Publickey cryptography principles.

→ public key Encryption structure.

public key cryptography is Asymmetric, involving the use of 2 separate keys: one is public & one is private.

symmetric-key	Asymmetric-key.
(i) A scheme which uses a single secret key for both encryption & decryption process.	(i) A scheme uses 2 different keys, one to encrypt the data (public key) and another one to decrypt the data (private key).
(ii) It is based on sharing secrecy.	(ii) It is based on personal secrecy.
(iii) Symbols are permuted or substituted.	(iii) It is based on applying mathematical functions to numbers.
(iv) Symmetric encryption algorithms: DES, 3DES & AES.	(iv) Asymmetric encryption algorithms: RSA, Diffie-Hellman.

→ Public key encryption scheme has 6 ingredients.

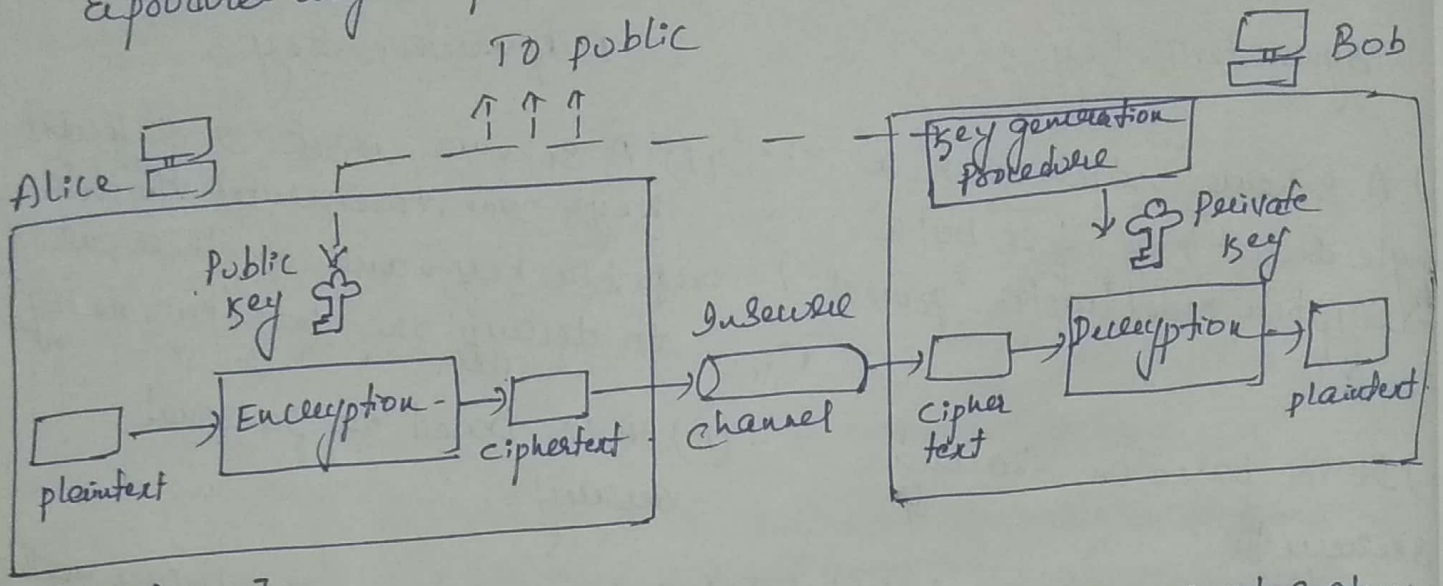
(i) plaintext: This is the readable message or data that is fed into an algorithm as an i/p.

(ii) Encryption algorithm: Performs various transformations on the plaintext.

(iii) Public & private key: This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.

(iv) Ciphertext : This is the scrambled message produced as o/p. It depends on the plaintext & key.
For a given message & different keys will produce & different ciphertexts.

(v) Decryption algorithm: It accepts ciphertext & the matching key & produces original plaintext.



[fig-1.]

→ The diagram illustrates the asymmetric-key cryptosystem.

(i) First, it emphasizes the asymmetric nature of cryptosystem. Bob needs to create 2 keys: one private & one public.
Bob is responsible for distributing the public key to the comm-unity. This can be done through public-key distribution channel. Although this channel is not required to provide secrecy, it must provide authentication & integrity.

(ii) Second, Asymmetric key cryptography means Bob & Alice can't use the same set of keys for 2-way communication. Diagram shows how Alice can use Bob's public key to send encrypted messages to Bob. If Bob wants to respond, Alice needs to establish her own private & public keys.

(iii) Third, Asymmetric-key cryptography ~~needs~~ means that Bob needs only one private key to receive all correspondance from anyone in the community, but Alice needs a public key

to communicate with n entities in the community, one public key for each entity. In other words, Alice needs a ring of public keys.

ciphertext $C = f(K_{public}, P)$ → for encryption.

plaintext $P = g(K_{private}, C)$ → for Decryption.

with this approach, all participants have access to public-keys. & private keys are generated locally by each participants & therefore need never be distributed.

To discriminate blw the two, we refer to the key used in symmetric encryption as a secret key.
The keys used for asymmetric encryption are referred to as the public key & private key.

Applications for public-key cryptosystems.

Use of public-key cryptosystems are catogarised into 3 categories.

→ Encryption/Decryption: The Sender encrypts a message with the recipient's public key.

→ Digital Signature: The Sender signs a message with its private key. Signing is achieved by cryptographic algorithm applied to the message or to a small block of data that is function of the message.

→ Key exchange: Two sides cooperate to exchange a session key.

Some algorithms are suitable for all 3 applications whereas others can be used for one or two of these applications.

Algorithm	Encryption/ Decryption	Digital Signature	Key exchange.
RSA.	yes	yes	yes.
Diffie-Hellman	NO	NO	yes.
DSS (Digital Signature Standard)	NO	yes	NO.
Elliptic curve.	yes	yes	yes.

Requirement for public key cryptography.

→ cryptosystem illustrated in [fig. 1] depends on a cryptographic algorithm. Conditions for that algorithms are

* It is computationally easy for partyB to generate a pair of (public & private) keys.
* It is computationally easy for a sender A, to know the public key of other party & message to be encrypted, M, to generate the corresponding ciphertext.

$$C = E(PU_b, M).$$

* It is computationally easy for the receiver B to decrypt the resulting ciphertext using private key to recover the original message.

$M = D(PR_b, C)$

* It is computationally infeasible for an opponent, to determine the private key (PR_b) by knowing the public key of the same party (PU_b).
* It is computationally infeasible for an opponent, to recover original message, M. by knowing PU_b & ciphertext.

Public key cryptographic algorithm.

The 2 most widely used public-key algorithms are RSA & Diffie-Hellman.

RSA public-key encryption algorithm

one of the first public-key scheme was developed in 1977 by Ron Rivest, Adi Shamir & Len Adleman at MIT

RSA is a block cipher in which plaintext & ciphertext are integers blw 0 & n-1 for some n.

Encryption & Decryption can be processed for some plaintext(M) & ciphertext (C) as follows:

$C = M^e \bmod n$.

$M = C^d \bmod n$.

Both Sender & receiver must know the values of n & e, and only the receiver knows the value of d.

For this algorithm to be satisfactory for Public-key encryption, the following requirements must be met.

→ It is possible to find values of e, d, n such that

$M^{ed} \bmod n = M, \forall M < n$.

→ It is relatively easy to calculate M^e & C^d for all values of $M < n$.

→ It is infeasible to determine d given e & n.

first 2 requirements are easily met. 3rd requirement can be met for large values of e & n.

RSA Algorithm.

Key generation

(i) Begin by selecting 2 prime nos such as p & q. p & q both should be of prime & $p \neq q$.

(ii) calculate $n = p \times q$.

(iii) calculate Euler's totien $\phi(n) = (p-1)(q-1)$ of n. which is the no of positive integers less than n & relatively prime to n.

(iv) Select an integer e i.e. relatively prime to $\phi(n)$
i.e. $\gcd(e, \phi(n)) = 1$.

(v) calculate d as the multiplicative inverse of e, modulo $\phi(n)$.
$de \bmod \phi(n) = 1$.

Suppose user A has published its public key & that user B wishes to send a message M to A. Then B calculates
$C = M^e \bmod n$.
& transmits C.

On receipt of this ciphertext, user A decrypts by calculating $M = C^d \pmod n$.

Encryption.

plaintext $M < n$.

ciphertext $C = M^e \bmod n$.

Decryption.

ciphertext C

plaintext $M = C^d \bmod n$.

[RSA is one of the cipher suites used in Transport layer Security, which is used by HTTPS, so, RSA may be used in any connection to an HTTPS: URL]

[Asymmetric cryptography either RSA or ECC is usually used in lot of s/w for verifying the s/w updates are from the original updates]

→ $p = 7$ $q = 11$.

→ $n = pq = 7 \times 11 = 77$.

→ $\phi(n) = (p-1)(q-1) = 6 \times 10 = 60$.

→ Select e. $\gcd(13, 60) = 1$.

$e = 13$.

→ Select d, $de \bmod 60 = 1$ ——— by inspection method.

$d\ 13 \bmod 60 = 1$

$37 \times 13 \bmod 60 = 1$

$481 \bmod 60 = 1$

→ consider a message $M = 5$.

ciphertext $C = M^e \bmod n$

$= 5^{13} \bmod 77$

$= (5^5 \bmod 77 \times 5^5 \bmod 77 \times 5^3 \bmod 77) \bmod 77$

$= 3125 \bmod 77 \times 3125 \bmod 77 \times 125 \bmod 77$

$= (45 \times 45 \times 48) \bmod 77 = 97200 \bmod 77 = 26.$

plaintext $M = C^d \bmod 77$

$= 26^{37} \bmod 77$

$= [26^5 \bmod 77 \times 26^5 \bmod 77 \times 26^5 \bmod 77 \times 26^5 \bmod 77 \times 26^5 \bmod 77$
$\times 26^5 \bmod 77 \times 26^5 \bmod 77 \times 26^2 \bmod 77] \bmod 77.$

$= (45 \times 45 \times 45 \times 45 \times 45 \times 45 \times 45 \times 60) \bmod 77 = 5.$

$2^5 \bmod 77 = 1188137^6 \bmod 77$

$= 45$

$2^{2^6} \bmod 77 = 676 \bmod 77$

$= 60.$

There are 2 possible approaches to defeating RSA algo

(i) Brute-force approach: Try all possible private keys. Thus, the larger the no of bits in e & d, the more secure the algorithm.

(ii) Tasks of factoring n into its 2 prime factors. For a large n with large prime factors, factoring is hard problem.

Currently, a 1024-bit key size (about 300 decimal digits) is considered strong enough for virtually all applications.

Diffie-Hellman key exchange

Purpose of this algorithm is to enable 2 users to exchange a secret key securely, then can be used for subsequent encryption of messages.

Summary of Algorithm

There are 2 public key known nos: A prime no q & an integer α i.e. primitive root of q.

Suppose the users A & B wish to exchange a key. User A selects a random integer $x_A < q$

Definition: Algorithm used to exchange a secret key btw users. For this exchange we use asymmetric cryptosystem.

→ Not an encryption algorithm.
→ Exchange secret / symmetric key btw Sender & receiver.
→ Uses Asymmetric encryption by using public key & private key.

Algorithm

→ Assume a prime no. q
→ Select an integer 'α', such that 'α' must be a primitive root of 'q' & $\alpha < q$.

primitive root of prime no p as one whose powers generate all the integers from 1 to p-1.
ie. if 'a' is a primitive root of the prime no p, then the nos

$a \bmod p, a^2 \bmod p \ldots\ldots a^{p-1} \bmod p$ are distinct & consists of integers from 1 to p-1 in some permutation.

→ ~~Assume~~ Select a private key of user A. (X_A)

$X_A < q$.

→ Calculate public key of user A (Y_A)

$$Y_A = \alpha^{X_A} \bmod q.$$

→ Select a private key of user B. (X_B)

$X_B < q$

Calculate public key of user B (Y_B)

$$y_B = \alpha^{x_B} \bmod q.$$

→ Generate a secret key by user A.

$$k = (y_B)^{x_A} \bmod q.$$

→ Generate a secret key by user B.

$$k = (y_A)^{x_B} \bmod q.$$

The above calculation produced same result. That result is exchanged as a secret value among them.

Since x_A & x_B are private, an adversary only has the following ingredients to work with: q, α, y_A & y_B.
Thus, adversary forced to use a discrete logarithm to determine the key.
To determine the private key of user B, an adversary must compute.

$$x_B = d\log_{\alpha,q}(y_B).$$

→ Let me consider $q = 11$.
→ α → primitive root of q.

mod 11

Power →	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	5	10	9	7	3	6	1
3	3	9	5	4	1	3	9	5	4	1
4										
5										
6										
7										
8										
9										

↑ not

→ 1, 2, 3 ..... 10.
⇒ not distinct

$\alpha = 2$.

→ Select private key X_A, such that $X_A < q$.

$X_A = 8$.

calculate $Y_A = \alpha^{X_A} \bmod q$.

$= 2^8 \bmod 11$

$= 256 \bmod 11 = 3$.

$Y_A = 3$.

→ Select private key X_B, such that $X_B < q$.

$X_B = 4$.

calculate $Y_B = \alpha^{X_B} \bmod q$.

$= 2^4 \bmod 11 = 16 \bmod 11 = 5$. $Y_B = 5$

→ Secret key by user A.

$K = (Y_B)^{X_A} \bmod q$.

$= (5)^8 \bmod 11 = 390625 \bmod 11 = (4)$.

→ Secret key by user B.

$K = (Y_A)^{X_B} \bmod q$.

$= (3)^4 \bmod 11 = 81 \bmod 11 = (4)$.

2) $q = 353$.

$\alpha = 3$.

A & B select a secret key as $X_A = 97$, $X_B = 233$ respectively

$X_A = 97$.

$Y_A = \alpha^{X_A} \bmod 353$.

$= (3)^{97} \bmod 353$.

$= [3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{10} \bmod 353 \times 3^{7} \bmod 353] \bmod 353$.

$3^{10} \bmod 353 = 59049 \bmod 353 = 98$.

$3^{7} \bmod 353 = 2187 \bmod 353 = 69$.

$= [(98 \times 98 \times 98) \bmod 353 \times (98 \times 98 \times 98) \bmod 353 \times (98 \times 98 \times 98) \bmod 353 \times 69 \bmod 353] \bmod 353$.

$= 941192 \bmod 353$

$= (94 \times 94 \times 94 \times 69) \bmod 353$

$= 57310296 \bmod 353 = 40$.

$X_B = 233$.

$Y_B = (3)^{233} \bmod 353$.

$= 248$.

secret key @ user A.

$K = (Y_B)^{X_A} \bmod q$.

$= (248)^{97} \bmod 353 = 160$.

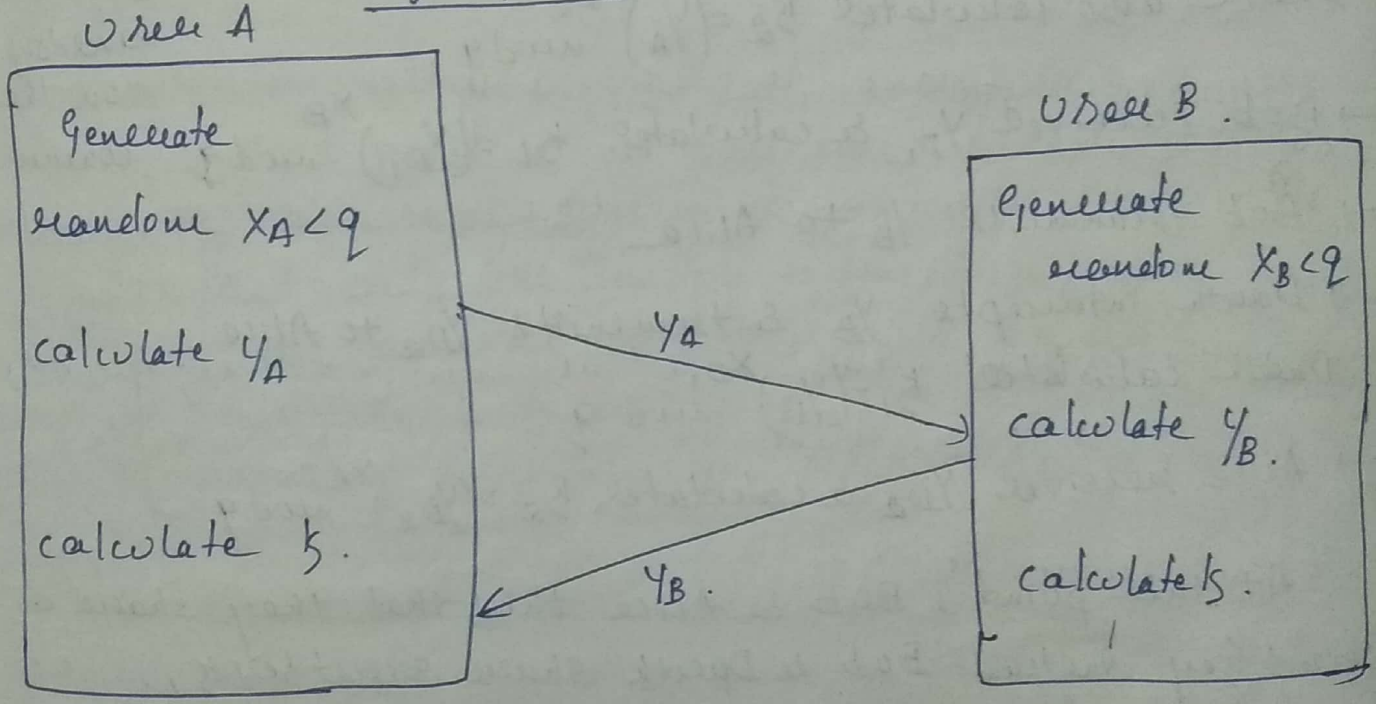
secret key @ user B.

$K = (Y_A)^{X_B} \bmod q$.

$= (40)^{233} \bmod 353$.

$= 160$.

Simple protocol



Suppose, that user A wishes to set up a connection with user B & uses a secret key to encrypt messages on that connection.

(i) User A generate one time private key X_A, calculate Y_A & send that to user B.

(ii) User B responds by generating a private key X_B, calculating Y_B & sending Y_B to user A.

Now, Both the users calculate secret key.

This simple protocol is insecure against Man-in-the middle attack.

The attack proceeds as follows.

→ Darth prepares for the attack by generating 2 random private keys X_{D_1} & X_{D_2} & then computes the corresponding public key Y_{D_1} & Y_{D_2}.

→ Alice transmits Y_A to Bob.

→ Darth intercepts Y_A & transmits Y_{D_1} to Bob.
Darth also calculates $K_2 = (Y_A)^{X_{D_2}} \bmod q$.

→ Bob receives Y_{D_1} & calculates $K_1 = (Y_{D_1})^{X_B} \bmod q$

→ Bob transmits Y_B to Alice

→ Darth intercepts Y_B & transmits Y_{D_2} to Alice.
Darth calculates $K_1 = (Y_B)^{X_{D_1}} \bmod q$.

→ Alice receives Y_{D_2} & calculates $K_2 = (Y_{D_2})^{X_A} \bmod q$

Darth simply eavesdrop on the communication.

At this point, Bob & Alice think that they share a secret key. Instead Bob & Darth share secret key K_1.
Alice & Darth share secret key K_2.

All future communication b/w them is as follows.

→ Alice sends an encrypted message $M: E(K_2, M)$

→ Darth intercepts the encrypted message & decrypts it to recover M.

→ Darth sends Bob $E(K_1, M)$. ~~And so forth~~

Darth wants to modify the message.

The key exchange protocol is vulnerable to such an attack because it doesn't authenticate the participants.

Message Authentication.

Approaches to message authentication.

Protection against active attacks is known as "message authentication".

It is a procedure that allows communicating parties to verify that received messages are authentic.

The 2 important aspects to verify are

* The contents of the messages haven't been altered.
* The source is authentic.

→ Authentication using conventional encryption.

It is possible to perform authentication simply by the use of symmetric encryption. If we assume that only the sender & receiver share a key then only the genuine sender would be able to encrypt a message successfully for the other participant, provided receiver can recognise a valid message.

Furthermore, ~~the receiver is assured~~ if the message includes an error detection code, & a timestamp (sequence no.), the receiver assured that - no alterations have been made, the sequencing is proper & message has not been delayed beyond the expected for network transit.

→ Message authentication without message encryption.

In this section, we examine several approaches to message authentication that do not rely on encryption. In this approach, an authentication tag is generated & appended to each message for transmission.

In this approach message is not encrypted, so message confidentiality is not provided. It is possible to combine authentication & confidentiality in a single algorithm by encrypting a message plus its authentication tag.

There are 3 situations in which message authentication without confidentiality is preferable.

Situation 1: There are no of applications in which the same message is broadcast to a no of destinations. Two examples are notification to users that the network is unavailable now & an alarm signal in a control center. It is cheaper & more reliable to have only one destination responsible for monitoring authenticity. Thus, message must be broadcast in plaintext with an associated message authentication tag. The responsible system performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.

Situation 2: Another possible scenario is an exchange in which one side has heavy load & can't afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis with messages being chosen at random for checking.

Situation 3: Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it everytime.

However, if a message authentication tag were attached to the program, it could be checked whenever assurance is required of the integrity of the program.

Message Authentication code.

In cryptography, a message authentication code sometimes known as a tag, is a short piece of information used to authenticate a message - in other words, to confirm that the message came from the stated sender & has not been changed.

"Authentication technique involves the use of a secret key to generate a small blocks of data, known as a message authentication code", which is appended to the message.

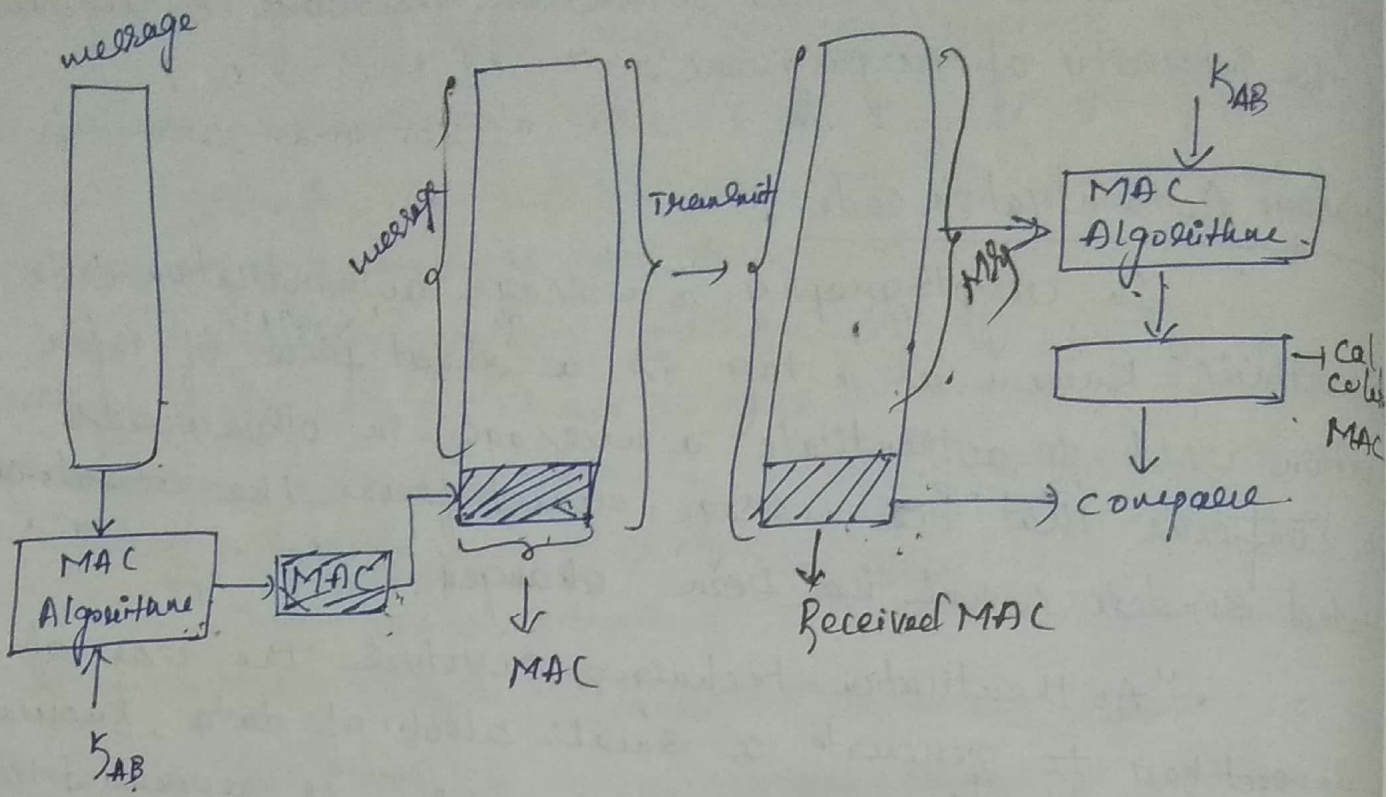
This technique assumes that 2 communicating parties say A & B, share a common secret key K_{AB}.

when 'A' has a message to send to 'B', it calculates the message Authentication code as a function of the message & the key: $MAC_M = F(K_{AB}, M)$.

The message plus code are transmitted to the intended recipient.

The recipient performs same calculation on the received message, using the same secret key, to generate a new message authentication code.

The received code is compared with the calculated code. [illustrated in the below diagram].



If we assume that only the receiver & Sender ~~the Sender~~ know the identity of the secret key, & if the received code matches the calculated code, then the following statements apply.

→ The receiver is assured that message has not been altered. If an attacker alters the message but doesn't alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker can't alter the code correspond to alteration in the message.

→ The receiver is assured that the message is from the alleged Sender. Because no one else knows the secret key, no one else could prepare a message with a proper code.

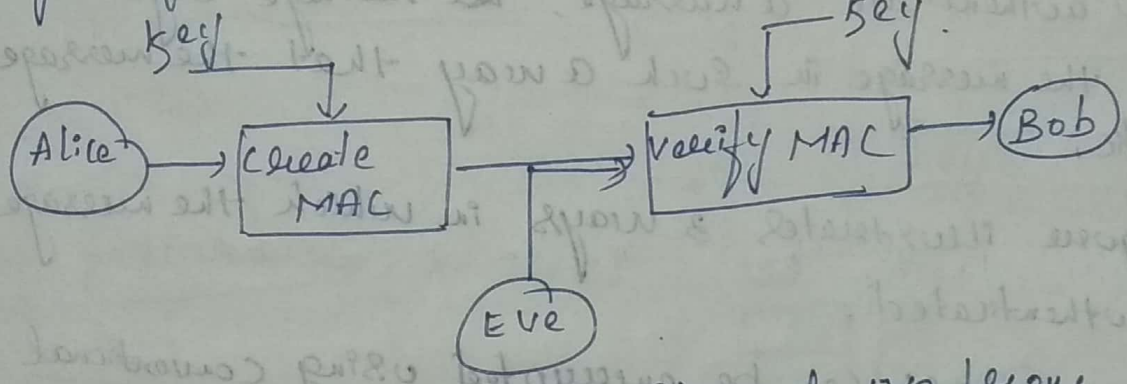
→ If the message includes a sequence number, then the receiver can be assured of the proper sequence, because an attacker can't successfully alter the seq no.

Number of algorithms could be used to generate the code. NIST specification recommends the use of DES. DES is used to generate an encrypted version of the message, and the last no of bits of ciphertext are used as the code.

The process just described is similar to encryption. one difference is that the authentication algorithm.

~~one-way hash function~~.

MAC ~~is created~~ is what you get from symmetric cryptography.



MAC is used to prevent EVE from creating anew message & inserting it instead of Alice's message.

one way hash function

A hash function which takes an i/p one message & returns a fixed-size alphanumeric string. The string is called "hash value" or "message digest" or "digital fingerprint" or "digest" or "checksum".

Hash functions.

A hash function maps a variable-length message into a fixed-length hash value or message digest.

An alternative to message authentication code is "one-way hash function".

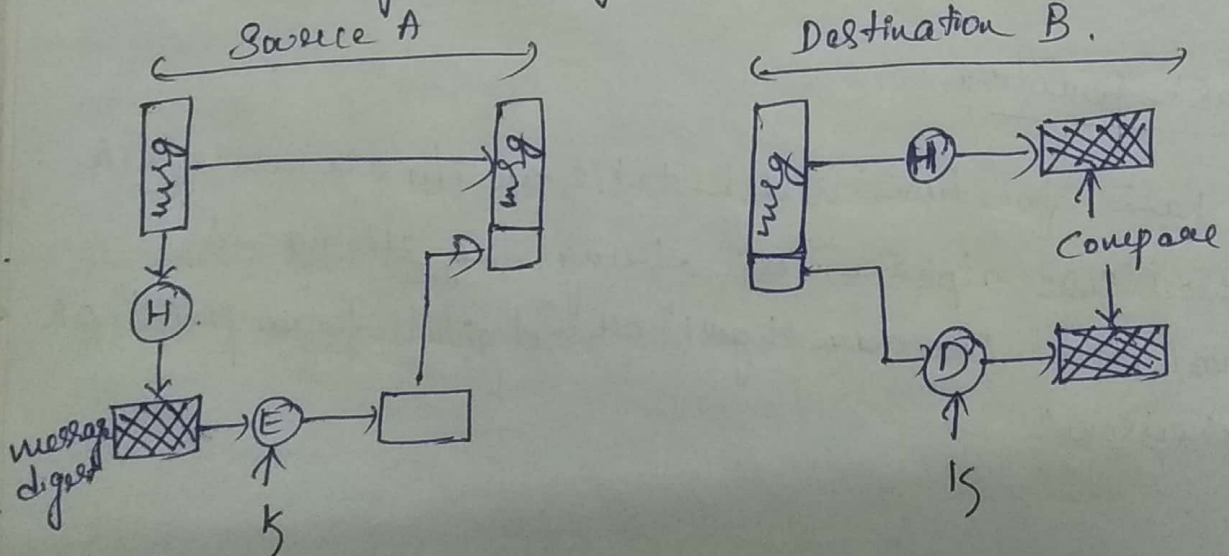
With message authentication code, a hash function accepts a variable-size message M as i/p & produces fixed-size message digest H(M) as o/p.

Unlike the MAC, a hash function doesn't take a secret key as i/p.

To authenticate a message, ~~the~~ a message digest is sent with the message in such a way that the message is authentic.

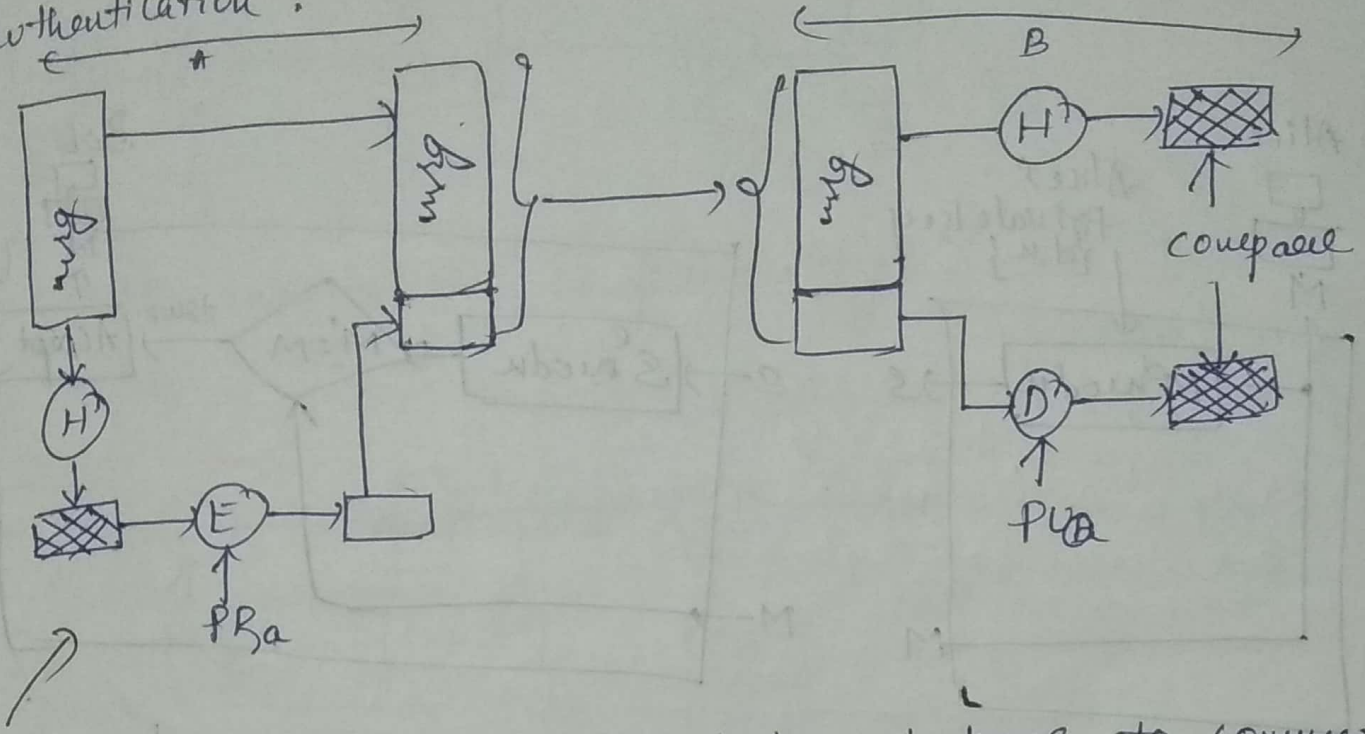
Figure illustrates 3 ways in which the message can be authenticated.

A message digest can be encrypted using conventional encryption, if it is assumed that only the sender & receiver share the encryption key, then authentication is assured.

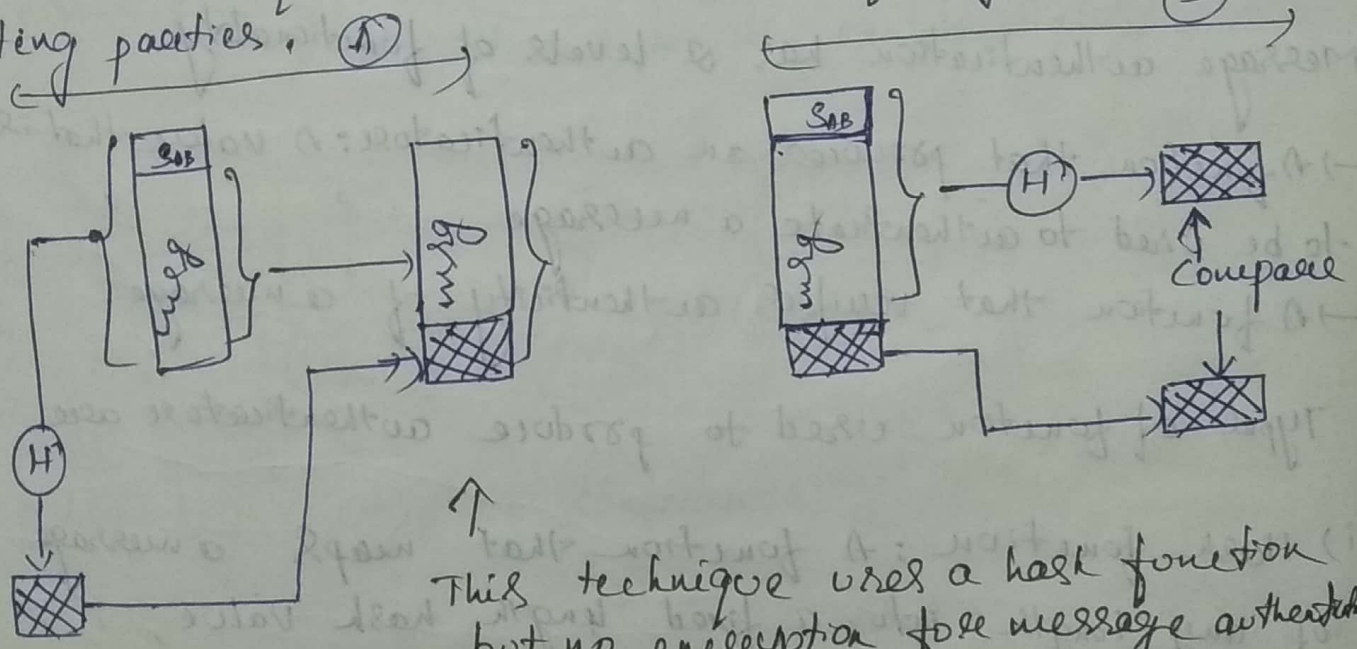


The message digest can be encrypted using public key encryption. The public key approach has 2 advantages.

(i) It provide digital signatures as well as message authentication.



(ii) It doesn't require the distribution of keys to communicating parties.



This technique uses a hash function but no encryption for message authentication

→ $MD_M = H(S_{AB} || M)$.

→ Sends $(M || MD_M) \Longrightarrow$ to B.

→ Receiver possesses S_{AB}, it can recompute $H(S_{AB} || M)$ & verify that with MD_M.

This technique is called "HMAC".

Secure Hash function.

One-way hash function or secure hash function is important not only in message authentication but in digital signatures.

→ Discuss requirements for a secure hash function.

→ Most important hash function.

Hash function requirements.

The main purpose of hash function is to produce a "fingerprint" of a file, message or other block of data.

To be useful for message authentication, a hash function H must have the following properties.

(i) H can be applied to a block of data of any size.

(ii) H produces fixed length output.

(iii) H(x) is relatively easy to compute for any given x, making both hardware & software implementations practical.

(iv) For any given code h, it is computationally infeasible to find x such that H(x) = h.
A hash function with this property is referred as "one-way" or "preimage resistant".

(v) For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x). A hash function with this property is referred as "second preimage resistant".

(vi) For any pair (x, y) such that H(x) = H(y), it is computationally infeasible to find that pair. This property is called "collision resistant".

The first 3 properties are requirements for the practical application of hash function to message authenti

Fourth property is one-way property: It is easy to generate a code given a message, but virtually impossible to generate a message given a code.

The 5th property guarantees that it is impossible to find an alternative message with the same hash value as a given message.

6th property protects against ~~birth~~ birthday attacks. ~~Details of~~ these attacks reduces strength of the hash function.

SHA (Secure hash function)

SHA was developed by National Institute of standards and technology (NIST)

SHA-0 was the first secure hash function. Revised version of this was referred as SHA-1.

SHA-1 produces a hash value of 160 bits.

Again NIST produced a revised version, that defined 3 newer versions of SHA with a hash value lengths of 256, 384 and 512 bits known as SHA-256, SHA-384 & SHA-512 respectively. collectively, these hash algorithms are known as "SHA-2".

In this section, we provide a description of SHA-512.

* This algorithm takes a message with a maximum length of $< 2^{128}$ bits as i/p & produces 512-bit message digest as o/p.
* Input is processed in 1024 bit blocks.
* Figure illustrates the overall processing of a message to produce a digest.
* The processing consists of following steps.

Step 1: Append padding bits

The message is padded so that its length is congruent to 896 modulo 1024

Length $\equiv$ 896 modulo 1024.

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

Thus, the no of padding bits is in the range of

1 to 1024.

The padding consists of single 1 bit followed by the necessary no of 0's bits.

minimum & maximum no of padding bits that can be added to a message are.

minimum length of padding is 0 & it happens when

$|M| + |P| + 128 = 0 \bmod 1024$.

$|P| = (-M - 128) \bmod 1024$ is 0.

maximum length of padding is 1023 & it happens when

$|P| = -M - 128 = 1023$.

i.e. $|M| = 897 \bmod 1024$.

Consider a message of length 2590.

$2590 + 128 = 2718$.

$2718 \bmod 1024 = 670$.

$896 - 670 = 226$

$226 + 128 = 354$.

Step 2: Append length.

A block of 128 bits is appended to a message. This block is treated as 128 bit integer & contains length of the original message before padding.

The outcome of first 2 steps yields a message that is an integer multiple of 1024 bits in length.

Step 3: Initialize hash buffer.

A 512 bit buffer is used to hold intermediate & final results of the hash function.

The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h).

These registers are initialized to the following 64-bit integers.

a = 6A09E667F3BCC908.

b = BB67AE8584CAA73B.

⋮

h = 5BE0CD19137E2179.

These values are calculated from the first 8 prime nos (2, 3, 5, 7, 11, 13, 17 & 19).

Each value is the fraction part of the square root of the corresponding prime nos after converting to binary & keeping only the first 64 bits.

For Ex: The 8th prime no is 19, $\sqrt{19} = 4.358898944$.

Converting the no to binary with only 64 bits in the fraction part, we get.

$(4.5BE0CD19137E2179)_{16}$.

Step 4: process message in 1024 bit blocks:

The heart of the algorithm is the module that consists of 80 rounds.

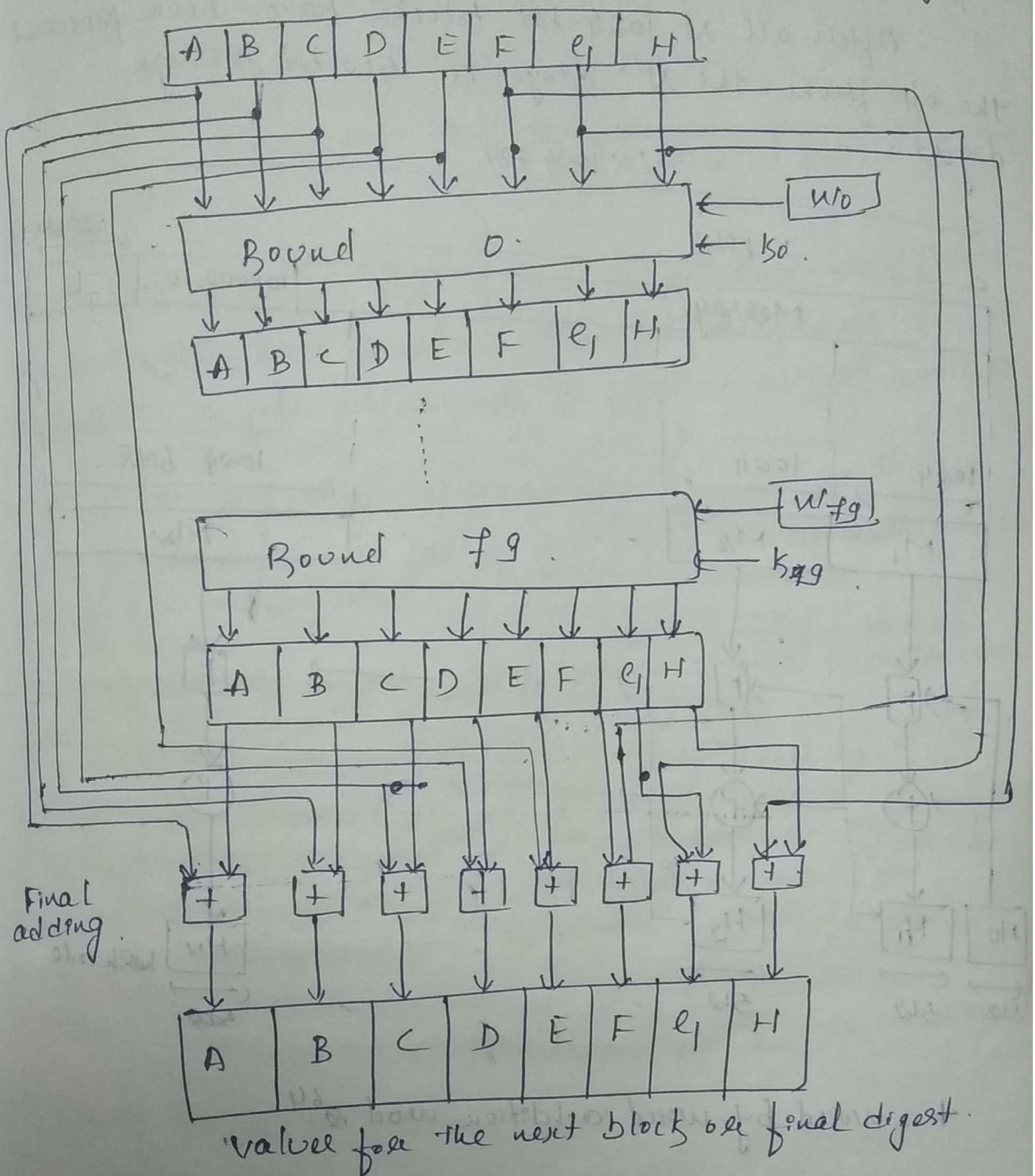
In each round, the contents of eight previous buffers, one word from the expanded block (W_i) & one 64-bit constant (K_i) are mixed together & then operated onto create a new set of eight buffers.

At the beginning of processing, the values of the 8 buffers are saved into 8 temporary variables.

At the end of processing, these values are added to the values created from Step 3.

We call the last operation as "final adding".

Result of the previous blocks or initial digest.

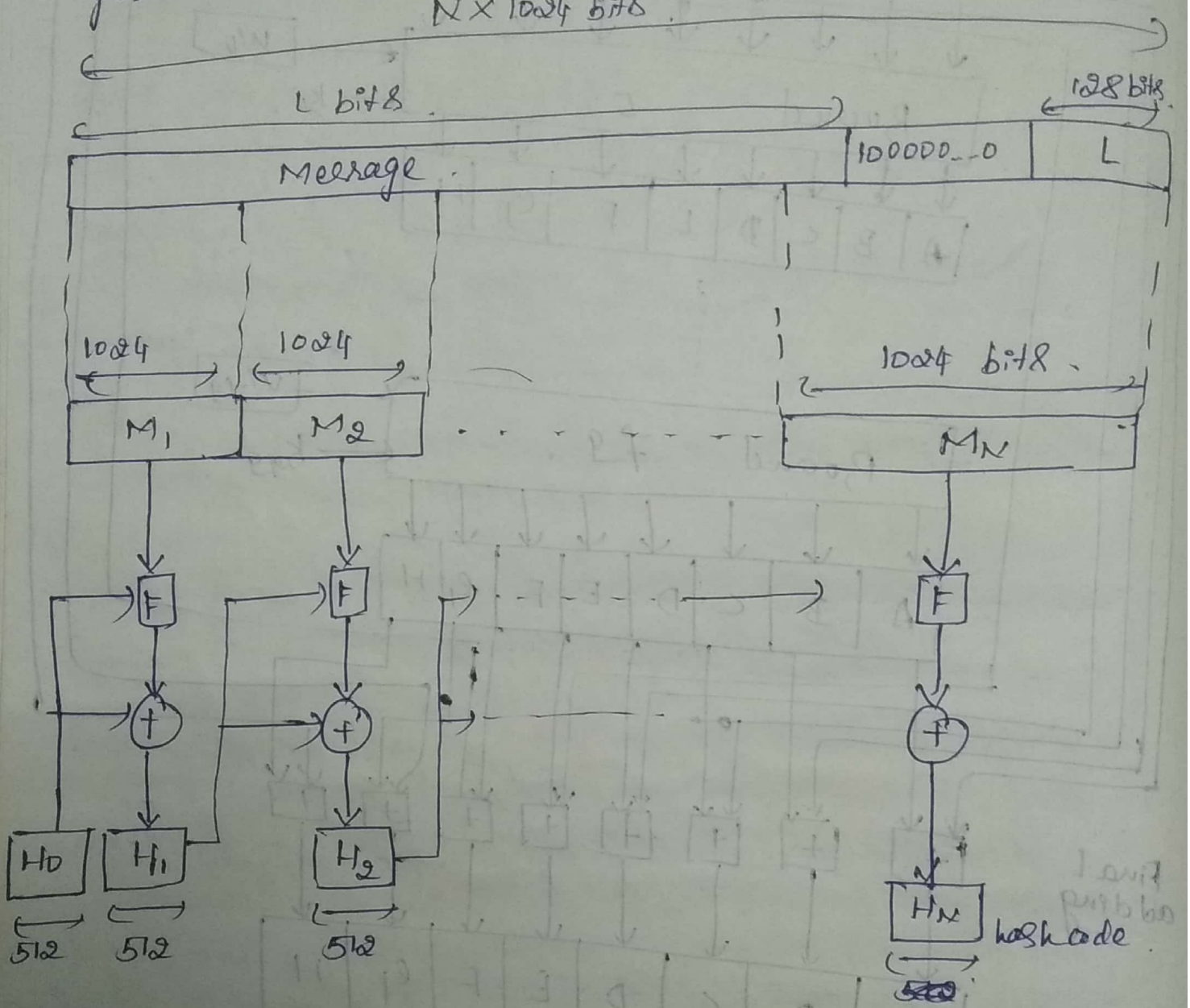


Final adding.

value for the next block or final digest.

Step 5: output

After all N 1024-bit blocks have been processed the o/p from the N^{th} stage is 512-bit message digest.



+ : word-by-word addition mod 2^{64}

minimum & maximum no of padding bits that can be added to a message are.

→ Minimum no of padding is 0.

It happens when $(-M-128) \bmod 1024 = 0$.

$M = 0$ (or)

$(-(2^{128}) - 128) \bmod 1024 = 0$.

$= -128 \bmod 1024$.

$-128 + 128 = 0 \bmod 1024$.

$896 - 0 = 896$.

The last block in the original message is 896 bits we add 128-bit length field to make the block complete.

→ Maximum no of padding is 1023.

It happens when $-M-128 \bmod 1024 = 1023 \bmod 1024$.

This means length of the original message

$-128 - 1023 \bmod 1024$.

$-1151 \bmod 1024$.

$= 1281$.

$896 - 1281 = 769$.

$769 + 128 = 897$.

In this case, we can't just add the length field because length of the last block exceeds one bit more than 1024. we need to add 897 bits to complete this block & create second block of 896 bits. Now length is added to make this block complete.

Word Expansion

→ Each block (msg) must be expanded.
→ A block is made of 1024 bits or 16 64-bit words.
→ We need 80 words in the processing
→ So, 16-bit word block must be expanded to 80 words from w_0 to w_{79}.

Uses of Hash function

* Hash functions are generally used to create one-way password file.
  A scheme which explains how hash of pwd is stored in an OS rather than the password itself. Thus, the actual password is not retrievable by hacker who gain access to pwd file.
  In simple terms, when user enters a password, hash of that pwd is compared to the stored hash value for verification. This approach to pwd protection is used by most of the OS.
* Hash can be used for intrusion detection & virus detection.
* cryptographic hash functions are used to construct a pseudorandom function. which is used to generate symmetric keys.

Message Authentication codes.

HMAC.

HMAC Design objectives

RFC 2104 lists the following design objectives for HMAC.

→ To use available hash functions, without modification. In particular, hash functions that perform well in software, & for which code is freely & widely available.

→ To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.

→ To preserve the original performance of the hash function without incurring a significant degradation.

→ To use & handle keys in simple way.

→ To have a well-understood cryptographic analysis of strength of the authentication mechanism based on reasonable assumptions on the embedded hash function.

The first & objectives are important to the accept-ability of HMAC.

HMAC Algorithm.

Diagram illustrates the overall operation of HMAC. The following terms are defined.

H → embedded hash function (eg: SHA-1)

M → Message input to HMAC (including padding specified in the embedded hash function).

Y_i → ith block of M $0 \le i \le (L-1)$

L → No of blocks in M.

b → No of bits in block.

n → length of hashcode produced by embedded function.

K → Secret key.
If key length is > b, the key is i/p to the hash function to produce an n-bit key.

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

ipad → 00110110 (36 in hexadecimal) repeated b/8 times.

opad → 01011100 (5C in hexadecimal) repeated b/8 times

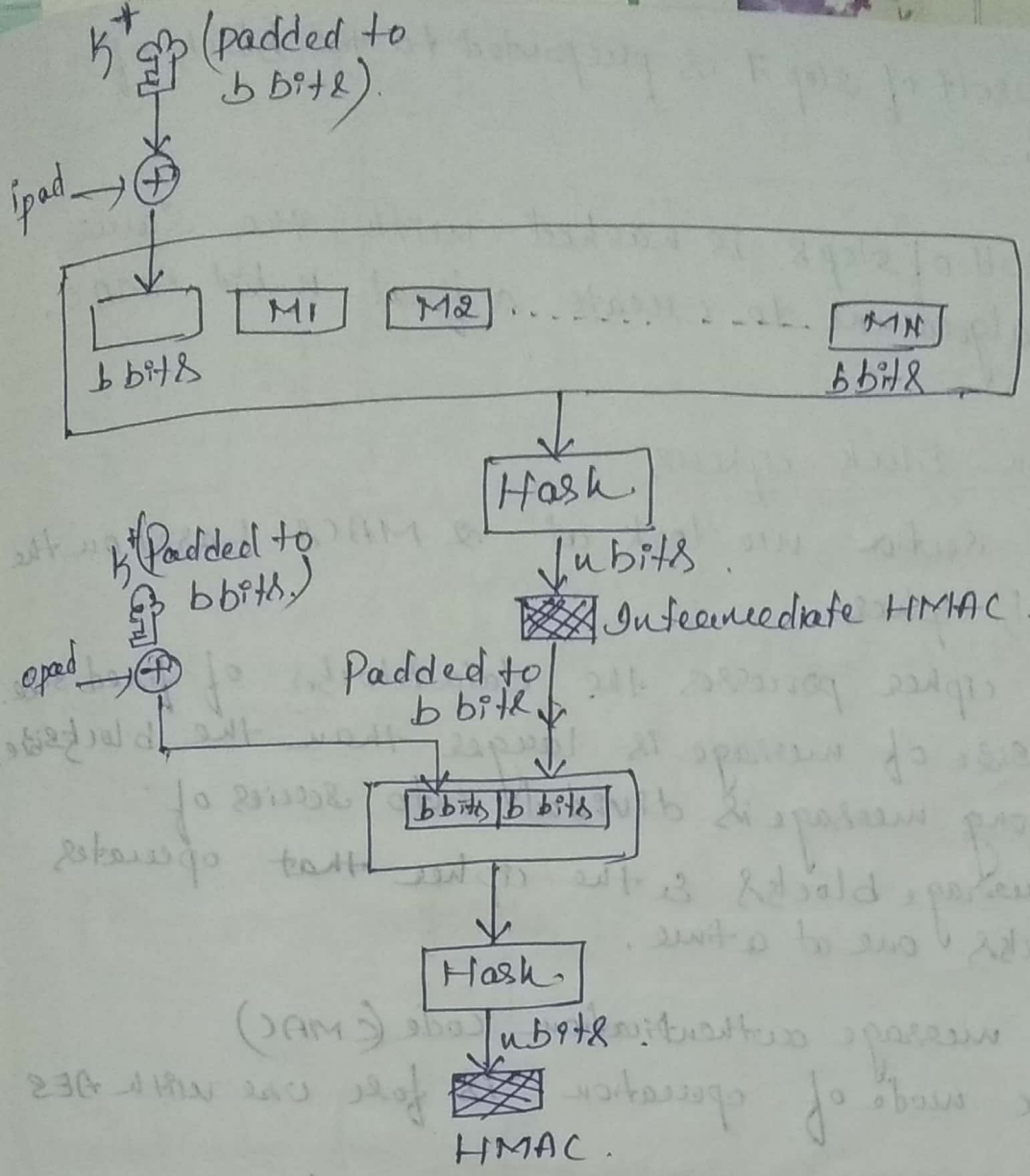
HMAC can be expressed as follows.

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

Step 1: The message is divided into N blocks, each of b bits.

Step 2: The secret key is left padded with 0's to create a b bit key.
Secret key (before padding) be longer than n bits.
where n is the size of HMAC.

Step 3: The result of step 2 is exclusively-ORed with a constant called ipad (input pad) to create a b-bit block.



The value of ipad is the b/8 ~~times~~ repetition of the sequence (00110110).

Step 4: The resulting block is prepended to the N-blocks message. The result is N+1 blocks.

Step 5: The result of step 4 is hashed ~~with~~ to create an n-bit digest. We call the digest as Intermediate HMAC.

Step 6: The intermediate n-bit HMAC is left padded with 0s to make b-bit blocks.

Step 7: Step 2 & 3 are repeated by a different constant opad (output pad). The value of opad is the b/8 repetition of sequence (01011100)

Step 8: The result of step 7 is prepended to the block of step 6.

Step 9: The result of step 8 is hashed with the same hashing algorithm to create a final n-bit HMAC.

MAC Based on Block ciphers.

In this section we look at 2 MACs based on the use of block cipher.

A block cipher processes the data blocks of fixed size. Usually, the size of message is larger than the block size. Hence, the long message is divided into series of sequential message blocks & the cipher that operates on these blocks one at a time.

(i) cipher based message authentication code (CMAC)

→ CMAC mode of operation is for use with AES & triple DES.

→ First, let us consider the operation of CMAC when the message is integer multiple of the cipher block length b.

→ For AES, b = 128 & for triple DES b = 64.

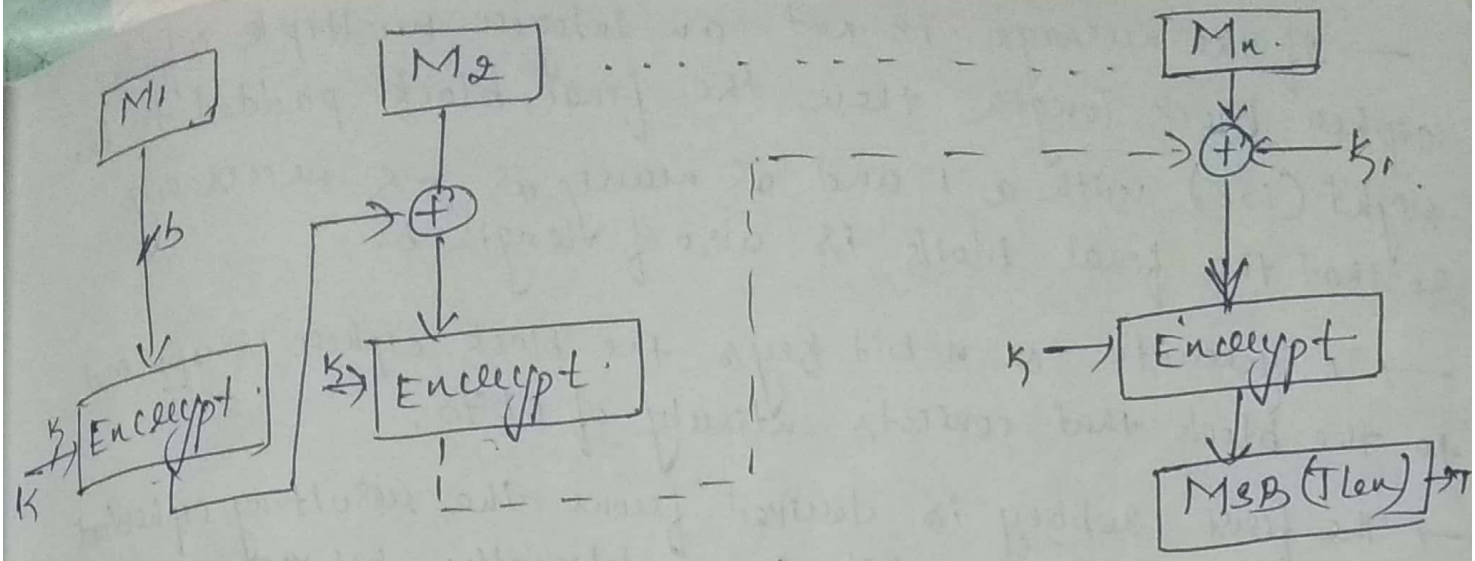
→ The message is divided into n blocks ($M_1, M_2 \ldots M_n$).

→ The Algorithm makes use of a K-bit encryption key K & an n-bit key, K_1.

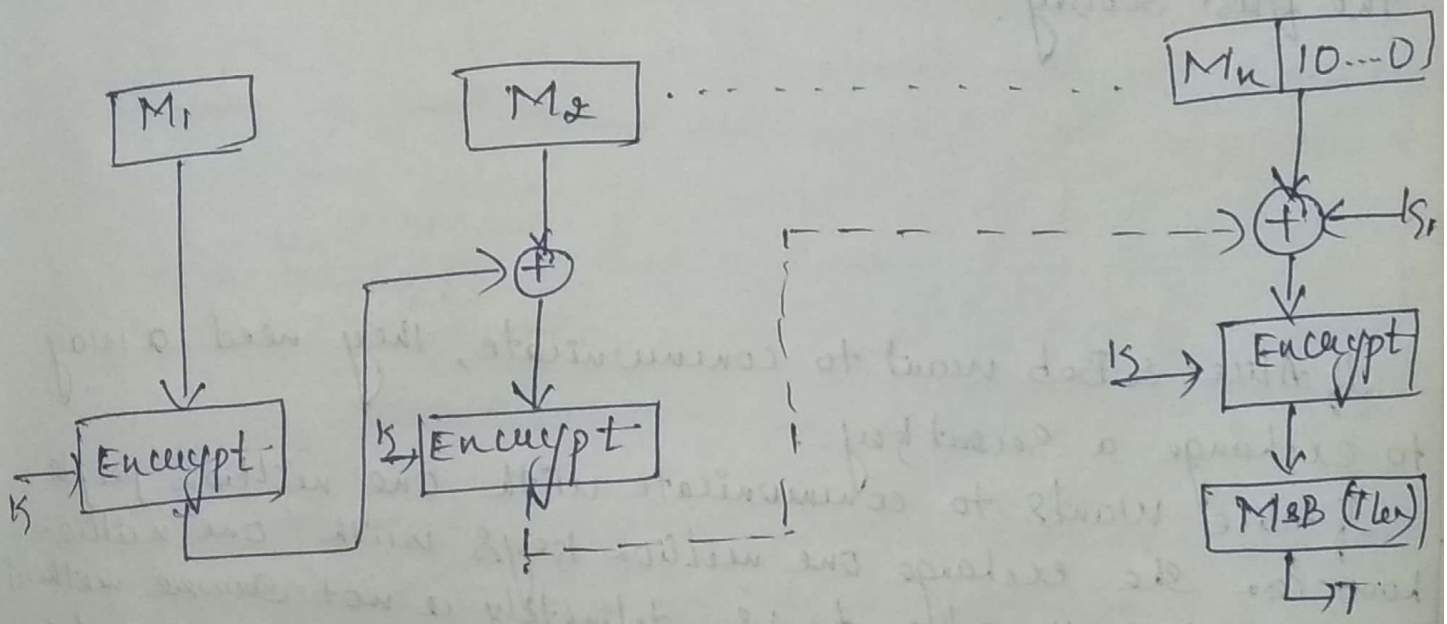
→ For AES, the key size K = 128 bits or 192 or 256 bits.

→ For Triple DES, the key size is 112 or 168 bits.

→ CMAC is calculated as follows:



(a) Message length is integer multiple of block size.



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

$C_1 = E(K, M_1)$

$C_2 = E(K, M_2 \oplus C_1)$

$C_3 = E(K, M_3 \oplus C_2)$

$C_n = E(K, [M_n \oplus C_{n-1} \oplus K_1])$

$T = MSB_{Tlen}(C_n)$.

T → MAC or Tag.

Tlen → bit length of T.

$MSB_s(X)$ → the s leftmost bits of the bit string X.

→ If the message is not an integer multiple of the cipher block length, then the final block padded to the right (LSB) with a 1 and as many as 0's necessary so that the final blocks is also of length b.

→ To generate the n-bit keys, the block cipher is applied to the blocks that consists entirely of 0bits.

→ The first subkey is derived from the resulting ciphertext by a left shift of one bit & conditionally by xoring the constant that depends on the block size.

→ The second key is derived in the same manner from the first subkey.

Digital Signature

When Alice sends a message to Bob, Bob needs to check the authenticity of the sender, he needs to be sure that the message comes from Alice not Eve.

Bob can ask Alice to sign the message electronically. In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer this type of signature as a "digital signature".

M: Message
S: Signature

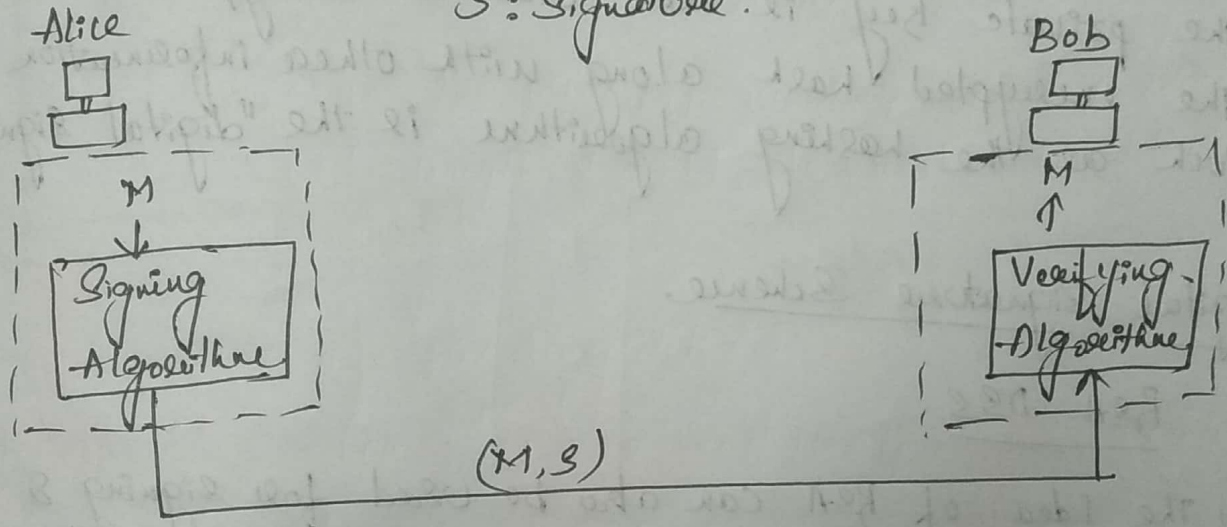


Diagram illustrates the digital signature process. The sender uses signing algorithm to sign the message. The message & the signature are sent to the receiver. The Receiver receives the message & the signature & applies the verifying algorithm to the combination. If the result is true, the message is accepted. Otherwise, it is rejected.

Digital Signature is an electronic forme of a signature that can be used to authenticate identity of the Sender of a message or the signer of the document.

Also ensure that the original content of the message or document that has been sent is unchanged.

To create a digital signature, signing software such as an email programe, creates a one-way hash of the electronic data to be signed.

The private key is then used to encrypt the hash.

The encrypted hash along with other information, such as the hashing algorithm is the "digital signature".

Digital Signature Scheme

RSA DSS

→ The idea of RSA can also be used for signing & verifying a message. In this case, it is called "RSA digital signature scheme".

→ The Digital signature scheme changes the roles of private & public keys.

→ private & public keys of the Sender are used.

→ Sender uses her own private key to sign the document, the receiver uses Sender's public key to verify it.

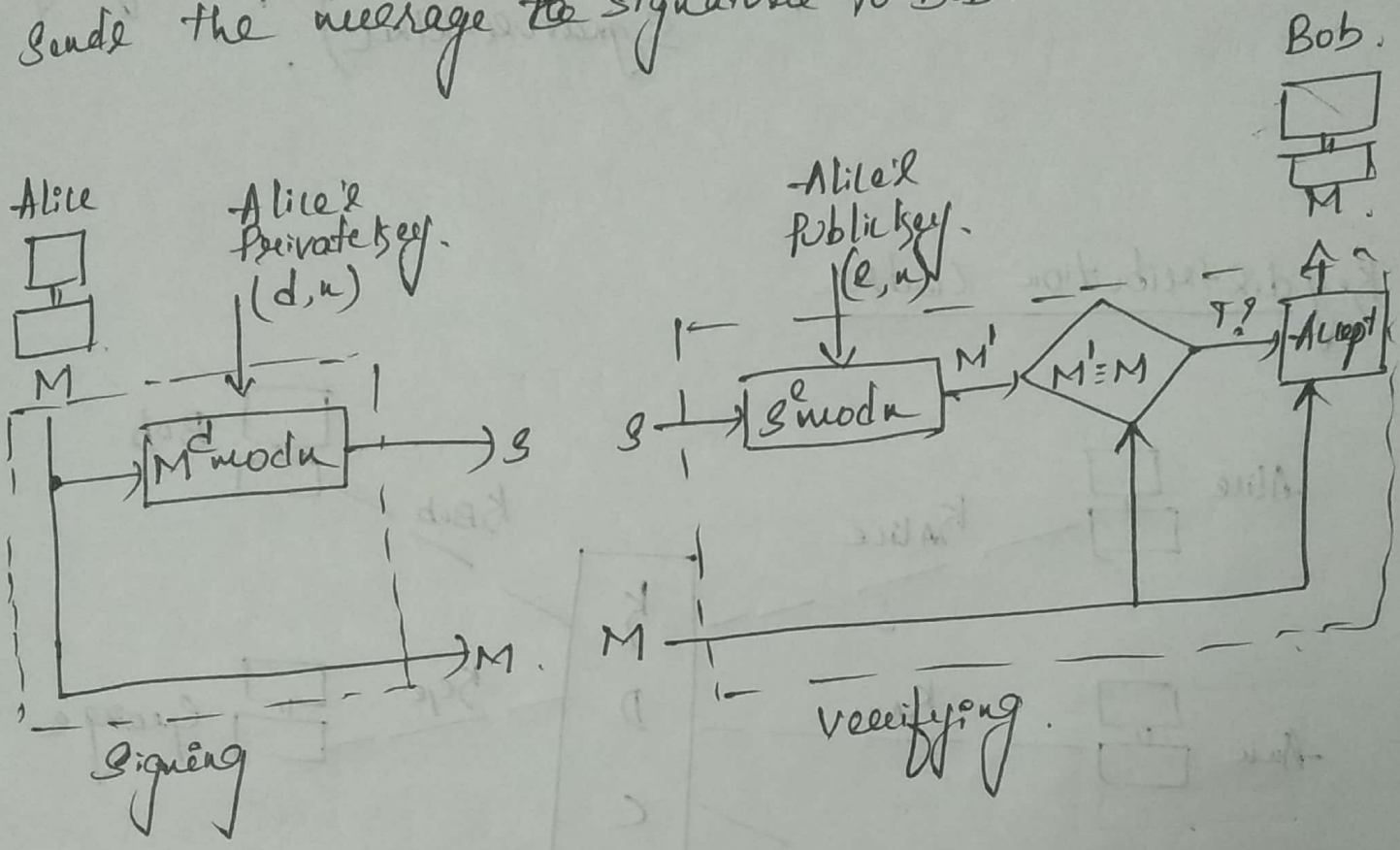
→ General idea behind RSA DSS.

Signing

Alice creates a signature out of message using her own private exponent.

$$S = M^d \bmod n$$

Sends the message & signature to Bob.



Verifying

Bob receives M & S. Bob applies Alice's public exponent to the signature to create a copy of the message

$M' = S^e \bmod n$.

Bob compares M' with M.

If the 2 values are congruent. Bob accepts the message.

Example of digital Signature

Suppose there is a company "xyz", which is a client of "ABC" consultancy.

(Refer computerfun4u.blogspot.com/2009/02/example-of-digital-signature.html).