Unit - 4

Message integrity

Describes the concept of ensuring that data has not been modified in transit.

Message authentication

If Alice needs to ensure the content of her document should not change, one way to preserve is through the use of her fingerprint.

Eve cannot modify the contents of the document or create a false document because she cannot forge Alice's fingerprint.

Methods for Message integrity

→ To preserve the integrity the electronic equivalent of the document and fingerprint is the message and digest pair.

→ The message digest needs to be safe from change.

→ Message digest generated through cryptographic hash function.

→ The function creates a compressed image of the message that can be used like a finger print.

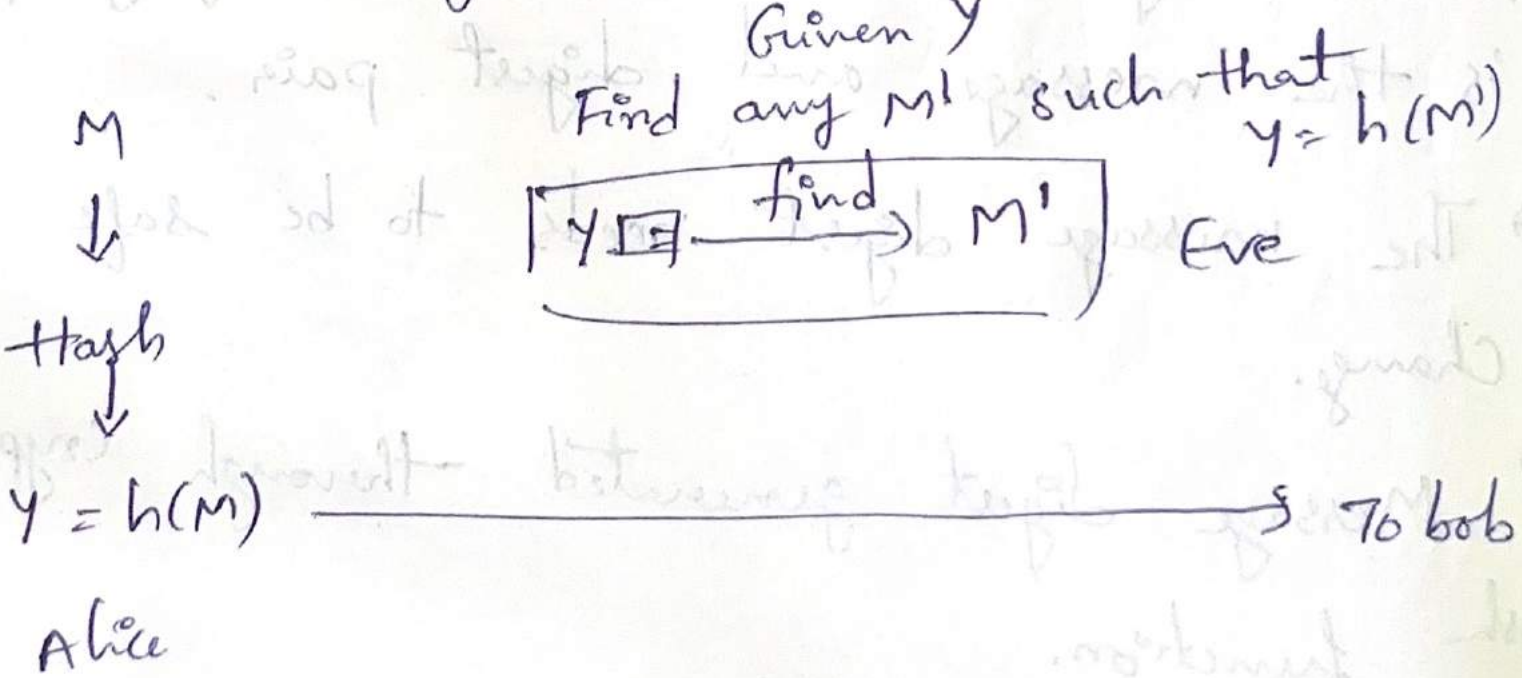
Cryptographic Hash function criteria

→ preimage resistance
→ Second preimage resistance
→ Collision resistance

pre-image Resistance

If a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
This property protects against an attacker who only has a hash value and is trying to find the input.

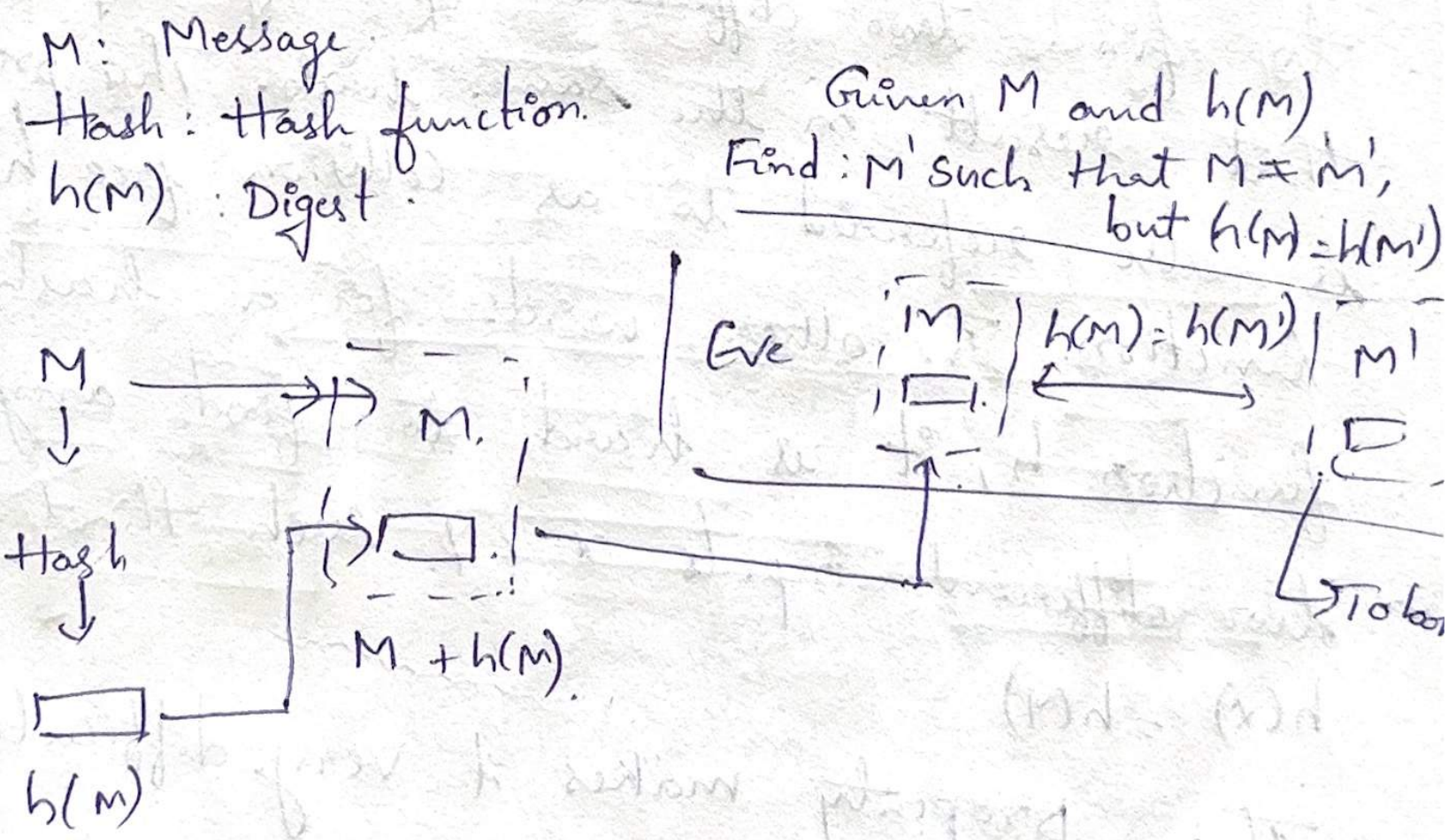
M: Message.
Hash: Hash function
h(M): Digest



Second pre-Image Resistance

If a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x)

This property of hash function protects against an attacker who has input value and its hash, and wants to substitute different value as legitimate value in place of original input value.



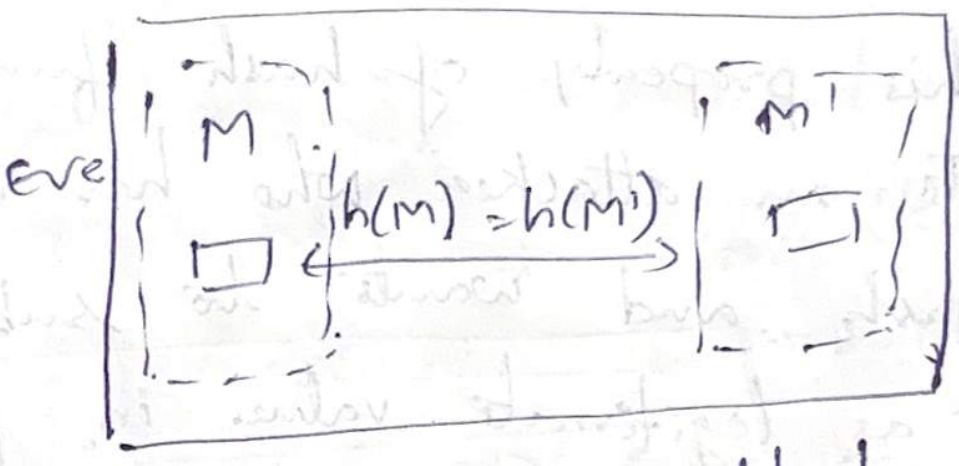
Collision Resistance

M: Message

Hash: Hash function

h(M) : Digest

Find M & M' such that M ≠ M'
but h(M) = h(M')



This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function. In other words for a hash function h, it is hard to find any two different inputs x & y such that

h(x) = h(y)

→ This property makes it very difficult for an attacker to find two input with same hash.

Applications of Hash function:

→ Password verification
→ Signature Generation and verification
→ Verifying file & Message integrity.

Message Authentication

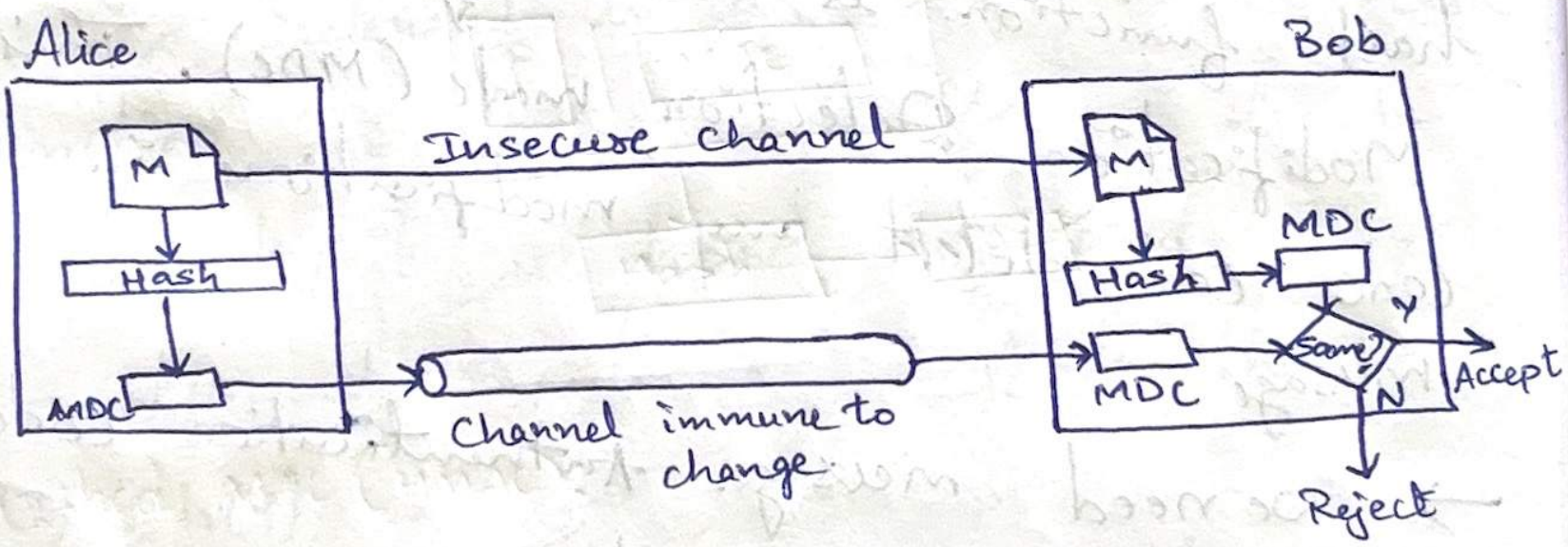
→ A message digest guarantees the integrity of a message. It guarantees that the message has not been changed.
→ To provide message Authentication, Sender needs to provide proof that it is send by the sender and not a fraud.
→ The digest created by a cryptographic hash function is normally called a Modification Detection mode (MDC). This code can detect any modifications in the message.
→ We need message Authentication code (MAC) for message Authentication

Modification Detection Code (MDC)

MDC is a message digest that can prove the integrity of the message, that message has not been changed.

→ If Alice needs to send a message to Bob and be sure that the message will not change during transmission.

→ Alice can create a message digest, MDC, and send both the message and the MDC to Bob. Bob can create a new MDC from the message and compare the received MDC and the new MDC. If they are the same, the message has not been changed.



M: Message

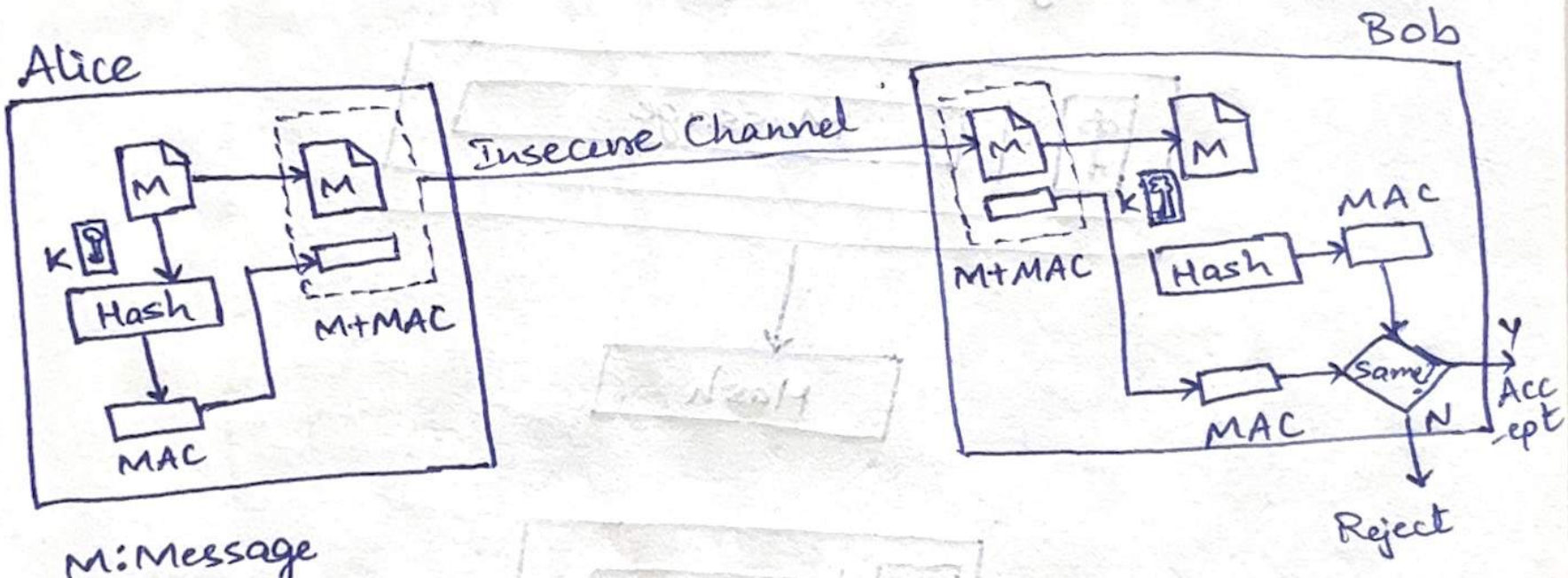
Hash: Cryptographic hash function

MDC: Modification detection code

Message Authentication Code (MAC):

→ To ensure the integrity of a message and the data origin authentication - we need to change a MDC to a MAC.

→ The difference between MDC and MAC is that the second include a secrete key between Alice and Bob.



M: Message

K: A shared secret key

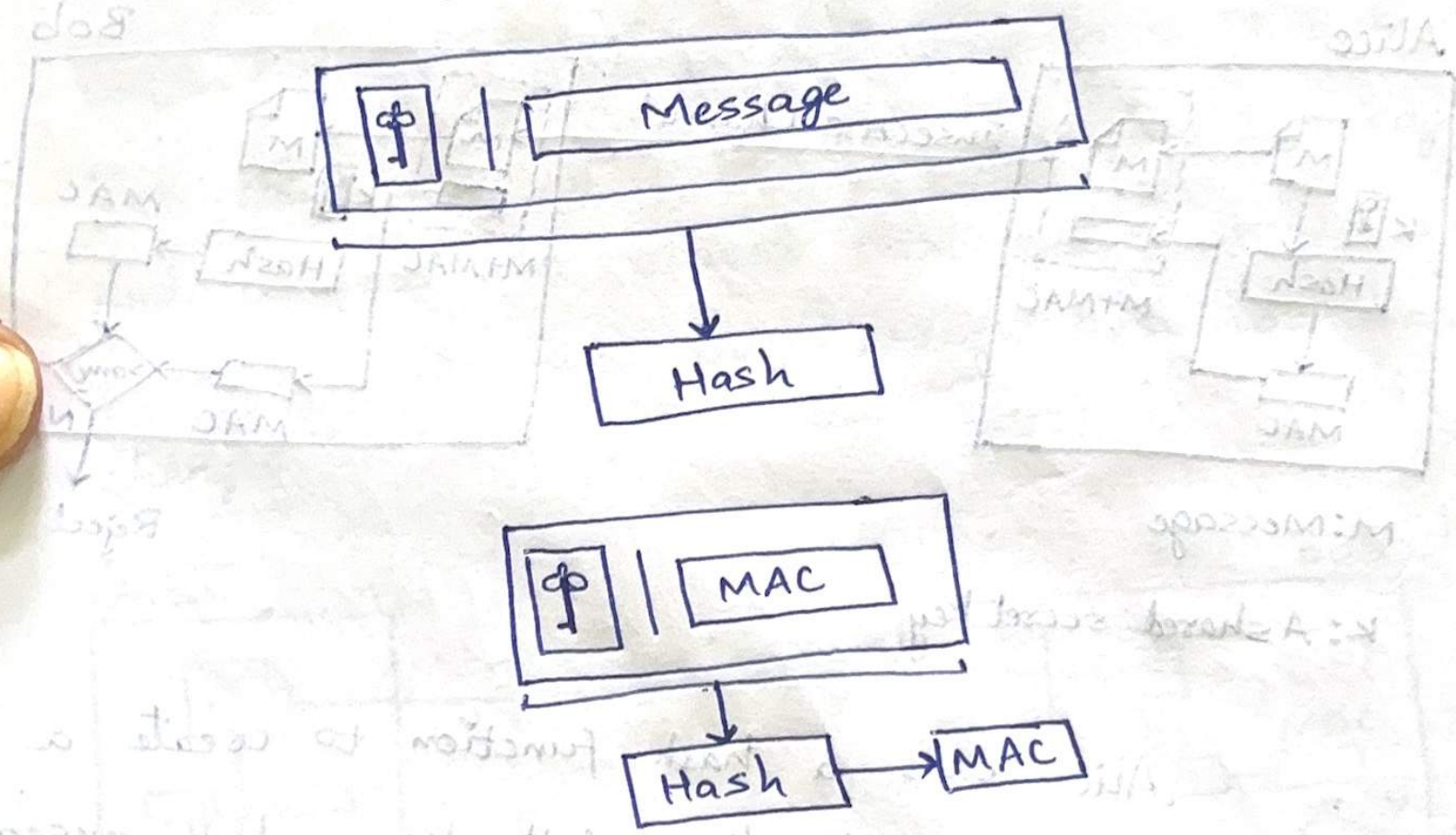
Alice uses a hash function to create a MAC from the concatenation of the key and the messages, h(k|M). She sends the message and the MAC to Bob over the insecure channel. Bob seperates the message from the MAC. He then makes a new MAC from the concatenation of the message and the secret key. Bob then compares the newly created MAC with the one received. If the two MACs match, the message is authentic and has not been modified by an adversary.

Nested MAC :: (NAM)

→ To improve MAC security, nested MAC's were designed in which hashing is performed twice.

→ In 1st step, the key is concatenated with the message and is hashed to create an intermediate digest.

→ In 2nd step, the key is concatenated with the intermediate digest to create the final digest.



Hashed MAC (HMAC):

→ HMAC algorithm stands for Hashed or Hash based MAC.

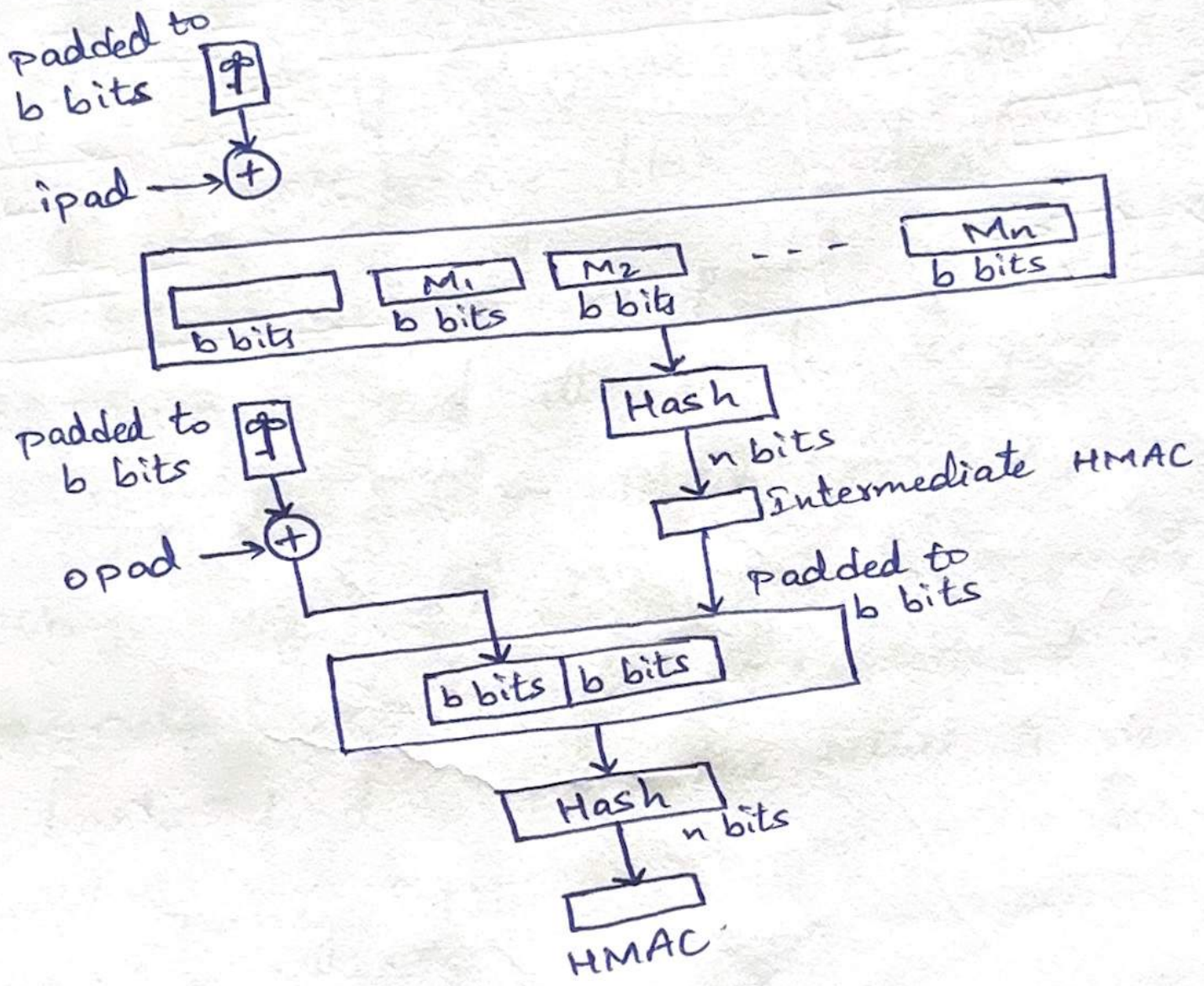
→ It uses the Hashing concept twice, so great resistant to attacker.

→ HMAC consists of twin benefits of Hashing & MAC.

→ The working of HMAC starts with taking a message M containing blocks of length b bits.

→ An input signature is padded to the left of the message and the whole is given as input to a hash function which gives us a intermediate HMAC.

→ Intermediate HMAC again is appended to an output signature and the whole is applied a hash function again, the result is our final HMAC of n bits.



Hashed MAC (HMAC)

→ Stands for Hashed or Hash based Message Authentication Code.

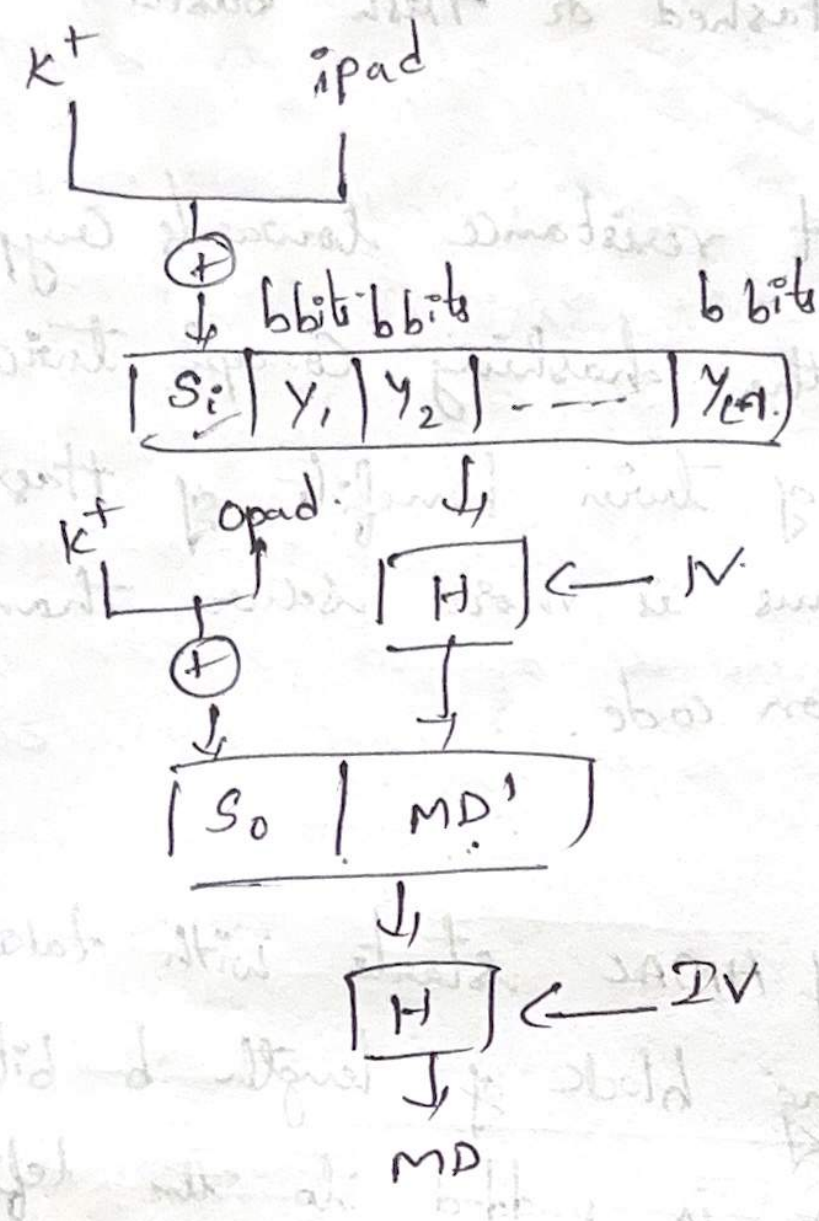
→ HMAC is a great resistance towards cryptanalysis attacks as it uses the hashing concept twice

→ HMAC consists of twin benefits of Hashing and MAC and thus is more secure than any other authentication code.

HMAC Algorithm

The working of HMAC starts with taking a message M containing block of length b bits. An input signature is padded to the left of the message and the whole is given as input to a hash function which give us a temporary message-digest MD'. MD' again is appended to an output signature and the whole is applied a hash function again, the result is our final message. MD

Structure of HMAC



K ($0 < K < b$)

H - Hashing function
M - Original message
S_i - input signature
S_o - output signature
Y_i - i^{th} block in original message M
L - Count of blocks in M
K - Secret key
IV - Initial vector (Some Constant)

$S_i = K^+ \oplus ipad$

$S_o = K^+ \oplus opad$.

K^+ is nothing but K padded with zeros on the left. so that the result is b bits in length.

ipad → 00110110
opad → 01011100

respectively taken b/8 times repeatedly

CMAC

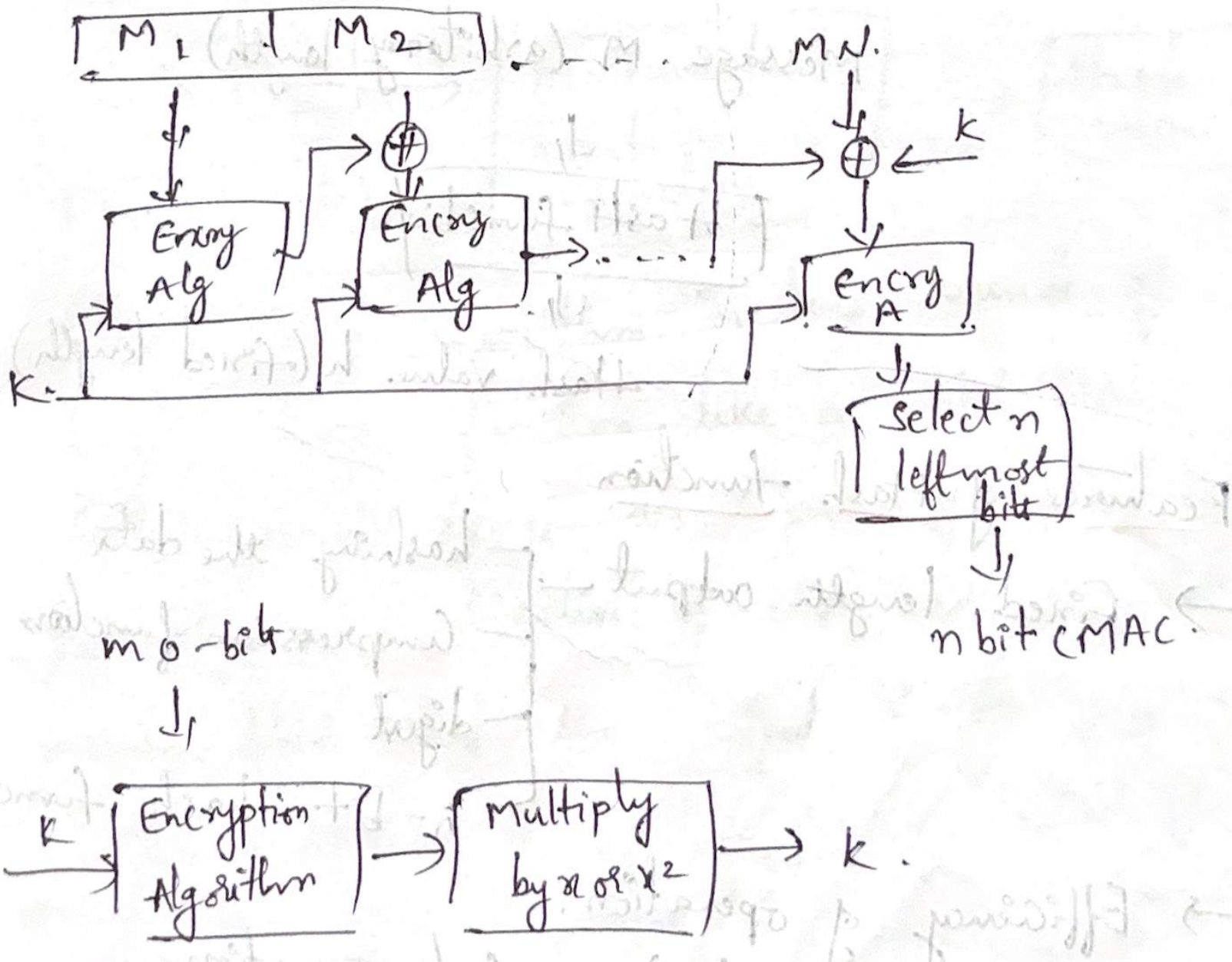
→ Similar to CBC (Cipher Block Chaining)

→ It takes N blocks of message but creates one block of MAC

→ The message is divided into N blocks of m-bit size. If last block is not m-bit size, then padded with start 1 then 000 ..., like 100000...

→ The block is encrypted with key k then its output is XOR with the next block for 2nd encryption and so on.

→ The last block is encrypted with some additional K value for more security.



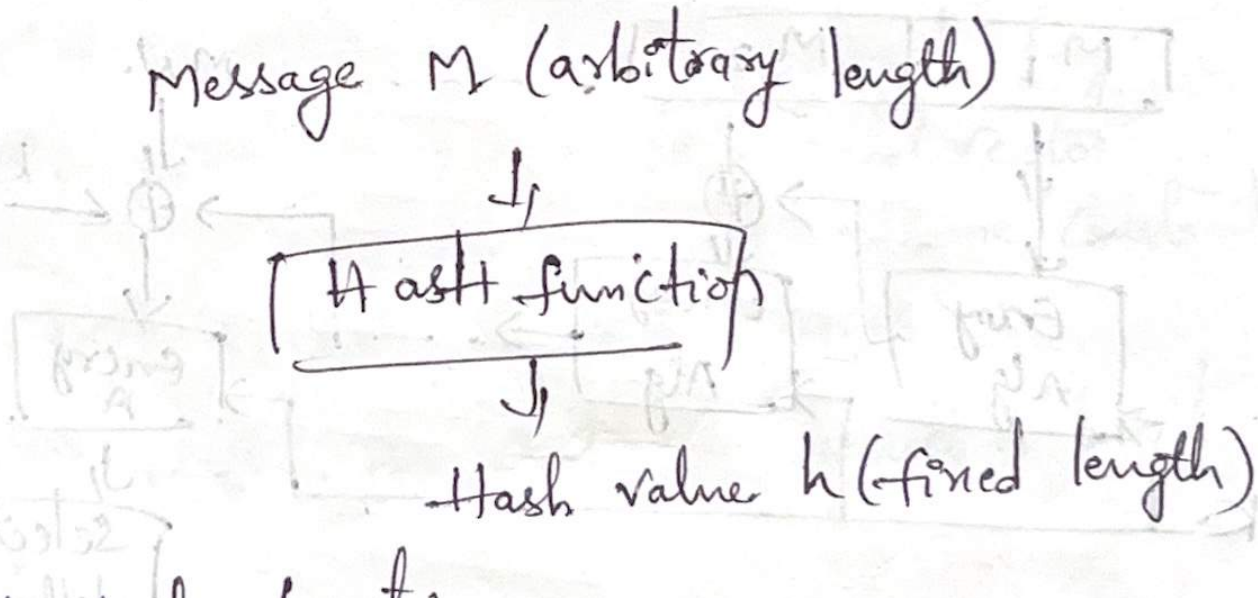
Cryptographic hash functions

A cryptographic hash function is a mathematic function used in cryptography.

→ Typical hash functions take inputs of variable length to return outputs of a fixed length.

→ Hash functions are extremely useful and appear in almost all information security applications.

→ Values returned by a hash function are called message digest



Features of Hash function

→ Fixed length output
- hashing the data
- Compression function
- digest
- n-bit hash function

→ Efficiency of operation.

h(x) — fast operation

Hash functions are much faster than a symmetric encryption.

Properties of Hash functions

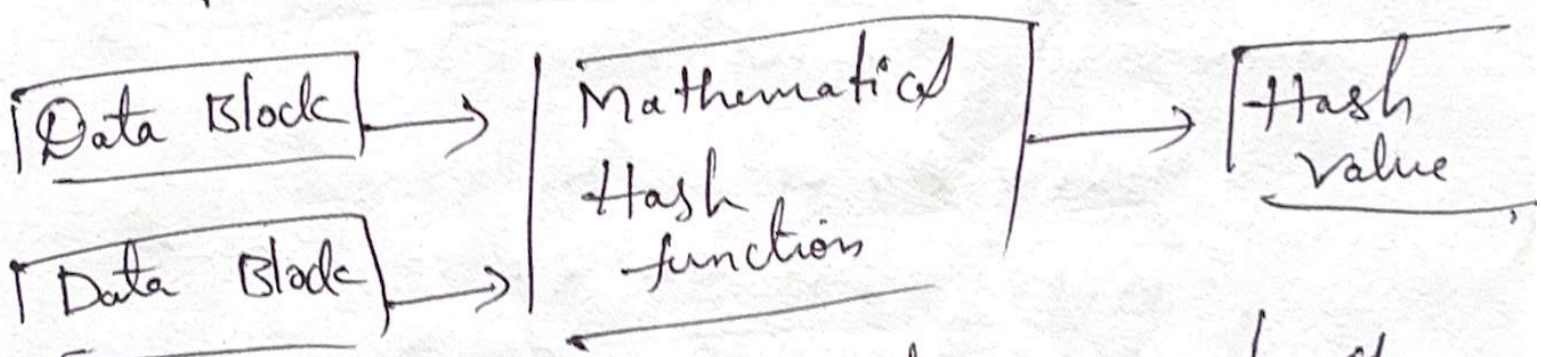
→ Pre-image Resistance
→ Second pre-image Resistance
→ Collision Resistance.

Designing of Hashing Algorithms

Hashing is a mathematical function that operates on two fixed-size blocks of data to create a hash code.

Hash function forms the part of the hashing algorithm.

→ Size of data block varies depending on the algorithm.



Hashing Algorithm involves rounds of above hash function like a block cipher.

Each round takes an input of fixed size typically a combination of the most recent message block and the output of the last round.

This process is repeated for as many rounds as are required to hash the entire message.

Popular Hash functions

→ Message Digest (MD) MD5
→ Secure Hash function (SHA)
→ Whirlpool

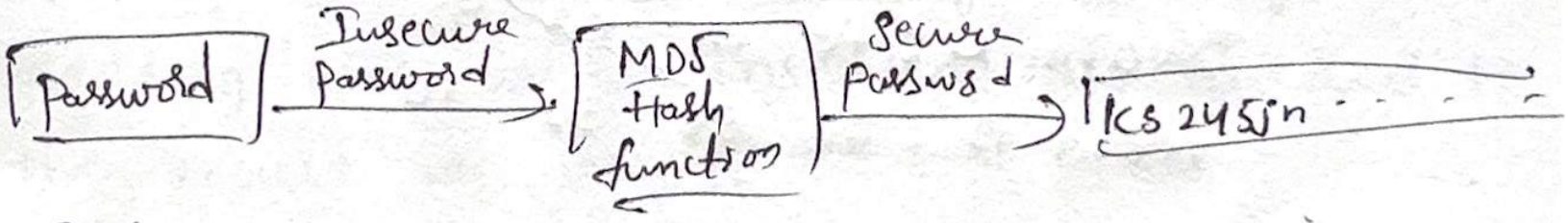
Message Digest 5

→ Cryptographic hash function algorithm that takes the message as input of any length and changes it into a fixed-length message of 16 bytes.

Output of MD5 alway – 128 bits (Digest size)

uses of MD5 algorithm

→ used for file authentication
→ security purpose
→ store our password in 128 bits format



Working of MD5

1) padding: original message + (padding) Extra bits so that total length is 64 bit less than exact multiple of 512

Example: Original msg = 1000 bits

512 × 1 = 512 bits
512 × 2 = 1024 bits
512 × 3 = 1536 bits – 64 – 1472

1536
64
1472

Working of the MD5 Algorithm

There are the following steps are performed to compute the message

Step 1 - Append padding bits -

The message is continued or padded in such a method that its total length in bits is congruent to 448 modulo 512. This operation, is continually implemented even if the message's length in bit is originally congruent to 448 modulo 512. 448 + 64 = 512, therefore the message is padded such that its length is now 64 bits less an integer multiple of 512.

Step 2 - Append length:

A 64 bit description of the length in bits of the original message M (before the padding bits, were inserted) is added to the result of step 1.

→ If the length of the original message is higher than 2^{64} = 184 467 440 73 709 551 616, therefore only the low order 64 bits of the length of message M are utlized.

→ Therefore, the field includes the length of the original message M modulo 2^{64}. These bits, are added as two 32 bit words and added low-order (least significant) word first.

→ The result of step 1 and step 2 is a message with a length in bits that is an integer multiple of 512 bits.

Step 3 - Initialize MD Buffer:

A 128-bit buffer can be used to hold intermediate and last result of the MD5 hash algorithm.

→ A four-word buffer (A, B, C, and D) can be used to evaluate the message digest.

→ Therefore, each A, B, C, D is a 32-bit register.

→ These registers are boot up to the following values in hexadecimal, low-order bytes first-

Word A : 01 23 45 67

Word B : 89 ab cd ef

Word C : fe dc ba 98

Word D : 76 54 32 10

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

A compression function includes 4 rounds of processing.

→ Each round creates an input the current 512-bit block being processed (Y_q) and the 128-bit buffer value ABCD and update the element of buffer.

→ It can describe 4 auxiliary functions that each create as input three 32-bit words and generate as output one 32-bit word

$$F(X, Y, Z) = XY \text{ v not } (X) Z$$

$$G(X, Y, Z) = XZ \text{ v } Y \text{ not } (Z)$$

$$H(X, Y, Z) = X \text{ xor } Y \text{ xor } Z$$

$$I(X, Y, Z) = Y \text{ xor } (X \text{ v not } (Z))$$

→ In each bit position, F acts as conditional: If X then Y else Z. The function F can have been represented using '+' instead of 'v' since XY and not X(Z) will never have 1's in the similias bit position

Step 5 - Output:

→ The message digest created an output including A, B, C, D.

→ The output from the final round is th 128-bit hash result of message digest it can be acquired after it can have incrementally processed all t 512-bit blocks of the message.

A	B	C	D.

Constant
. 32 bit Constant

1-16 each step - single Constant

X - part of plain text

Logical function - F G H I

Constants

p.t

16 blocks of 32 bit

512

A B C D 32

F , T, [1-16] X[]
16 steps

A B C D

G , T [17-32] X[]
16 steps

A B C D

H T [32--48] X[]
16 steps

A B C D.

I T[49-64] X[]
16 steps

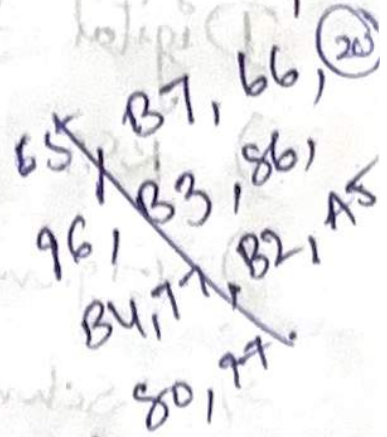
Addition modulo 32.

First Block

Message Digest

first block output given as second block output

A B C D.



$F(B,C,D) - (B \wedge C) \vee (\neg B \wedge D)$

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

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

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

$\vee$ - OR

$\neg$ - Not

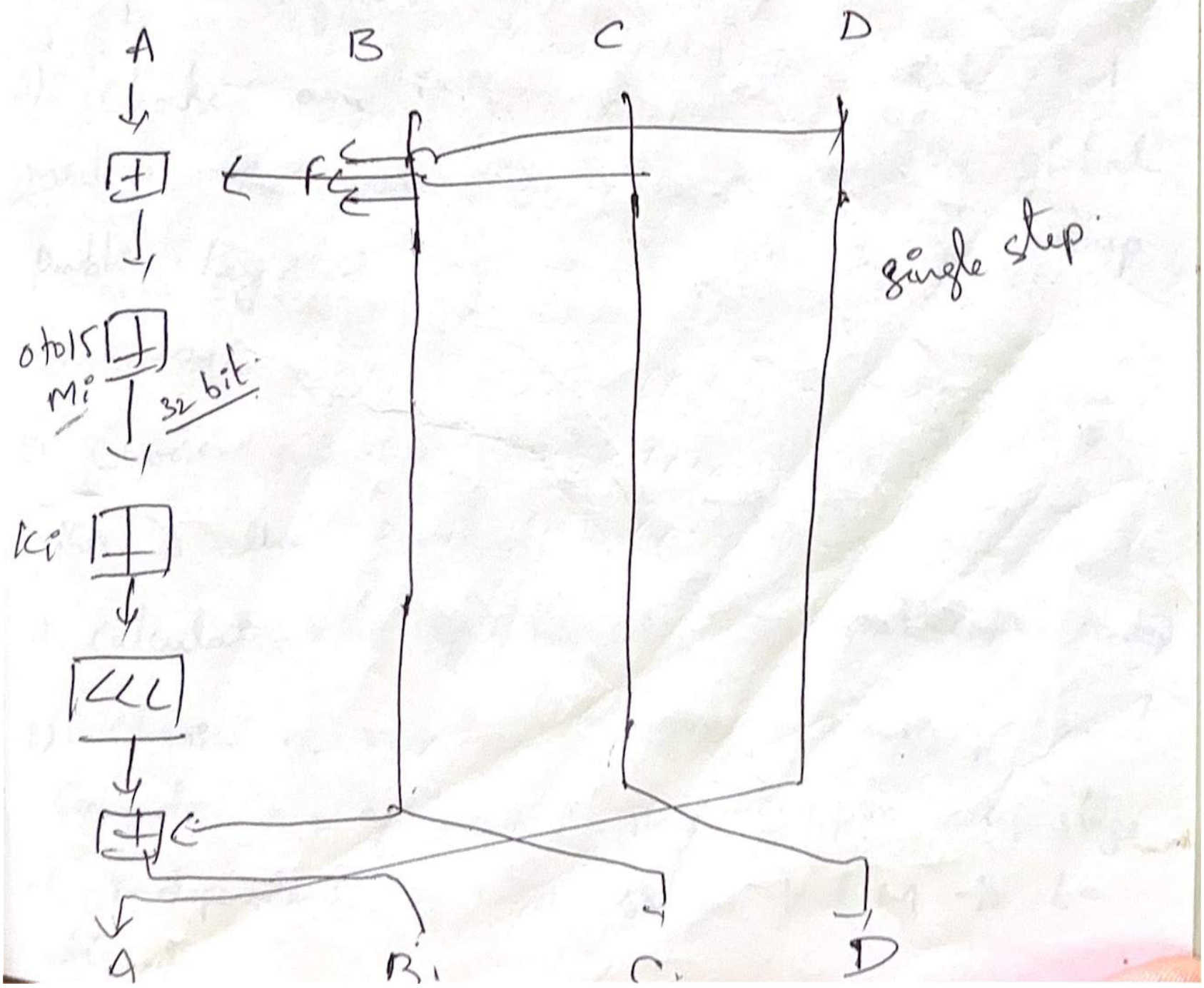
$\oplus$ - XOR

$\boxed{+}$ - Addition modulo 32

A ← B $\boxed{+}$ ((A $\boxed{+}$ Function (B,C,D) $\boxed{+}$ X[ ] $\boxed{+}$ T[i] <<<s

P.T. Constant

T[1] .. T[16]



Secure Hash algorithm. 512

Secure Hash algorithms (SHA) are cryptographic functions that transform data into a fixed length hash value.

O/P
↓
128 bit - SHA-1
256 bit - SHA 256
512 bit - SHA 512

512 bit - SHA 512
512 bit hash code.

$H(M) = h(512\ bit)$

SHA - 512

→ plain text Block size = 1024 bits
→ No. of Rounds / steps = 80
→ Each Round → Qword = 64 bit
↳ Generated from plain text
→ Each Round → Constant K
→ Buffers = 8 Buffers. (size = 64 bit)
↓
Store intermediate Results
Store o/p (hash code)

$\frac{512}{64} = 8$ buffers

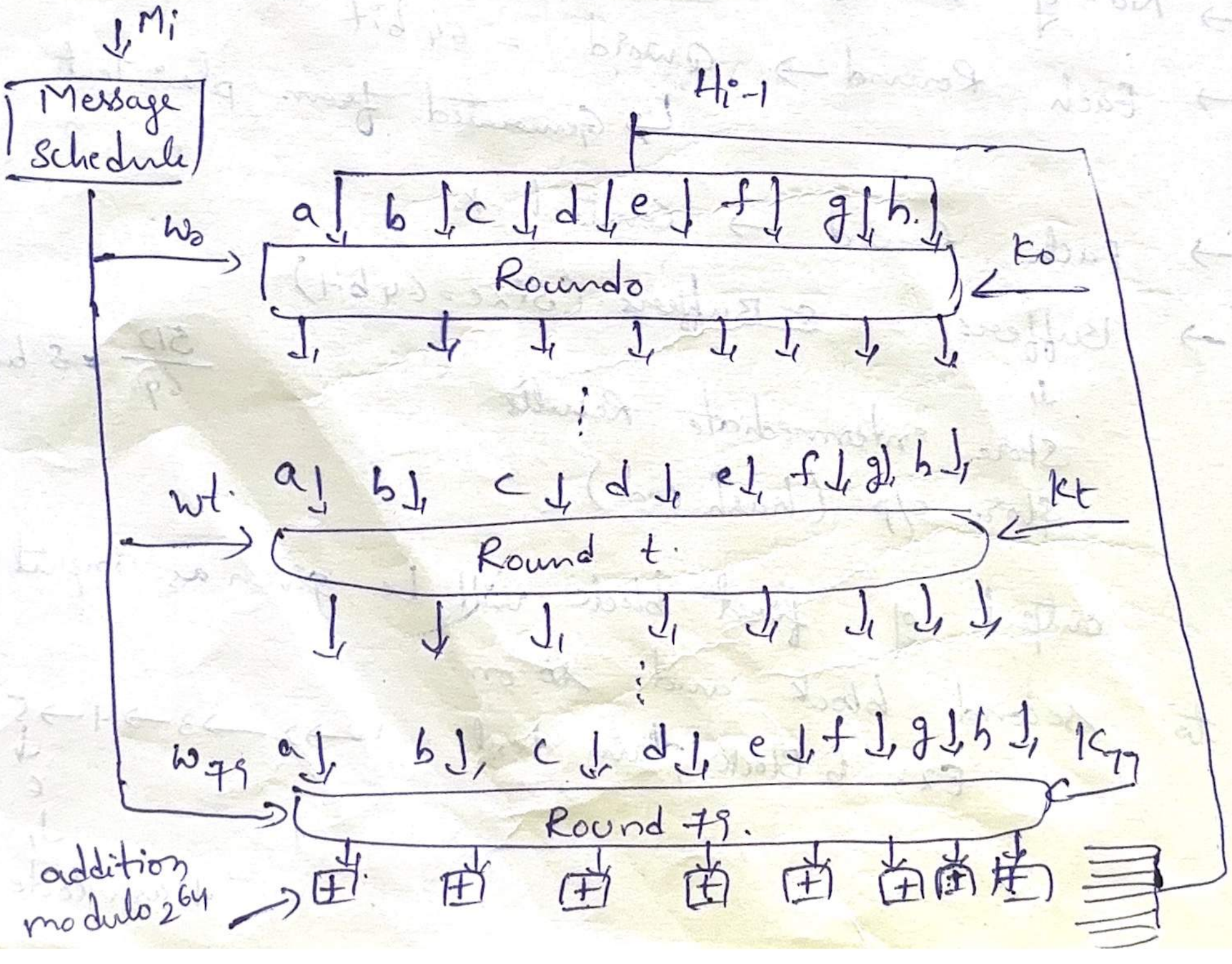
output of first block will be given as input to second block and so on

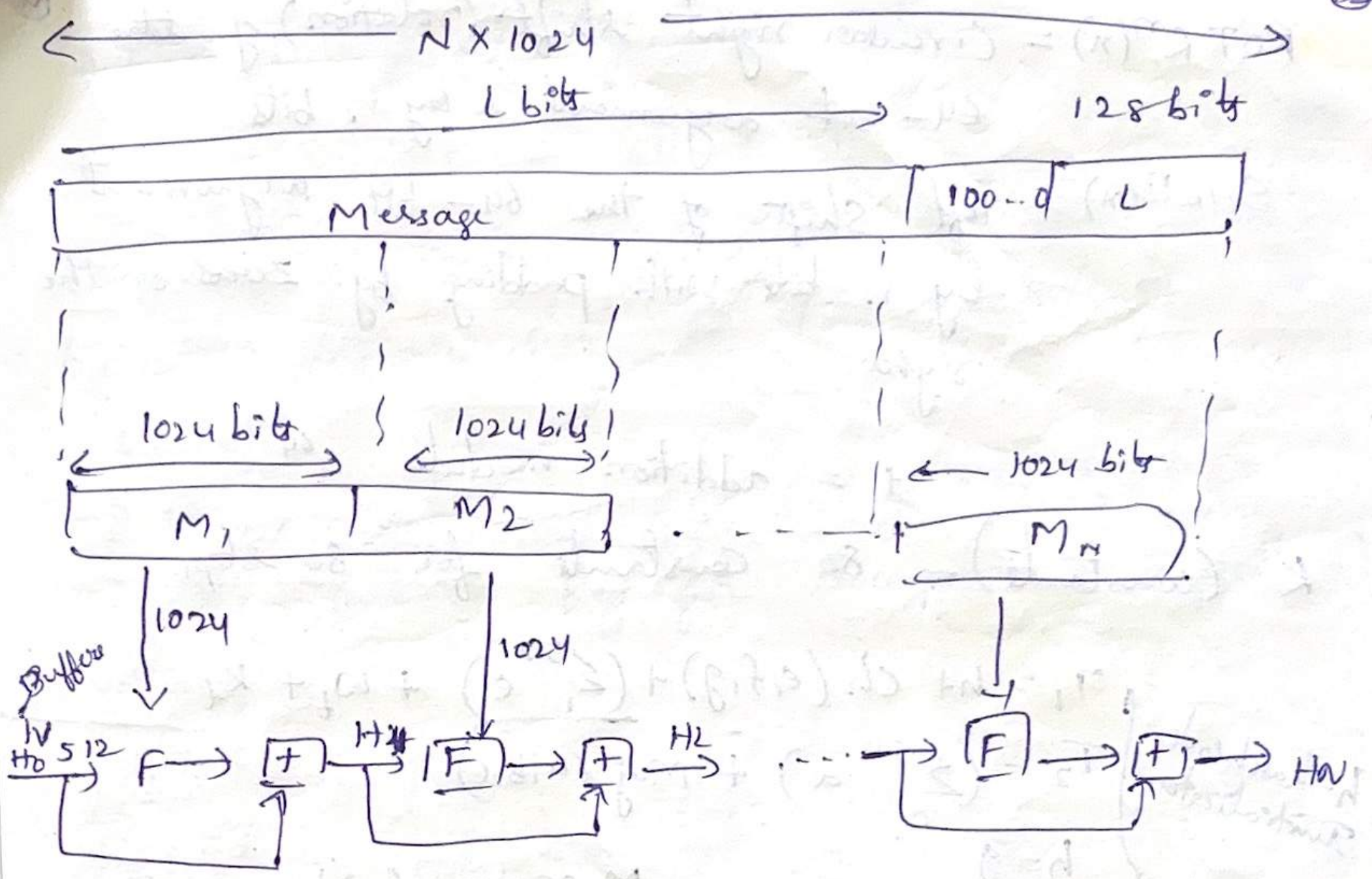
Ex: 6 Block plain text 1 → 2 → 3 → 4 → 5 → 6
↓
Hash code.

SHA - 512

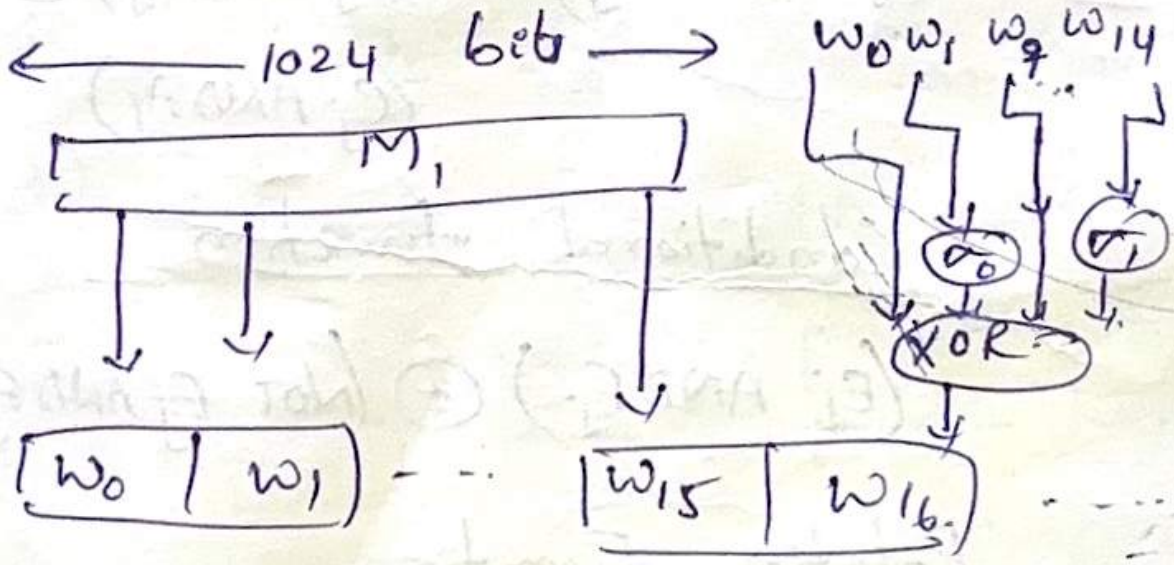
1) pad the bits 100... so that length of P.T is multiple of 1024 bits
2) Append 128 bit Representation of Original plain text such that length = Multiple of 1024 bits
3) Initialize the buffers (a, b, c, d, e, f, g, h) 64 bit in hexa decimal
4) process each block of plain text in 80 Rounds/steps
5) output in Buffers is Hash code. (512 bit)

process in Round function.





Word generation



$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16} \qquad (16 \text{ to } 79)$$

Where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$

$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$

$ROTR^n(x)$ = Circular right shift (rotation) of the 64-bit argument x by n bits

$SHR^n(x)$ = left shift of the 64-bit argument x by n bits with padding by zeros on the right

$+$ = addition modulo 2^{64}

K (constants) – 80 constants for 80 steps

h-hashcode Initialization

ch – Conditional function

e-buffer.

Maj – Majority.

each & every Round.

$T_1 = h + ch(e,f,g) + (\Sigma_1^{512} e) + w_t + k_t$

$T_2 = (\Sigma_0^{512} a) + Maj(a,b,c)$

$h = g$

$g = f$

$f = e$

$e = d + T_1$

$d = c$

$c = b$

$b = a$

$a = T_1 + T_2$

Majority function.

$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$

Conditional function

$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$

Rotation Function.

$\text{Rotate}(A) = RotR_{28}(A) \oplus RotR_{34}(A) \oplus RotR_{21}(A)$

$\text{Rotate}(E) = RotR_{28}(E) \oplus RotR_{34}(E) \oplus RotR_{21}(E)$

Digital signature schemes

1) RSA Digital signature scheme
2) Elgamal Digital signature scheme
3) Schnorr Digital signature scheme
4) Digital Signature standard
5) Elliptic curve Digital signature scheme

1) Elgamal Digital Signature Scheme

This scheme allows that a verifier can confirm the authenticity of a message m sent by the signer sent to him over an insecure channel.

→ uses private key for encryption (Signing)
→ uses publick key for decryption (verification)

Working

1) select a prime number q.
2) select a primitive root α of q.
3) Generate a random Integer (x_A)
$1 < x_A < q-1$
4) Compute $Y_A = (\alpha)^{x_A} \bmod q$
5) Generate keys for user A.
user A private key → x_A
user A public key → $\{q, \alpha, Y_A\}$.
6) Generate hash code (m) for the plain text (M
$m = H(M)$ $0 \leq m \leq q-1$
7) Generate a random Integer k
$1 \leq k \leq q-1$ and
$\gcd(k, q-1) = 1$

Now calculate S_1 and S_2

$$S_1 = \alpha^k \bmod q$$

$$S_2 = k^{-1}(m - X_A S_1) \bmod q-1$$

Now we will get signature pair (S_1, S_2)

Now, at user B's side user B can verify by calculating V_1 & V_2

$$V_1 = \alpha^m \bmod q.$$

$$V_2 = (Y_A)^{S_1} \cdot (S_1)^{S_2} \bmod q$$

If $V_1 = V_2 \rightarrow$ Signature valid

If $V_1 \neq V_2 \rightarrow$ Signature not valid

Let $q = 19$

$\alpha = 10$

Random Integer x_A

$1 < x_A < q-1$

$1 < x_A < 18$.

Let $x_A = 16$.

$V_1 = 16.$

$V_2 = 16.$

$Y_A = \alpha^{x_A} \bmod q$

$= (10)^{16} \bmod 19$ $\boxed{Y_A = 4}$

Keys for user A $X_A - 16$ — private key

$\{q, \alpha, Y_A\} - \{19, 10, 4\}$

public key.

Generate Hash code (m)

$m = H(M)$

$0 \leq m \leq q-1$

$0 \leq m \leq 18$

$m = 14$

$m = 4$

Generate K

$0 \leq K \leq q-1$ and $\gcd(K, q-1) = 1$

$0 \leq K \leq 18$ and $\gcd(K, 18) = 1$ $\boxed{K = 5}$

Calculate $S_1 = \alpha^K \bmod q \Rightarrow (10)^5 \bmod 19$, $S_1 = 100000 \bmod 19$

$\boxed{S_1 = 3}$

$S_2 = K^{-1}(m - X_A S_1) \bmod q-1$

$K^{-1} = K^{-1} \bmod q-1$

$5X ? = 1 \bmod 18.$

$K^{-1} = 11$

$S_2 = K^{-1}(m - X_A S_1) \bmod q-1$

$= 11(14 - 6X3) \bmod 18$

$= -374 \bmod 18 = 4$

$S_2 = 4.$

$(S_1, S_2) = (3, 4)$

Schnorr Digital Signature scheme

Schnorr signature is a digital signature produced by the schnorr signature algorithm that was described by Claus schnorr. It is a digital signature scheme known for its simplicity, is efficient and generates short signatures.

Working of Schnorr signature scheme.

Calculating first at the sender side:

1) Choose primes P and q, such that q is a prime factor of P-1

2) Choose an integer y such that $y^q = 1 \bmod p$. The value y, P, q comprises a global public key that can be common to a group of users.

3) Choose random integer s with $0 < s < q$, this is the sender's private key.

4) Calculate $v = y^{-s} \bmod P$. → public key (Sender)

5) Choose a random integer r $0 < r < q$ and Compute $x = y^{-r} \bmod P$. This preprocessing stage is independent of the simple text M to be signed

6) Attach the message M with x and hash the attached resultant to compute the value

$e = H(M||x)$ (attaching simple text M with x). H is the hash functions

7) E is the hashed value of the simple text M.

8) Compute $y = (r + s * e) \bmod q$

9) Here signature pair are (y, e)

Calculating at the receiver's side:

1) Compute $x' = \gamma^{y} * v^{e} \bmod p$ here

$v = \gamma^{-s} \bmod p$.

$\rightarrow \gamma^{y} * \gamma^{-se} \bmod p$

$\rightarrow \gamma^{-(y-se)} \bmod p$, $\{y = r + se\}$

$\rightarrow \gamma^{r} \bmod p$

2) we can see that $x' = x$

3) Also, verify $e = H(M||x)$

4) $H(M||x) = H(M||x')$

Digital Signature standard. (DSS)

→ This DSS uses secure hash algorithm & Digital signature Algorithm (DSA)

→ Let p be a prime number. Length of Prime no will be $2^{L-1} < p < 2^L$

→ Let q be another prime such that $(p-1) \bmod q = 0$.

→ Calculate $g = h^{(p-1)/q} \bmod p$.

where $1 < h < (p-1)$

L = 512 to 1024 bits multiple of 64

→ Choose random integer x such that $0 < x < q$. x is private key $d_A = x$

→ Calculate $y = g^x \bmod p$.

y is public key $\therefore C_A = y$.

q = 160 bit prime divisor of p-1

→ Choose random integer k $0 < k < q$ — secret key.

Signing Algorithm

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

$s = k^{-1}(H(m) + x \times r) \bmod q$.

Verification

$t = H(m)s^{-1} \bmod q$

$u = rs^{-1} \bmod q$

$v = (g^t y^u \bmod q) \bmod q$

$v = r \rightarrow$ Accepted

Example: $H(m) = 3$ $h = 2$ $p-1 \bmod q = 0$. 34

$6 \bmod 3 = 0$.

$\boxed{x = 5}$ → private

$p = 7$

$q = 3$

$k = 2$

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

$= 2^{(6)/3} \bmod 7$

$= 4 \bmod 7$

$g = 4$

$y = g^x \bmod p$

$= 4^5 \bmod 7$

$= 2$

↓ public

Signing Algorithm.

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

$= (4^2 \bmod 7) \bmod 3$

$= 2 \bmod 3$

$\boxed{r = 2}$

$s = k^{-1} (H(m) + x r) \bmod q$

$= 2^{-1} (3 + 10) \bmod 3$

$= 2^{-1} (13) \bmod 3$

$= 26 \bmod 3$

$s = 2$.

Verification Algorithm

$t = H(m)\, s^{-1} \bmod q$

$= 3 \times 2^{-1} \bmod 3$

$= 3 \times 2 \bmod 3$

$= 6 \bmod 3$

$t = 0$

$u = r\, s^{-1} \bmod q$

$= 2 \times 2^{-1} \bmod 3$

$= 2 \times 2 \bmod 3 = 4 \bmod 3$

$\boxed{u = 1}$

$v = (g^t \times y^u \bmod p) \bmod q$

$= (4^0 \times 2^1 \bmod 7) \bmod 3$

$= 2 \bmod 3$

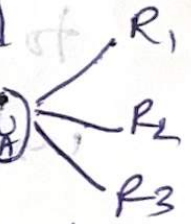
$= v = 2$.

$\boxed{V = r}$ → accepted.

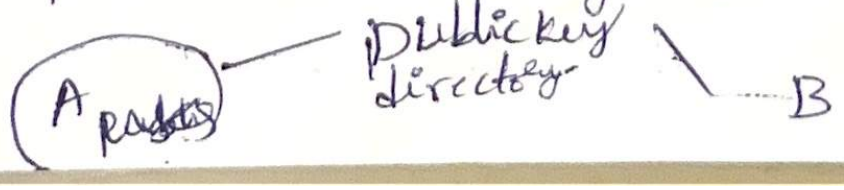
Key Management: Complex task to distribute the public & private keys between Sender and Receiver. If the key is known to the third party then the whole Security mechanism becomes worthless. So, there comes the need to secure the exchange of keys. Two aspects
- Distribution of public keys
- use of public-key Encryption to distribute secrets

Distribution of public key
- public announcement
- public available directory
- public key authority
- public key certificates

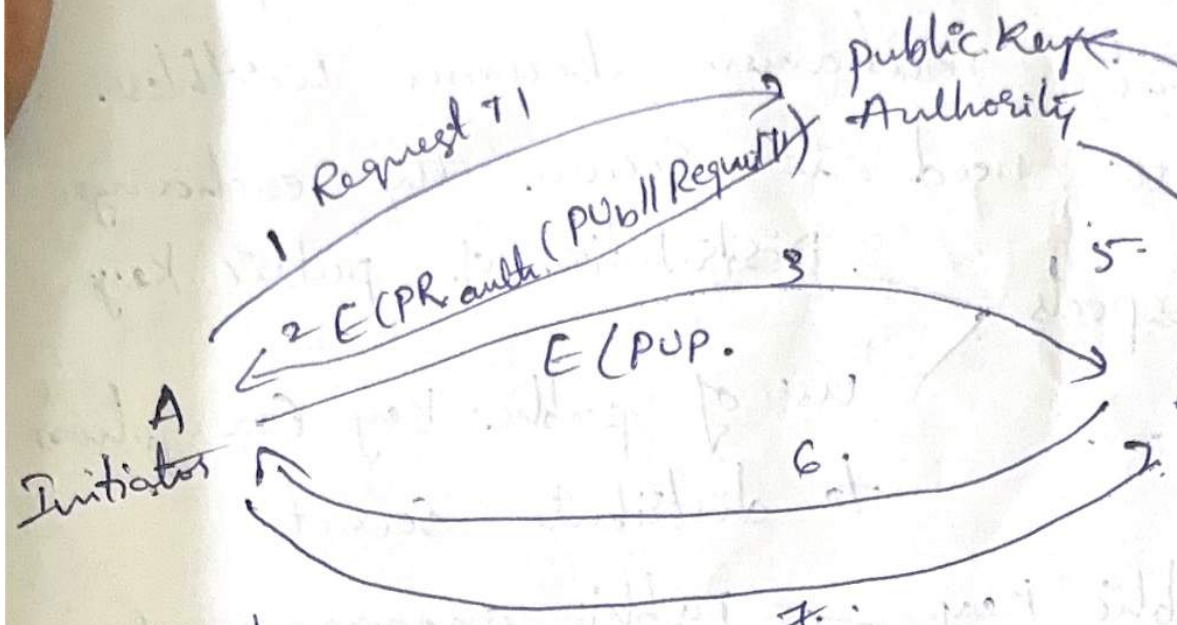
public announcement
public key - known to everyone
Some user pretends to be user A and send a public key to another participant. until user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.



Public Available Directory: Authority maintains directory (with a name, public key) entry for each participant.
→ Participant registers a public key with directory authority.
→ participant can replace public key at any time
→ participants can access the directory electronically.
For this secure authenticated communication from the authority to the participant is mandatory.
→ more secure than public annoncements. but if attacker succeed in computing directory private key authority he could pass out public keys to any participant



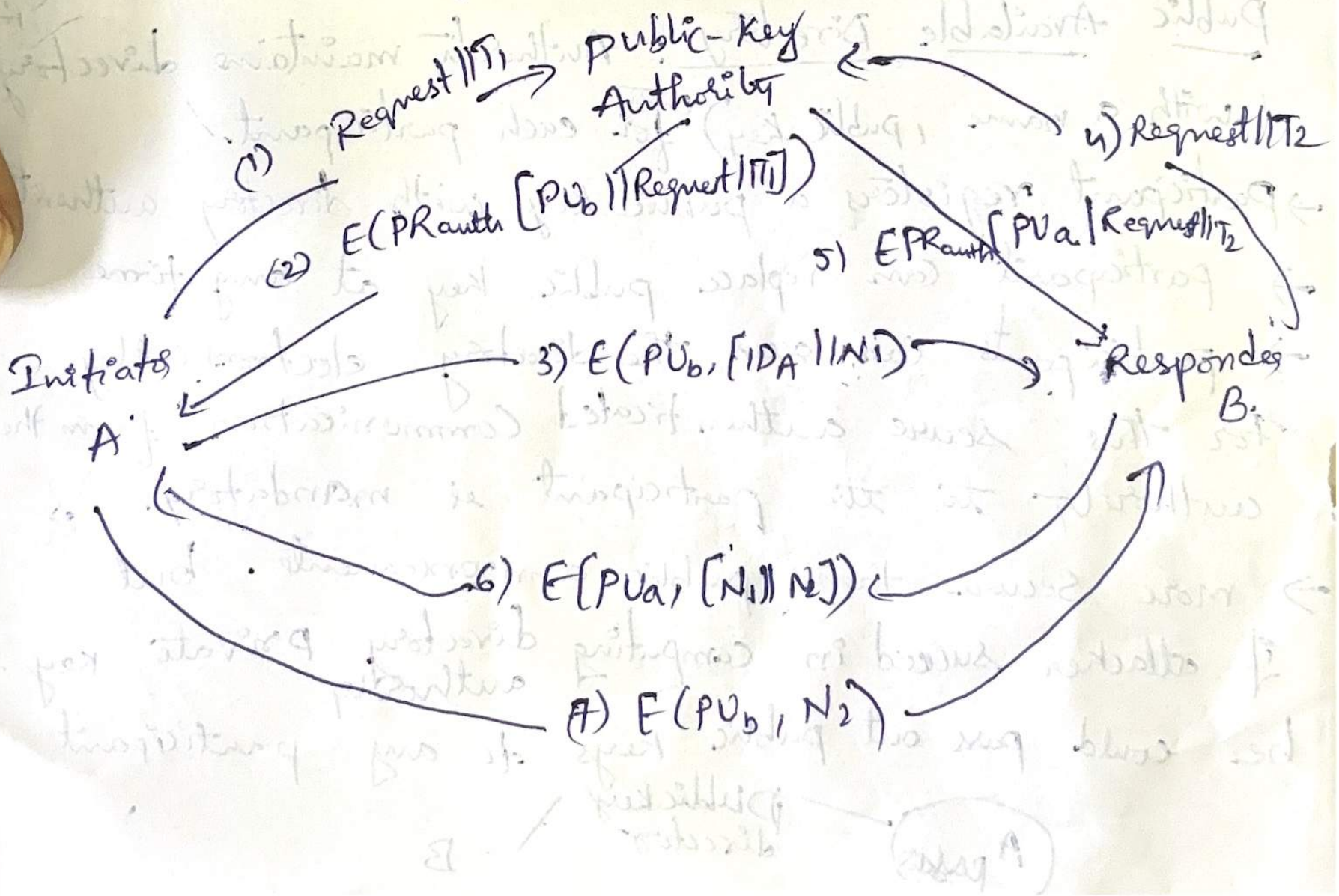
public key authority : similar to the directory but improves security by providing tighter control over the distribution of public keys from the directory



1- sends Timestap mess to Authority Containing public key of B. request for.

2- Authority responds with a message encrypted using Authority private key; decrypt using authority public key (User A): Assured that received from authority B's public key used to encrypt message destined for B.

3- A's stores B's public key and encrypts a message to B contains identifier of A and a nonce (N1) used to identify this transaction uniquely.



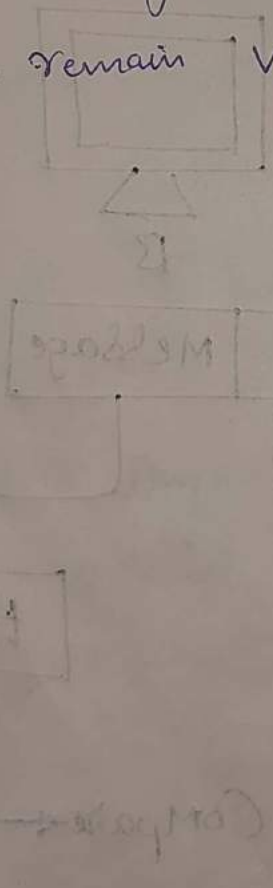
Public-key Certificates

This time authority provides a certificate which binds an identity to the public key) to allow key exchange without real-time acess to the public authority each time. The certificate is accompained by some other info such as period of validity, rights of use etc. All of this content is signed by the private key of the certificate authority and it can be verified by anyone possessing the authority's public key. First sender and receiver both request CA for a certificate which contains a public key and other information and then they can exchange these certificates and can start Communication.

Digital Signature Schemes:

A Digital signature scheme will have two components, a private signing algorithm which permits a user to securely sign a message and a public verification algorithm which permits anyone to verify that the signature is authentic.

The signing algorithm needs to bind a signature to a message in such a way that the signature cannot be pulled out and used to sign another document or have the original message modified and the signature remain valid.



RSA Digital Signature Scheme

The RSA idea can also be used to sign and verify a msg. In this case, it is called the RSA digital signature scheme.

→ The digital signature scheme changes the roles of the private and public keys.

Note: Firstly the private & public keys of the sender are used (not of the receiver)

Second → the sender uses his/her own private key to sign the document and receiver uses the senders public key to verify it.

Key Generation.

(Same as RSA algorithm)

Eg: Alice chooses 2 prime no's 'p' and 'q'

Calculates $n = p * q$

$\phi(n) = (p-1) * (q-1)$

She then chooses e (the public Exponent) and calculates d (the private Exponent)

such that $ed \equiv 1 \bmod \phi(n)$

Alice keeps d and publically announces n & e

In Normal RSA

Encryption $C = M^e \bmod n$ (e, n) public key

Here, for signing

$S = M^d \bmod n$ uses her private key (or) we can say she uses her private Exponent "d" to create her signature

Now, this signature "S" & Message "M" is sent to Bob.

Verifying

Bob receives "M" and "S"

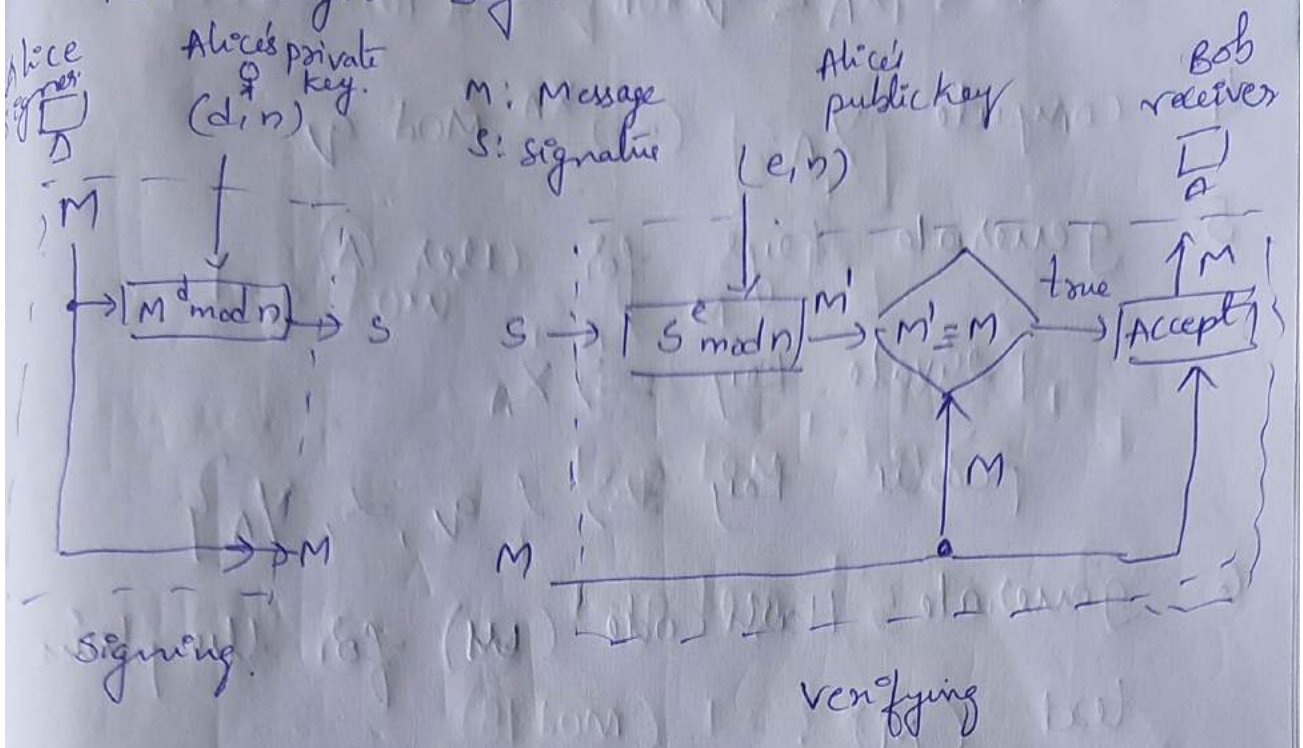
Bob applies Alice's public exponent(e) to the signature to create a copy of message

$M' = S^e \bmod n$

Bob Compares the value of m' with m

→ If both values Congruent Bob accepts the message else not

RSA digital signature scheme



Elgamal Digital Signature Scheme

→ Digital Signature scheme

Encryption → public key

Decryption → private key

The global elements of Elgamal are prime number q & a which is primitive root of q

* Working:

1. Select a prime number (q).
2. Select a primitive root (α) of q.
3. generate a random integer (X_A).

$1 < X_A < q-1$

4. Compute $Y_A = (\alpha)^{X_A} \text{ Mod } q$

5. generate keys for user (A)

private key $\Rightarrow X_A$

public key $\Rightarrow \{q, \alpha, Y_A\}$

6. generate Hash code (M) for the plain text (M)

$m = H(M) \quad 0 \leq m \leq q-1$

7. generate a random integer (k)

$1 \leq k \leq q-1$ and $\gcd(k, q-1) = 1$

8. Now calculate S_1 and S_2 $\quad S_1 = \alpha^k \text{ Mod } q$

$S_2 = k^{-1}(M - X_A S_1) \text{ Mod } q$

9. Now we got the signature pair (S_1, S_2)

Now, at user B's side,

Calculate V_1 and V_2

$V_1 = \alpha^M \text{ mod } q$

$V_2 = (Y_A)^{S_1} \ldots (S_1)^{S_2} \text{ Mod } q$

If $V_1 = V_2$

$\Rightarrow$ Signature is valid

if $V_1 \neq V_2$

$\Rightarrow$ Not valid.

Example:

let $q = 19$ and $\alpha = 10$

Now, Random integer X_A $(1 < X_A < q-1)$

$$1 < X_A < 18$$

$$\Rightarrow X_A = 16$$

$$Y_A = \alpha^{X_A} \bmod q = (10)^{16} \bmod 19$$

$$= 4$$

$$\boxed{Y_A = 4}$$

Keys:

private key $\Rightarrow X_A \Rightarrow 16$

public key $\Rightarrow \{q, \alpha, Y_A\} \Rightarrow (19, 10, 4)$

Now, generate Hash code (m)

$$m = H(M) \quad 0 \leq M \leq q-1$$

$$0 \leq M \leq 18$$

$$\therefore M = 14$$

generate (k), $0 \leq k \leq q-1$ and $(k, q-1) = 1$

$$0 \leq k \leq 18 \text{ and } \gcd(k, 18) = 1$$

$$\therefore \boxed{k = 5}$$

Calculate $S_1 = \alpha^k \bmod q = (10)^5 \bmod 19$

$= 3$

$\boxed{S_1 = 3}$

$S_2 = k^{-1} (M - X_A S_1) \bmod q-1$

$k^{-1} \Rightarrow k^{-1} \bmod q-1$

$= 5^{-1} \bmod q-1$

$= 5 \times ? = 1 \pmod{18}$

$\therefore \boxed{k^{-1} = 11}$

$S_2 = k^{-1} (M - X_A S_1) \bmod q-1$

$= 11 (14 - 16 \times 3) \bmod 18$

$= -374 \bmod 18 = 4$

$\boxed{S_2 = 4}$

$\therefore S_1, S_2 = (3, 4)$

At B' send:

$V_1 = \alpha^M \bmod q$

$= 10^{14} \bmod 19 = 16$

$\boxed{V_1 = 16}$

$V_2 = (Y_A)^{S_1} (S_1)^{S_2} \bmod q$

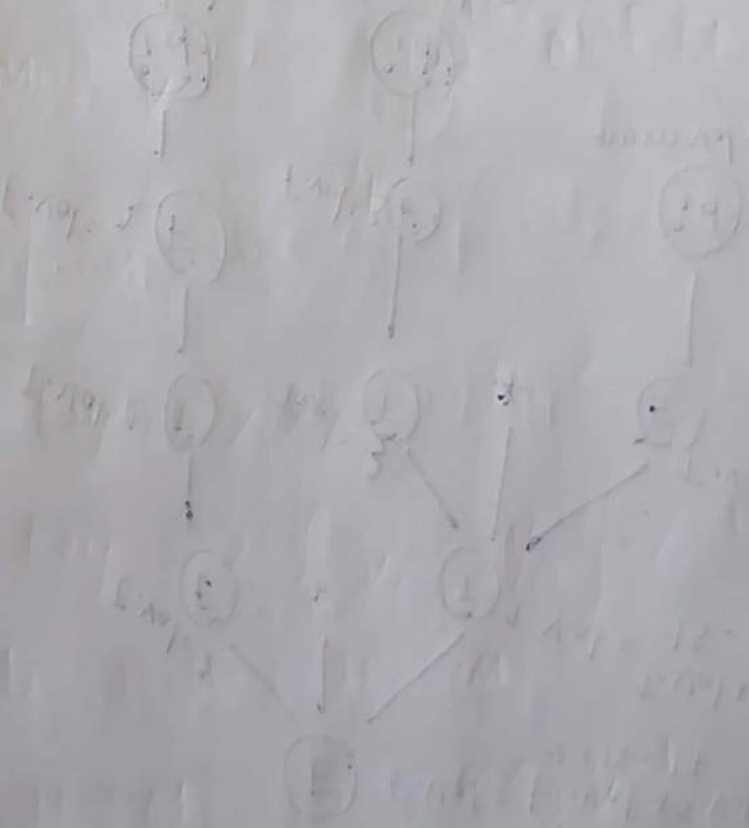
$= 4^3 \times 3^4 \bmod 19$

$= 5184 \bmod 19 = 16$

$$\boxed{V_2 = 16}$$

Now $V_1 = V_2$

$\therefore$ signature is valid



Key management

⇒ The main aim of key management is to generate a secret key between two parties and store.

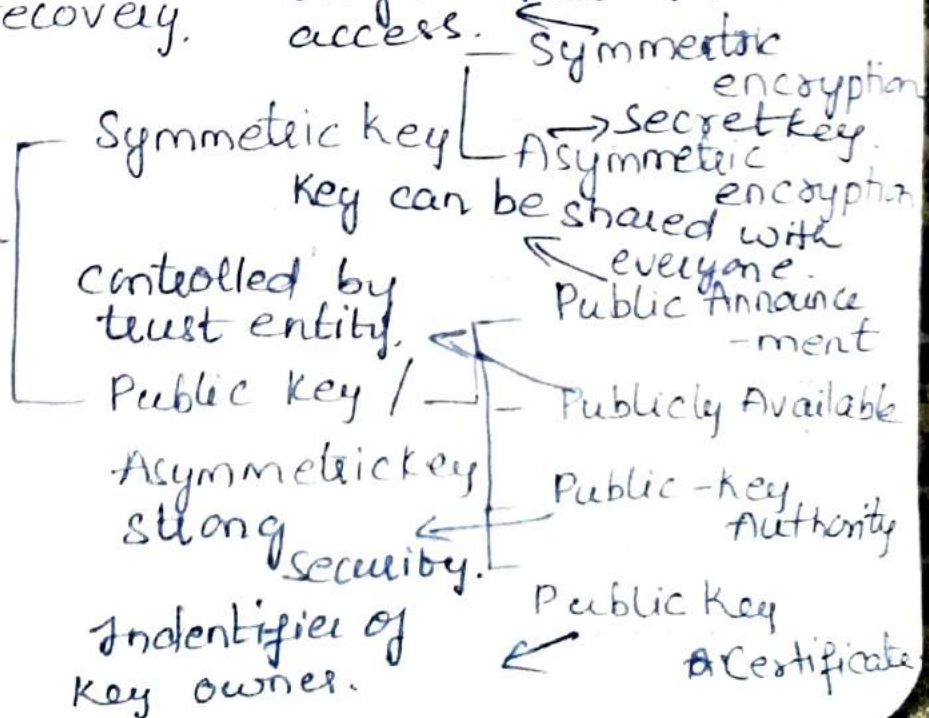
1. Symmetric
2. Asymmetric

1. public key
2. Private key

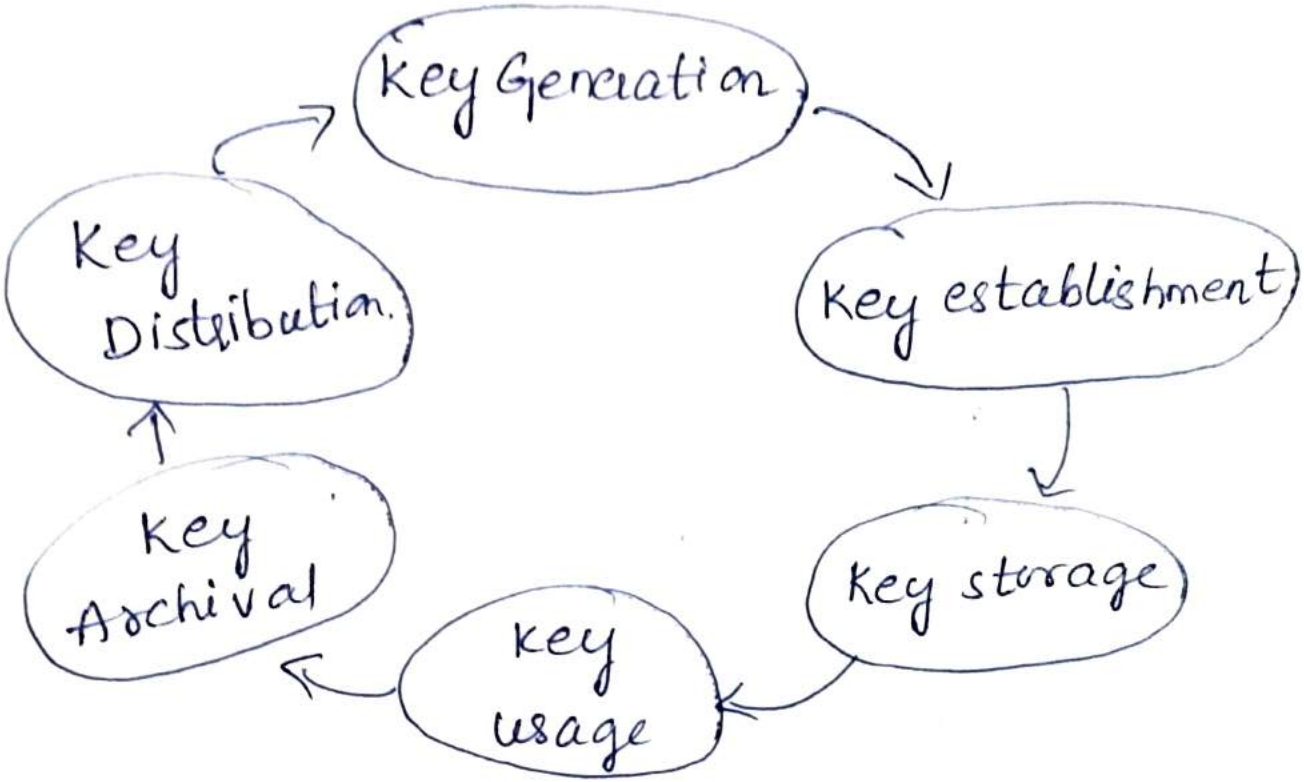
→ It plays important role in securing cryptographic goals like confidentiality → Sender is known to receiver → No change in data authentication, data integrity and digital signatures → confidentiality, authentication, digital integrity.

⇒ Two different keys are used for encryption and decryption.

⇒ Basic purpose of key management is key generation, key distribution, controlling the use of keys, updating, destruction of keys and key backup / recovery.



Key management structure



Protocols used in key management

1. Kerberos authentication protocol
2. X-509 authentication protocol

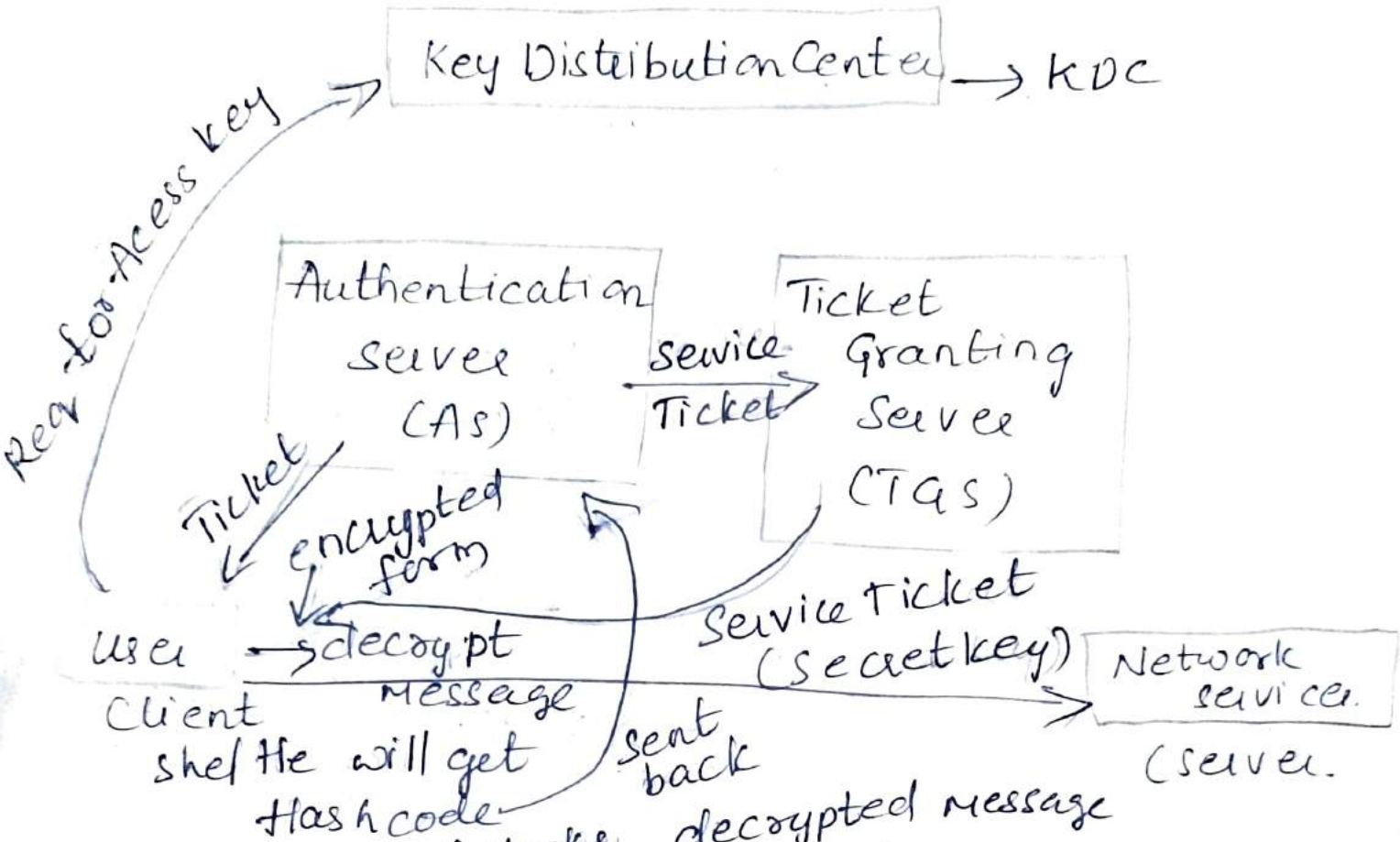
1. Kerberos

— It is a network authentical protocol
If you want to use a network it will provide all the services if you are certified user.

— Client server Architecture

— Symmetrickey
Same key used for both encryption & decryption.

— Requires a third party for key
To provide keys it needs third party
KDC key distribution center (trusted party)
database of all the secret keys.



2. X-509

⇒ Digital signature certificate accepted Internationally.

⇒ doesnot generate any keys but provides a way to access public keys

↓ It provides certificate for the user.

There are several elements in X509 certificate it has (3) versions

Versions → 1, 2, 3.

Serial Number → certificate no.

Signature Algorithm Identifier

Issuer Name → Name of user

validity Period → date & Time

Subject Name

Public key Information → key is issued for encryption & decryption

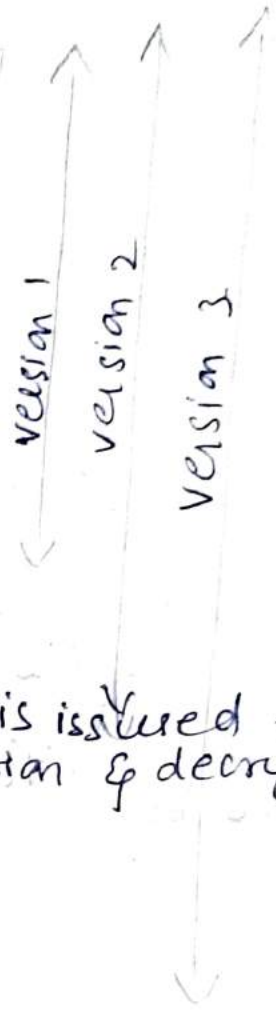
Issue ID ← Issue Unique ID } version 2

subject ID ← Subject Unique ID

Extension. → Version 3

↓

To add any description (optional.



version - versions - 1, 2, 3

It is a public key informative.

version 1 is upto version - public key information.

version 2 includes - Issue unique ID
Subject Unique ID

version 3 includes - Extension.

⇒ Signature algorithm -> which algorithm is used ~~used~~ by user.

⇒ Subject Name :- To whom you are issuing the certificate.