UNIT- IV

Cryptographic Hash functions :- (Compression fn)

Simple Hash function → (Authentication)

Msg. authentication is a mechanism or service used to verify the integrity of a message. Msg. authentication assures that data recieved are exactly as sent by sender and that the purported identity of the sender is valid.

The two most common cryptographic techniques for msg. authentication are a msg. authentication code and secure hash function.

Hash functions – A hash value $\overset{\text{hash code}}{\overset{\uparrow}{'h'}}$ is generated by a function 'H' of the form -

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

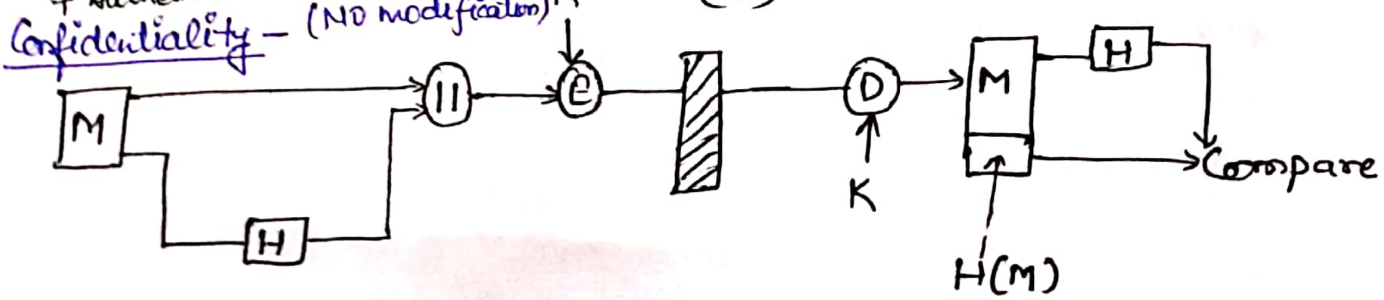
where 'M' is a variable-length message and H(M) is the fixed -length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct.

Unlike MAC, a hash code is also 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 in any bit or bits in the message results in a change in the hash code.

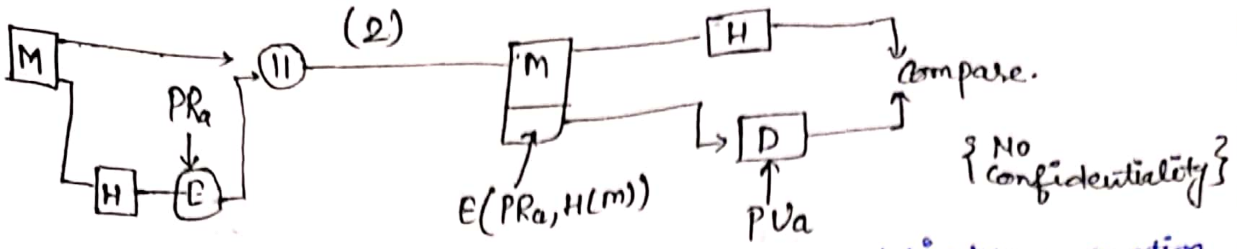
(1)

+ Authentication

Confidentiality – (No modification)

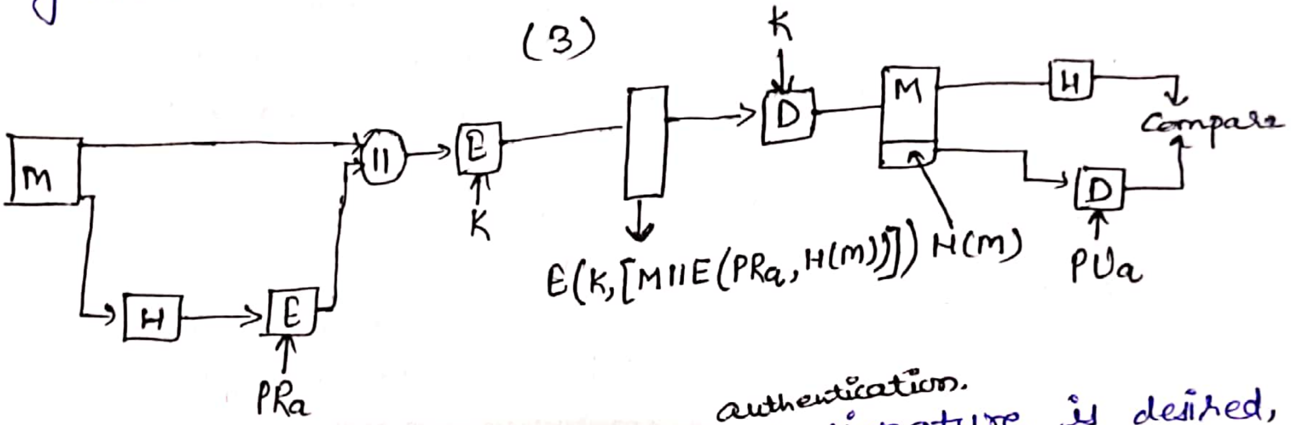


Internal error control strategy) (Symmetric key Encryption)



{No confidentiality}

(2) → Only 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. Infact, this is the essence of the digital signature.



(3)- If confidentiality as well as a digital signature (authentication.) is desired, then the msg plus the private key - encrypted hash code can be encrypted using a symmetric secret key. This is the common technique.

Requirement ~~for~~ Hash function:-

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for msg authentication, H must have following property:-

1)- H can be applied to a block of data of any size.

2)- H produces a fixed-length O/P.

3)- H(x) is relatively easy to compute for any given x, making both H/W & S/W implementation practical.

4)- For any given value h, it is computationally infeasible to find 'x' such that $H(x) = h$. This is sometimes referred to in the literature as the one-way property. (for secret info^n security) [Difficult to find a input that maps to a given hash o/p]

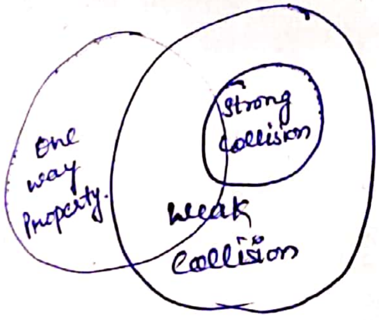
5)- For any given block x, it is computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$. This is sometimes referred to as weak collision resistance. [remove forgery] [Difficult to find two inputs mapping to same hash o/p]

6)- It is computationally infeasible to find pair (x, y) such that $H(x) = H(y)$. This is sometimes referred to as strong collision resistance. [remove birthday attack]

Birthday Attack→

for finding how 2 people has same birthday date.

We generally think - 365 days → 365+1 ... people. 100% probability. But acc. to Birthday Attack - 23 people will give more than 50% probability.



Hash f^n. based On Cipher Block chaining -

-) Although simple XOR or rotated XOR is insufficient, if only the hash code is encrypted, you may still feel that such a simple f^n could be useful when the message together with the hash code is encrypted. But you must be careful. A technique originally proposed by the National Bureau of Standards used a simple XOR applied to 64 bits block of the message and then encryption of the entire message that used CBC mode.

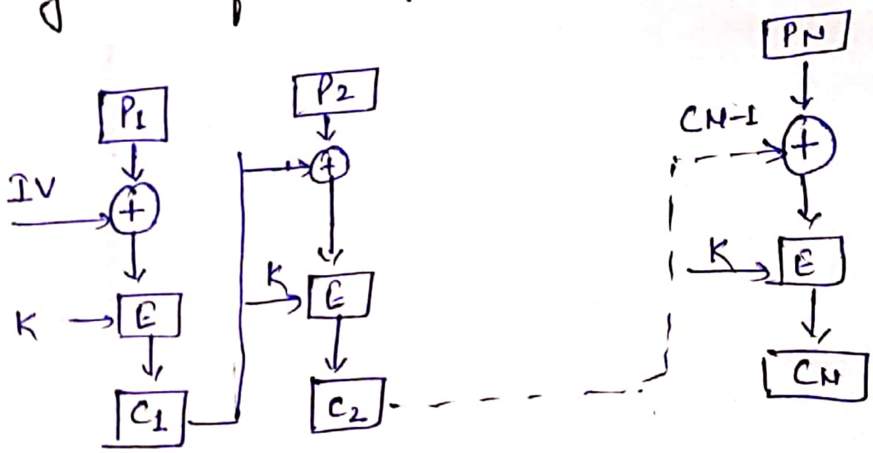
→ We can define the scheme as –

- Given a message M consisting of a sequence of 64-bit blocks $P_1, P_2, P_3 \cdots P_N$, define the hash code $h = H(M)$ as the block by block XOR of all the blocks and append the hash code as the final block:

$$h = P_{N+1} = P_1 \oplus P_2 \oplus P_3 \oplus \cdots \oplus P_N.$$

- Next Encrypt the entire message plus the hash code using CBC mode to produce the encrypted message $C_1, C_2, C_3 \cdots C_{N+1}$. There are several ways the ciphertext can be manipulated in such a way that it is not detectable by hash code.

By the definition of CBC-



we have –

$$P_1 = IV \oplus D(K, C_1)$$
$$P_i = C_{i-1} \oplus D(K, C_i)$$
$$P_{N+1} = C_N \oplus D(K, C_{N+1})$$

$$h = P_{N+1} = [IV \oplus D(K, C_1)] \oplus [C_1 \oplus D(K, C_2)] \oplus \cdots$$
$$[C_{N-1} \oplus D(K, C_N)]$$

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

Secure Hash Algorithm (SHA) → It was developed by the National Institute of standards & Technology (NIST) in 1993.

↳ SHA is a modified version of MD5.

↳ SHA-1 produces a hash value of 160 bits. (1995)

↳ In 2002, NIST produced a revised version of the standard with three versions — with hash value lengths of 256, 384 & 512 bits known as SHA-256, SHA-384 and SHA-512.

Comparison of SHA parameters —

	SHA-1	SHA-256	SHA-384	SHA-512
Message Digest	160	256	384	512
Message Size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Steps in algo	80	64	80	80

SHA properties —

i)- Generating original message from digest } Infeasible.

ii)- Finding two messages generating same digest

Before understanding SHA, we need to understand MD5.

MD5 —

↳ Developed by Ron Rivest.

↳ Fast & Produces 128-bits message digest.

Working - i)- Original message | Padding (10000 ....)

↳ Padding is done such that total length is 64 bit less than exact multiple of 512.

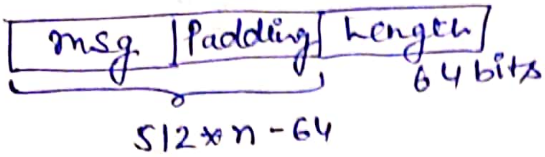
ex- 1000 bits + 472 bits → [Exact 64 bit less than multiple of 512]

512 * 2 = 1024 (this is not less than 64 bit of multiple of 512)

512 * 3 = 1536 - 64 = 1472

ii)- Append Original length before Padding - (Modulo 64).

⇒ 1000 bits



generally length will be like msg is 2^8 then length

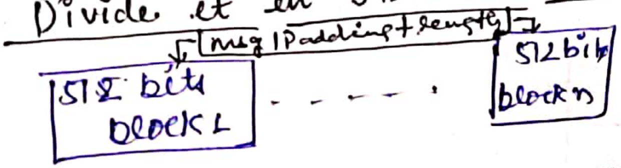
⇒ 1111LLLL000 - - - .64 bits

if msg. is 2^{64} then length is all 1LLL - - - -64-

if msg. is greater than 2^{64} than take modulo 64 and then use the length.

↳ By this step we finally get exact multiple of 512 bits.

iii)- Divide it in 512 bits blocks -



iv)- Initialize A - chaining variables :- (32 bits, A, B, C & D)

Values of these variables (registers) are predefined in Hexa-decimal.

A = 0 1 2 3 4 5 6 7

B = 8 9 a b c d e f

C = f e d c b a 9 8

D = 7 6 5 4 3 2 1 0

v)- Process each 512 - bit block - [Each 512 bit block goes to 64 operations]

in the sequence of 16 - 16 - 16 - 16 operations.

first 16 operations will use function - 'F'

second " " " " " - 'G'

third " " " " " - 'H'

fourth " " " " " - 'I'

$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$ → Not

$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$

$H(B,C,D) = B \oplus C \oplus D$

$I(B,C,D) = C \oplus (B \vee \neg D)$

Single step of Operation

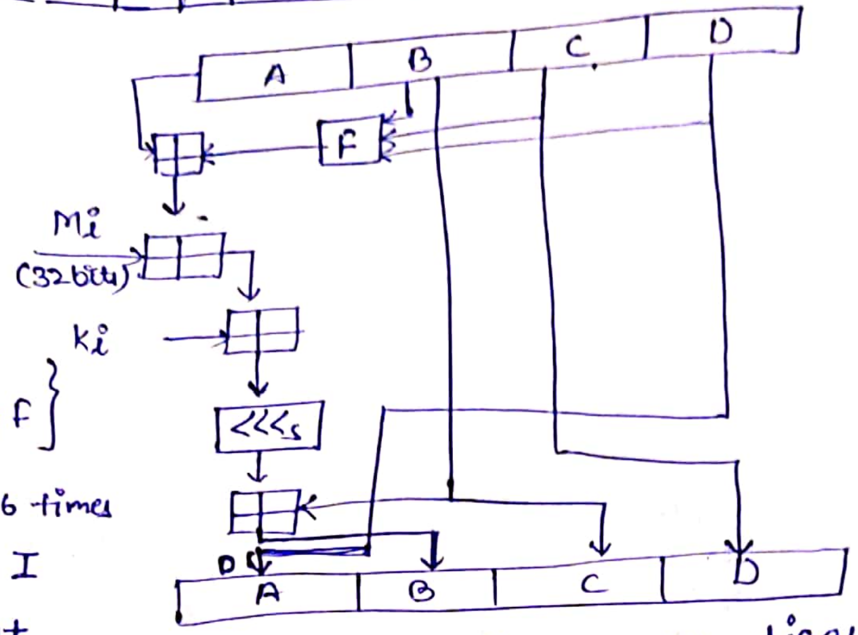
⊞ → Addition modulo 2^{32}

M_i → msg. input of 32 bits

$<<<_s$ → s bits left circular shift

K_i → 32 bit key constant

32 ×16 / 192 / 32 / 512



{16-f times using f} then using G .16 times again H & I

the last O/P will be of - 32×4 = 128 bits message digest.

Coming Back to SHA-1 – (Gives -160 bits O/P)

Working – i)- Padding [64 bit less than exact multiple of 512]

ii) – Append Length

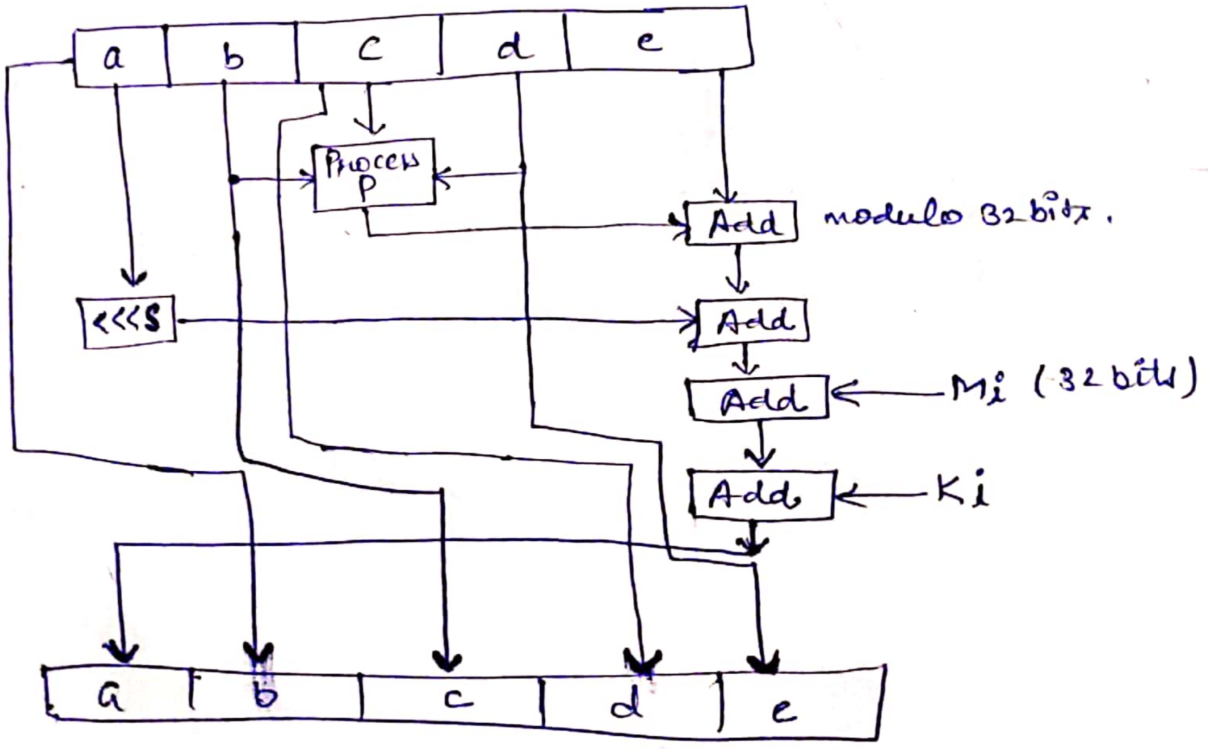
iii) - Divide the I/P into 512 bit blocks.

iv) - five chaining variables (A, B, C, D, E)

v) - Process Blocks. [4-Rounds and each of 20 steps or operations] total - 80.

32 ×20 / 640

32



Round	Process P
1	$(b \wedge c) \vee (\neg b \wedge d)$
2	$(b \oplus c \oplus d)$
3	$(b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
4	$b \oplus c \oplus d$

Difference b/w MD5 & SHA-1

	MD5	SHA-1
① Message Digest length in bits	128	160
②- Attack to try and find the original message given a message digest	Requires 2^{128} operations to break in.	Requires 2^{160} bits.
③- Attack to try and find two messages producing same message digest	Requires 2^{64} operations to break in	2^{80} operations
④. Speed	Faster	Slower.
⑤- Successful attempts so far	There have been reported attempts to some extent	No such claims so far.

UNIT-IV

Message Authentication Code :- MAC is a block of a few bytes that is used to authenticate a message.

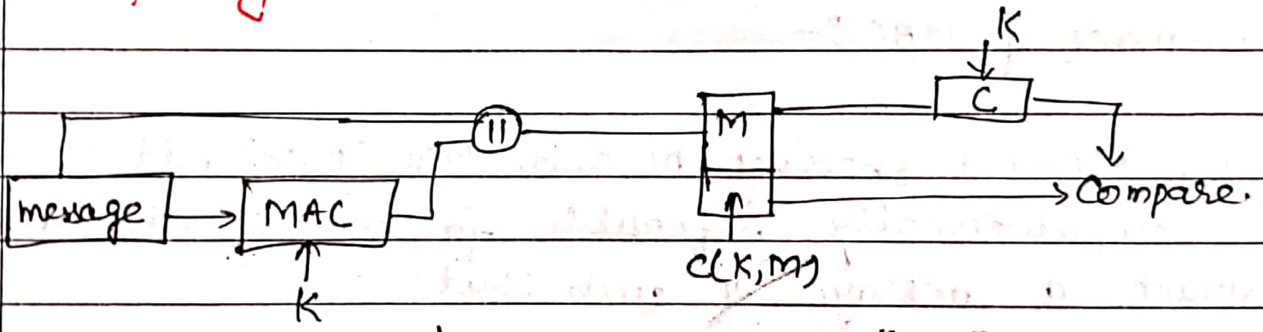
For establishing MAC process, the sender and reciever share a symmetric key K.

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

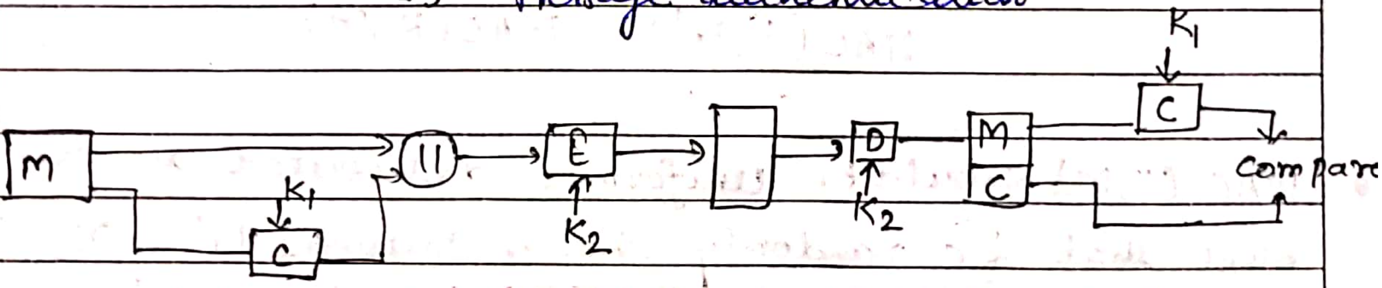
$$MAC = C(K, M)$$

M= variable length message, K = is a secret key (Shared) & C(K,M) is a fixed length authenticator.

The MAC is appended to the message at the source at a time when message is assumed or known to be correct. The reciever authenticates that messages by recomputing the MAC.

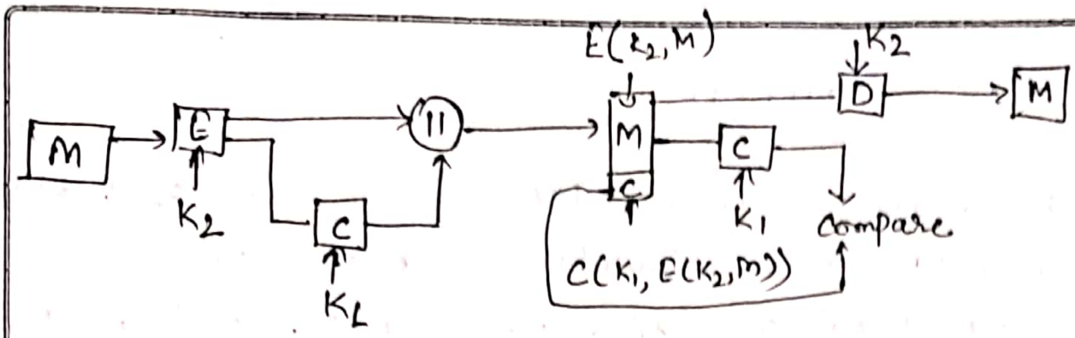


1)- Message authentication



2)- Msg. Authentication & Confidentiality.

authentication tied to plain Text.



(3) Msg. authentication & Confidentiality ; authentication tied to ciphertext.

Limitation of MAC -

1)- It can provide message authentication among pre-decided legitimate users who have shared key.

2)- This requires establishment of shared secret prior to use of MAC.

Requirements of MAC :-

- If an opponent observes M and MAC(K,M), it should be computationally infeasible for the opponent to construct a message M' such that.

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

- MAC (K,M) should be uniformly distributed in the sense that for randomly chosen messages M' & M, the probability that MAC(K,M) = MAC(K,M') is 2^{-n}, where n is the no. of bits in the tag.

- Let M' be equal to some known transformation on M. that is, M' = f(M) for ex- f may involve inverting one or more specific bits. In that case- $Pr[C(K,M) = C(K,M')] = 2^{-n}$

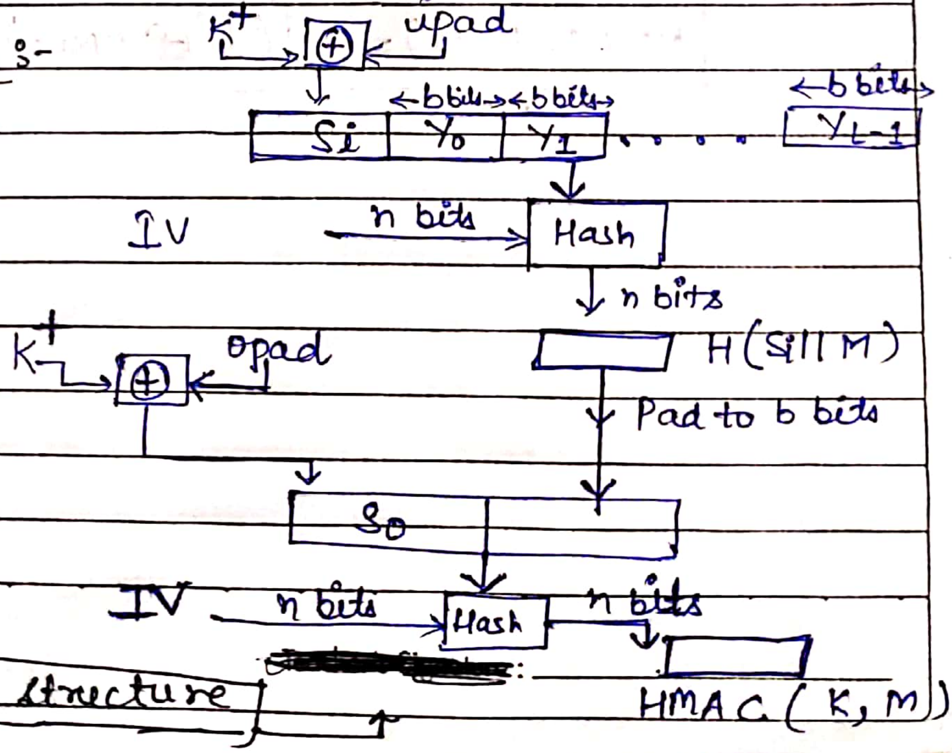
MAC based on Hash function :- (HMAC)- HMAC stands for Hashed or Hash based Message Authentication Code. It is a result of work done on developing a MAC derived from cryptographic hash functions.

HMAC is a great resistant towards cryptanalysis attacks as it uses the hashing concept twice. HMAC consists of twin benefits of hashing and MAC and thus is more secure than any other authentication codes. HMAC has been made compulsory to implement in IP security.

Objectives →

- As the Hash function, HMAC is also aimed to be one way, i.e, easy to generate output from input but complex the other way round.
- It aims at being less effected by collisions than the hash functions.
- HMAC tries to handle the keys in more simple manner.

HMAC algorithm :-



HMAC structure

- H = embedded hash function (eg. MD5, SHA-1)
- IV = initial value input to hash function
- M = message input to HMAC (including the padding specified in the embedded hash function)
- Y_i = ith 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 function.
- K = secret key recommended length is $\geq n$, if key length is 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.

HMAC can be expressed as follows :-

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

CMAC (MAC based on Cipher Block Chaining) -
(CBC-MAC) -

Let us consider the operation of CMAC 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 is divided into 'n' blocks, $M_1, M_2 \cdots M_n$. The algorithm makes use of a k-bit encryption key K and an n-bit constant K_1.

For AES, the key size K is 128, 192 or 256 bits for triple DES, the key is 112 or 168 bits.

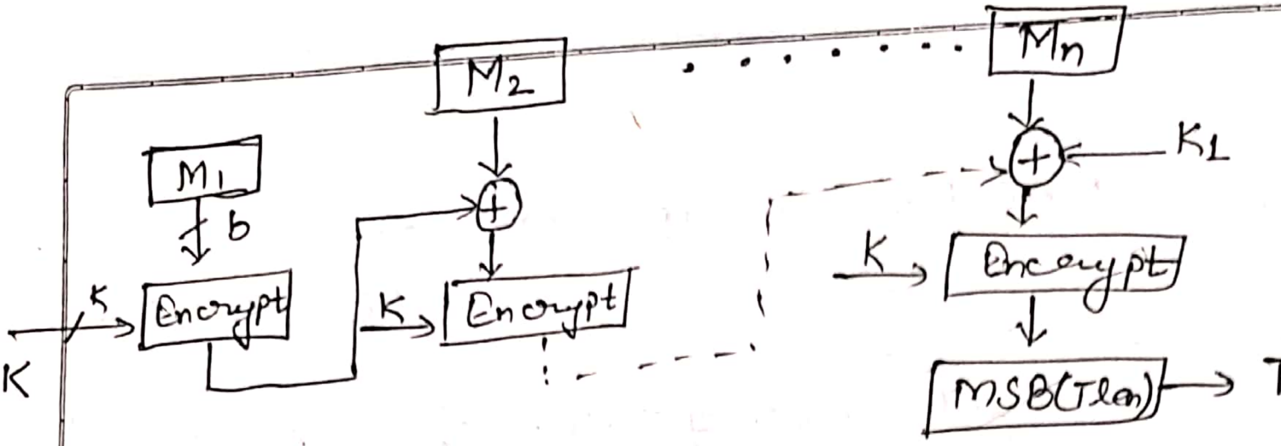
$$C_1 = E(K, M_1)$$
$$C_2 = E(K, [M_2 \oplus C_1])$$
$$C_3 = E(K, [M_3 \oplus C_2])$$
$$\vdots$$
$$C_n = E(K, [M_n \oplus C_{n-1} \oplus K_1])$$
$$T = MSB_{Tlen}(C_n)$$

where

T = MAC, also referred to as the tag.

Tlen = bit length of T.

$MSB_s(x)$ = the S leftmost bits of the bit String x.



Digital Signature :-

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.

The digital signature standard (DSS) is an NIST standard that uses the secure hash algorithm (SHA).

DS work because public key cryptography depends on two manually authenticating cryptographic keys. The individual who is creating the digital signature uses their own ~~data~~ private key to encrypt signature related data.
The only way to decrypt that data is with the signer's public key.

~~forev~~- Properties of digital signature :-

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

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

→ It must be verifiable by third party, to resolve dispute.

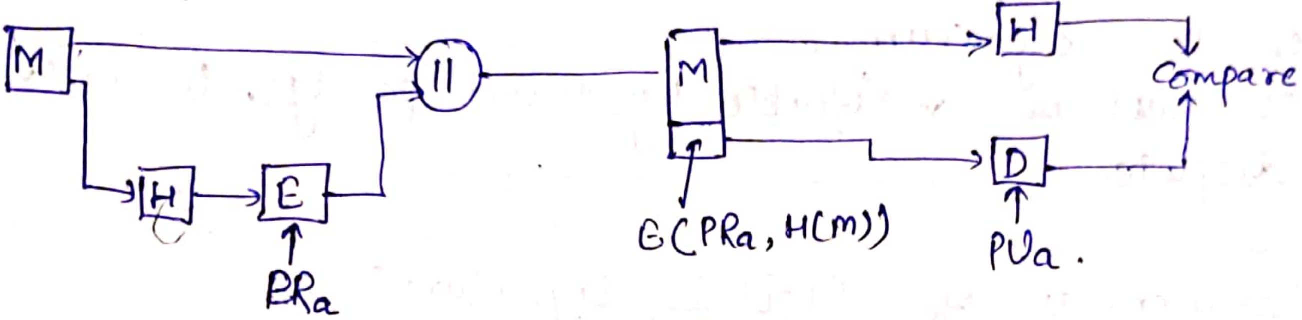
Requirements of Digital Signature :-

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender to prevent both forgery and denial.
- It must be computationally infeasible to forge a digital signature, either by constructing new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.

Digital Signature Standard :-

The DSS makes use of the secure hash algorithm and presents a new digital signature technique, the digital signature algorithm (DSA).

- RSA approach-



- DSS Approach - This approach also makes use of hash function.

The hash code is provided as input to a signature function along with a random number 'k' generated for this particular signature.

Sender Side -
The signature fn also depends on the sender's private key (PRa) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PUG). The result is a signature consisting of two components, labeled s and r.

Reciever Side - At this end, the hash code of the incoming message is generated. This plus the signature is input to a verification function.

The O/P of the verification is a value that is equal to the signature component 'r', if the signature is valid.

DSS Algorithm –

Global Public-key components :–

p = prime no., where $2^{L-1} < p < 2^{L}$, for $512 \le L \le 1024$ and L a multiple of 64.

q = prime divisor of $(p-1)$, where $2^{159} < q\, 2^{160}$. i.e. bit length of 160 bits.

$$\boxed{g = ~~h^{(p-1)/q}~~ \; h^{(p-1)/q} \bmod p}$$

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 No. :–

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)

Scanned by CamScanner

Verification At reciever side –

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

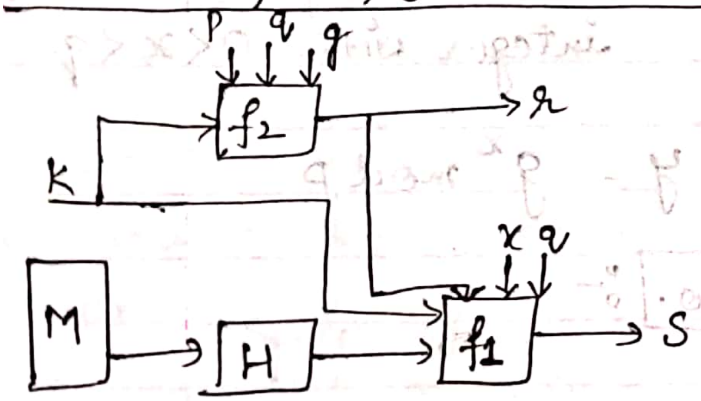
$$u_1 = [H(M') w] \bmod q$$

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

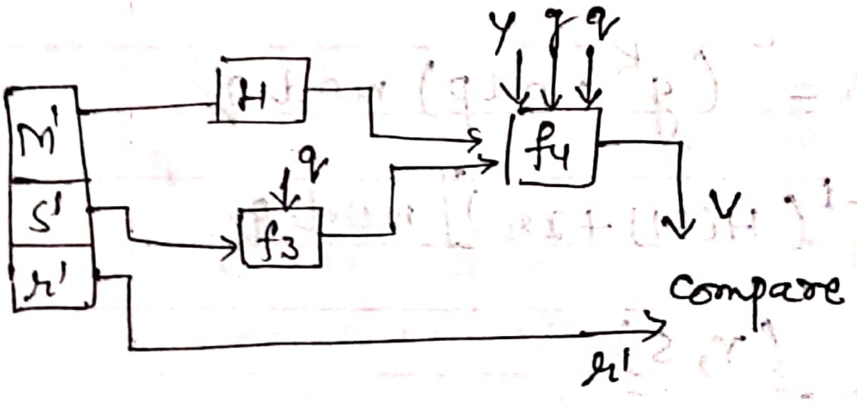
$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

Test $\boxed{v = r'}$

M', r', s' = Recieved version of M, r, S



DSS signing



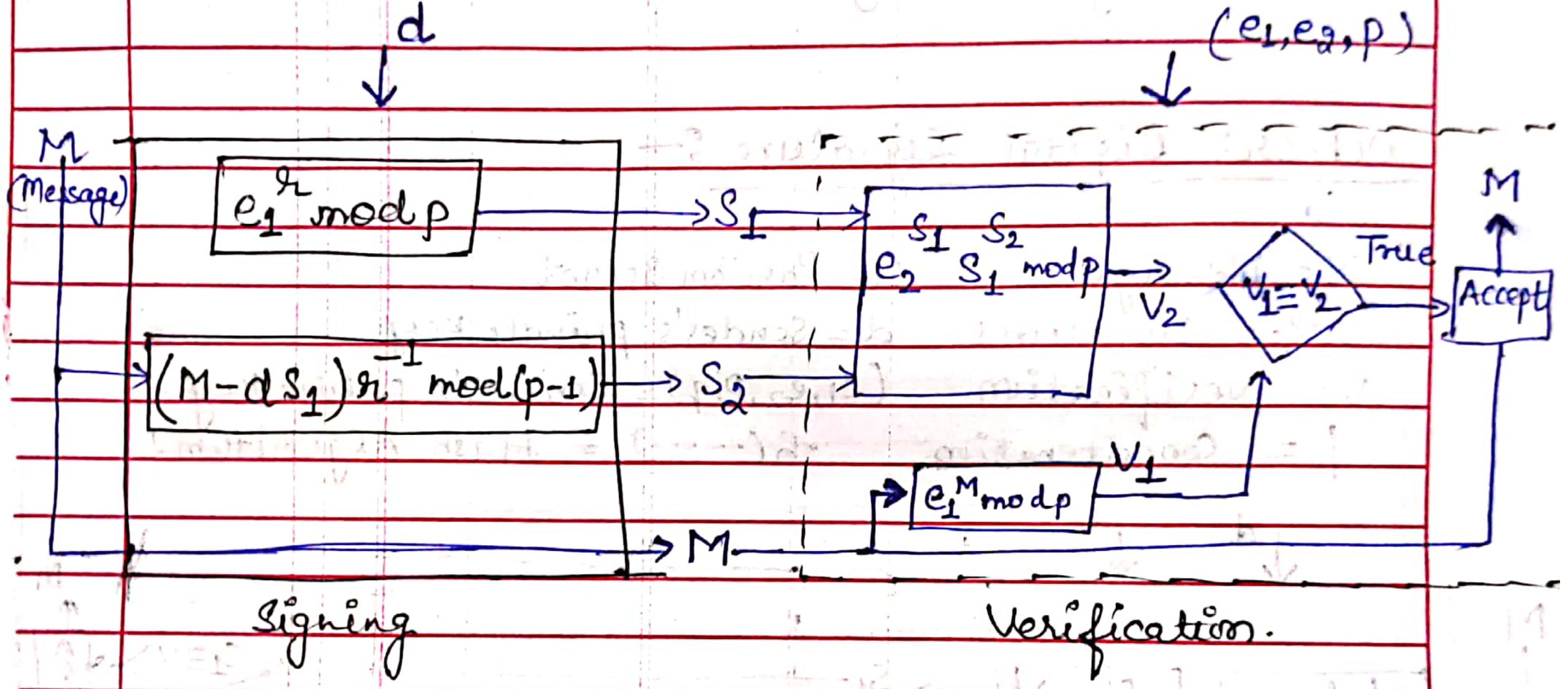
DSS verifying

Elgamal based Digital signature :-

M = Message r = Random Secret

S_1, S_2 = Signatures d = Sender's private key

V_1, V_2 = Verifications (e_1, e_2, p) = Sender's public key.



- In the signing process, two functions create two signatures. in the verifying process the output of two functions are compared for verification.

- One function is used both for signing and verifying but the function uses different inputs.

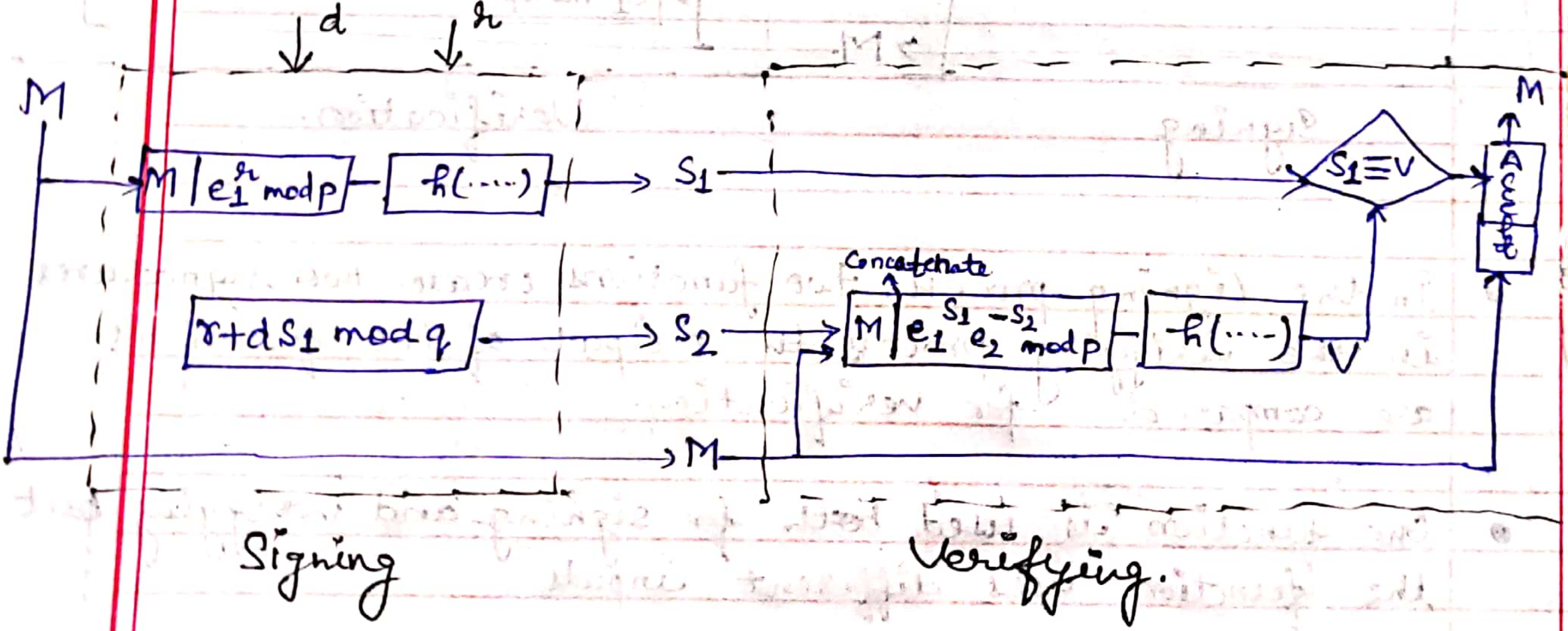
- The message is part of the input to function 2 when signing, it is part of the input to function 1 when verifying.

- p should be atleast 1024 bits. This could make the signature as large as 2048 bits.

- To reduce the size of the signature, schnorr proposed a new scheme based on Elgamal, but with a reduced signature size.

Schnorr Digital Signature :-

M = Message	r = Random Secret
S_1, S_2 = Signatures	d = Sender's private key
V = Verification	(e_1, e_2, p, q) = Sender's public key.
| = Concatenation	h(.....) = Hash Algorithm.



Signing - 1)- Sender chooses a random number 'r'.

(2)- Sender calculates $S_1 = h(M \mid e_1^r \bmod p)$.

(3)- Sender calculates $S_2 = r + d * S_1 \bmod q$

(4)- Sender sends M, S_1 & S_2.

Verifying – 1)- Reciever calculates

$$V = h(M | e_1^{S_2} \, e_2^{-S_1} \bmod p)$$

2)- If S_1 is congurent to V modulo p, the message is accepted, otherwise rejected.