Module - IV

Cryptographic Hash Function:

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

$h = H(M)$

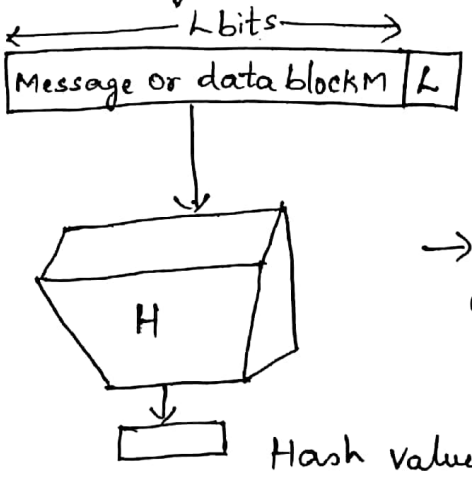
A good hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed & apparently random.

The principal object of a hash function is data integrity i.e. a change to any bit or bits in M results, with high probability, in a change to the hash code.

A Cryptographic hash function (CHF) is an algorithm for which it is computationally infeasible to find either

a) data object that maps to a pre-specified hash result.
   ↳ one-way property
b) two objects that map to the same hash result.
   ↳ Collision-free property

Hash functions are often used to determine whether the data has changed or not.



M - variable length

→ Block diagram of Cryptographic Hash function
$h = H(M)$

The input is padded out to an integer multiple of some fixed length & the padding includes the value of the length of the original message in bits.

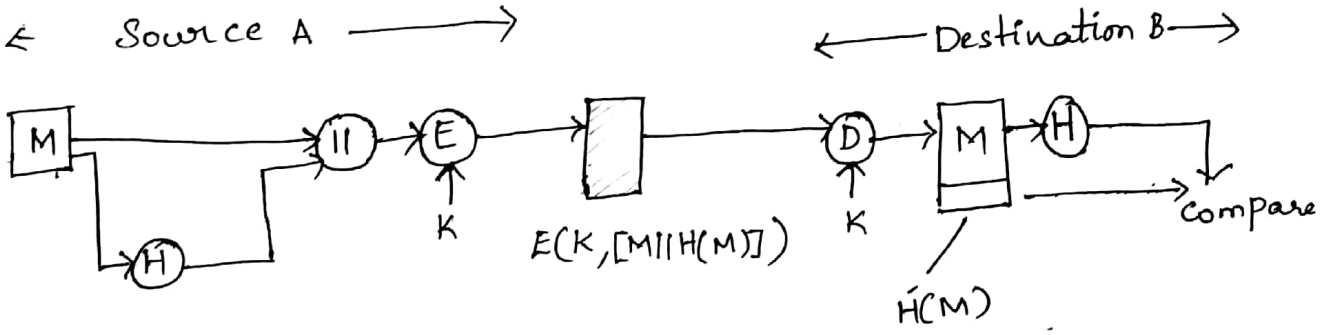
→ The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

Applications Of Cryptographic Hash Functions

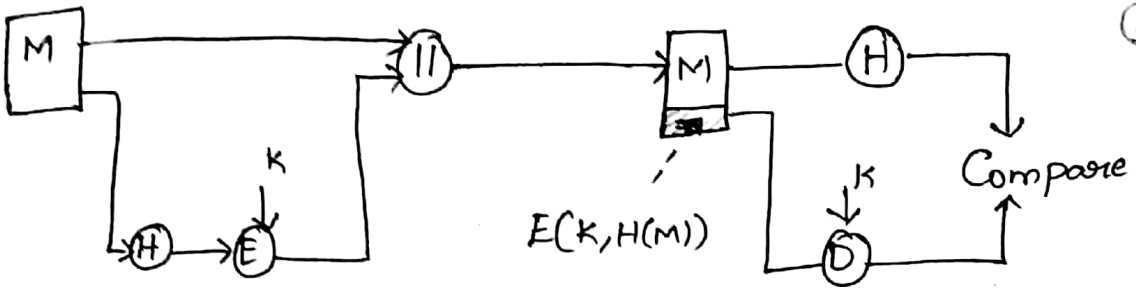
→ Message authentication is a mechanism or service used to verify the integrity of a message.

Message authentication (MA) anures that data received are exactly as sent (i.e. contain no modification, insertion, deletion or replay)

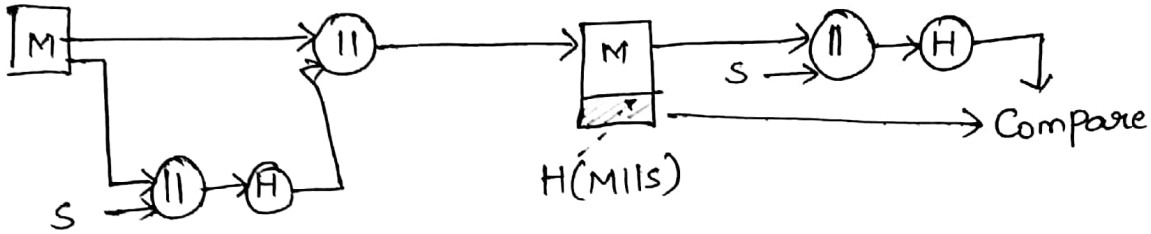
When a hash function is used to provide MA, the hash function value is often referred to as a message digest.



The message plus concatenated hash value is encrypted using symmetric encryption. Because only A & B share the secret key, the message must have come from A & 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, confidentality is also provided.



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



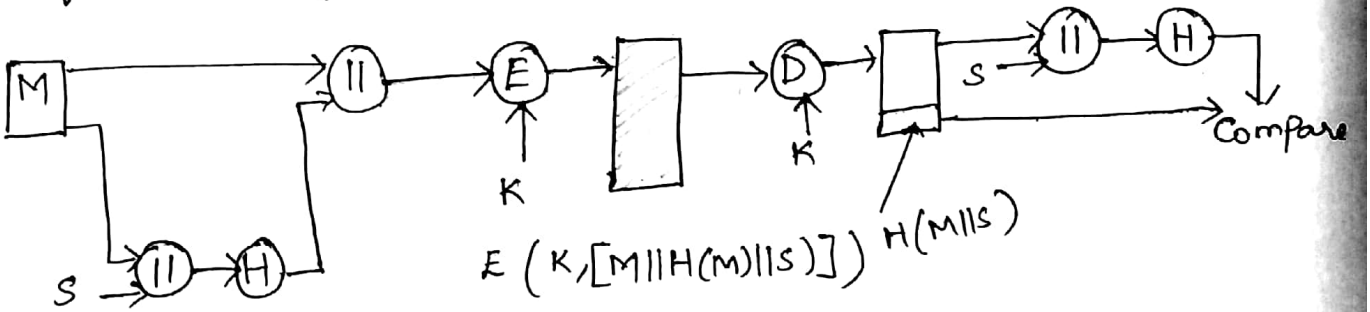
It is possible to use a hash function but no encryption for message authentication.

This technique assumes 2 communicating parties share a common secret value S.

A computes the hash value over the concatenation of M & S and appends the resulting hash value to M

B has S, so it can recompute the hash value to verify.

An opponent cannot alter the message & cannot generate a false message as the secret value is not sent.



Confidentality can be added to the prior method by encrypting the entire message plus the hash code.

MA is achieved using a message authentication code (MAC). also known as a keyed hash function. MACs are used b/w 2 parties that share a secret key to authenticate information exchanged b/w Parties

MAC function takes as input a secret key & a data blocks & produces a hash value referred to as the MAC. This can then be transmitted with or stored with the protected message.

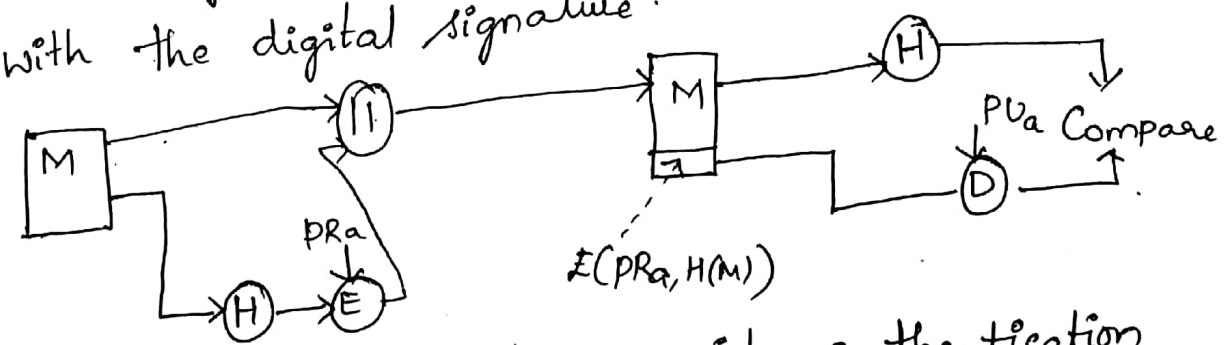
To check the integrity of the message, the MAC function can be applied to the message & the result compared with the stored MAC value.

Combination of hashing & encryption results in function MAC.

$E(K, H(M))$ is a function of variable - length message M & a secret key K, & it produces a fixed-size o/p i.e. secure against an opponent who does not know the secret key.

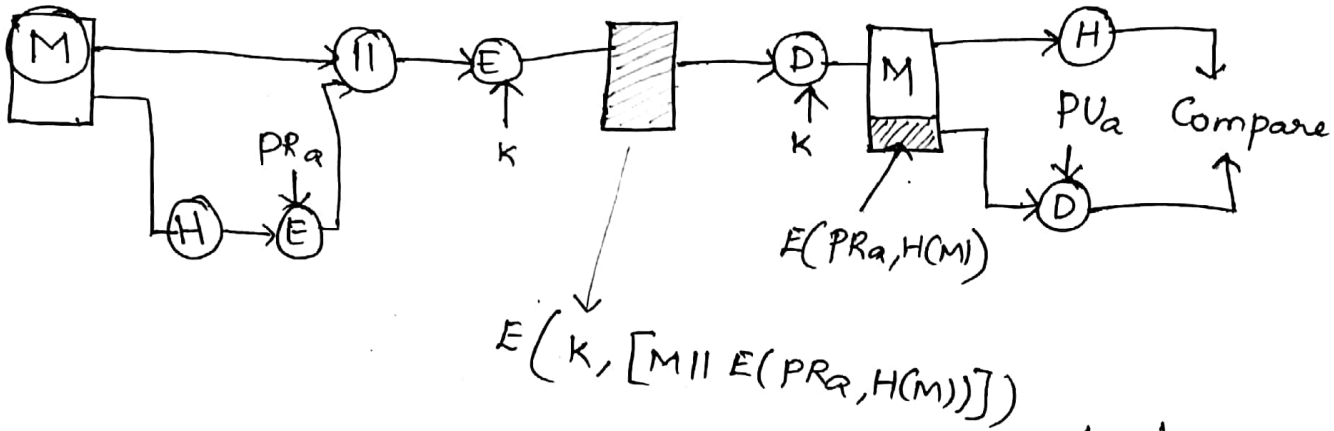
Digital Signatures:

The hash value of a msg is encrypted with a user's private key. Person aware of user's public key can verify the integrity of the message that is anociated with the digital signature.



↳ Provides authentication

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



Here the message + private key encrypted hash code is encrypted using symmetric secret key. This ensures Confidentality & digital signature.

Other Applications

→ Hash functions are commonly used to create a one-way password file. The hash of the password is stored by an operating system rather than the password itself.

→ A hacker who gain access to the password file can't retrieve the actual password. When a user enters a password the hash of that password is compared to the stored hash value for verification.

→ Hash functions can be used for intrusion detection & virus detection. Store H(F) for each file on a system & secure the hash values. We can determine if a file has been modified by recomputing H(F).

→ CHF can be used to produce PRF or PRNG.

→ Hash based PR

Two simple Hash Functions:

One of the simplest hash function is the bit-by-bit XOR of every block which is expressed as

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

Where C_i = ith bit of the hash code $1 \le i \le n$

m = number of n-bit blocks in the input

b_{ij} = i^{th} bit in j^{th} block

$\oplus$ = XOR operation

This operation produces a simple parity for each bit position and is known as a longitudinal redundancy check.

A simple way to improve matters is to perform a 1-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows
   a. Rotate the current hash value to the left by 1 bit.
   b. XOR the block into the hash value.

This has randomizing effect on the input more completely & overcoming any regularities that appear in the input.

A simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted.

A simple function could be useful when the message together with 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 & then an encryption of the entire message that used the cipher block chaining (CBC) mode. (7)

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

$$h_* = X_{N+1} = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$

Encrypt the entire msg plus hash code using CBC mode to produce the encrypted msg $Y_1, Y_2, \cdots Y_{N+1}$

The ciphertext of the message can be manipulated in such a way that it is not detectable by the hash code.

CBC definition

$$X_1 = IV \oplus D(K, Y_1)$$
$$X_i = Y_{i-1} \oplus D(K, Y_i)$$
$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

But X_{N+1} is the hash code

$$X_{N+1} = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$
$$= [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \cdots$$
$$\oplus [Y_{N-1} \oplus D(K, Y_N)]$$

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.

Message Authentication Codes

A message authentication code (MAC) is an algorithm that requires the use of a secret key. A MAC takes a variable-length msg & secret key as input & produces an authentication code. A recipient in possession of the secret key can generate an authentication code to verify the integrity of the message.

1. Disclosure: Release of msg 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 applications, the frequency & duration of connections could be determined.

These 2 attacks can be dealt with the measures of data confidentality.

3. Masquerade: Insertion of msgs into n/w from a fraudulent source. This includes creation of msgs by an opponent which appear as coming from authorized entity. Fraudulent ack of msgs receipt or non-recipient by someone other than msg recipient.
4. Content modification: Insertion, deletion, transposition & modification to the contents of the msg.
5. Sequence modification: Any modification to a seq. of msgs b/w parties, including insertion, deletion & reordering.
6. Timing modification: Delay or replay of msgs. In a connection-oriented application, an entire session or seq. of msgs could be a replay of some previous valid session, or individual messages in the seq. could be delayed or replayed.

In a connectionless application, an individual msg could be delayed or replayed.

Measures to deal with these items are regarded as msg authentication.

7. Source repudiation : Denial of transmission of msg by source.
8. Destination repudiation : Denial of receipt of msg by destination.

7 can be dealt with digital signatures
8 needs a protocol + " " to deal this attack.

Message Authentication Functions:

MA or digital signature mechanism has 2 levels of functionality.

At lower level, a fn produces authenticator ; a value to authenticate a msg.
This fn is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a msg.

An authenticator can be produced with 3 types of fns.

- Hash function : Fn which maps Msg M of any variable length into a fixed-length hash value.
- Msg encryption : Ciphertext of the entire msg
- MAC : fn of the msg + a secret key which produces a fixed-length value

Message Encryption: - This act as measure of authentication.

Symmetric Encryption: // Provides Confidentality & Authentication

source A — Destination B (C) (A)

Msg M → M → C → M

Confidentality is provided.

A & B possess K. When a msg is sent to B, receiver assumes the msg sender is A since they share the symmetric key.

So, it provides C & A.

Given a decryption fn D & a secret key K, the destination will accept any i/p x & produce o/p $Y = D(K, X)$

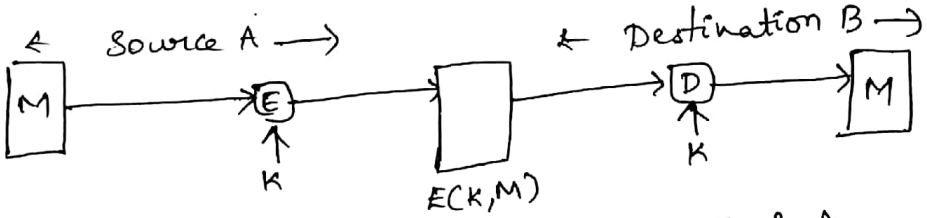
If X is Ciphertext of a legitimate msg M,

Y is some plaintext msg M.

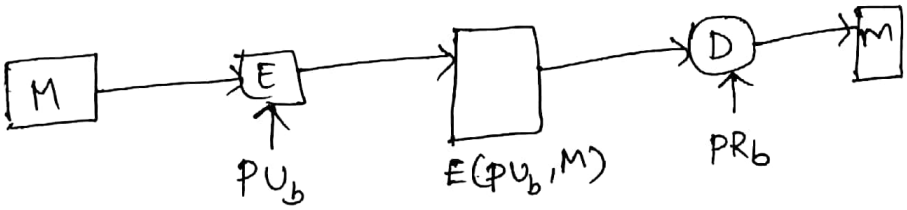
otherwise, Y will likely be a meaningless sequence of bits.

If M is any arbitary bit pattern

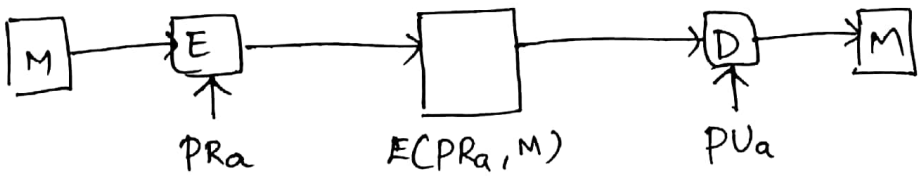
→ If M can be any bit pattern, then regardless of value of x, the value $Y = D(K, X)$ is some bit pattern & therefore must be accepted as authentic plaintext.



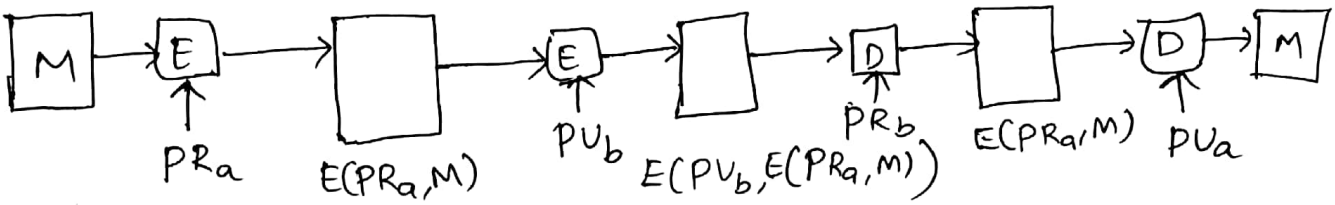
a) Symmetric Encryption: C & A.



b) public-key encryption: C



c) Public-key encryption: A & Signature



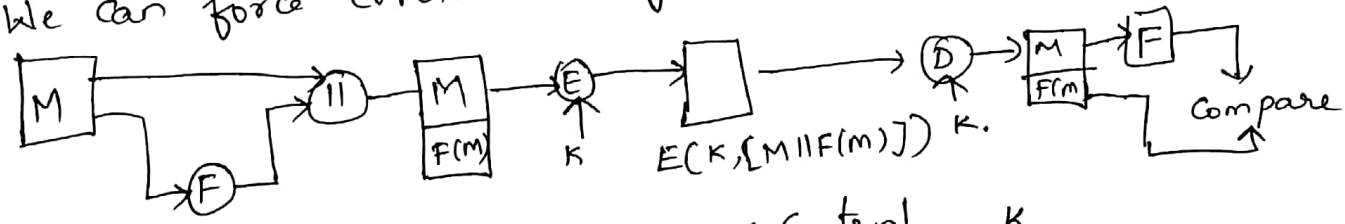
d) public-key encryption: C, A & S

We require that only a small subset of all possible bit patterns be considered legitimate plaintext.

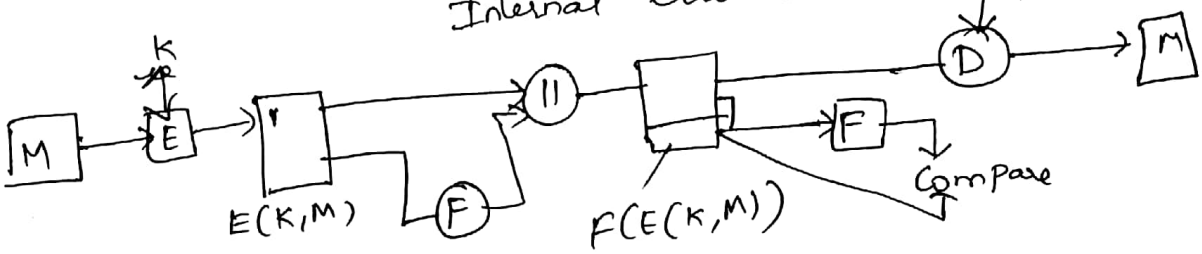
It is difficult to determine automatically if all the ciphertext corresponds to intelligible plaintext. An opponent could achieve a certain level of disruption.

We can force the plaintext to have some structure that is easily recognized but that cannot be replicated w/o recourse to the encryption fn.

We can force error-detecting code known as Frame check sequence (FCS)



Internal error control



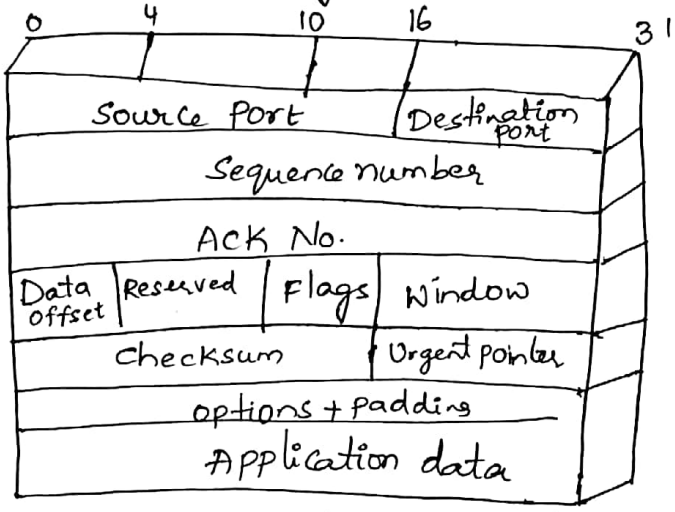
b) External error control

→ With internal error control, authentication is provided because an opponent would have difficulty generating ciphertext that when decrypted would have valid error control bits

→ With external error control, an opponent can construct messages with valid error control bits. Even though the opponent does not possess any knowledge of what the decrypted plaintext will be but opponent may create confusion & disrupt operations.

→ Any sort of structuring like error-control code added to the transmitted msg strengthen the authentication capability. Such structure is exhibited by architecture which contain layered protocol.

Ex: Msg transmitted using the TCP/IP protocol architecture



Format of a TCP segment

Each pair of hosts used the unique secret key, so that all exchange b/w a pair of hosts use the same key regardless of applications. We encrypt all of the datagram except Ip header.

→ If an opponent substituted some arbitary bit pattern for the encrypted TCP segment, the resulting plaintext would not include a meaningful header.

The header not includes checksum, other useful info such as SeqNo.

Public-Key Encryption: // provides confidentality but not authentication

→ To provide authentication, A uses its Private Key to encrypt the msg & B uses A's public key to decrypt.

→ There must be some internal structure to the plaintext so that the receiver can distinguish b/w well-formed plaintext and random bits.

→ To provide both Confidentiality & Authentication, user A can encrypt M using PR_a which provides the digital structure & then using PU_b, which provides Confidentality.

→ Public-Key algorithm which is complex, must be exercised 4 times rather than 2 in each Communication. //Disadvantage

MAC

Authentication technique which uses a secret key to generate a small-fixed-size block of data known as Cryptographic Checksum or MAC which is appended to msg.

$$MAC = MAC(K, M)$$

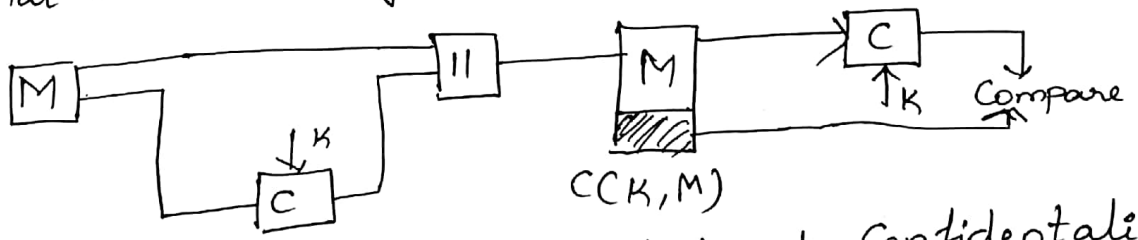
Where M = i/p Msg
C = MAC fn
K = shared secret key
MAC = msg authentication code

Msg + MAC are sent to intended recipient. The recipient Performs the same calculation on the received msg, using the same secret key, to generate a new MAC. Received MAC is compared to calculated MAC.

If the received MAC matches the calculated MAC, then

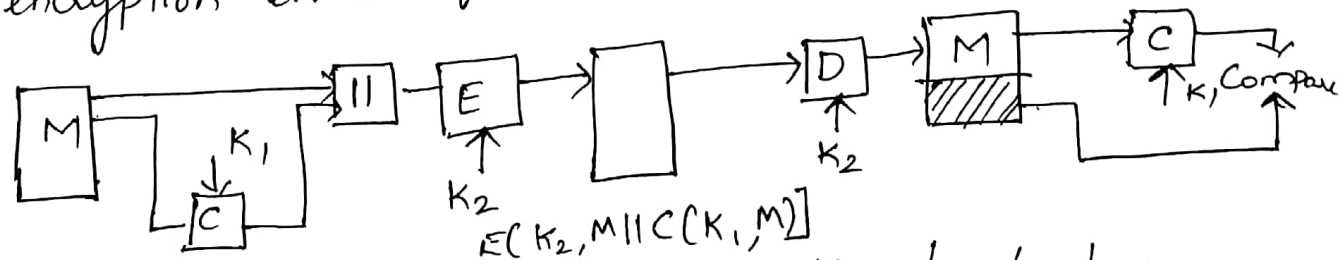
1. Receiver is assured that the msg has not been altered. If attacker alters the msg but does not alter the MAC then calculated MAC will differ since opponent doesn't possess secret key.
2. Receiver is assured that the msg is from the alleged sender.
3. If the msg includes a seq. No. then the receiver can be assured of proper sequence because opponent can alter the seq. No. (HDLC, X.25, TCP)

A MAC fn is similar to encryption. One diff is that the MAC algorithm need not be reversible.

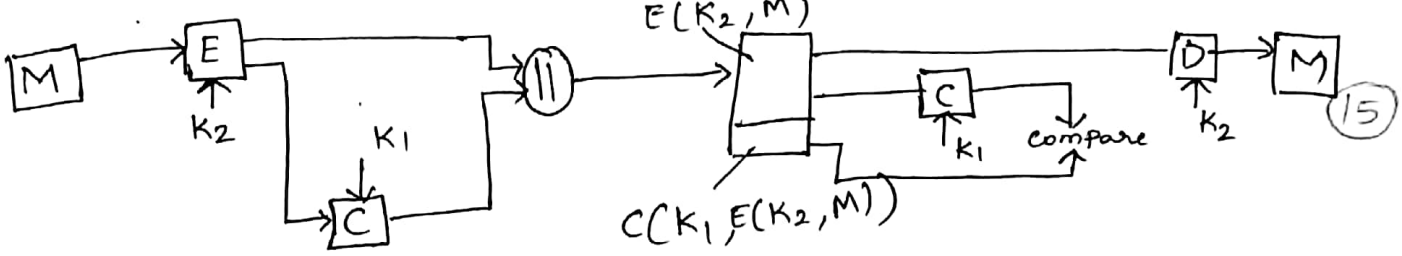


Provides authentication but not confidentality.

Confidentality can be provided by performing msg encryption either after or before MAC algorithm.



A & C, A tied to plaintext



Uses of MA

1. There are no. of app's in which the same msg is broadcast to a no. of destinations.
   e.g. Notification of n/w unavailability or alarm signal in a military control centre.
2. A Scenario in which there is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming msgs. Authentication is carried out on a selective basis, msgs being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer " can be executed w/o having to decrypt it every time, which would be wasteful of Processor resources. If MAC is attached to the program it could be checked whenever integrity needs to be checked.
4. SNMPv3 (Simple N/w Mgmt Protocol Version 3 (SNMPV3), which separates the fns of C & A. It is imp. to authenticate SNMP msgs, particularly if the msg contains a command to change parameters at the managed system.
5. Separation of A & C fns affords architectural flexibility.
6. User can prolong the period of protection beyond the time of recipient and allow processing of message content.

MAC does not Provide a digital signature.

Requirements for MAC:

A MAC, also known as cryptographic checksum, is generated by a fn C of the form

$$T = MAC(K, M)$$

where M - variable length msg

K - Secret Key

MAC(K,M) - fixed length authenticator called Tag of size n

Tag is appended to msg at the source.

Receiver is authenticates that msg by recomputing the Tag.

In encryption

A brute force attack may require $2^{(K-1)}$ attempts for a K-bit key.

But, for MAC fn is a many-to-1 function.

→ Using Brute force method, the key with which MAC is calculated may be discovered.

→ The opponent has access to plaintext & anociated MAC's

if $K > n$ Keysize > MAC size

Given a known M_1 & T_1 with $T_1 = MAC(K, M_1)$

Cryptanalyst can perform $T_i = MAC(K_i, M_1)$ for $\forall K_i$

Total of 2^K tags will be produced

but there are only $2^n < 2^K$ different tag values.

No. of Keys will produce the correct tag & opponent has no way of knowing which is the correct key.

Total of $2^K/2^n = 2^{(K-n)}$ Keys will produce a match.

opponent must iterate the attacks.

Round 1

Given : $M_1, T_1 = MAC(K, M_1)$

Compute $T_i = MAC(K_i, M_1)$ for all 2^K keys

No. of matches $\approx 2^{(K-n)}$

Round 2

Given : $M_2, T_2 = MAC(K, M_2)$

Compute $T_i = MAC(K_i, M_2)$ for $2^{(K-n)}$ keys from Round 1

No. of matches $\approx 2^{(K-2\times n)}$

α rounds will be needed if $K = \alpha \times n$.

if K = 80bit key & T is 32 bit

Round (1) $\approx 2^{(80-32)} \approx 2^{48}$

(2) $\approx 2^{(80-2\times 32)} \approx 2^{16}$

Round (3) should produce only a single key which is used by the sender.

Case-2

If the Key length $\leq$ Tag length

then first round will produce a single match.

So, Brute force attacks is may happen to discover the authentication key.

If $M = (X_1 \| X_2 \| \ldots \| X_m)$ Concatenation of 64 bit blocks X_i

$\Delta(M) = X_1 \oplus X_2 \oplus \cdots \oplus X_m$

$MAC(K, M) = E(K, \Delta(M))$

$\oplus$ – XOR

Encryption alg is DES in Electronic Codebook mode.

Key length is 56 bits & Tag = 64 bits

If an opponent observes $\{M \| MAC(K,M)\}$

brute-force attempt $- 2^{56}$ encryptions.

But by just replacing

$X_1 \ldots X_{m-1}$ with any $Y_1 \ldots Y_{m-1}$ and replacing X_m with Y_m where Y_m is calculated as

$$Y_m = Y_1 \oplus Y_2 \oplus \ldots \oplus Y_{m-1} + \Delta f(M)$$

Now opponent can now concatenate the new msg with $Y_1 \ldots Y_m$ with the tag to form a msg which will be authentic by the receiver.

To overcome the attacks which are mounted on MAC fn, MAC fn should satisfy these requirements

1. Opponent has M & MAC(K,M), it should be infeasible to construct a msg M' such that

$$MAC(K, M') = MAC(K, M)$$

2. MAC(K,M) should be uniformly distributed.

$$Pr[MAC(K,M)] = Pr[MAC(K,M')] = 2^{-n}$$

n - Tag bits M, M' - different msgs

3. $M' = f(M)$ // Transformation of M i.e. inverting 1 or more bits

$$Pr[MAC(K,M)] = Pr[MAC(K,M')] = 2^{-n}$$

Requirement 2

Opponent can try various msgs until finding a msg that fits a given tag - needs 2^{n-1} attempts

Security of MACs :

1. Brute force attacks (BFA):

BFA on MAC is more difficult than BFA on hash fn ∵ since it requires known message - tag pairs.

Desired security property of MAC algorithm:

Computation resistance: Given 1 or more [M, MAC pairs $_{(K, x_i)}$] then it is difficult to compute any [M, MAC pair $_{K, x}$]

for any new i/p $x \neq x_i$

2 lines of attack possible — Key space
\ MAC value

Attack takes a level of effort — 2^K (Known MAC Key $_{\text{possible key values}}$
\ 2^n (Known tag value

So, $\min(2^K, 2^n)$ is effort for brute-force attack.

$\therefore \min(K, n) \geq N$ N is in the range of 128 bits

Cryptanalysis:

Way to measure the resistance of MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack.

HMAC

MAC derived from a Cryptographic Hash Fn. (CHF)

1. CHF such as MD5 & SHA execute faster in s/w than symmetric block ciphers such as DES
2. Library code for CHF is widely available

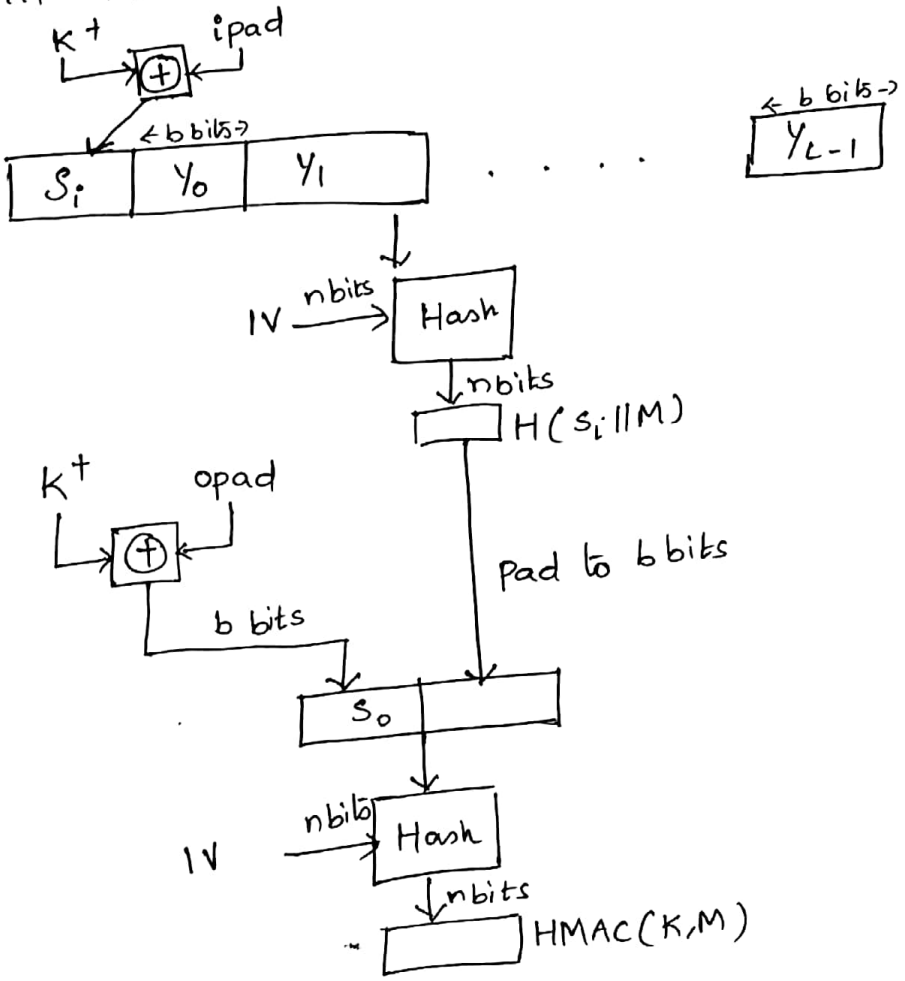
HMAC Design Objectives

1, 2
To use & handle keys in a simple way.

Existing implementation of a hash function can be used as a module in implementing HMAC.

So, HMAC code is prepackaged & ready to use w/o modification.

→ We can replace a given hash fn in an HMAC implementation We will remove the existing hash fn & insert new module.



HMAC structure

H = embedded Hash fn (e.g. MD5, SHA-1, RIPEMD-160)

IV = initial value i/p to hash function

M = msg i/p to HMAC

Y_i = i^{th} block of M, $0 \le i \le (L-1)$

L = no. of blocks in M

b = no. of bits in a block

n = length of hash code produced by embedded hash fn

K = Secret Key ($\ge n$)

K^+ = K padded with 0's on the left so that the result is b bits in length (21)

ipad = 00110110 (36 in hex) repeated b/8 times

opad = 01011100 (5C in hex) " " "

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

Alg:

1. Append zero's to the left end of K to create K^+
2. XOR K^+ with ipad to generate 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 & o/p the result

→ XOR with ipad results in flipping one-half of bits of K.

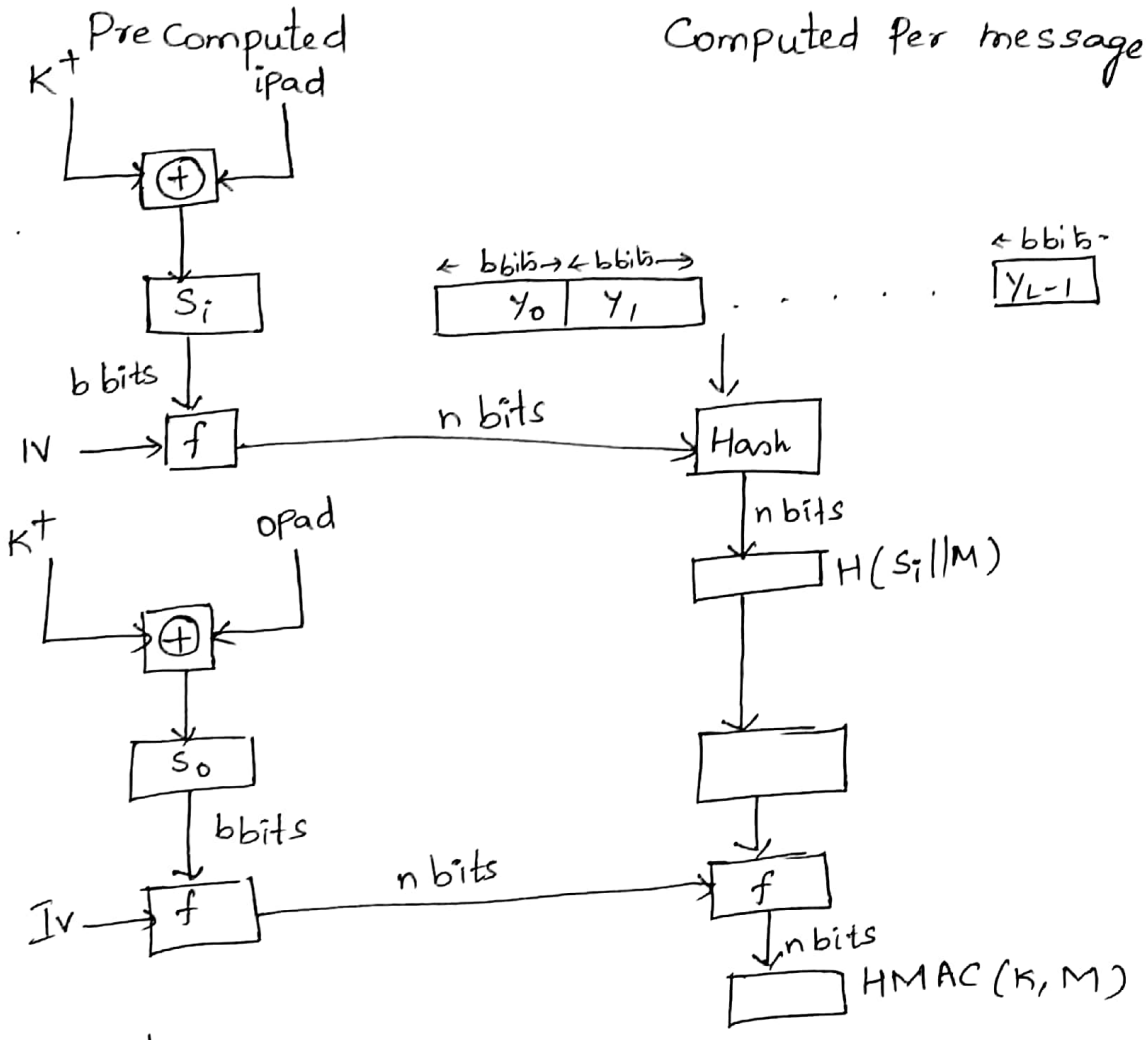
→ XOR with opad results in " " " " " "
using a different set of bits.

By passing S_i & S_o through Compression fn of hash alg; we have pseudorandomly generated 2 keys from K.

$f(IV, (K^+ \oplus ipad))$

$f(IV, (K^+ \oplus opad))$

where $f(cv, block)$ is the compression fn for the hash fn which takes as arguments a chaining variable of n bits & a block of b bits & produces a chaining variable of n bits. These quantities need to be computed initially & every time key changes.



Security of HMAC:

1. Bruteforce attack on the key order 2^n.
2. Birthday attack

(1) Attacker is able to compute an o/p of the compression fn even with an IV that is random, secret & unknown to the attacker.

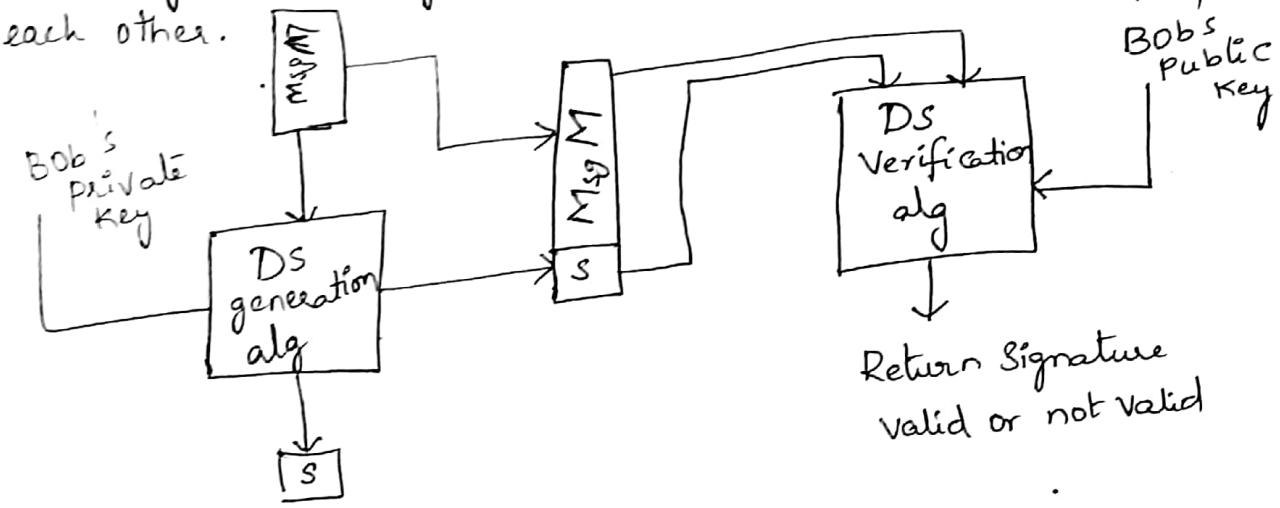
(2) Attacker looks for 2 msgs M & M' that produce the same hash $H(M) = H(M')$ – level of effort of $2^{n/2}$

Digital Signatures : (DS)

A DS is an authentication mechanism that enables the creator of a msg to attach a code that act as signature. Signature is formed by taking the hash of the msg & encrypting it with Creator's private key. Signature guarantees the source & integrity of the msg.

Properties :

→ MA protects 2 parties who exchange msgs ~~b/w~~ ~~each other~~. from any third party, but does not protect 2 parties against each other.



Generic Model of DS Process

Ds must have following properties

→ It must verify the author & the date and time of the signature.

→ It must authenticate the contents at the time of the signature.

→ It must be verifiable by third parties, to resolve disputes

Digital signature function includes authentication fn.

Attacks & Forgeries:

A denotes the user whose signature method is being attacked and C denotes the attacker.

- Key-Only Attack: C only knows A's public key.
- Known msg Attack: C is given access to a set of msgs & their signatures.
- Generic Chosen msg Attack: C chooses a list of msgs before attempting to break A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. Attack is not dependent on A's public key.
- Directed Chosen msg Attack: Similar to generic attack but attacks after C knows A's public key.
- Adaptive Chosen msg Attack: C is allowed to use A as an 'oracle' i.e. A may request signatures of msgs that depend on previously obtained msg-signature pairs.

Forgeries:

→ Total-break: C determines A's private key.

→ Universal forgery: C finds an efficient signing alg that provides an equivalent way of constructing signatures on arbitary msgs.

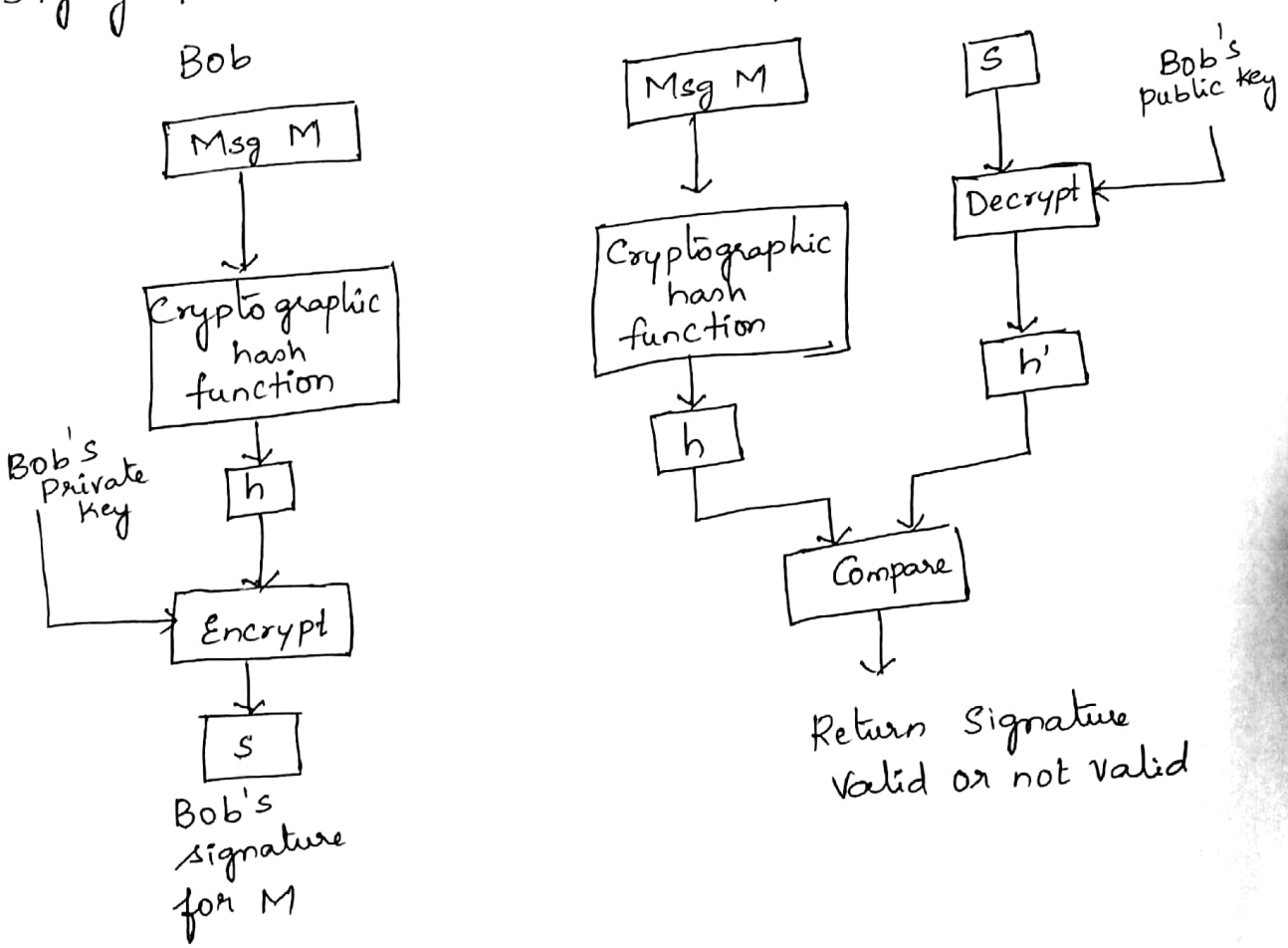
→ Selective forgery: C ~~finds~~ forges a signature for a particular msg chosen by C.

→ Existential forgery: C forges a signature for at least 1 msg. C has no control over the msg.

Digital Signature Requirements:

1) The signature must be a bit pattern that depends on the msg being signed.
2) The signature must use some info. unique to the sender to prevent both forgery & denial.
3) Relatively easy to produce Ds
4) Relatively easy to recognize & verify the digital signature.
5) Computationally infeasible to forge a digital signature, either by constructing a new msg for an existing Ds or by constructing a fraudulent Ds for a given msg.
6) It must be practical to retain a copy of the digital signature in storage.

A SHA algorithm embedded in Ds provides a basis for satisfying these requirements.



Direct Digital Signature:

→ Involves only the communicating parties.

→ Destination knows the public key of the source.

→ Confidentiality can be provided by encrypting the entire msg plus signature with a shared secret key.

→ Perform the signature function first & then an outer confidentiality function.

→ If the signature is the inner operation, then the recipient can store the plaintext msg & its signature for later use in dispute resolution.

→ Every signed message should include a timestamp & to require prompt reporting of compromised keys to a central authority.

Threat

Some private key might actually be stolen from X at time T. The opponent can then send a msg signed with X's signature & stamped with a time before or equal to T.

Digital Signature standard:

→ Dss makes use of SHA & presents a new digital signature technique, the Digital Signature Algorithm (DSA).

→ Dss uses an algorithm to provide digital signature function, it can't be used for encryption or key exchange. ← like RSA

→ It is a public-key technique.

Digital Signatures have 2 approaches

1. RSA approach
2. Dss Approach

RSA approach:

→ Msg to be signed is i/p to a hash function which produces a secure hash code of fixed length.

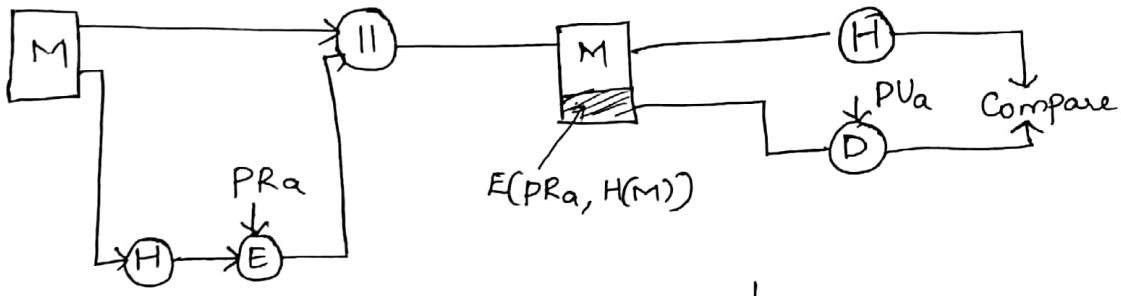
→ Hash Code is then encrypted using the sender's private key to form the signature.

→ Msg + signature are transmitted

→ Recipient takes the msg & produces hash code.

→ Recipient also decrypts the signature using sender's Public Key.

→ If the calculated hash code matches the decrypted signature, the signature is accepted as valid.



RSA Approach

DSS Approach:

→ This also uses hash function.

→ The hash code is provided as i/p to signature function along with a random number k generated.

→ Signature function also depends on sender's private key (PRa) & a set of parameters known to a group of Communicating Principals. (or global public key PU_G)

→ Result is signature consisting of 2 components labeled s & r.

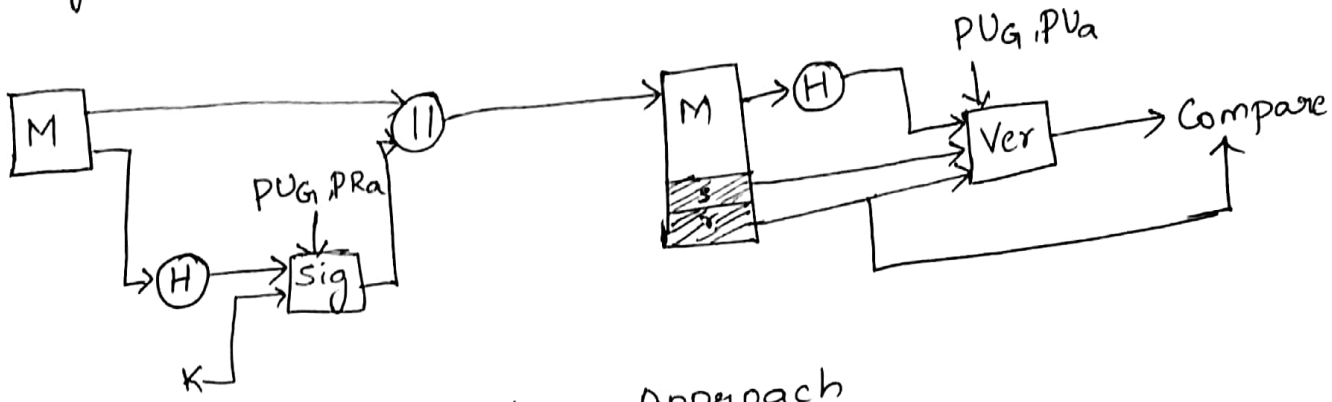
Receiver end

→ Hash code of incoming msg is generated.

→ signature is also given as i/p to Verification fn.

→ Verification function also depends on global public Key & sender's public key (PUa) which is paired with sender's private key.

→ Verification fn produces a value which is equal to signature component r if the signature is valid.



b) DSS Approach

Digital Signature Algorithm:

1. Select a 160-bit prime number q
2. select a prime number p such that $512 \leq L[p] \leq 1024$ such that q divides p-1
3. Choose $g = h^{(p-1)/q} \bmod p$ where h is an integer with $1 < h < (p-1)$ such that $h^{(p-1)/q} \bmod p > 1$
4. Each User selects a private key & generates a public key.
5. Private key x is chosen randomly or pseudorandomly & lies b/w 1 & q-1.
6. public key $y = g^x \bmod p$ is calculated.
7. Choose K randomly or pseudorandomly & is unique for each signing.

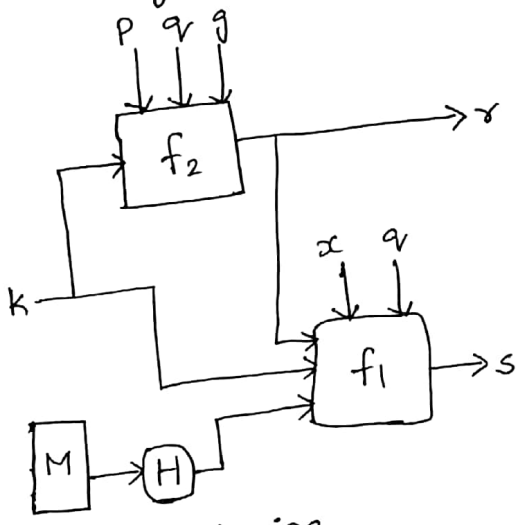
Signing:

→ To create a signature, a user calculates 2 quantities r & s, that are fns of Public-Key Components (p, q, g), user's private key (x), hash code of the msg H(M) & k.

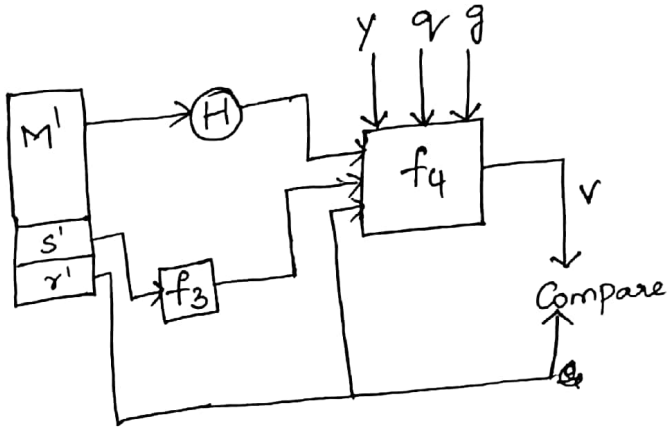
$$r = (g^k \bmod p) \bmod q$$

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

Signature = (r, s)



(a) Signing



(b) Verifying

Verifying:

→ Verification is performed at the receiving end.

→ Receiver generates a quantity v that is a function of the public key components, the sender's public key & hash code of the incoming msg.

→ Eq:

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

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

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

$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

Test : $v = r'$

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

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

→ Signature test at verification process is depend on r but does not depend on the msg.

→ v is a fn of K & P, q & g

→ Multiplicative inverse of K (mod q) is passed to a fn that also has as i/ps the msg hash code & user's private key.

→ Receiver can recover r using the incoming msg & signature, the public key of the user & global public key.