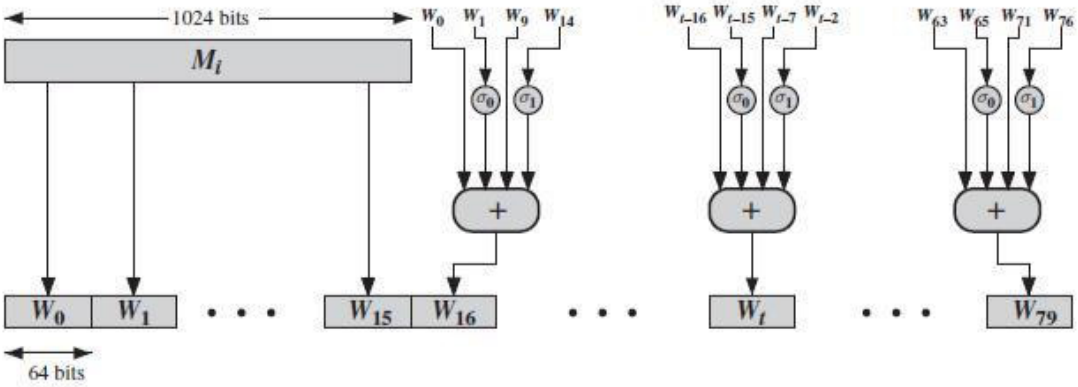


Fig. Creation of 80-word Input Sequence

4. HMAC and CMAC

a. MACs BASED ON HASH FUNCTIONS (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, in available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- 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.

HMAC Algorithm:

HMAC defines the following terms.

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, $0 \leq i \leq (L - 1)$

L = number of blocks in M

b = number of bits in a block

n = length of hash code produced by embedded hash fu ction

K = secret key; recommended length is $\geq n$; if key length s 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

Then HMAC can be expressed as

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

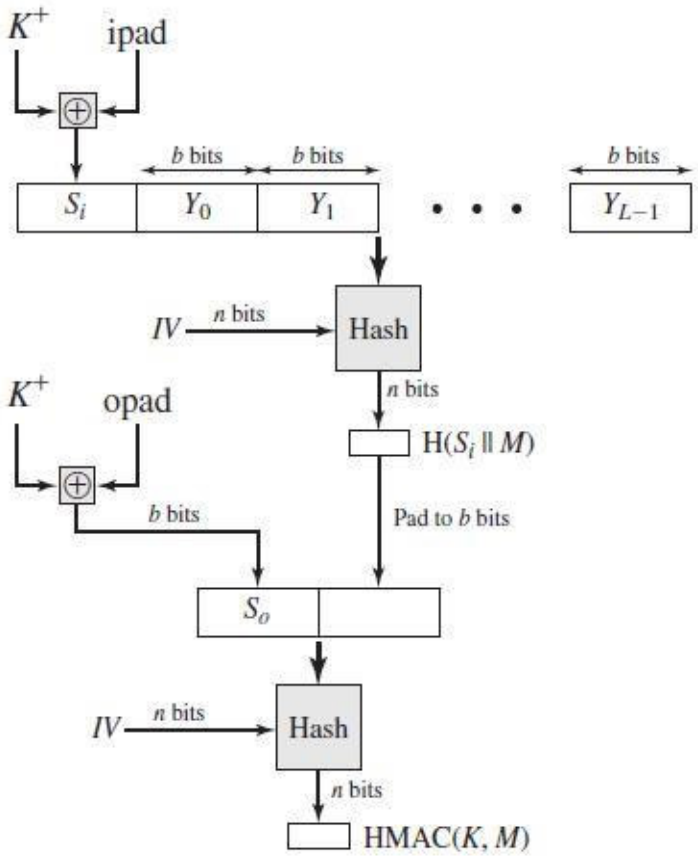


Figure: HMAC Structure

The algorithm is as follows:

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zeroes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

The XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping o e-half of the bits of K, using a different set of bits. In effect, by passing S_i and S_o through the compression function of the hash algorithm, we have pseudo randomly generated two keys from K.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for S_i, S_o, and the block produced from the inner hash). A more efficient, two quantities are precomputed:

$$\mathbf{f}(IV, (K^+ \oplus \mathbf{ipad}))$$
$$\mathbf{f}(IV, (K^+ \quad \mathbf{opad}))$$

where 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.

In effect, the precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This more efficient implementation is especially worthwhile if most of the messages for which a MAC is computed are short.

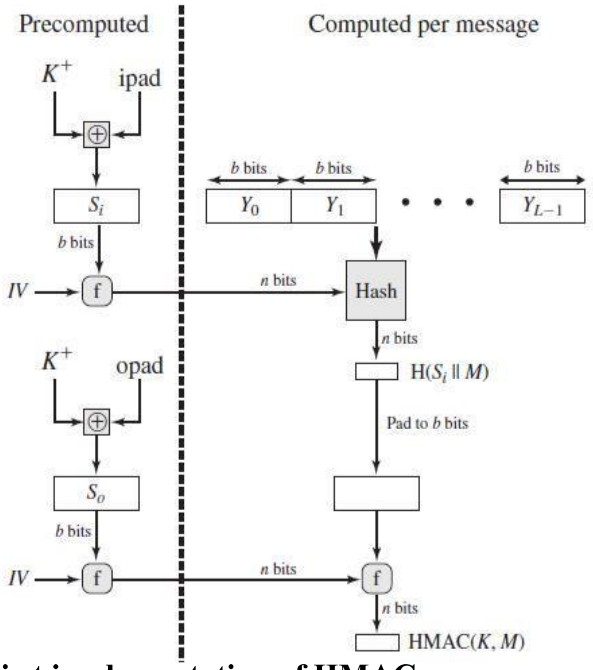


Figure: Efficie t implementation of HMAC

Security of HMAC :

The security of any MAC function based on an embedded hash function. The security of a MAC function is generally 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. 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.

In the first attack, we can view the 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.

In the second attack, the attacker is looking for two messages *M* and *M'* that produce the same hash: H(*M*) = H(*M'*). This is the birthday attack. This requires a level of effort of $2^{n/2}$ for a hash length of *n*.

b. CIPHER-BASED MESSAGE AUTHENTICATION CODE (CMAC)

CMAC is a block cipher-based message authentication code algorithm. It may be used to provide assurance of the authenticity and, hence, the integrity of binary data.

When the message is an integer multiple *n* of the cipher block length *b*. For AES, *b* = 128, and for triple DES, *b* = 64. The message s divided into *n* blocks (M_1, M_2, …, M_n). The algorithm makes use of a *k*-bit enc yption key *K* and a *b*-bit constant, K_1. For AES, the key size *k* is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows.

$$C_1 = \mathrm{E}(K, M_1)$$
$$C_2 = \mathrm{E}(K, [M_2 \quad C_1])$$
$$C_3 = \mathrm{E}(K, [M_3 \quad C_2])$$
$$\vdots$$
$$C = \mathrm{E}(K, [M_n \quad C_{n-1} \quad K_1])$$
$$T = \mathrm{MSB}_{Tlen}(C_n)$$

where

T	= m ssage authentication code, also referred to as the tag
Tlen	= bit length of T
MSBs(*X*)	= the *s* leftmost bits of the bit string *X*

If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many 0s as necessary so that the final block is also of length b. The CMAC operation then proceeds as before, except that a different b-bit key K_2 is used instead of K_1.

The two b-bit keys are derived from the k-bit encryption key as follows.

$$L = E(K, 0^b)$$
$$K_1 = L \cdot x$$
$$K_2 = L \cdot x^2 = (L \cdot x) \cdot x$$

where multiplication ($\cdot$) is done in the finite field GF(2^b) and x and x^2 are first and second-order polynomials that are elements of GF(2^b).

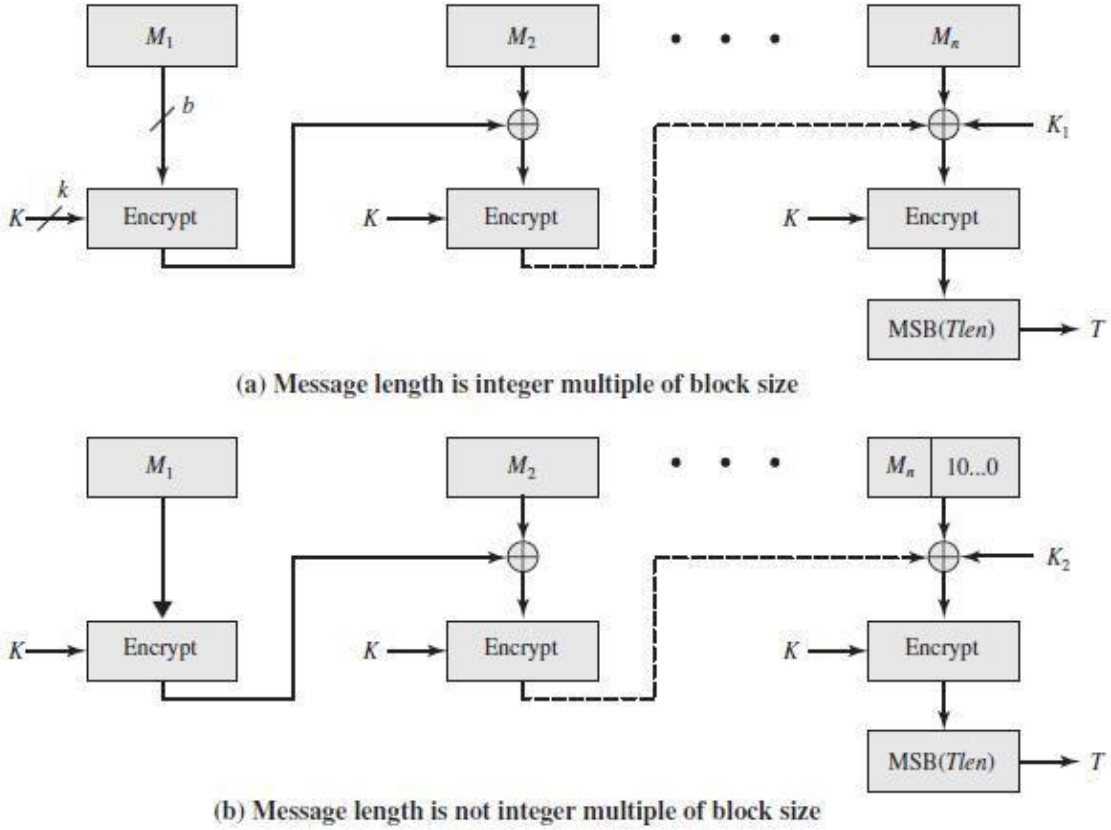


Figure: Cipher-Based Message Authentication Code

To generate K_1 and K_2, the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting ciphertext by a left shift of one bit and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.

AUTHENTICATION PROTOCOLS:

Two general areas are

- Mutual authentication and
- One-way authentication

a. Mutual Authentication

✓ An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other‟s identity and to exchange session keys.

- Central to the problem of authenticated key exchange are two issues:
  - Confidentiality &
  - Timeliness

✓ To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form.

✓ At minimum a successful replay can disrupt operatio s by presenting parties with messages that appear genuine but are not.

The following are examples of replay attacks:

- Simple replay: The opponent simply copies a message and replays it later.
- Repetition that can be logged: An opponent can replay a time stamped message within the valid time window.

✓ **Repetition that cannot be detected:** This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arr ves.

- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recog ize the difference between messages sent and messages received on the basis of content.

General approaches:

- **Timestamps:**
  - Party A acc pts a message as fresh only if the message contains a timestamp that, in A‟s judgment, is close enough to A‟s knowledge of current time.
  - This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:**
  - **p** Party A, expecting a fresh message from B, first sends B a nonce(challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

Symmetric Encryption Approaches:

A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC).

- Each party in the network shares a secret key, known as a master key, with the KDC.
- The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution.

1. A → KDC: $ID_A||ID_B||N_1$
2. KDC → A: $E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. A → B: $E(K_b, [K_s||ID_A])$
4. A → A: $E(K_s, N_2)$
5. A → B: $E(K_s, f(N_2))$

- Secret keys K_a and K_b are shared betw n A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K, to A and B.
- Suppose that an opponent, X, has been able to compromise an old session key. Her proposal assumes that the master keys, K_a and K_b, are secure, and it consists of the following steps:

1. A → KDC: $ID_A||ID_B$
2. KDC → A: $E(K_a, [K_s||ID_B||T||E(K_b, [K_s||ID_A||T])])$
3. A → B: $E(K_b, [K_s||ID_A||T])$
4. B → A: $E(K_s, N_1)$
5. A → B: $E(K_s, f(N_1))$

- T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$$|\text{Clock } T| < \Delta t_1 + \Delta t_2$$

where $t1$ is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and $t2$ is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted

using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

1. A → B: $ID_A||N_a$
2. B → KDC: $ID_B||N_b||E(K_b, [ID_A||N_a||T_b])$
3. KDC → A: $E(K_a, [ID_B||N_a||K_s||T_b])||E(K_b,[ID_A||K_s||T_b])||N_b$
4. A → B: $E(K_b, [ID_A||K_s||T_b])||E(K_s, N_b)$

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.
2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message

to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.

3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authenticatio s, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a) and it provides A with a session key (K_s) and the time limit on its use (T_b).
4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session k y. The ticket provides B with the secret key that is used to decrypt E(K_s, N_b) to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

A desires a new session with **B**. The following protocol ensues:

1. A ⟶ B: $E(K_b, [ID_A||K_s||T_b])||N'_a$
2. B ⟶ A: $N'_b||E(K_s, N'_a)$
3. A ⟶ B: $E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces *N'a* and *N'b* assure each party that there is no replay attack.

Public key encryption approaches

This protocol assumes that each of the two parties is in possession of the current public key of the other.

1. A ⟶ AS: $ID_A||ID_B$
2. AS ⟶ A: $E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
3. A ⟶ B: $E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$ $||E(PU_b, E(PR_a, [K_s||T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates.

The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against eplays of compromised keys.

1. A ⟶ KDC: $ID_A||ID_B$
2. KDC ⟶ A: $E(PR_{auth}, [ID_B||PU_b])$
3. A ⟶ B: $E(PU_b, [N_a||ID_A])$
4. B ⟶ KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$
5. KDC ⟶ B: $E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_B]))$
6. B ⟶ A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_B)||N_b]))$
7. A ⟶ B: $E(K_s, N_b)$

This seems to be secure protocol that takes into account the various attacks. The authors themselves spotted a flaw and submitted a revised version of the algorithm is as follo s.

1. A ⟶ KDC: $ID_A||ID_B$
2. KDC ⟶ A: $E(PR_{auth}, [ID_B||PU_b])$
3. A ⟶ B: $E(PU_b, [N_a||ID_A])$
4. B ⟶ KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$
5. KDC ⟶ B: $E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_A||ID_B])$
6. B ⟶ A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_A||ID_B)||N_b]))$
7. A ⟶ B: $E(K_s, N_b)$

b. One way Authentication

Symmetric Encryption Approach

This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message. The KDC strategy is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated.

For a message with content *M*, the seque ce is as follows:

1. A ⟶ KDC: $ID_A||ID_B||N_1$
2. KDC ⟶ A: $E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. A ⟶ B: $E(K_b, [K_s||ID_A])||E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides level of authentication that the sender is A.

Public Key encryption Approaches

This approach requires that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or t ice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \longrightarrow B: E(PU_b, K_s)||E(K_s, M)$$

If authentication is the primary concern, then a digital signature may suffice.

$$A \longrightarrow B: M||E(PR_a, H(M))$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud.

Both the message and signature can be encrypted with the recipient's public key:

$$A \longrightarrow B: E(PU_b, [M||E(PR_a, H(M))])$$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate.

$$A \longrightarrow B: M||E(PR_a, H(M))||E(PR_{as}, [T||ID_A||PU_a])$$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obta n the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted w th a one-time secret key; the secret key is also transmitted, encrypted with B's public key.

DIGITAL SIGNATURE ALGORITHM

NIST DIGITAL SIGNATURE ALGORITHM

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Algorithm (DSA).
The DSA makes use of the Secure Hash Algorithm (SHA).

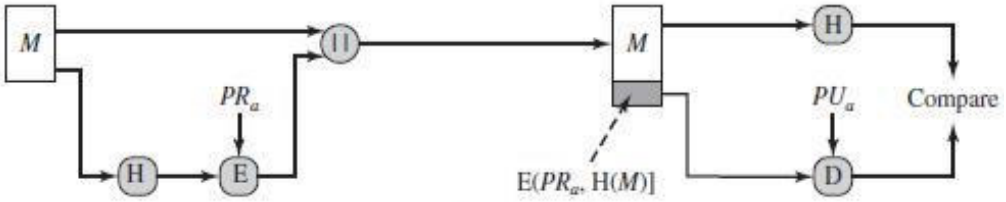
The DSA as originally proposed in 1991 and revised in 1993, 1996 and then 2000 an expanded version of the standard was issued as FIPS 186-2, subsequently updated to FIPS 186-3 in 2009.

The DSA uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

The RSA Approach:

- In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length.
- This hash code is then encrypted using the sender‟s private key to form the signature.
- Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code.
- The recipient also decrypts the signature using the sender‟s public key.

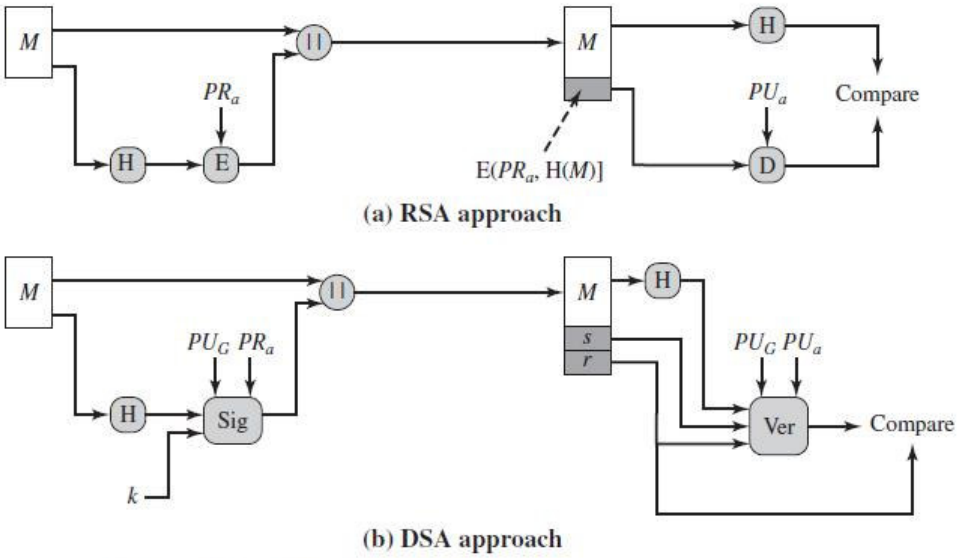
- If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.



(a) RSA approach

The DSA Approach:

- The DSA approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number *k* generated for this particular signature.
- The signature function also depends on the sender‟s private key (PR_a) and a set of parameters known to a g oup of communicating principals. We can consider this set to constitute global public key (PU_G). The result is a signature consisting of two components, labeled *s* and *r*.
- At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function.
- The verification function also depends on the global public key as well as the sender‟s public key (PU_a), which is paired with the sender‟s private key.
- The output of the verification function is a value that is equal to the signature component *r* if the signature is valid.
- The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.



(a) RSA approach

(b) DSA approach

Figure 13.3 Two Approaches to Digital Signatures

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$ i.e., bit length of N bits

g $= h(p - 1)/q \bmod p$, where h is any integer with $1 < h < (p - 1)$ such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1} (H(M) + xr)] \bmod q$

Signature $= (r, s)$

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$

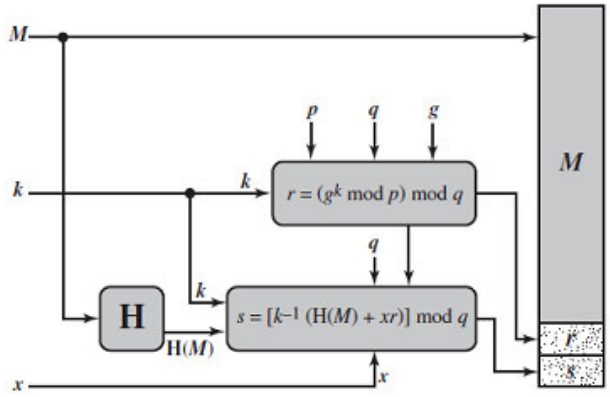
TEST: $v = r'$

M = message to be signed

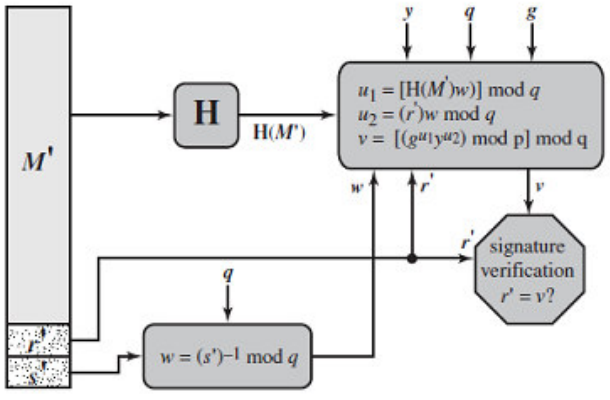
H(M) = hash of M using SHA-1

M', r', s' = received versions of M, r, s

Figure 13.4 The Digital Signature Algorithm (DSA)



(a) Signing



(b) Verifying

Figure 13.5 DSA Signing and Verifying

ELGAMAL DIGITAL SIGNATURE SCHEME

Elgamal encryption scheme is designed to enable encryption by a user's public key with decryption by the user‟s private key. The Elgamal signature scheme involves the use of the private key for encryption and the public key for decryption.
For a prime number q, if α is a primitive root of q, then

$\alpha, \alpha^2, \cdots, \alpha^{q-1}$ are distinct (mod q).

It can be shown that, if a is a primitive root of α, then
1. For any integer m, $\alpha^m \equiv 1 \pmod q$ if and only if $m \equiv 0 \pmod{q-1}$.
2. For any integers, i, j, $\alpha^i \equiv \alpha^j \pmod q$ if and only if $i \equiv j \pmod{q-1}$.

As with Elgamal encryption, the global elements of **Elgamal digital signature** are a prime number q and a, which is a primitive root of q.
User A **generates a private/public key pair** as follows.
1. Generate a random integer X_A, such that $1 < X_A < q - 1$.
2. Compute $Y_A = \alpha^{X_A} \bmod q$.
3. A‟s private key is X_A; A‟s pubic key is $\{q, \alpha, Y_A\}$.

To sign a message *M*, user A first computes the hash $m = H(M)$, such that m is an integer in the range $0 \le m \le q - 1$. A then **forms a digital signature** as follows.
1. Choose a random integer K such that $1 \le K \le q - 1$ and $\gcd(K, q - 1) = 1$. That is, K is relatively prime to $q - 1$.
2. Compute $S_1 = \alpha^K \bmod q$. Note that this is the same as the computation of C_1 for Elgamal encryption.
3. Compute $K^{-1} \bmod (q - 1)$. That is, compute the inverse of K modulo $q - 1$.
4. Compute $S_2 = K^{-1}(m - X_A S_1) \bmod (q - 1)$.
5. The signature consists of the pair (S_1, S_2).

Any user B can **verify the signature** as follows.
1. Compute $V_1 = \alpha^m \bmod q$.
2. Compute $V_2 = Y_A^{S_1} S_1^{S_2} \bmod q$.

The signature is valid if $V_1 = V_2$. Assume that the equality is true. Then we have

$\alpha^m \bmod q = Y_A^{S_1} S_1^{S_2} \bmod q$ assume $V1 = V2$

$\alpha^m \bmod q = \alpha^{X_A S_1} \alpha^{KS_2} \bmod q$ substituting for Y_A and S_1

$\alpha^{m - X_A S_1} \bmod q = \alpha^{KS_2} \bmod q$ rearranging terms

$m - X_A S_1 \equiv KS_2 \bmod (q - 1)$ property of primitive roots

$m - X_A S_1 \equiv KK^{-1}(m - X_A S_1) \bmod (q - 1)$ substituting for $S2$

Example
Assume, $q = 19$. It has primitive roots {2, 3, 10, 13, 14, 15}. We choose $\alpha = 10$.
Alice **generates a key pair** as follows:
1. Alice chooses $X_A = 16$.
2. Then $Y_A = \alpha^{X_A} \bmod q = \alpha^{16} \bmod 19 = 4$.
3. Alice‟s private key is 16; Alice‟s pubic key is $\{q, a, Y_A\} = \{19, 10, 4\}$.

Suppose Alice wants to **sign a message** with hash value $m = 14$.
1. Alice chooses $K = 5$, which is relatively prime to $q - 1 = 18$.
2. $S_1 = \alpha^K \bmod q = 10^5 \bmod 19 = 3$.
3. $K^{-1} \bmod (q - 1) = 5^{-1} \bmod 18 = 11$.
4. $S_2 = K^{-1}(m - X_A S_1) \bmod (q - 1) = 11\,(14 - (16)(3)) \bmod 18 = -374 \bmod 18 = 4$.

Bob can **verify the signature** as follows.
1. $V_1 = \alpha^m \bmod q = 10^{14} \bmod 19 = 16$.
2. $V_2 = Y_A^{S_1} S_1^{S_2} \bmod q = (4^3)(3^4) \bmod 19 = 5184 \bmod 19 = 16$.

Thus, the signature is valid.

b. SCHNORR DIGITAL SIGNATURE SCHEME

- The Schnorr signature scheme is based on discrete logarithms. The Schnorr scheme minimizes the message-dependent amount of computation required to generate a signature.
- The main work for signature generation does not depend on the message and can be done during the idle time of the processor.
- The message-dependent part of the signature generation requires multiplying a $2n$-bit integer with an n-bit integer.
- The scheme is based on using a prime modulus p, with $p - 1$ having a prime factor q of appropriate size; that is, $p - 1 \equiv \pmod{q}$. Typically, we use $p \approx 2^{1024}$ and $q \approx 2^{160}$. Thus, p is a 1024-bit number, and q is a 160-bit number, which is also the length of the SHA-1 hash value.

The first part of this scheme is the **generation of a private/public key pair**, which consists of the following steps.

1. Choose primes p and q, such that q is a prime factor of $p - 1$.
2. Choose an integer a, such that $\alpha_q = 1 \bmod p$. The values a, p, and q comprise a global public key that can be common to a group of users.
3. Choose a random integer s with $0 < s < q$. This is the user‟s private key.
4. Calculate $v = a^{-s} \bmod p$. This is the user‟s public key.

A user with private key s and public key v **generates a signature** as follows.

1. Choose a random integer r with $0 < r < q$ and compute $x = a^r \bmod p$. This computation is a preprocessing stage independent of the message M to be signed.
2. Concatenate the message with x and hash the result to compute the value e: $\boldsymbol{e = H(M \,||\, x)}$
3. Compute $y = (r + se) \bmod q$. The signature consists of the pair (e, y).

Any other user can **verify the signature** as follows.

1. Compute $x' = a^y v^e \bmod p$.
2. Verify that $e = H(M \,||\, x')$.

$x' \equiv a^y v^e \equiv a^y a^{-se} \equiv a^{y-se} \equiv a^r \equiv x \pmod{p}$

Hence, $\boldsymbol{H(M \,||\, x') = H(M \,||\, x).}$