CRYPTOGRAPHY AND NETWORK SECURITY
UNIT-4

Syllabus: Application of Cryptographic Hash Functions, Requirements & Security, Secure Hash Algorithm, Message Authentication Functions, Requirements & Security, HMAC & CMAC. Digital Signatures, NIST Digital Signature Algorithm. Key management & distribution.

4.1. Hash Functions:

A hash function H accepts a variable-length block of data *M* as input and produces a fixed-size hash value

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

Principal object is data integrity

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

Cryptographic hash function

- 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)

(b) two data objects that map to the same hash result (the collision-free property)

4.2.Applications of cryptographic hash functions:

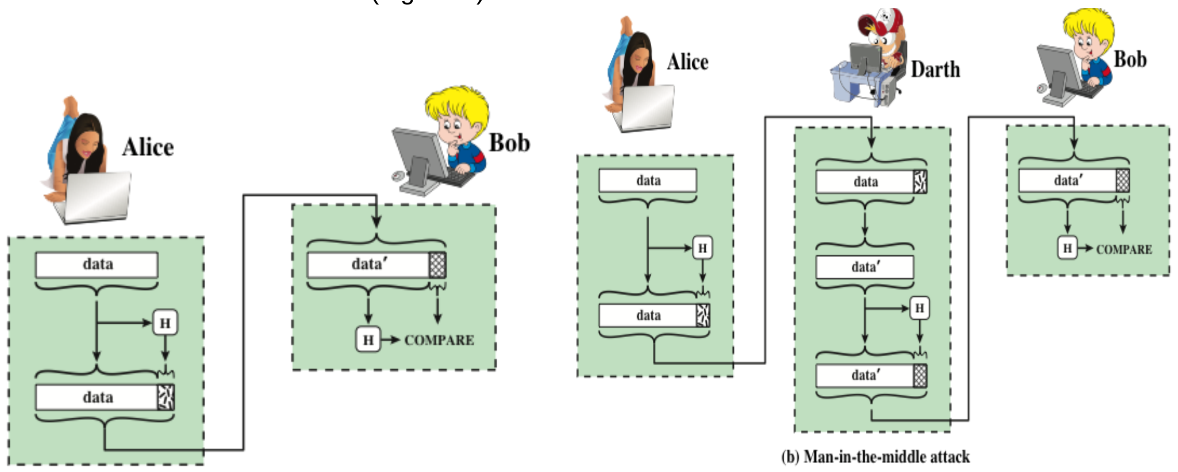
- Message Authentication
- Digital signature
- Other applications

4.2.1Message 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.**

The essence of the use of a hash function for message authentication is as follows.

- 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 (Figure a).



(b) Man-in-the-middle attack

(a) Use of hash function to check data integrity

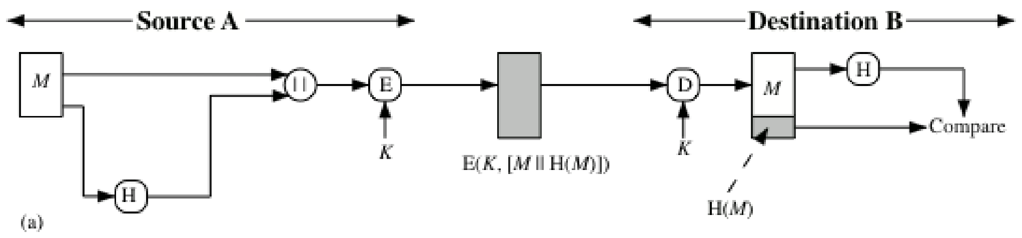
Figure 11.2 Attack Against Hash Function

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 Figure b. 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.

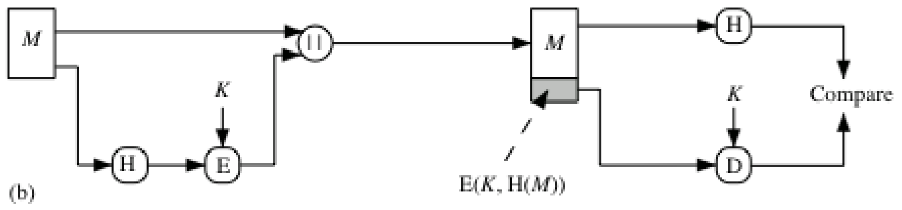
Ways Hash Code Can Be Used to Provide Message Authentication:

A variety of ways in which a hash code can be used to provide message authentication, as follows:

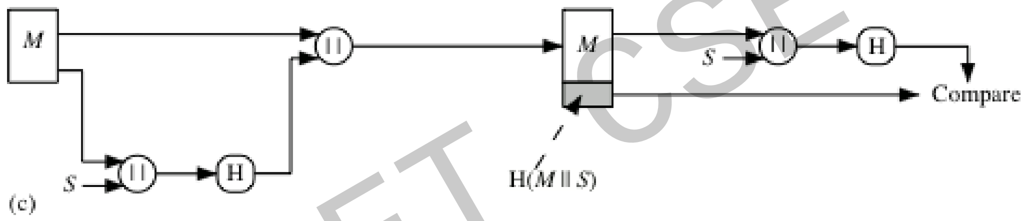
a) The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied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



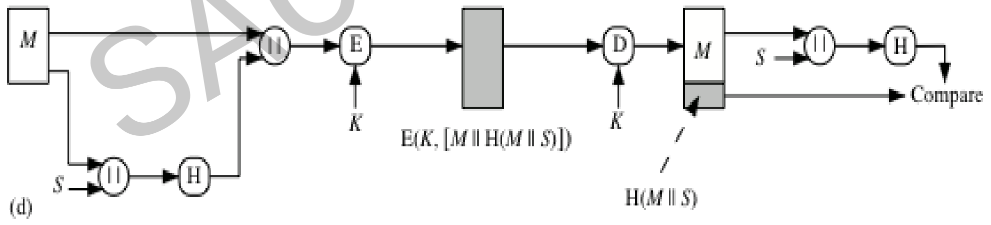
b) Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



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 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.



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

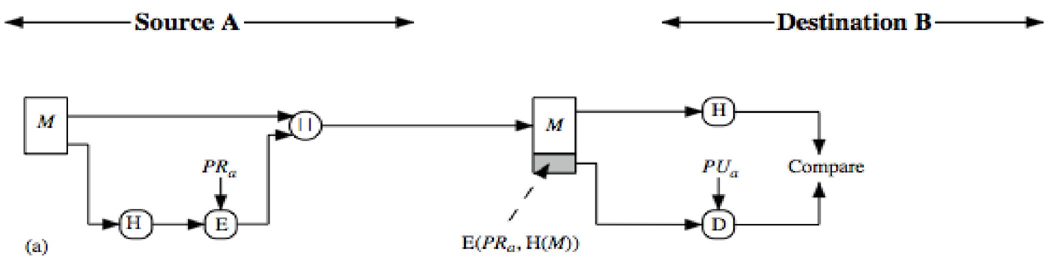


Digital Signatures:

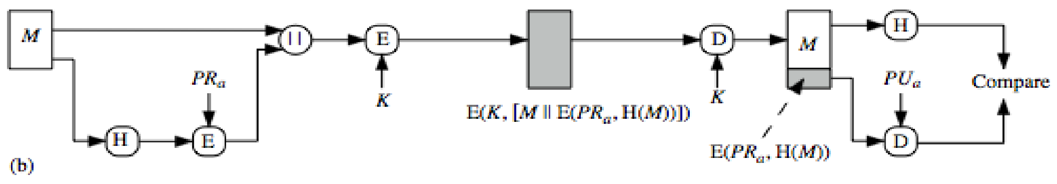
In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature.

In this case an attacker who wishes to alter the message would need to know the user's private key. There are **two types** how a hash code is used to provide a digital signature:

a. The hash code is encrypted, using public-key encryption and using 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:

- to create a one-way password file that store hash of password not actual password
- for intrusion detection and virus detection it keep & check hash of files on system
- pseudorandom function (PRF) or pseudorandom number generator (PRNG).

4.3. Requirements and Security

Here there are two terms we need to define.
For a hash value h = H (x), we say that x is the **preimage of h**. That is, x is a data block whose hash function, using the function H, is h. Because H is a many-to-one mapping, for any given hash value h, there will in general be multiple preimages.
A **collision** occurs if we have x ≠ y and H (x) = H (y). Because we are using hash functions for data integrity, collisions are clearly undesirable.

Requirements for a Cryptographic Hash Function H:

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

Table lists the generally accepted requirements for a cryptographic hash function.
The first three properties are requirements for the practical application of a hash function. The fourth property, **preimage (**for a hash value *h = H(x),* we say that x is the **preimage** of ***h***) **resistant**, is the one-way property: 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 fifth property, **second preimage resistant**, guarantees that it is impossible to find an alternative message with the same hash value as a given message. This prevents forgery when an encrypted hash code is used. A hash function that satisfies the first five properties in is referred to as a **weak hash function**. If the sixth property, collision resistant, is also satisfied, then it is referred to as a **strong hash function**. A strong hash function protects against an attack in which one party generates a message for another party to sign. The final requirement, **pseudorandomness**, has not traditionally been listed as a requirement of cryptographic hash functions, but is more or less implied.

Attacks on hash functions:

As with encryption algorithms, there are two categories of attacks on hash functions:

1. brute-force attacks and
2. Cryptanalysis

- A **brute-force attack** does not depend on the specific algorithm but depends only on bit length. In the case of a hash function, a brute-force attack depends only on the bit length of the hash value.
- A **cryptanalysis**, in contrast, is an attack based on weaknesses in a particular cryptographic algorithm.

Birthday Attacks:

- For a collision resistant attack, an adversary wishes to find two messages or data blocks that yield the same hash function
  - The effort required is explained by a mathematical result referred to as the ***birthday paradox***
- How the birthday attack works:?
  - The source (A) is prepared to sign a legitimate message *x* by appending the appropriate *m*-bit hash code and encrypting that hash code with A's private key
  - Opponent generates $2^{m/2}$ variations *x'* of *x*, all with essentially the same meaning, and stores the messages and their hash values
  - Opponent generates a fraudulent message *y* for which A's signature is desired
  - Two sets of messages are compared to find a pair with the same hash
  - The opponent offers the valid variation to A for signature which 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 and the opponent is assured of success even though the encryption key is not known

Hash Function Cryptanalysis:

As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. In recent years, have much effort, and some successes, in developing cryptanalytic attacks on hash functions. Must consider the overall structure of a typical secure hash function, referred to as an iterated hash function.

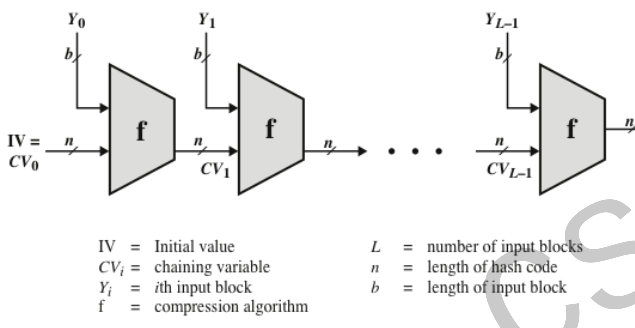


Figure 11.8 General Structure of Secure Hash Code

This was proposed by Merkle and is the structure of most hash functions in use today. 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. 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 part of the algorithm. The final value of the chaining variable is the hash value. Often, $b > n$; hence the term compression. The motivation for this iterative structure stems from the observation by Merkle and Damgard 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. 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. The attacks that have been mounted on hash functions are rather complex.

4.4Secure Hash Algorithm(SHA):

SHA was originally designed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993.Was revised in 1995 as SHA-1. Based on the hash function MD4 and its design closely models MD4.Produces 160-bit hash values. In 2002 NIST produced a revised version of the standard that defined three new versions of SHA with hash value lengths of 256, 384, and 512.Collectively known as SHA-2.

Table Comparison of SHA Parameters

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

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.

The processing consists of the following steps:

- **Step 1: Append padding bits**, the message is padded so that its length is congruent to 896 modulo 1024 [length = 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
- **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 represented as eight 64-bit registers (a, b, c, d, e, f, g, h).
- **Step 4: Process the message in 1024-bit** (128-word) blocks, which forms the heart of the algorithm. This contains 80 rounds.
- **Step 5: Output the final state value as the resulting hash**

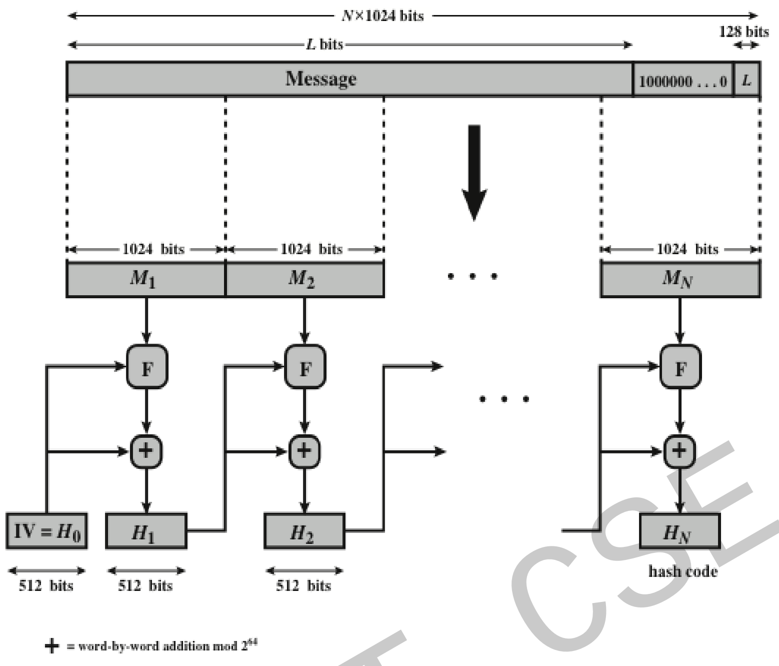


Figure 11.9 Message Digest Generation Using SHA-512

SHA-512 Compression Function:

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
    - updating a 512-bit buffer
    - using a 64-bit value derived from the current message block
    - and a round constant based on cube root of first 80 prime numbers

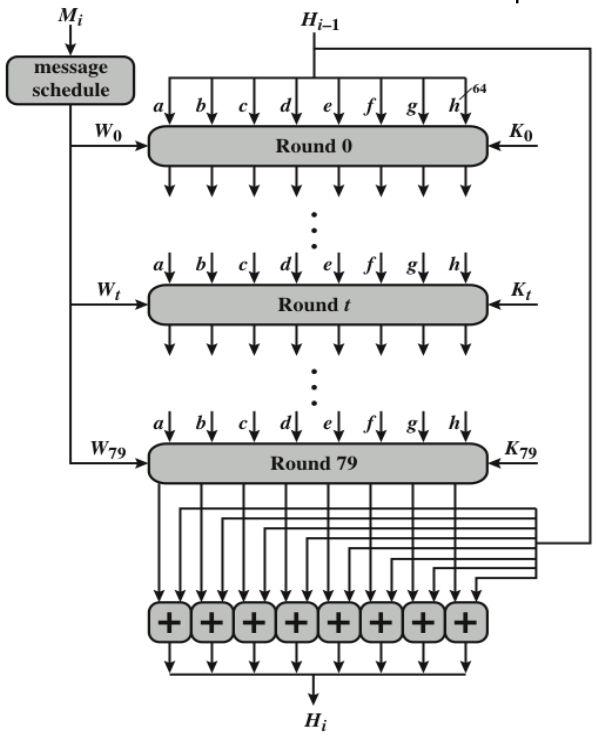


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

SHA-512 Round Function:

The structure of each of the 80 rounds is shown in Stallings Figure 11.10. Each 64-bit word is shuffled along one place, and in some cases manipulated using a series of simple logical functions (ANDs, NOTs, ORs, XORs, ROTates), in order to provide the avalanche & completeness properties of the hash function. The elements are:

Ch(e,f,g) = (e AND f) XOR (NOT e AND g)

Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c)

∑(a) = ROTR(a,28) XOR ROTR(a,34) XOR ROTR(a,39)

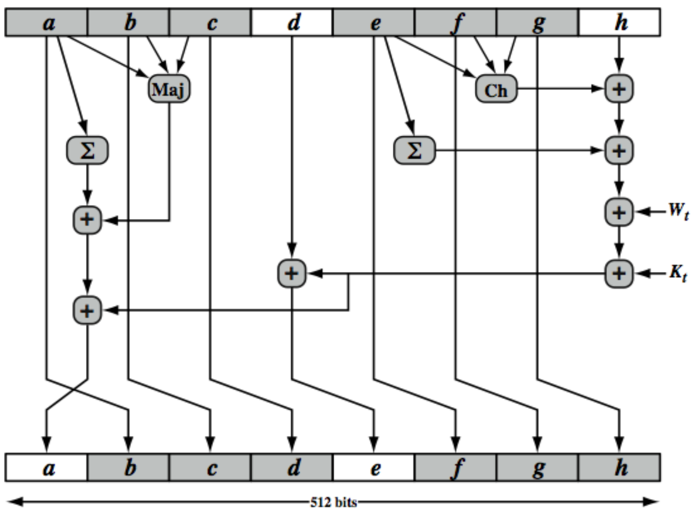
∑(e) = ROTR(e,14) XOR ROTR(e,18) XOR ROTR(e,41)

+ = addition modulo 2^64

Kt = a 64-bit additive constant

Wt = a 64-bit word derived from the current 512-bit input block.

Six of the eight words of the output of the round function involve simply permutation (*b, c, d, f, g, h*) by means of rotation. This is indicated by shading in Figure. Only two of the output words (*a, e)* are generated by substitution. Word e is a function of input variables *d, e, f, g, h,* as well as the round word W t and the constant Kt. Word a is a function of all of the input variables, as well as the round word W t and the constant Kt.



4.5. Message Authentication:

message authentication is concerned with: protecting the integrity of a message, validating identity of originator, non-repudiation of origin (dispute resolution)

Message Security Requirements

In the context of communications across a network, the following attacks can be identified.

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.

2. **Traffic analysis**: Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

3. **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.

4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.

5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

6. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.

7. **Source repudiation:** Denial of transmission of message by source.

8**. Destination repudiation:** Denial of receipt of message by destination.

4.6 Message Authentication Functions:

There are three functions used:

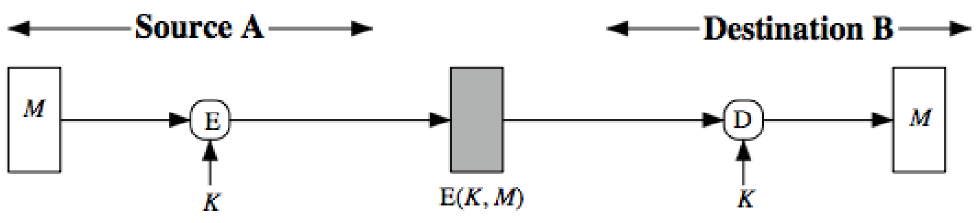
- ▶ **hash function**: A function that maps a message of any length into a fixed-length hash value which serves as the authenticator
- ▶ **message encryption**: The ciphertext of the entire message serves as its authenticator
- ▶ **message authentication code (MAC)**: A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message Encryption:

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

Symmetric Message Encryption:

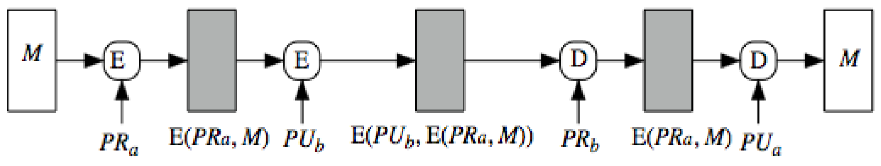
- encryption can also provide authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver now key used
  - know content cannot of been altered
  - if message has suitable structure, redundancy or a checksum to detect any changes



(a) Symmetric encryption: confidentiality and authentication

Public-Key Message Encryption:

- if public-key encryption is used:
  - encryption provides no confidence of sender
    - since anyone potentially knows public-key
  - however, if
    - sender **signs** message using their private-key
    - then encrypts with recipient's public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

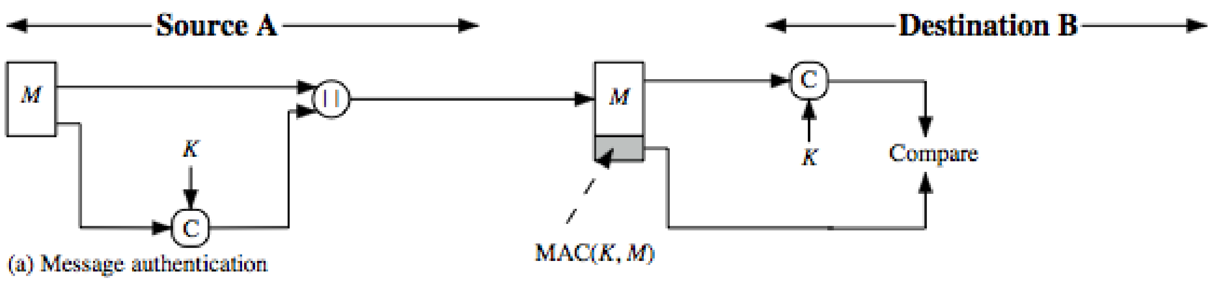


(d) Public-key encryption: confidentiality, authentication, and signature

Message Authentication Code (MAC):

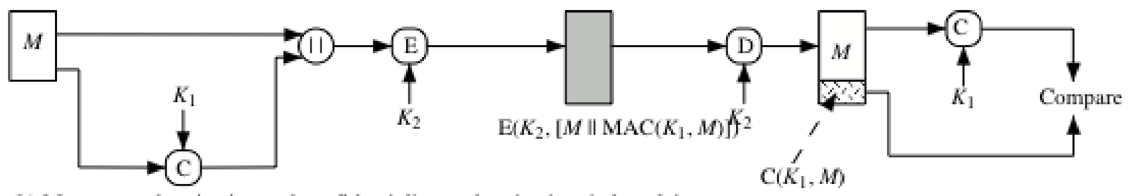
An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has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

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 Figure a. 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 the receiver is assured that the message has not been altered, is from the alleged sender, and if the message includes a sequence number then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number. A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must for decryption. In general, the MAC function is a many-to-one function. A MAC function is similar to encryption, except that the MAC algorithm need not be reversible, as it must for decryption.

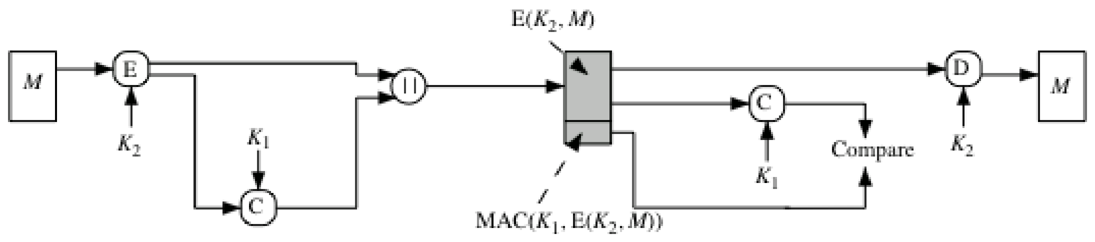


(a) Message authentication

The process depicted on the above provides authentication but not confidentiality, because the message as a whole is transmitted in the clear. Confidentiality can be provided by performing message encryption either after (see Figure b) or before (see Figure c) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure 12.4b is used. Can use MAC in circumstances where just authentication is needed (or needs to be kept), see text for examples (e.g. such as when the same message is broadcast to a number of destinations; when one side has a heavy load and cannot afford the time to decrypt all incoming messages; or do not need to keep messages secret, but must authenticate messages). Finally, note that the MAC does not provide a digital signature because both sender and receiver share the same key.



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



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

Figure: Basic Uses of Message Authentication Code(MAC)

<u>MAC Properties:</u>

A MAC (also known as a cryptographic checksum, fixed-length authenticator, or tag) is generated by a function C. The MAC 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 re-computing the MAC.
The MAC function is a many-to-one function, since potentially many arbitrarily long messages can be condensed to the same summary value, but don't want finding them to be easy.

<u>4.7.HMAC Design Objectives:</u>

RFC 2104 lists the following design objectives for HMAC:

- To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available.
- To allow for easy replace ability 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.

<u>HMAC:</u>

The idea of a keyed hash evolved into HMAC, designed to overcome some problems with the original proposals. It involves hashing padded versions of the key concatenated with the message, and then with another outer hash of the result prepended by another padded variant of the key. The hash function need only be used on 3 more blocks than when hashing just the original message (for the two keys + inner hash). HMAC can use any desired hash function, and has been shown to have the same security as the underlying hash function. Can choose the hash function to use based on speed/security concerns.

<u>HMAC Overview:</u>

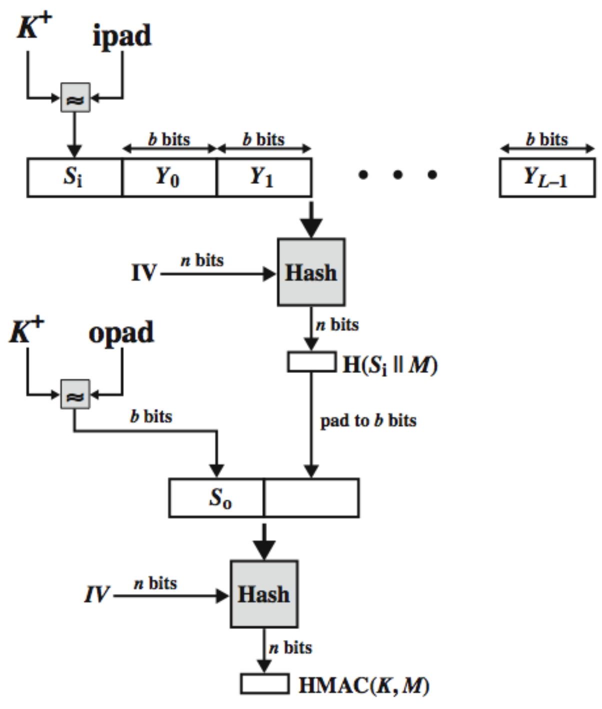


Figure illustrates the overall operation of HMAC:

$HMAC_K = Hash[(K^+ \text{ XOR opad}) \,||\, Hash[(K^+ \text{ XOR ipad}) \,||\, M)]$

where:

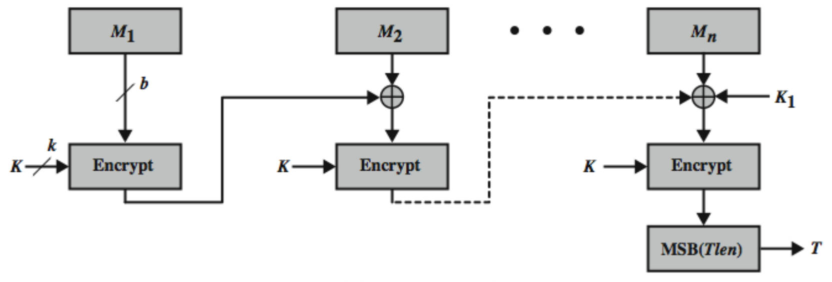
K^+ is K padded with zeros on the left so that the result is b bits in length

ipad is a pad value of 36 hex repeated to fill block
opad is a pad value of 5C hex repeated to fill block
M is the message input to HMAC (including the padding specified in the embedded hash function)
Note that the XOR with ipad results in flipping one-half of the bits of *K*. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, pseudorandomly 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 *Si, So*, and the block produced from the inner hash). A more efficient implementation is possible by precomputing the internal hash function on (K^+ XOR opad) and (K^+ XOR ipad) and inserting the results into the hash processing at start & end. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This is especially worthwhile if most of the messages for which a MAC is computed are short.

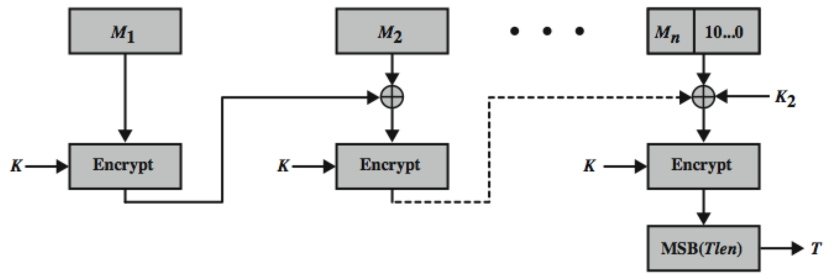
4.8. CMAC:

The Data Authentication Algorithm cipher-based MAC has been widely adopted in government and industry. Has been shown to be secure, with the following restriction. Only messages of one fixed length of *mn* bits are processed, where *n* is the cipher block size and *m* is a fixed positive integer. This limitation can be overcome using multiple keys, which can be derived from a single key. This refinement has been adopted by NIST as the cipher-based message authentication code (CMAC) mode of operation, for use with AES and triple DES. It is specified in NIST Special Publication 800-38B.

CMAC Overview:



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Figure shows the structure of CMAC. It uses the blocksize of the underlying cipher (ie 128-bits for AES or 64-bits for triple-DES). The message is divided into n blocks M1..Mn, padded if necessary. The algorithm makes use of a k-bit encryption key K and an n-bit constant K1 or K2 (depending on whether the message was padded or not). For AES, the key size k is 128,192, or 256 bits; for triple DES, the key size is 112 or 168 bits. The two constants K1 & K2 are derived from the original key K using encryption of 0 and multiplication in GF(2^n).

4.9DIGITAL SIGNATURES

The most important development from the work on public-key cryptography is the **digital signature**. Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other either fraudulently creating, or denying creation, of a message. A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way. It must have the following properties:
• 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
Thus, the digital signature function includes the authentication function.

DIGITAL SIGNATURE MODEL

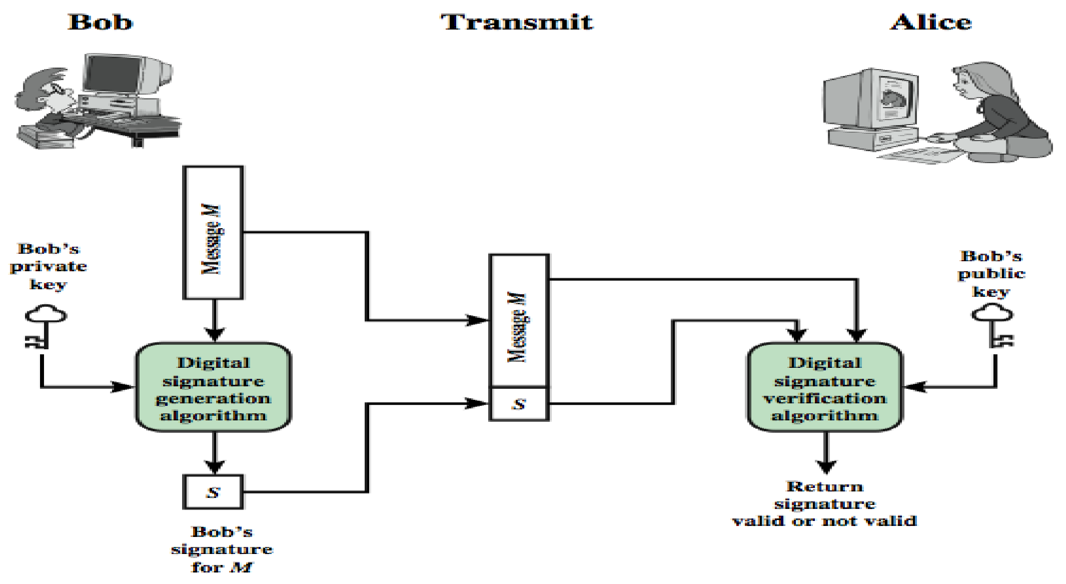

Bob
Transmit
Alice
Bob's private key
Message M
Digital signature generation algorithm
S
Bob's signature for M
Message M
S
Bob's public key
Digital signature verification algorithm
Return signature valid or not valid

Figure is a generic model of the process of making and using digital signatures. Bob can sign a message using a digital signature generation algorithm. The inputs to the algorithm are the message and Bob's private key. Any other user, say Alice, can verify the signature using a verification algorithm, whose inputs are the message, the signature, and Bob's public key.

Attacks and Forgeries:

lists the following types of attacks, in order of increasing severity. Here A denotes the user whose signature is being attacked and C denotes the attacker.

- **Key-only attack**: C only knows A's public key.
- **Known message attack**: C is given access to a set of messages and signatures.
- **Generic chosen message attack**: C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. The attack is generic because it does not depend on A's public key; the same attack is used against everyone.
- **Directed chosen message attack**: Similar to the generic attack, except that the list of messages is chosen after C knows A's public key but before signatures are seen.
- **Adaptive chosen message attack**: C is allowed to use A as an "oracle." This means the A may request signatures of messages that depend on previously obtained message-signature pairs.

Then defines success as breaking a signature scheme as an outcome in which C can do any of the following with a non-negligible probability:

- **Total break**: C determines A's private key. • Universal forgery: C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
- **Selective forgery**: C forges a signature for a particular message chosen by C.
- **Existential forgery**: C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

Digital Signature Requirements:

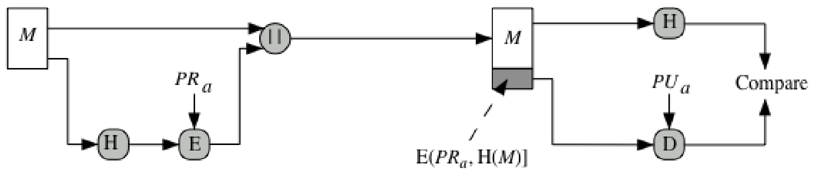
- ➢ must depend on the message signed
- ➢ must use information unique to sender
  - to prevent both forgery and denial
- ➢ must be relatively easy to produce
- ➢ must be relatively easy to recognize & verify
- ➢ be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for given message
- ➢ be practical save digital signature in storage

Direct Digital Signatures:

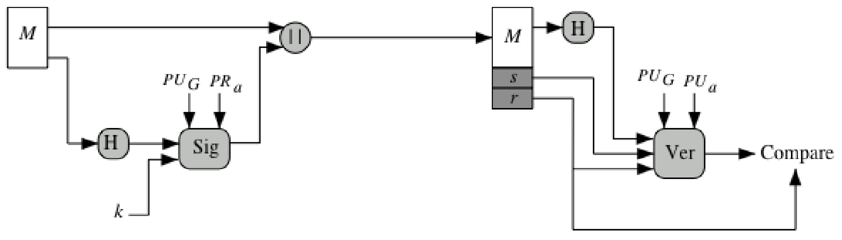
The term *direct digital* signature refers to a digital signature scheme that involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source. Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender's private key, or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by further encrypting the entire message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message and its signature. But these approaches are dependent on the security of the sender's private-key. Will have problems if it is lost/stolen and signatures forged. The universally accepted technique for dealing with these threats is the use of a digital certificate and certificate authorities. Also need time-stamps and timely key revocation.

4.10.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 was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2, subsequently updated to FIPS 186-3 in 2009. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography.



(a) RSA Approach



(b) DSA Approach

Figure 13.3 Two Approaches to Digital Signatures

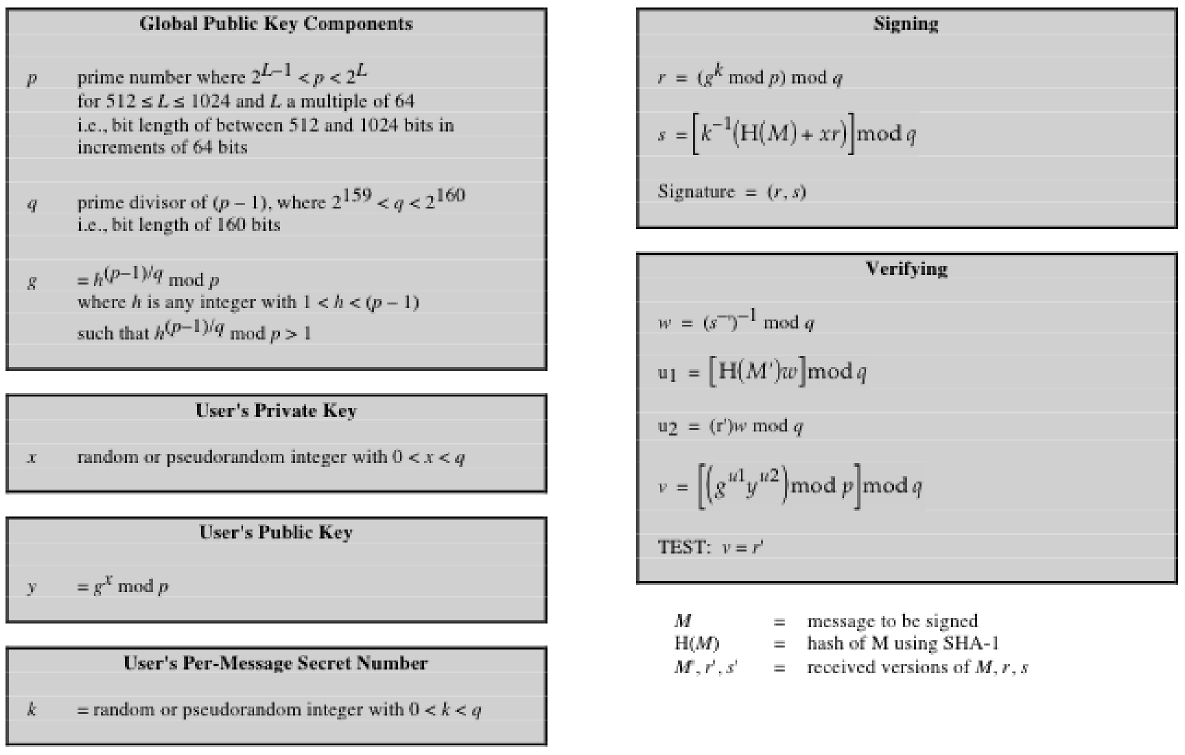
The DSA uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

Above Figure contrasts the DSA approach for generating digital signatures to that used with RSA. 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.

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 group of communicating principals. We can consider this set to constitute a 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.

The Digital Signature Algorithm:



Global Public Key Components

p	prime number where $2^{L-1} < p < 2^L$ for $512 \le L \le 1024$ 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 $2^{159} < q < 2^{160}$ i.e., bit length of 160 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

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

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

Signature $= (r, s)$

Verifying

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

$u_1 = \left[\mathrm{H}(M')w\right] \bmod q$

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

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

TEST: $v = r'$

M	=	message to be signed
$\mathrm{H}(M)$	=	hash of M using SHA-1
M', r', s'	=	received versions of M, r, s

Figure 13.4 The Digital Signature Algorithm (DSS)

The DSA is based on the difficulty of computing discrete logarithms and is based on schemes originally presented by ElGamal and Schnorr. The DSA signature scheme has advantages, being both smaller (320 vs 1024bit) and faster (much of the computation is done modulo a 160 bit number), over RSA. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

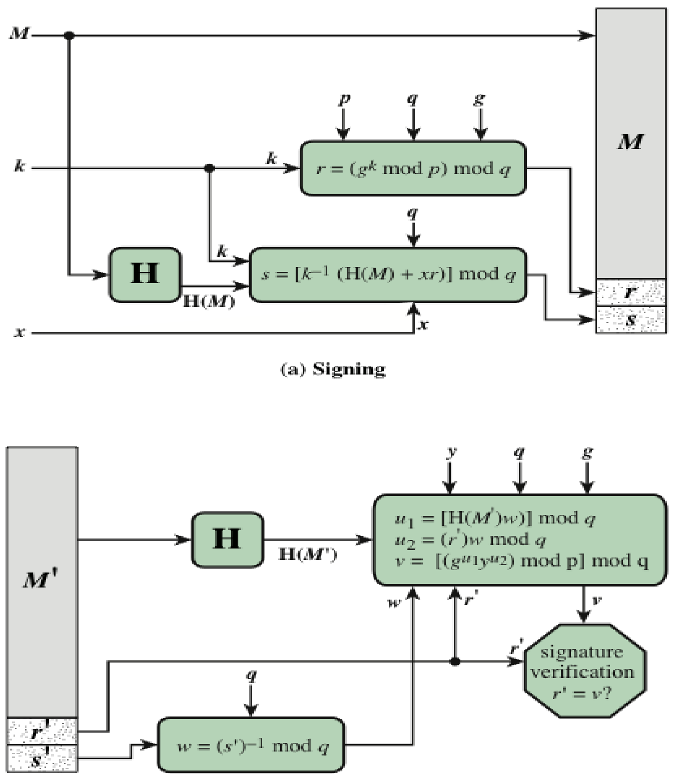
DSA typically uses a common set of global parameters (p,q,g) for a community of clients, as shown. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p – 1). Finally, g is chosen to be of the form $h^{(p-1)/q}$ mod p where h is an integer between 1 and (p – 1) with the restriction that g must be greater than 1. Thus, the global public key components of DSA have the same for as in the Schnorr signature scheme.

Then each DSA uses chooses a random private key x, and computes their public key as shown. The calculation of the public key y given x is relatively straightforward. However, given the public key y, it is computationally infeasible to determine x, which is the discrete logarithm of y to base g, mod p.

To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p,q,g), the user's private key (x), the hash code of the message H(M), and an additional integer k that should be generated randomly or pseudo-randomly and be unique for each signing. This is similar to ElGamal signatures, with the use of a per message temporary signature key k, but doing calculations first mod p, then mod q to reduce the size of the result. The signature (r,s) is then sent with the message to the recipient. Note that computing r only involves calculation mod p and does not depend on message, hence can be done in advance. Similarly with randomly choosing k's and computing their inverses.

At the receiving end, verification is performed using the formulas shown. The receiver generates a quantity v that is a function of the public key components, the sender's public key, and the hash of the incoming message. If this quantity matches the r component of the signature, then the signature is validated. Note that the difficulty of computing discrete logs is why it is infeasible for an opponent to recover k from r, or x from s. Note also that nearly all the calculations are mod q, and hence are much faster save for the last step.

The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious that such a scheme would work.



(a) Signing

(b) Verifying

Figure 13.5 DSA Signing and Verifying

4.11.Key management:

One of the major roles of public-key encryption is to address the problem of key distribution. There are actually two distinct aspects to the use of public-key encryption in this regard:

- The distribution of public keys
- The use of public-key encryption to distribute secret keys

Public-Key Certificates

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large. Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

The solution to this problem is the public-key certificate. In essence, a certificate consists of a public key plus a User ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA) that is trusted by the user community, such as a government agency or a financial institution. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. Figure 3.12 illustrates the process.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security

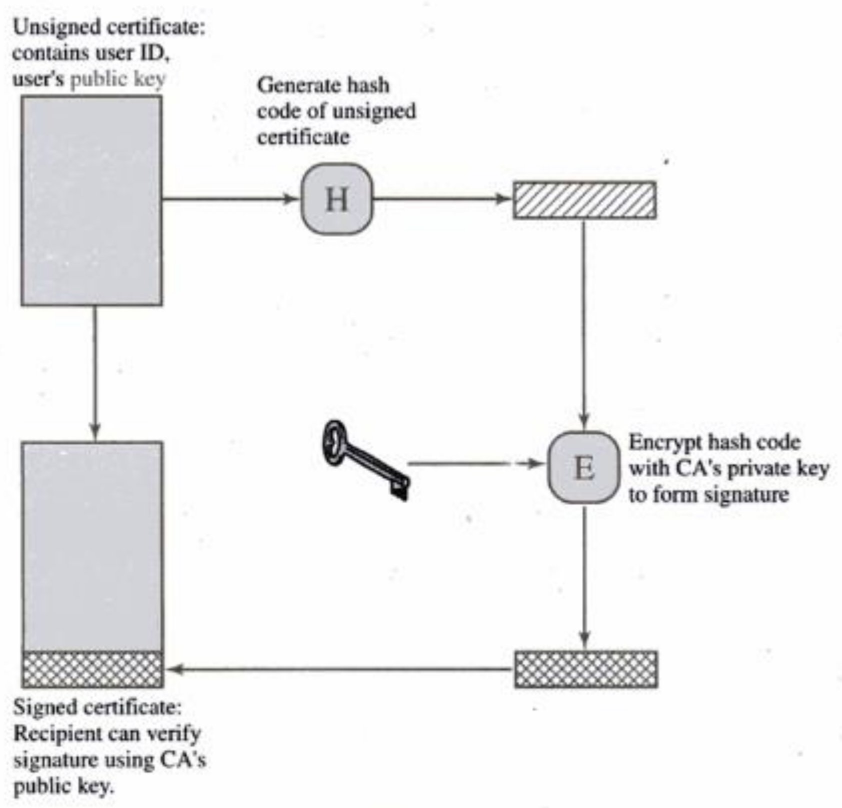

Unsigned certificate: contains user ID, user's public key
Generate hash code of unsigned certificate
H
E
Encrypt hash code with CA's private key to form signature
Signed certificate: Recipient can verify signature using CA's public key.

Figure 3.12 Public-Key Certificate Use

applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME, all of which are discussed in Part Two. X.509 is examined in detail in Chapter 4.

Public-Key Distribution of Secret Keys

With conventional encryption, a fundamental requirement for two parties to communicate securely is that they share a secret key. Suppose Bob wants to create a messaging application that will enable him to exchange e-mail securely with anyone who has access to the Internet or to some other network that the two of them share. Suppose Bob wants to do this using conventional encryption. With conventional encryption, Bob and his correspondent, say, Alice, must come up with a way to share a unique secret key that no one else knows. How are they going to do that? If Alice is in the next room from Bob, Bob could generate a key and write it down on a piece of paper or store it on a diskette and hand it to Alice. But if Alice is on the other side of the continent or the world, what can Bob do? He could encrypt this key using conventional encryption and e-mail it to Alice, but this means that Bob and Alice must share a secret key to encrypt this new secret key. Furthermore, Bob and everyone else who uses this new e-mail package faces the same problem with every potential correspondent: Each pair of correspondents must share a unique secret key.

One approach is the use of Diffie-Hellman key exchange. This approach is indeed widely used. However, it suffers the drawback that, in its simplest form, Diffie-Hellman provides no authentication of the two communicating partners.

A powerful alternative is the use of public-key certificates. When Bob wishes to communicate with Alice, Bob can do the following:

1. Prepare a message
2. Encrypt that message using conventional encryption with a one-time conventional session key.
3. Encrypt the session key using public-key encryption with Alice's public key.
4. Attach the encrypted session key to the message and send it to Alice.

Only Alice is capable of decrypting the session key and therefore of recovering the original message. If Bob obtained Alice's public key by means of Alice's public-key certificate, then Bob is assured that it is a valid key.