SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMMUNICATION ENGINEERING

UNIT - 3
NETWORK SECURITY – SCS1316

3.1: PUBLIC KEY CRYPTO SYSTEM:

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. The first problem is that of key distribution, and the second one related to digital signature. Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic.

It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key. A public-key encryption scheme has six ingredients Plaintext: This is the readable message or data that is fed into the algorithm as input.

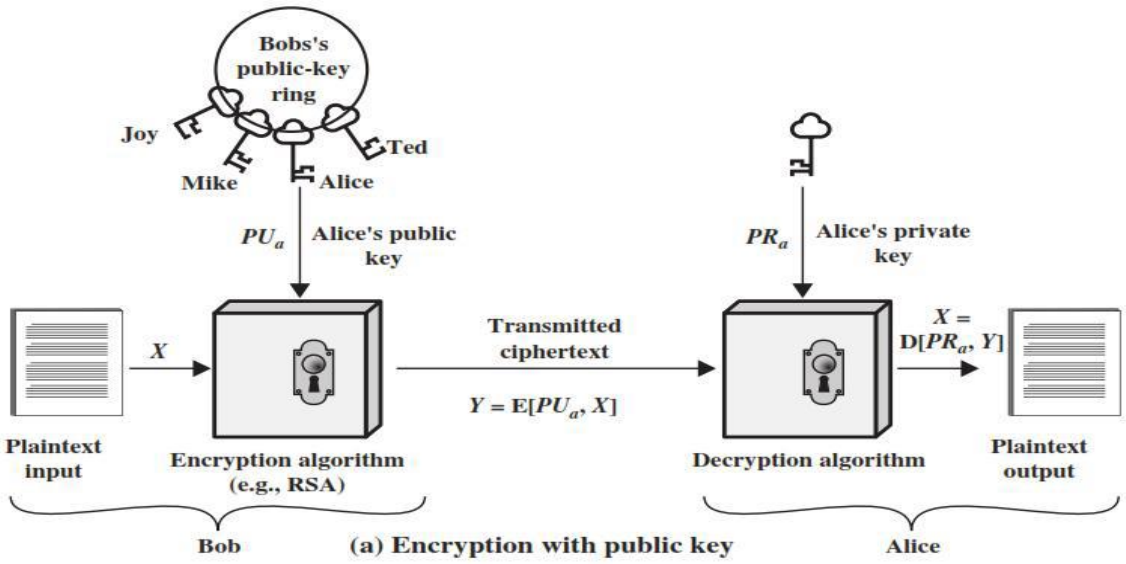


Fig 3.1: Encryption with Public key

Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.

• Public and private keys: 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 algorithm depend on the public or private key that is provided as input.

• Cipher text: This is the scrambled message produced as output. It depends on the plaintext

and the key. For a given message, two different keys will produce two different cipher texts.

• Decryption algorithm: This algorithm accepts the cipher text and the matching key and produces the original plaintext.

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 3.1, suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key. With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed.

As long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

3.2 : DIFFIE-HEILMAN KEY EXCHANGE/AGREEMENT ALGORITHM:

Whitefield Diffie and Martin Hellman devised an amazing solution to the problem of key agreement, or key exchange in 1976. This solution is called as the Diffie-Hellman Key Exchange /Agreement Algorithm. The beauty of this scheme is that the two parties, who want to communicate securely, can agree-on a symmetric key using this technique.

It might come as a surprise, but Kl is actually equal to K2! This means that Kl =K2 = K is the symmetric key, which Alice and Bob must keep secret and can henceforth usefor encrypting/decrypting their messages with. The mathematics behind this is quite interesting.

Let us try to understand what this actually means, in simple terms. (a) Firstly, take a look at what Alice does in step 6. Here, Alice computes: K1 = BX mod n. What is B? From step 4, we have: B = gYmod n. Therefore, if we substitute this value of B in step 6, we will have the following equation: Kl = (g)yr mod n =(g)YXmod n. (b) Now, take a look at what Bob does in step 7. Here, Bob computes: K2 = AY mod n. What is A? From step 2, we have:

A = gX mod n. Therefore, if we substitute this value of A in step 7, we will have the following equation: K2 = (gX)Y mod n = gXY' mod n. Now, basic mathematics says that: KYX = KXY Therefore, in this case, we have: K I = K2 = K.

1. Firstly, Alice and Bob agree on two large prime numbers, n and g. These two integers need not be kept secret. Alice and Bob can use an insecure channel to agree on them.
2. Alice chooses another large random number x, and calculates A such that:
   $A = g^x \bmod n$
3. Alice sends the number A to Bob.
4. Bob independently chooses another large random integer y and calculates B such that:
   $B = g^y \bmod n$
5. Bob sends the number B to Alice.
6. A now computes the secret key K1 as follows:
   $K1 = B^x \bmod n$
7. B now computes the secret key K2 as follows:
   $K2 = A^y \bmod n$

Fig. *Diffie-Hellman key exchange algorithm*

Fig 3.2: Diffie- Hellman Key Exchange algorithm

1. Firstly, Alice and Bob agree on two large prime numbers, n and g. These two integers need not be kept secret. Alice and Bob can use an insecure channel to agree on them.
   Let n = 11, g = 7.
2. Alice chooses another large random number x, and calculates A such that:
   $A = g^x \bmod n$
   Let x = 3. Then, we have, $A = 7^3 \bmod 11 = 343 \bmod 11 = 2$.
3. Alice sends the number A to Bob.
   Alice sends 2 to Bob.
4. Bob independently chooses another large random integer y and calculates B such that:
   $B = g^y \bmod n$
   Let y = 6. Then, we have, $B = 7^6 \bmod 11 = 117649 \bmod 11 = 4$.
5. Bob sends the number B to Alice.
   Bob sends 4 to Alice.
6. A now computes the secret key K1 as follows:
   $K1 = B^x \bmod n$
   We have, $K1 = 4^3 \bmod 11 = 64 \bmod 11 = 9$.
7. B now computes the secret key K2 as follows:
   $K2 = A^y \bmod n$
   We have, $K2 = 2^6 \bmod 11 = 64 \bmod 11 = 9$.

Fig. *Example of Diffie-Hellman key exchange*

Fig 3.3: Example of Diffie-Hellman Key exchange

3.3 KEY MANAGEMENT:

Key management refers to management of cryptographic keys in a cryptosystem. This includes dealing with the generation, exchange, storage, use and replacement of keys. Once keys are inventoried, key management typically consists of three steps: exchange, storage and use. Key exchange (also key establishment) is any method in cryptography by which cryptographic keys are exchanged between two parties, allowing use of a cryptographic algorithm. If the cipher is a symmetric key cipher, both will need a copy of the same key. If an asymmetric key cipher with the public/private key property, both will need the other's public key.

3.3.1: Key storage: However distributed, keys must be stored securely to maintain communications security. Security is a big concern[4] and hence there are various techniques in use to do so. Likely the most common is that an encryption application manages keys for the user and depends on an access password to control use of the key.

Key use : The major issue is length of time a key is to be used, and therefore frequency of replacement. Because it increases any attacker's required effort, keys should be frequently changed. This also limits loss of information, as the number of stored encrypted messages which will become readable when a key is found will decrease as the frequency of key change increases.

Challenges

Several challenges IT organizations face when trying to control and manage their encryption keys are:

1. Scalability: Managing a large number of encryption keys.
2. Security: Vulnerability of keys from outside hackers, malicious insiders.
3. Availability: Ensuring data accessibility for authorized users.
4. Heterogeneity: Supporting multiple databases, applications and standards.
5. Governance: Defining policy-driven access control and protection for data

Distribution of public Keys:

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

• Public announcement • publicly available directory

• Public-key authority. • Public-key certificates

3.3.2: PUBLIC ANNOUNCEMENT OF PUBLIC KEYS :

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large (Refer fig below)

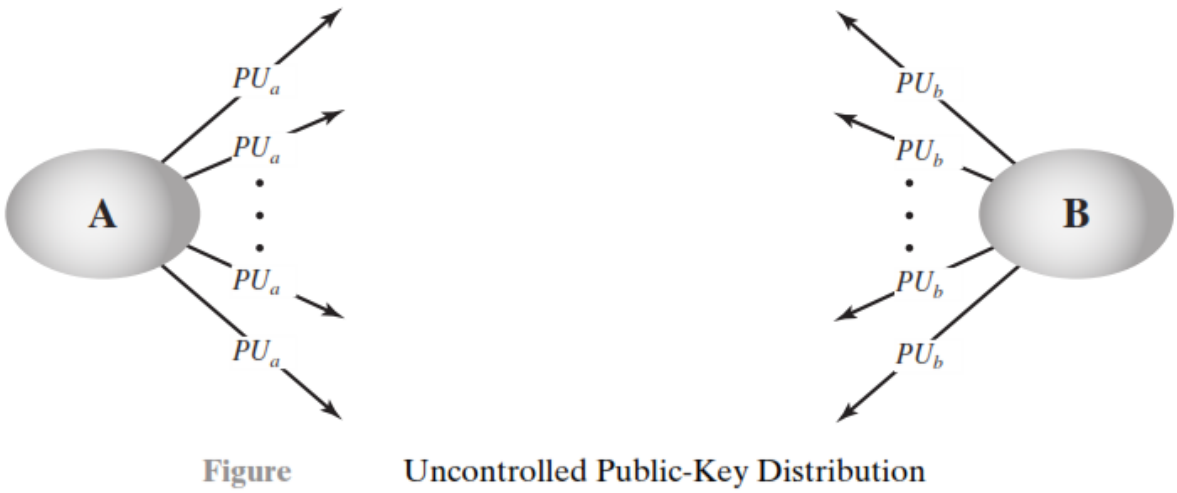


Figure Uncontrolled Public-Key Distribution

Fig 3.4: Uncontrolled public key distributions

Publicly Available Directory

A greater degree of security can be achieved by maintaining a publicly availabledynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization (Figure below). Such a scheme would include the following elements:

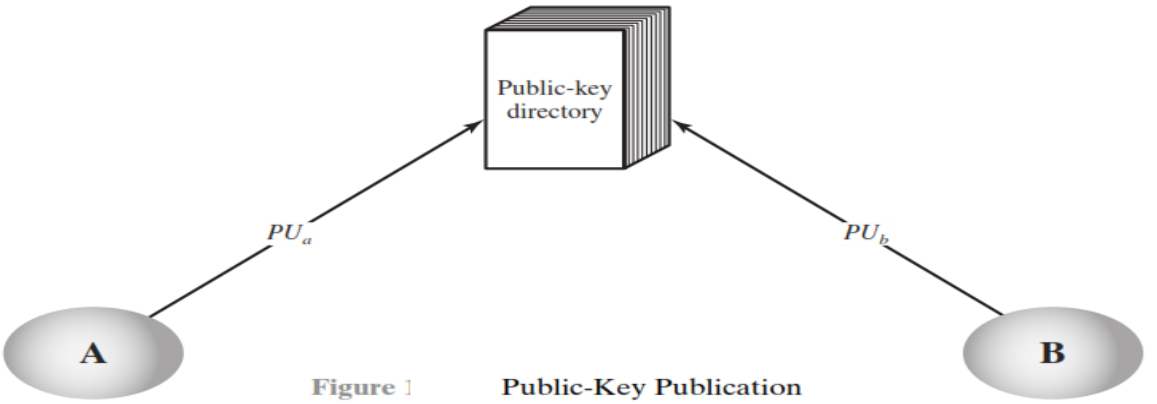


Fig 3.5: Public-Key Publication

1.The authority maintains a directory with a {name, public key} entry for each participant.

2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.

3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.

4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

3.3.3: **PUBLIC-KEY AUTHORITY:**

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure below.

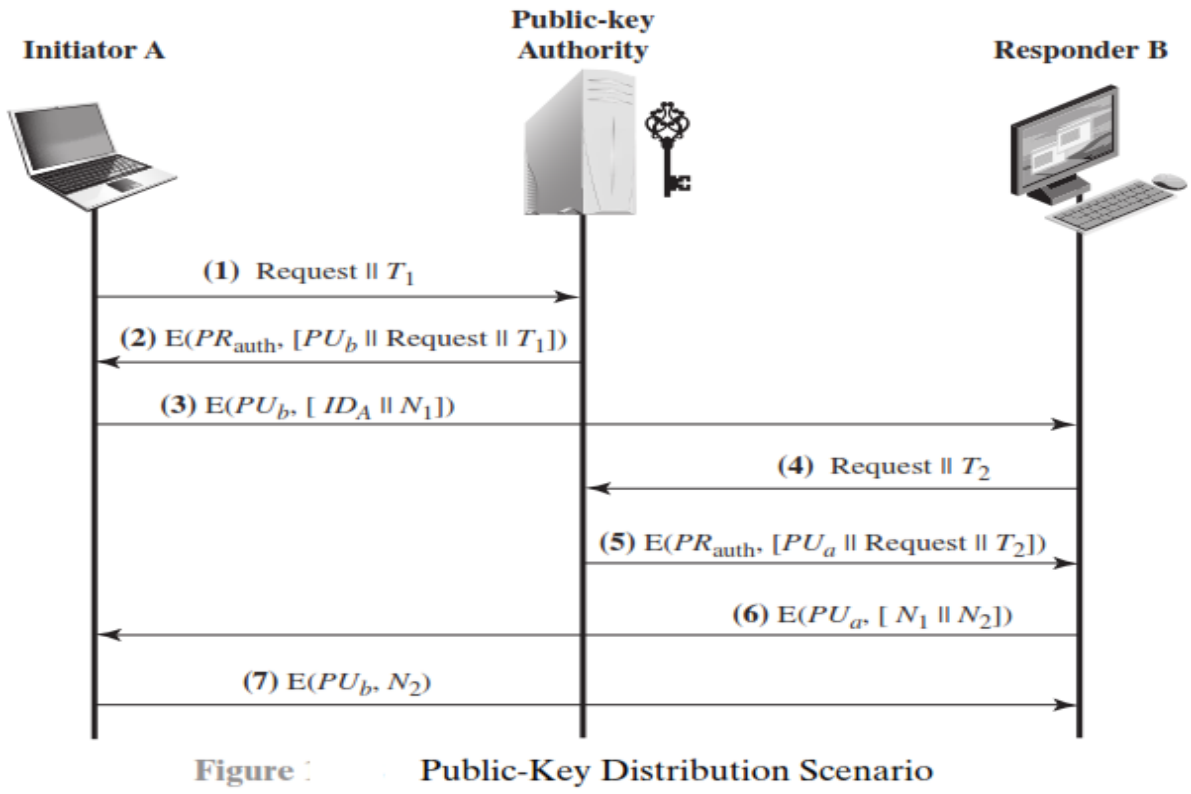


Fig 3.6: Public Key Distribution Scenario

1. A sends a time stamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, PR Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

• B's public key, PUb, which A can use to encrypt messages destined for B

• The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority

• The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (IDA) and a nonce (N), which is used to identify this transaction uniquely.

4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's

public key. At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (3), the presence of N_1 in message (6) assures A that the correspondent is B.

7. A returns N2, which is encrypted using B's public key, to assure B that its correspondent is A.

Thus, a total of seven messages are required.

3.4: PUBLIC-KEY CERTIFICATES:

The public key certificate relies on certificates that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party.

Typically, the third party is a certificate authority, such as a government agency or a financial institution that is trusted by the user community. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature.

We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the currency of the certificate.

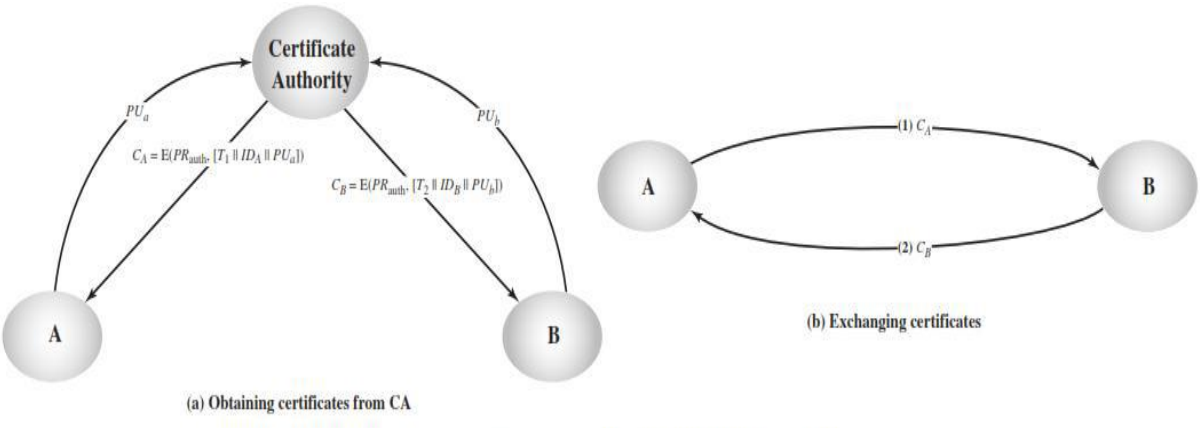


Figure Exchange of Public-Key Certificates

Fig 3.7:Exchange of public key exchange

A certificate scheme is illustrated in Figure above. Each participant applies to the certificate authority, supplying a public key and requesting a certificate. Application must be in person or by some form of secure authenticated communication.

For participant A, the authority provides a certificate of the form

$$C_A = \mathrm{E}(PR_{\mathrm{auth}}, [T \| ID_A \| PU_a])$$

where PRauth is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$\mathrm{D}(PU_{\mathrm{auth}}, C_A) = \mathrm{D}(PU_{\mathrm{auth}}, \mathrm{E}(PR_{\mathrm{auth}}, [T \| ID_A \| PU_a])) = (T \| ID_A \| PU_a)$$

The recipient uses the authority's public key, PUauth, to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements IDA and PUa provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, transport layer security (TLS), and S/MIME.

3.4.1: Message digest:

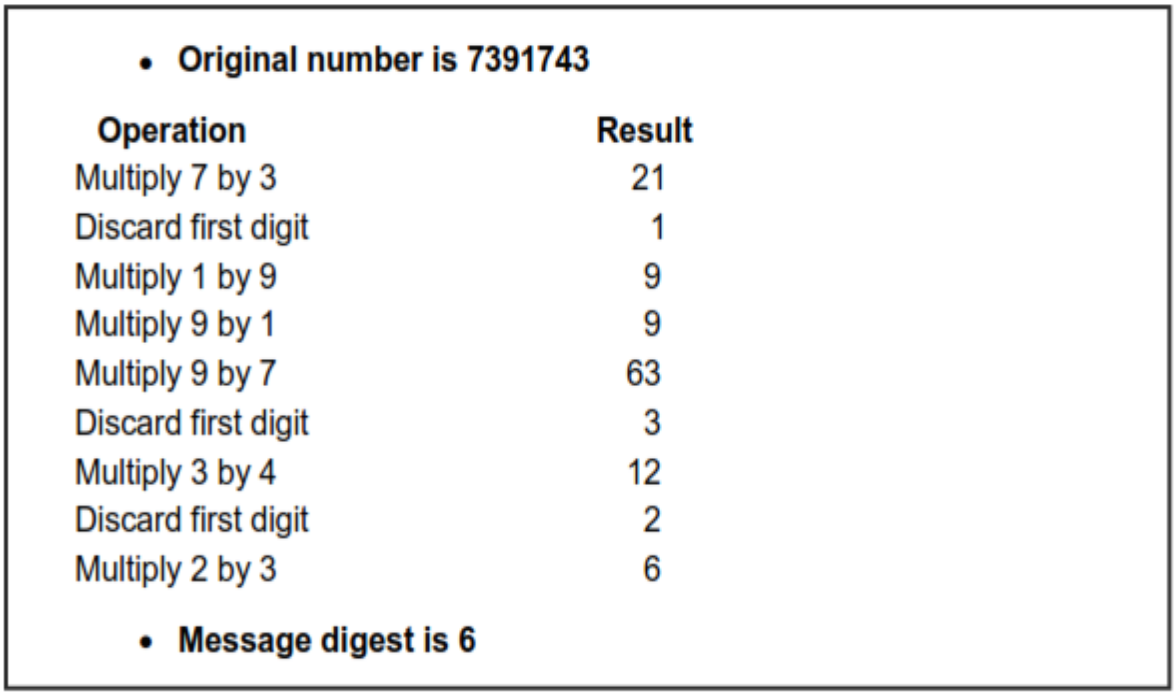
Message digest (also called as hash) is a fingerprint or the summary of a message. It is used to verify the integrity of the data (i.e. to ensure that a message has not been tampered with after it

leaves the sender but before it reaches the receiver).

Idea of a Message Digest:

The concept of message digests is based on similar principles. However, it is slightly wider in scope. For instance, suppose we have a number 4000 and we divide it by 4 to get 1000, 4 becomes a fingerprint of the number 4000. Dividing 4000 by 4 will always yield 1000. If we change either 4000 or 4, the result will not be 1000. Another important point is, if we are simply given the number 4, but are not given any further information, we would not be able to trace back the equation 4 x 1000 =4000. Thus, we have one more important concept here. The fingerprint of a message (in this case, the number 4) does not tell anything about the original message (in this case, the number 4000). This is because there are infinite other possible equations, which can produce the result 4.

Another simple example of message digest is shown in Fig. below.



- **Original number is 7391743**

Operation	Result
Multiply 7 by 3	21
Discard first digit	1
Multiply 1 by 9	9
Multiply 9 by 1	9
Multiply 9 by 7	63
Discard first digit	3
Multiply 3 by 4	12
Discard first digit	2
Multiply 2 by 3	6

- **Message digest is 6**

Fig. *Simplistic example of message digest*

Fig 3.8: Example of message digest

Let us assume that we want to calculate the message digest of a number 7391753. Then, we multiply each digit in the number with the next digit (excluding it if it is

0) and disregarding the first digit of the multiplication operation, if the result is a two-digit number.

3.4.2: Requirements of a Hash function (Message digest):

1. It should be a one way function. That means given the message it should be easy to find out its digest, and the reverse should be impossible.(getting back the message from its digest must be infeasible)
2. No two different messages should produce a same digest. This requirement is stated as collision free property.

3.4.3: MD5 :

MD5 is a message digest algorithm developed by Ron Rivest. MD5 is quite fast and produces 128-bit message digests. Over the years, researchers have developed potential weaknesses in MD5. However, so far, MD5 has been able to successfully defend itself against collisions. This may not be guaranteed for too long, though.

After some initial processing, the input text is processed in 512-bit blocks (which are further divided into 16 32-bit sub-blocks). The output of the algorithm is a set of four 32-bit blocks, which make up the 128-bit message digest.

3.5.4 How MD5 Works?

Step 1: Padding :The first step in MD5 is to add padding bits to the original message. The aim of this step is to make the length of the original message equal to a value, which is 64 bits less than an exact multiple of 512. For example, if the length of the original message is 1000 bits, we add a padding of 472 bits to make the length of the message 1472 bits. This is because, if we add 64 to 1472, we get 1536, which is a multiple of 512 (because 1536 = 512 x 3).

Thus, after padding, the original message will have a length of 448 bits (64 bits less than 512), 960 bits (64 bits less than 1024), 1472 bits (64 bits less than 1536), etc. The padding consists of a single 1-bit, followed by as many 0-bits, as required. Note that padding is always added, even if the message length is already 64 bits less than a multiple of 512. Thus, if the message were already of length say 448 bits, we will add a padding of 512 bits to make its

length 960 bits. Thus, the padding length is any value between 1 and 512.

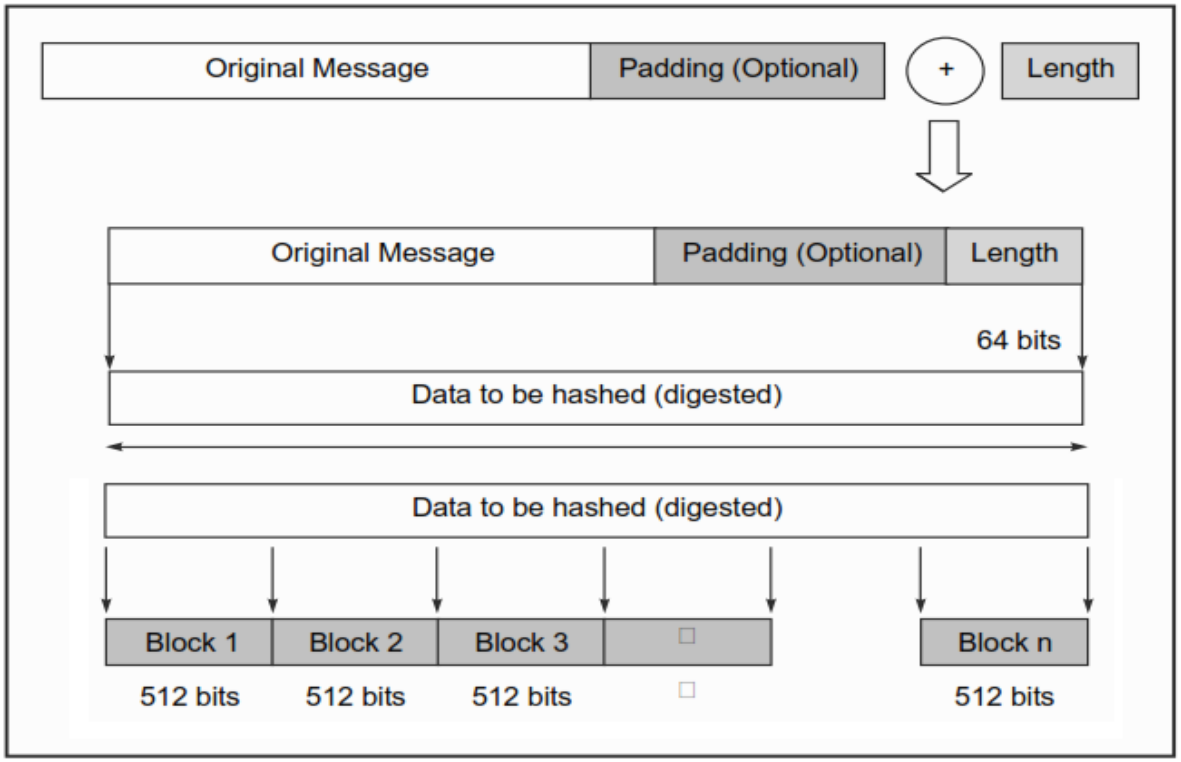


Fig 3.9: Message Digest

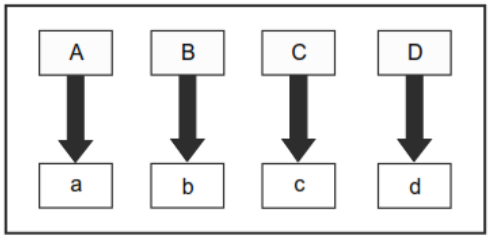
Step 2: Append length After padding bits are added, the next step is to calculate the original length of the message and add it to the end of the message, after padding. The length of the message is calculated, excluding the padding bits. This length of the original message is now expressed as a 64-bit value and these 64 bits are appended to the end of the original message + padding.

Step 3: Divide the input into 512-bit blocks Now, we divide the input message into blocks, each of length 512 bits. (Refer fig)

Step 4: Initialize chaining variables In this step, four variables (called as chaining variables) are initialized. They are called as A, B, C and D. Each of these is a 32-bit number. The initial hexadecimal values of these chaining variables are shown in Fig. below.

A	Hex	01	23	45	67
B	Hex	89	AB	CD	EF
C	Hex	FE	DC	BA	98
D	Hex	76	54	32	10

Fig. *Chaining variables*



Copying chaining variables into temporary variables

Fig 3.10: Chaining Variables

Step 5: Process blocks :

Copy the four chaining variables into four corresponding variables, a, b, c and d. After all the initializations, the real algorithm begins. There is a loop that runs for as many 512-bit blocks as are in the message. Now, we have four rounds. In each round, we process all the 16 sub-blocks belonging to a block. The inputs to each round are: (a) all the 16 sub-blocks, (b) the variables a, b, c, d and (c) some constants, designated as t.

All the four rounds vary in one major way: Step 1 of the four rounds has different processing. The other steps in all the four rounds are the same. • In each round, we have 16 input sub-blocks, named M[0], M[1], …, M[15] or in general, M[i], where i varies from 0 to 15. As we know, each sub-block consists of 32 bits.

• Also, t is an array of constants. It contains 64 elements, with each element consisting of 32 bits. We denote the elements of this array t as t[1], t[2], … t[64] or in general as t[k], where k varies from 1 to 64. Since there are four rounds, we use 16 out of the 64 values of t in each round.

Let us summarize these iterations of all the four rounds. In each case, the output of the intermediate as well as the final iteration is copied into the register abcd. Note that we have 16 such iterations in each round.

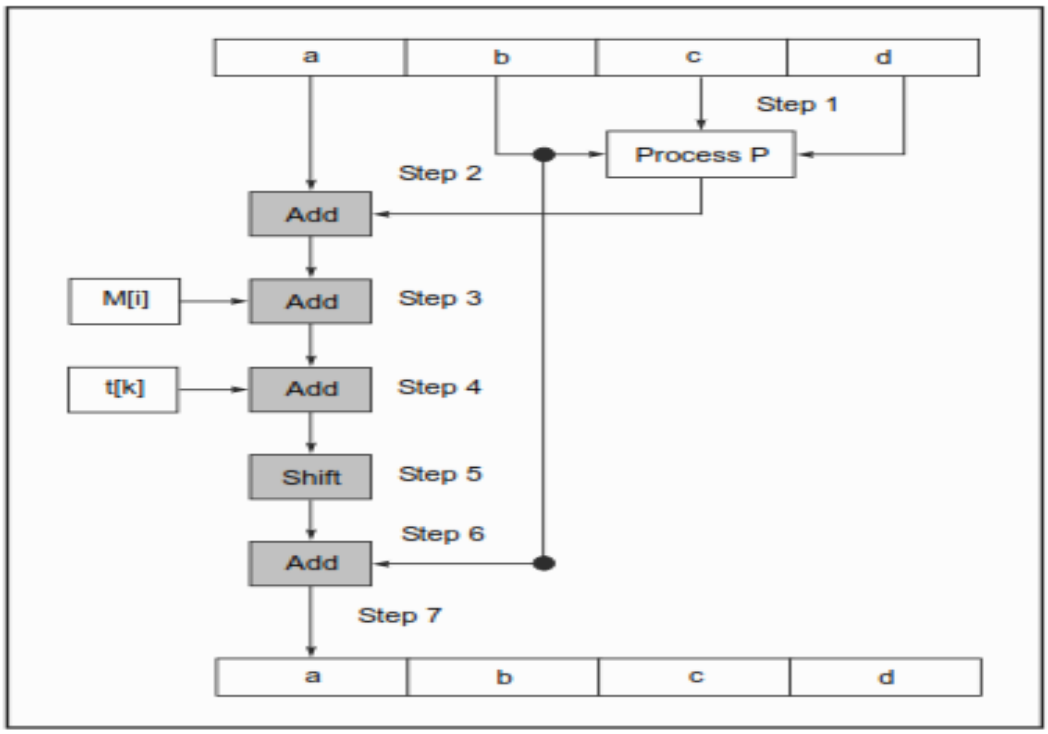


Fig. *One MD5 operation*

Fig 3.11:One MD5 operation

1. A process P is first performed on b, c and d. This process P is different in all the four rounds.
2. The variable a is added to the output of the process P (i.e. to the register abcd).
3. The message sub-block M[i] is added to the output of Step 2 (i.e. to the register abcd).
4. The constant t[k] is added to the output of Step 3 (i.e. to the register abcd).
5. The output of Step 4 (i.e. the contents of register abcd) is circular-left shifted by s bits. (The value of s keeps changing).
6. The variable b is added to the output of Step 5 (i.e. to the register abcd).
7. The output of Step 6 becomes the new abcd for the next step.

3. 5 : SECURE HASH ALGORITHM (SHA):

The National Institute of Standards and Technology (NIST) along with NSA developed the Secure Hash Algorithm (SHA). In 1993. SHA is a modified version of MD5 and its design closely resembles MD5. SHA works with any input message that is less than 2
64 bits in length. The output of SHA is a message digest, which is 160 bits in length (32 bitsmo re than the message digest produced by MD5). The word Secure in SHA was decided based on two features. SHA is designed to be computationally infeasible to:
(a) Obtain the original message, given its message digest and
(b) Find two messages producing the same message digest

Step1 to 3 MD5 and SHA are same. In step 4 and process SHA differs from MD5.

Step 4: Initialize chaining variables Now, five chaining variables A through E are initialized. Remember that we had four chaining variables, each of 32 bits in MD5 (which made the total length of the variables 4 x 32 = 128 bits). Recall that we stored the intermediate as well as the final results into the combined register made up of these four chaining variables, i.e. abcd. Since in the case of SHA, we want to produce a message digest of length 160 bits, we need to have five chaining variables here (5x32 = 160 bits). In SHA, the variables A through D have the same values as they had in MD5. Additionally, E is initialized to Hex C3 D2 E1 F0.

Step 5: Process blocks Now the actual algorithm begins. Here also, the steps are quitesimilar to those in MD5. SHA has four rounds, each round consisting of 20 steps. Each round takes the current 512- bit block, the register abcde and a constant K[t] (where t = 0 to 79) as the three inputs. It then updates the contents of the register abcde using the SHA algorithm steps. Also notable is the fact that we had 64 constants defined as t in MD5. Here, we have only four constants defined for K[t], one used in each of the four rounds.

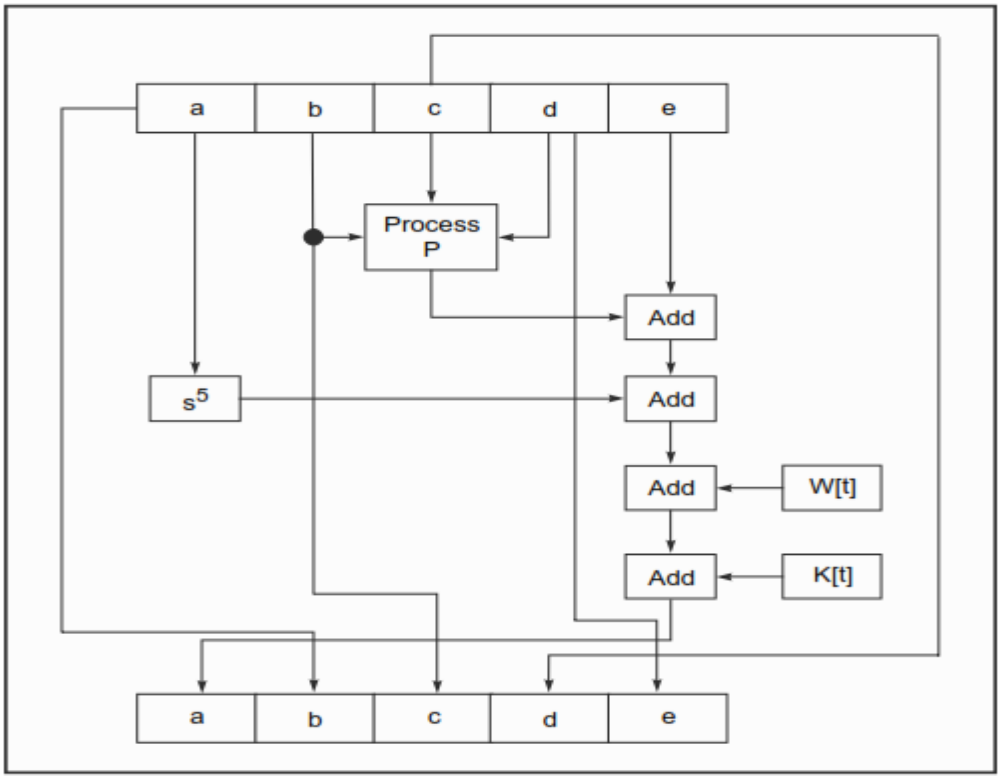


Fig. *Single SHA-1 iteration*

Fig. 3.12: Single SHA-1 iteration

Table 3.1: Comparison of MD5 and SHA-1

Table *Comparison of MD5 and SHA-1*

Point of discussion	*MD5*	*SHA-1*
Message digest length in bits	128	160
Attack to try and find the original message given a message digest	Requires 2^{128} operations to break in	Requires 2^{160} operations to break in, therefore more secure
Attack to try and find two messages producing the same message digest	Requires 2^{64} operations to break in	Requires 2^{80} operations to break in
Successful attacks so far	There have been reported attempts to some extent (as we discussed earlier)	No such claims so far
Speed	Faster (64 iterations and 128-bit buffer)	Slower (80 iterations and 160-bit buffer)
Software implementation	Simple, does not need any large programs or complex tables	Simple, does not need any large programs or complex tables

3.6: MESSAGE AUTHENTICATION CODE (MAC):

The concept of Message Authentication Code (MAC) is quite similar to that of a message digest. However, there is one difference. As we have seen, a message digest is simply a fingerprint of a message. There is no cryptographic process involved in the case of message digests. In contrast, a MAC requires that the sender and the receiver should know a shared symmetric (secret) key, which is used in the preparation of the MAC. Thus, MAC involves cryptographic processing. Let us assume that the sender A wants to send a message M to a receiver B. How the MAC processing works is shown in Fig. below.

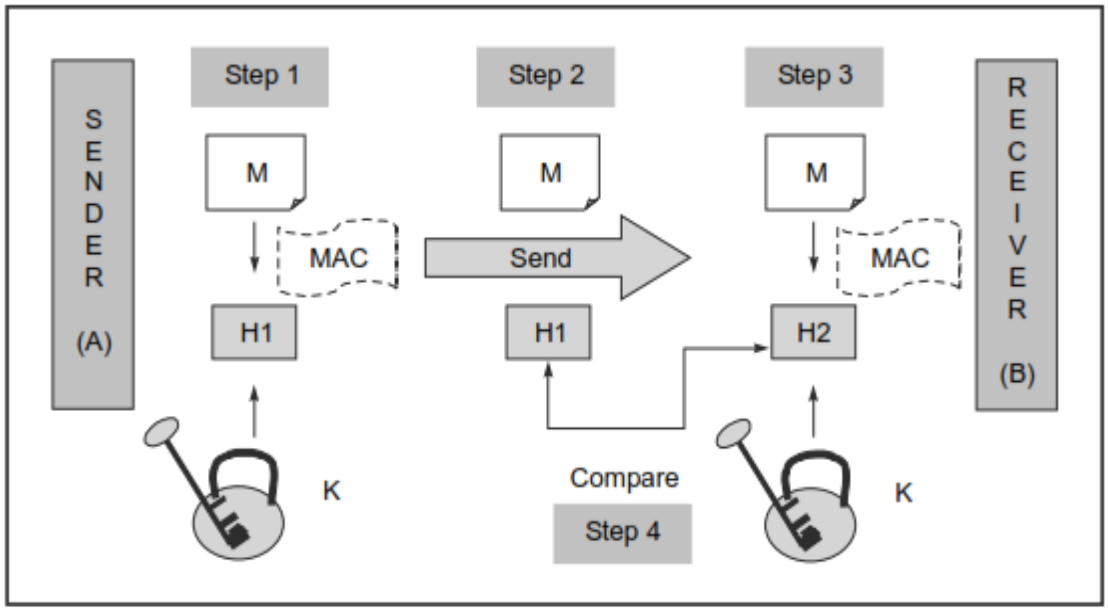


Fig. *Message authentication code (MAC)*

Fig.3.13: Message Authentication Code

1. A and B share a symmetric (secret) key K, which is not known to anyone else. A calculates the MAC by applying key K to the message M.
2. A then sends the original message M and the MAC H1 to B.
3. When B receives the message, B also uses K to calculate its own MAC H2 over M.
4. B now compares H1 with H2. If the two match, B concludes that the message M has not been changed during transit. However, if H1 ≠ H2, B rejects the message, realizing that the message was changed during transit.

3.7 : HMAC

HMAC stands for Hash-based Message Authentication Code. HMAC has been chosen as a mandatory security implementation for the Internet Protocol (IP) security and is also used in the Secure Socket Layer (SSL) protocol, widely used on the Internet. The fundamental idea behind HMAC is to reuse the existing message digest algorithms, such as MD5 or SHA-1.It treats the message digest as a black box. Additionally, it uses the shared symmetric key to encrypt the message digest, which produces the output MAC. This is shown in Fig.below.

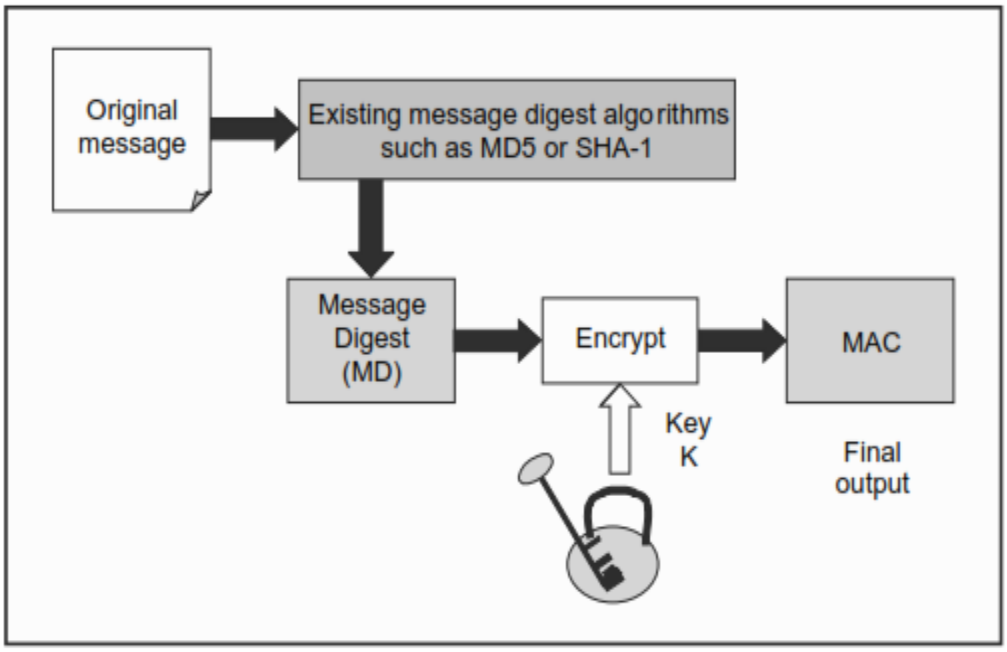


Fig. *HMAC concept*

Fig.3.14: HMAC Concept

How HMAC Works?

Let us now take a look at the internal working of HMAC. For this, let us start with the various variables that will be used in our HMAC discussion. MD = The message digest/hash function used (e.g. MD5, SHA-1, etc.) M = The input message whose MAC is to be calculated L = The number of blocks in the message M b = The number of bits in each block K = The shared symmetric key to be used in HMAC ipad = A string 00110110 repeated b/8 times opad = A string 01011010 repeated b/8 times

Step 1: Make the length of K equal to b

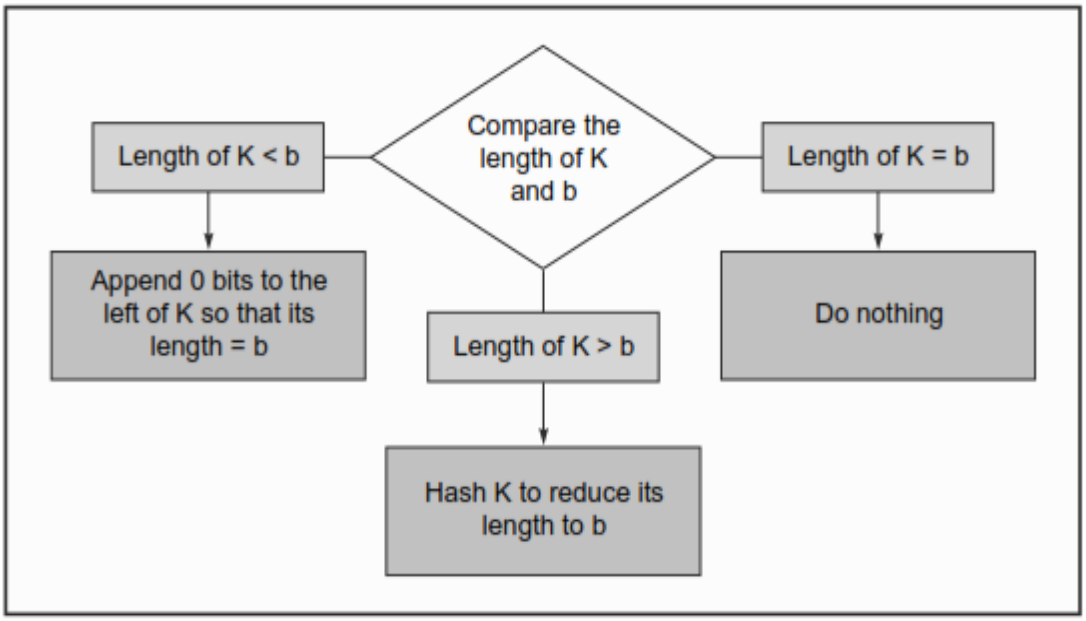


Fig. *Step 1 of HMAC*

Fig. 3.14: Step1 of HMAC

Step 2: XOR K with ipad to produce S1 We XOR K (the output of Step 1) and ipad to produce a variable called as S1.

Step 3: Append M to S1 We now take the original message (M) and simply append it to the end of S1 (which was calculated in Step 2).

Step 4: Message digest algorithm Now, the selected message digest algorithm (e.g. MD5, SHA-1, etc) is applied to the output of Step 3 (i.e. to the combination of S1 and M). Let us

call the output of this operation as H.

Step 5: XOR K with opad to produce S2 Now, we XOR K (the output of Step 1) with opad to produce a variable called as S2.

Step 6: Append H to S2 In this step, we take the message digest calculated in step 4 (i.e. H) and simply append it to the end of S2 (which was calculated in Step 5).

Step 7: Message digest algorithm Now, the selected message digest algorithm (e.g. MD5, SHA-1, etc) is applied to the output of Step 6 (i.e. to the concatenation of S2 and H). This is the final MAC that we want.

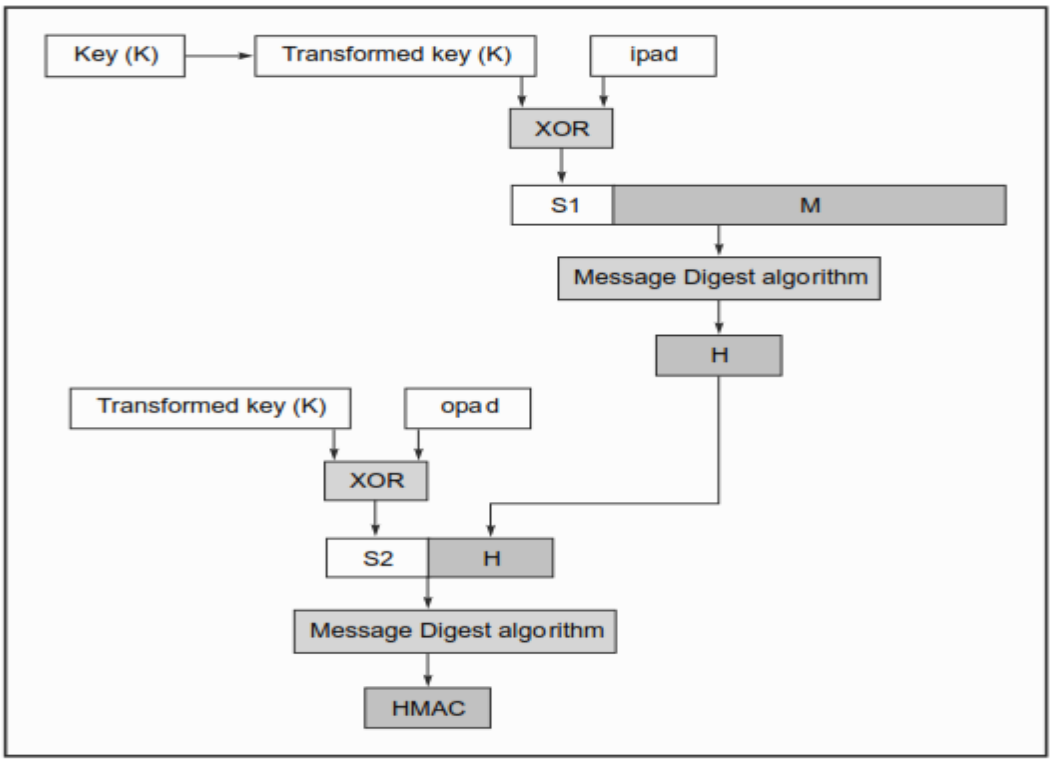


Fig. *Complete HMAC operation*

Fig.3.15: Complete HMAC Operation

3.8: DIGITAL SIGNATURES:

These are the public-key primitives of message authentication. In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to bind signatory to the message. Similarly, a digital signature is a technique that binds a person/entity to the digital data. This binding can be independently verified by receiver as well as any third party. Digital signature is a cryptographic value that is calculated from the data and a secret key known only by the signer.

The exchange of data is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces. In real world, the receiver of message needs assurance that the message belongs to the sender and he should not be able to repudiate the origination of that message. This requirement is very crucial in business applications, since likelihood of a dispute over exchanged data is very high.

As mentioned earlier, the digital signature scheme is based on public key cryptography. The model of digital signature scheme is depicted in the following illustration.

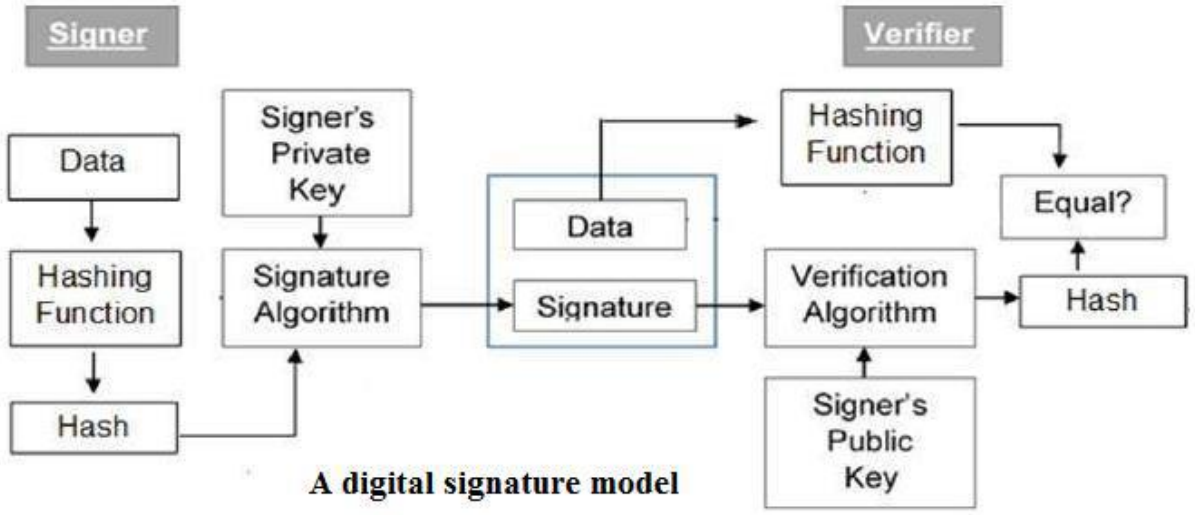


Fig 3.16: A digital Signature Model

The following points explain the entire process in detail −

- Each person adopting this scheme has a public-private key pair.
- Generally, the key pairs used for encryption/decryption and signing/verifying are different. The private key used for signing is referred to as the signature key and the public key as the verification key.
- Signer feeds data to the hash function and generates hash of data.
- Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash. Signature is appended to the data and then both are sent to the verifier. Verifier feeds the digital signature and the verification key into the verification algorithm. The verification algorithm gives some value as output.
- Verifier also runs same hash function on received data to generate hash value.
- For verification, this hash value and output of verification algorithm are compared.

Based on the comparison result, verifier decides whether the digital signature is valid.

- Since digital signature is created by 'private' key of signer and no one else can have this key; the signer cannot repudiate signing the data in future.

It should be noticed that instead of signing data directly by signing algorithm, usually a hash of data is created. Since the hash of data is a unique representation of data, it is sufficient to sign the hash in place of data. The most important reason of using hash instead of data directly for signing is efficiency of the scheme.

Let us assume RSA is used as the signing algorithm. As discussed in public key encryption chapter, the encryption/signing process using RSA involves modular exponentiation.

Signing large data through modular exponentiation is computationally expensive and time consuming. The hash of the data is a relatively small digest of the data, hence signing a hash is more efficient than signing the entire data.

3.8.1: DIGITAL SIGNATURE REQUIREMENTS

We can formulate the following requirements for a 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 relatively easy to produce the digital signature.

• It must be relatively easy to recognize and verify the digital signature.

• It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.

• It must be practical to retain a copy of the digital signature in storage.

3.9: KERBEROS

Many real-life systems use an authentication protocol called as Kerberos, to allow the workstations to allow network resources in a secure manner. The name Kerberos signifies a multi-headed dog in the Greek mythology (apparently used to keep outsiders away). Version 4 of Kerberos is found in most practical. implementations. However, Version 5 is also in use

now.

3.9.1 How does Kerberos Work?

There are four parties involved in the Kerberos protocol:

Alice: The client workstation
Authentication Server (AS): Verifies (authenticates) the user during login
Ticket Granting Server (TGS): Issues tickets to certify proof of identity
Bob: The server offering services such as network printing, file sharing or an application program

The job of AS is to authenticate every use at the login time. AS shares a unique secret password with every user.
The job of TGS is to certify to the servers in the network that a user is really what she claims to be. For proving this, the mechanism of tickets (which allow entry into a
Server, just as a ticket allows parking a car or entering a music concert) is used.

Step 1: Login To start with, Alice, the user, sits down at an arbitrary public workstation and enters her name. The work station sends her name in plain text to the AS.

In response, the AS performs several actions. It first creates a package of the user name (Alice) and a randomly generated session key (KS). It encrypts this package with the symmetric key that the AS shares with the Ticket Granting Server (TGS). The output of this step is called as the Ticket Granting Ticket (TGT). Note that the TGT can be opened only by the TGS, since only it possesses the corresponding symmetric key for decryption. The AS then combines the TGT with the session key (KS), and encrypts the two together using a symmetric key derived from the password of Alice (KA). Note that the final output can, therefore, be opened only by Alice.

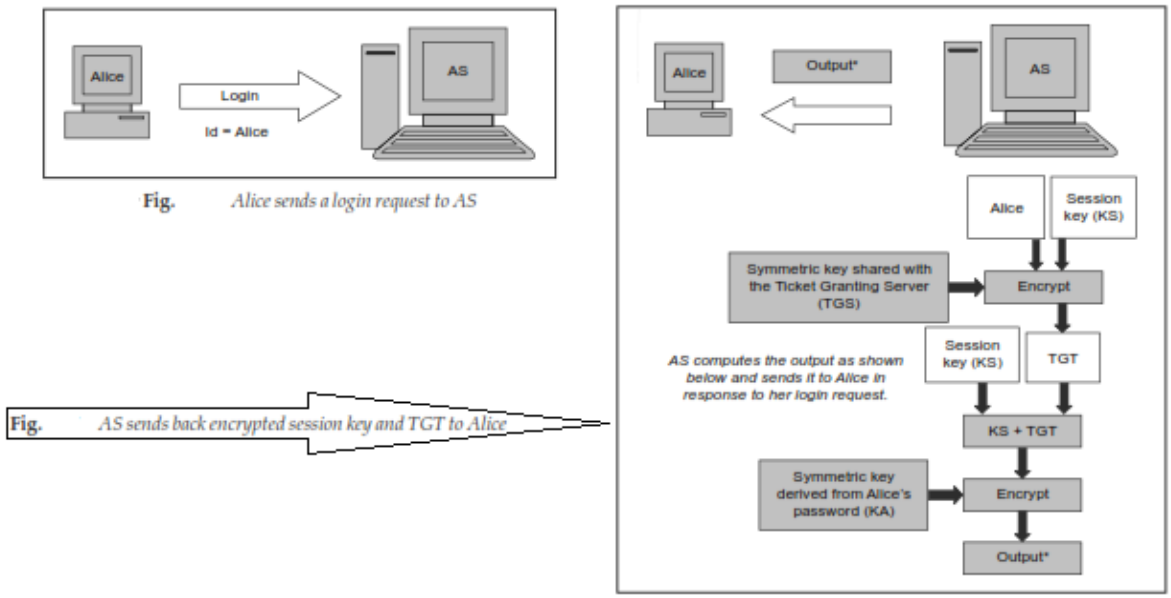


Fig 3.17: Kerberos

Step 2: Obtaining a service granting ticket (SGT) Now, let us assume that after a successful login, Alice wants to make use of Bob – the email server, for some email communication. For this, Alice would inform her workstation that she needs to contact Bob. Therefore, Alice needs a ticket to communicate with Bob. At this juncture, Alice's workstation creates a message intended for the Ticket Granting Server (TGS), which contains the following items:

• The TGT as in step 1
• The id of the server (Bob) whose services Alice is interested in
• The current timestamp, encrypted with the same session key (KS)

As we know, the TGT is encrypted with the secret key of the Ticket Granting Server (TGS). Therefore, only the TGS can open it. This also serves as a proof to the TGS that the message indeed came from Alice. Why? This is because, if you remember, the TGT was created by the AS (remember that only the AS and the TGS know the secret key of TGS). Furthermore, the TGT and the KS were encrypted together by the AS with the secret key derived from the password of Alice. Therefore, only Alice could have opened that package and retrieved the TGT. Once the TGS is satisfied of the credentials of Alice, the TGS creates a session key KAB, for Alice to have secure communication with Bob. TGS sends it twice to Alice: once combined

with Bob's id (Bob) and encrypted with the session key (KS) and a second time, combined with Alice's id (Alice) and encrypted with Bob's secret key (KB). This is shown in Fig. below.

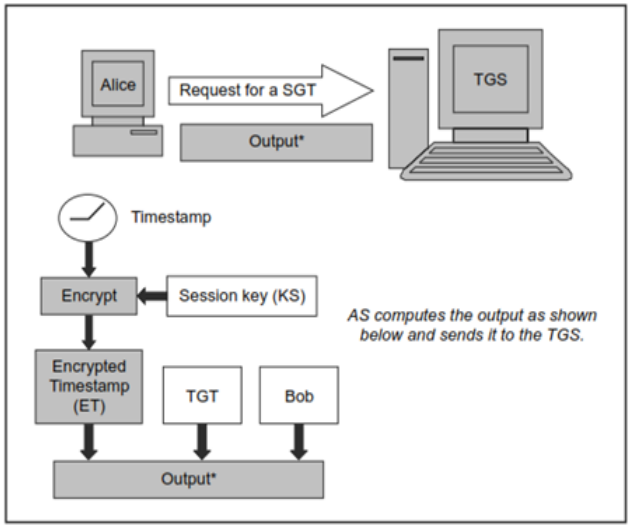


Fig. *Alice sends a request for a SGT to the TGS*

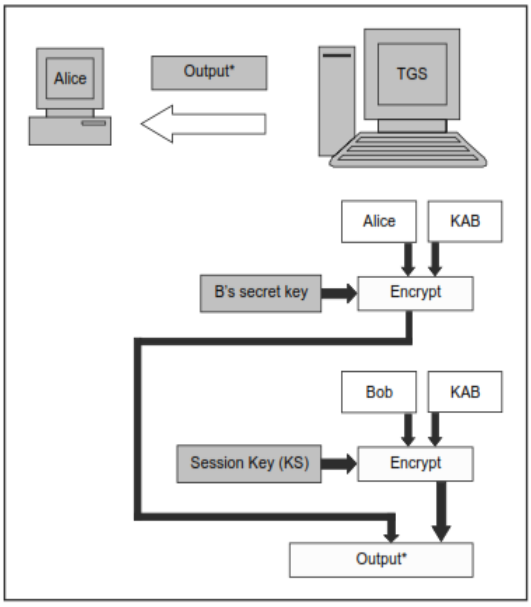


Fig. *TGS sends response back to Alice*

Fig 3.8: Kerberos Working

Step 3: User contacts Bob for accessing the server Alice can now send KAB to Bob in order to enter into a session with him. Since this exchange is also desired to be secure, Alice can simply forward KAB encrypted with Bob's secret key (which she had received from the TGS in the previous step) to Bob. This will ensure that only Bob can access KAB. Furthermore, to guard against replay attacks, Alice also sends the timestamp, encrypted with KAB to Bob. Since only Bob has his secret key, he uses it to first obtain the information (Alice + KAB). From this, it gets the key KAB, which he uses to decrypt the encrypted timestamp value.

Now how would Alice know if Bob received KAB correctly or not? In order to satisfy this query, Bob now adds 1 to the timestamp sent by Alice, encrypts the result with KAB and sends it back to Alice. This is shown in Fig. above. Since only Alice and Bob know KAB, Alice can open this packet and verify that the timestamp incremented by Bob was indeed the one sent by her to Bob in the first place. Now, Alice and Bob can communicate securely with each other. They would use the shared secret key KAB to encrypt messages before sending and also to decrypt the encrypted messages received from each other.

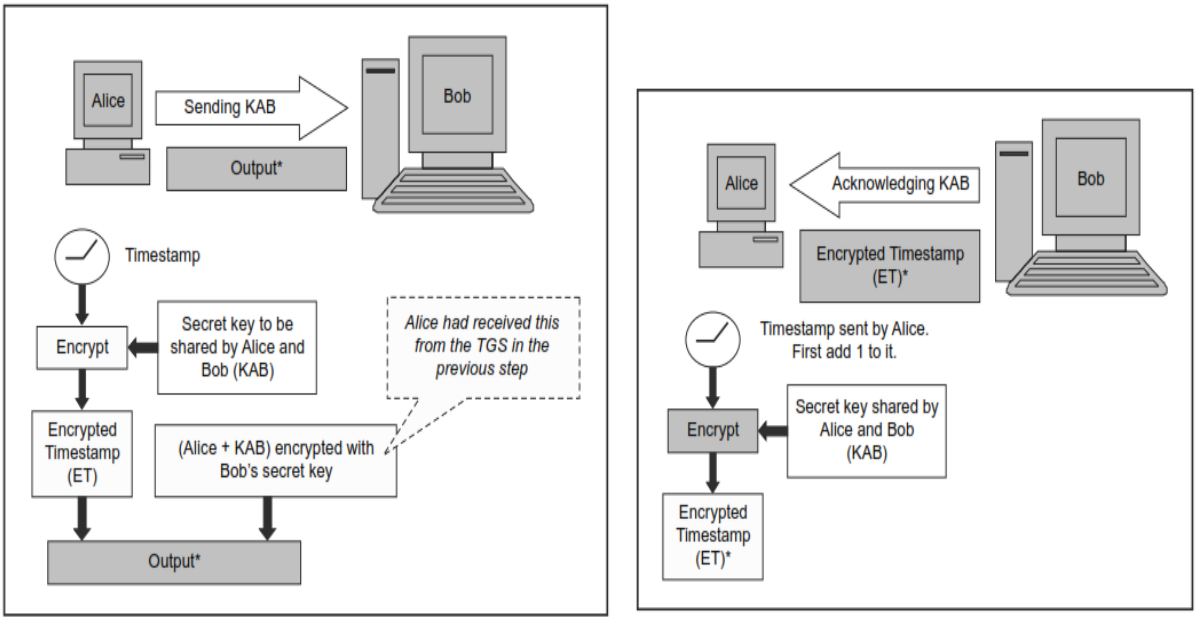


Fig. Alice sends KAB securely to Bob

Fig. Bob acknowledges the receipt of KAB

Fig.3.19: Acknowledgement

3.10: X.509 CERTIFICATES

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates. X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME, IP Security, and SSL/TLS Certificates The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates

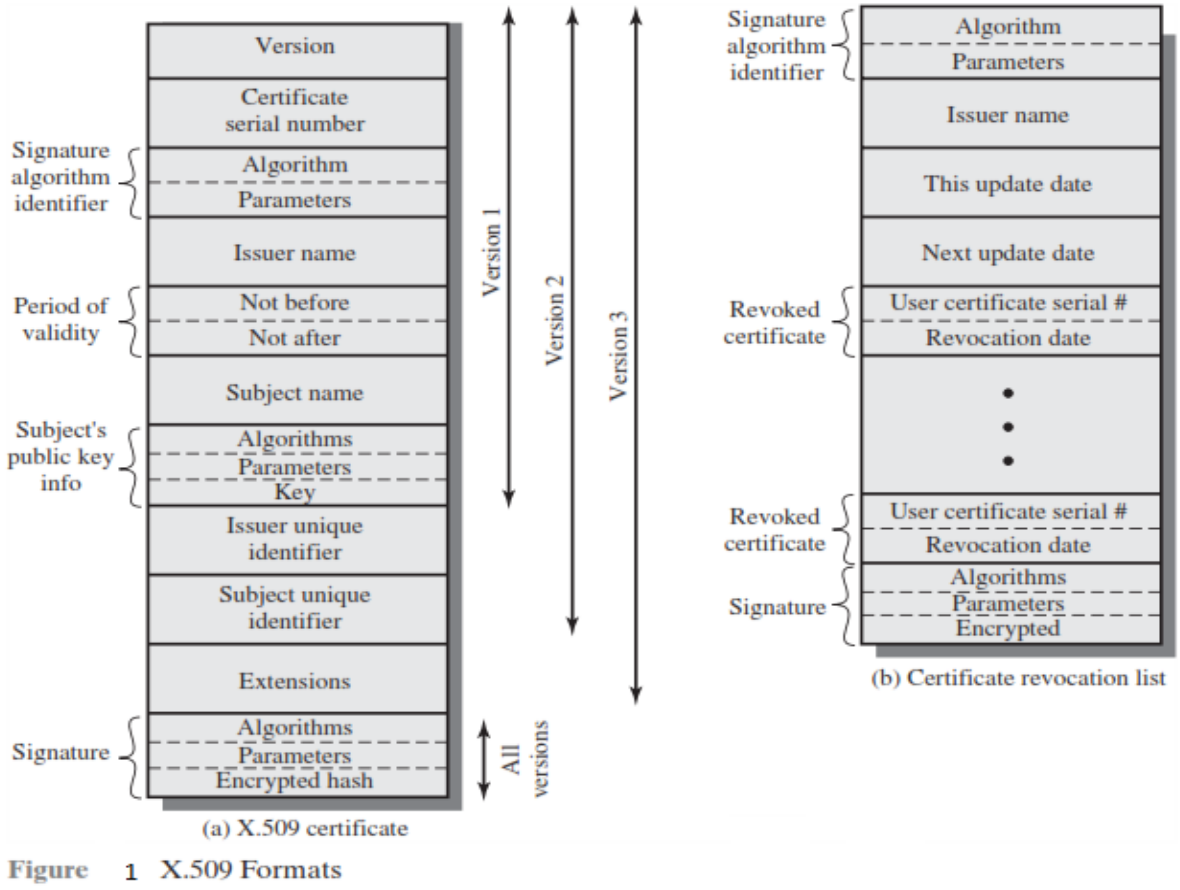


Fig 3.20: General form of a certificate

The above figure shows the general format of a certificate, which includes the following elements.

• **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

• **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.

• **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.

• **Issuer name:** X.500 name of the CA that created and signed this certificate.

• Period of validity: Consists of two dates: the first and last on which the certificate is valid.

• **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

• **Subject's public-key information:** The public key of the subject, plus an identifier of the

algorithm for which this key is to be used, together with any associated parameters.

• **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.

• **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.

• **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.

• **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier.

3.11: PUBLIC-KEY INFRASTRUCTURE:

Public-key infrastructure (PKI) is the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

• **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.

• **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more registration authorities.

• **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end entity registration process, but can assist in a number of other areas as well

. • **CRL issuer:** An optional component that a CA can delegate to publish CRLs.

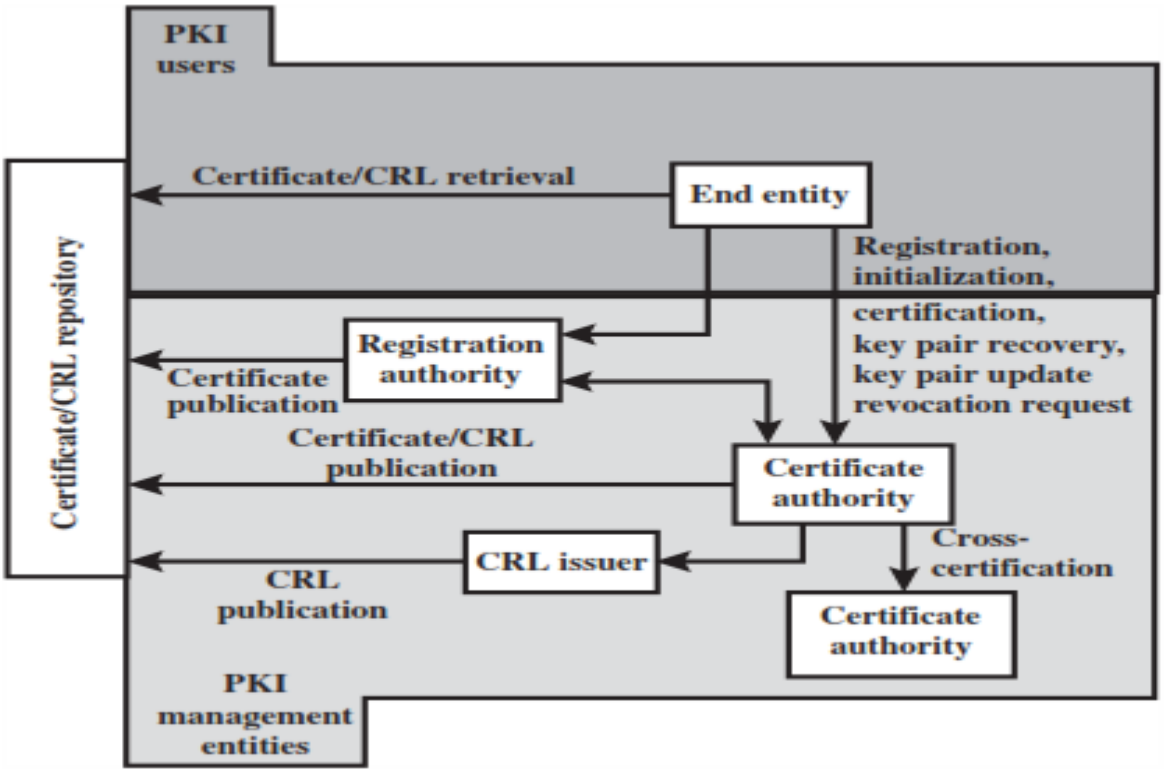


Figure 3 PKIX Architectural Model

Fig. 3.21: PKIX Architectural Model

Figure 3.21 shows the interrelationship among the key elements of the PKIX model. These elements are

• **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities. PKIX Management Functions PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 3 and include the following:

• **Registration:** This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some off-line or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.

• **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s) to be used in validating certificate paths.

• **Certification:** This is the process in which a CA issues a certificate for a user's public key and returns that certificate to the user's client system and/or posts that certificate in a repository.

• **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).

• **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.

• **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.

• **Cross certification:** Two CAs exchange information used in establishing a cross certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.