

Unit-4

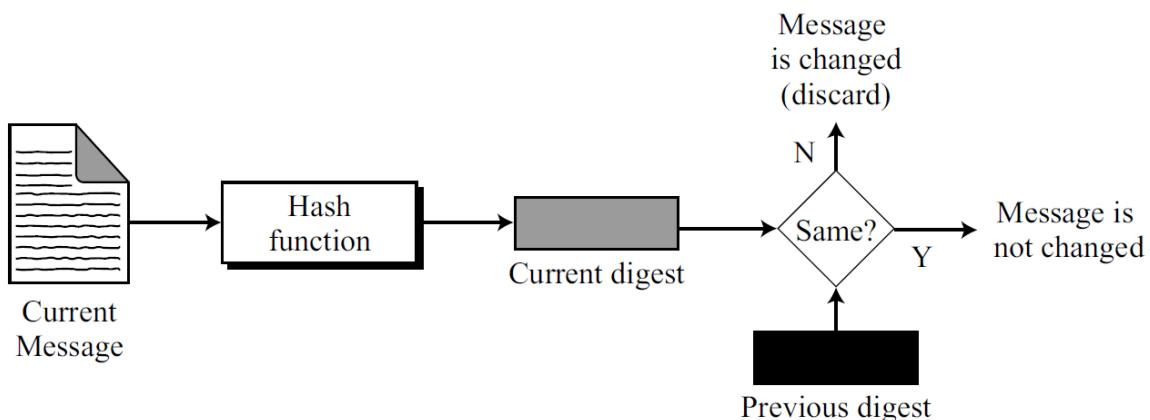
Syllabus: Data Integrity: Introduction, Message Integrity, Cryptographic Hash Functions, Message Authentication, Digital Signature, Key Management

4.1 Introduction to Data Integrity

Symmetric and Asymmetric Cryptosystems are used to achieve the confidentiality of information. When sensitive information is exchanged, the receiver must have the assurance that the message has come intact from the intended sender and is not modified inadvertently or otherwise.

Cryptography systems only provide secrecy, or confidentiality, but not integrity. However, there are occasions where we may not even need secrecy but instead, we need data integrity.

Simple technique to preserve the integrity of a document is done using a message digest. If sender needs to be sure that the contents of the message/document will not be changed, he can create a message digest by using hash function and attaches the message digest to the message/document. Adversary cannot modify the contents of this message/document or create a false document because he cannot forge sender's digest. To ensure that the document has not been changed, receiver creates message digest for the received message/document and compares it with sender's message digest. If they are same, the message/document is from sender, if not same, message/document is not from exact sender or message lost integrity characteristic.



Cryptographic Hash Function Criteria

A cryptographic hash function must satisfy four criteria:

- **Deterministic function:** For a given message M , hash function must generate same message digest every time.
- **Preimage resistance:** A cryptographic hash function must be preimage resistant. Given a hash function ' h ' and $y = h(M)$, it must be extremely difficult for the adversary to find any message, M' , such that $h(M')$ generates the same ' y '.

Preimage Attack

Given: $y = h(M)$

Find: M' such that $y = h(M')$

- **Second preimage resistance:** Second preimage resistance, ensures that, adversary cannot modify the given message to create the original digest. i.e., if adversary knows the original message and message digest, he cannot modify the original message to create the same message digest.

Second Preimage Attack

Given: M and h(M)

Find: M' ≠ M such that h(M) = h(M')

- **Collision resistance:** collision resistance, ensures that Eve cannot find two messages that hash to the same digest.
-

Collision Attack

Given: none

Find: M' ≠ M such that h(M) = h(M')

4.2 Message Integrity

MD5 Hash Function

The MD5 algorithm is a widely used hash function that produces a 128-bit (16-byte) hash value. It's commonly used to check the integrity of files/data. Designed by Ronald Rivest in 1991 to replace the older hash function MD4, and it was defined in RFC 1321 in 1992.

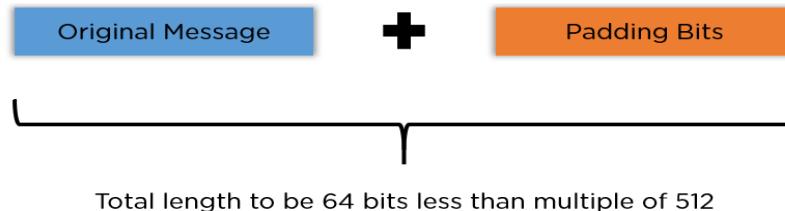
MD5 hash function takes 512-bit block input and generates a fixed size 128bit hash code.



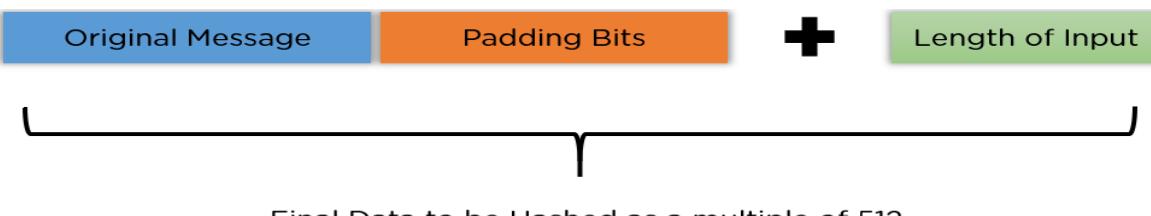
MD5 Hashing Process:

MD5 algorithm processes data in a multi-step procedure that includes padding, appending length, initializing buffers, and manipulating 512-bit block.

1. **Padding Bits:** The input data is padded with bits to ensure its length is equal to 448 modulo 512. Padding always begins with a single '1' bit followed by enough '0' bits to meet the required length.



2. **Appending Length:** A 64-bit representation of the original input length is appended to the result of the padded input. This ensures the total length is exactly a multiple of 512 bits.



3. **Initialize MD Buffer (Initialization Vector):** Four buffers (A, B, C, D) of 32-bit each are initialized with specific hexadecimal values. These buffers are used in the processing of each 512-bit block.

$$A = \boxed{01 \ 23 \ 45 \ 67}$$

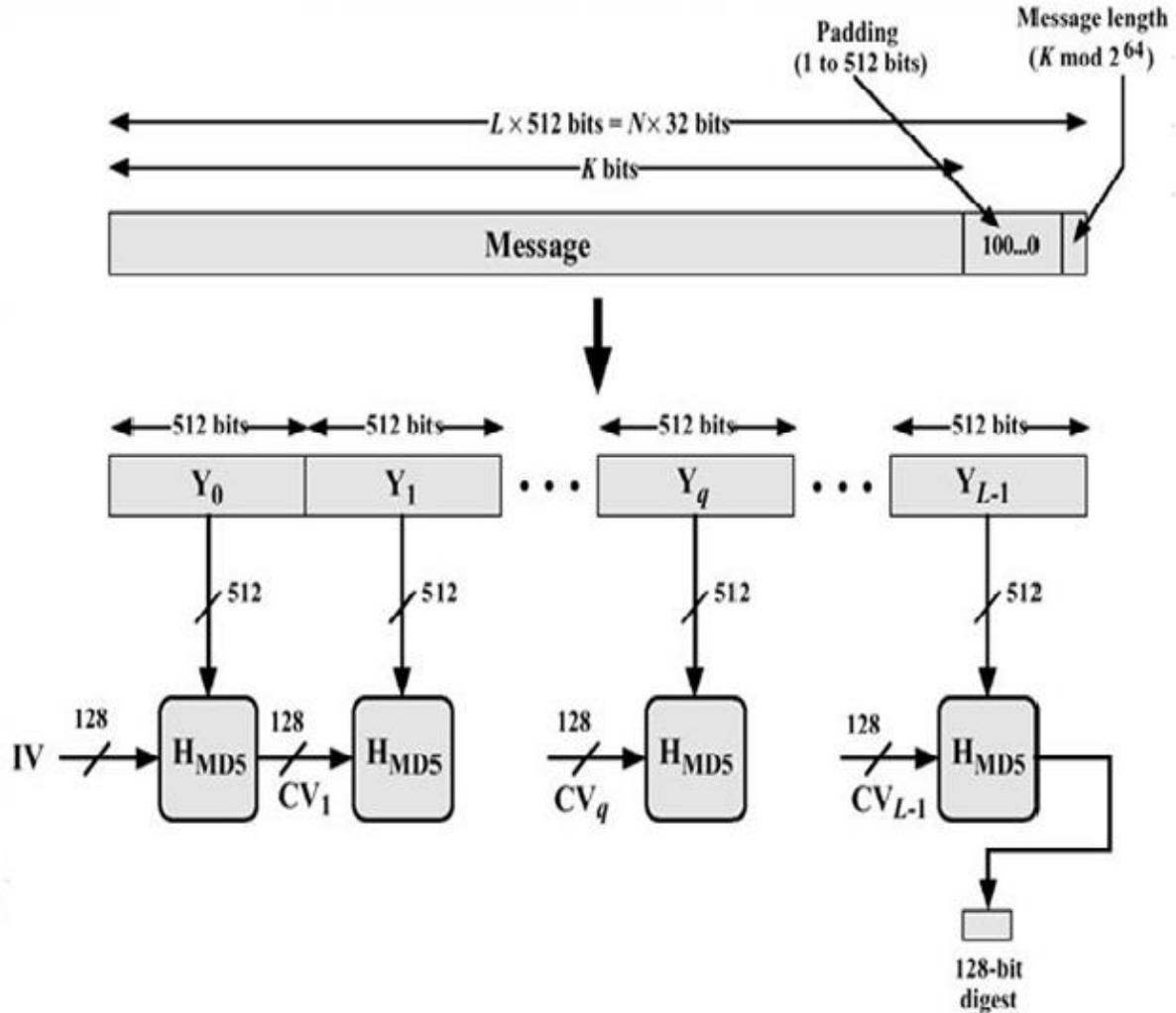
$$C = \boxed{FE \ DC \ BA \ 98}$$

$$B = \boxed{89 \ AB \ CD \ EF}$$

$$D = \boxed{76 \ 54 \ 32 \ 10}$$

4. Process Each 512-bit Block: The main MD5 algorithm operates on each 512-bit block in four rounds, each consisting of 16 operations. Different non-linear functions (F, G, H, I) are applied in each round, and function involves bitwise operations (AND, OR, XOR, NOT), addition modulo (2^{32}), and left bit rotations(<<< by s bits).

Below figure depicts the complete **structure of MD5 Hash algorithm**:



Each 512-bit block conversion process: Every 512-bit block is divided into 16 parts of 32-bit each. This step contains 4-rounds, each round uses separate non-linear function, and each rounds performs 16 operations for each 32-bit part of 512-bit block.

Each round utilizing all the 32-bit sub-blocks of input, the initialization buffers, and a constant array value.

Constant array can be denoted as $T[1] \rightarrow T[64]$.

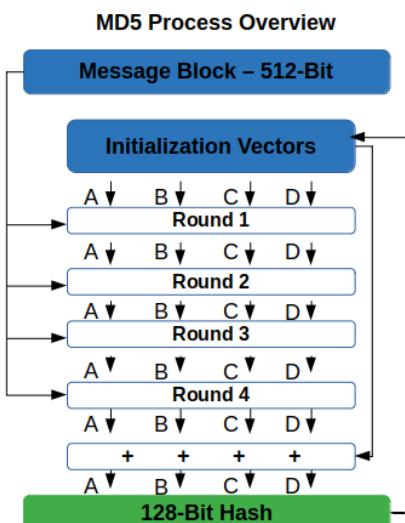
Each of the sub-blocks are denoted as M : $M[0] \rightarrow M[15]$.

Round 1: F, T[1-16], M_i

Round 2: G, T[17-32], M_i

Round 3: H, T[33-48], M_i

Round 4: I, T[49-64], M_i



Round Functions:

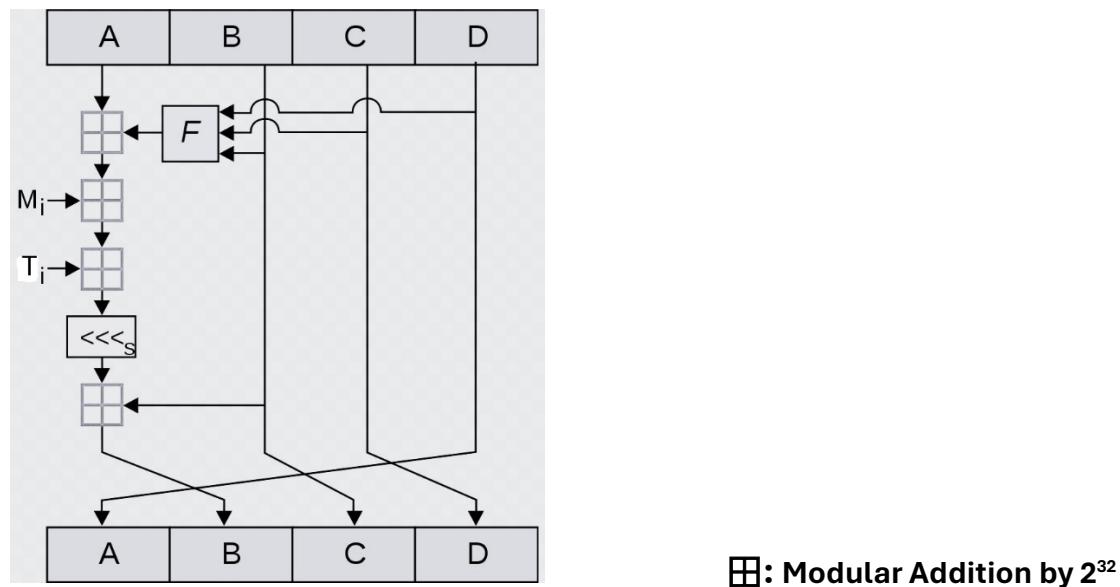
$$F = (B \wedge C) \vee (\neg B \wedge D) \quad \wedge \text{ Logical AND}$$

$$G = (B \wedge D) \vee (C \wedge \neg D) \quad \vee \text{ Logical OR}$$

$$H = B \oplus C \oplus D \quad \oplus \text{ XOR}$$

$$I = C \oplus (B \vee \neg D) \quad \neg \text{ Logical NOT}$$

Round Function Process:



In Round Function F: $A = B \boxplus ((A \boxplus F(B, C, D)) \boxplus M_i \boxplus T_i)$)<<< by S bits

- A is copied into B, B is copied into C, C is copied into D, D is copied into A. This process is repeated for 16 times for each part of 512-bit Block of input.

In Round Function G: $A = B \boxplus ((A \boxplus G(B, C, D)) \boxplus M_i \boxplus T_i)$)<<< by S bits

- A is copied into B, B is copied into C, C is copied into D, D is copied into A. This process is repeated for 16 times for each part of 512-bit Block of input.

In Round Function H: $A = B \boxplus ((A \boxplus H(B, C, D)) \boxplus M_i \boxplus T_i)$)<<< by S bits

- A is copied into B, B is copied into C, C is copied into D, D is copied into A. This process is repeated for 16 times for each part of 512-bit Block of input.

In Round Function I: $A = B \boxplus ((A \boxplus I(B, C, D)) \boxplus M_i \boxplus T_i)$)<<< by S bits

- A is copied into B, B is copied into C, C is copied into D, D is copied into A. This process is repeated for 16 times for each part of 512-bit Block of input.

After 16 operations in each round, i.e., after 64 operations, final values of A, B, C, D vectors are added (\boxplus) with Initialization Vector. The resultant A, B, C, D vectors become input to next 512-bit block (i.e., Initialization vector for next 512-bit block).

Note: In 2004, collisions were found in MD5. An analytical attack was reported to be successful only in an hour by using computer cluster. This collision attack resulted in compromised MD5 and hence it is no longer recommended for use.

4.3 Cryptographic Hash Functions

The Secure Hash Algorithm (SHA) is a family of cryptographic hash functions designed to provide a secure means of verifying the integrity of data. The SHA algorithm takes an input and produces a fixed-size hash value, often referred to as a message digest. This digest is typically rendered as a hexadecimal number.

SHA-0: The original version is SHA-0, a 160-bit hash function, was published by NIST in 1993. It had few undisclosed weaknesses and did not become very popular. Later in 1995, SHA-1 was designed to correct the weaknesses of SHA-0.

SHA-1 Algorithm: SHA-1 is one of the earliest versions of the SHA algorithm, producing a 160-bit hash value. It was developed by the National Security Agency (NSA) and has been widely used for various security applications. However, SHA-1 has been considered insecure since 2005, and major technology companies have phased out its use in SSL certificates.

SHA-2 and SHA-256: SHA-2 is a successor to SHA-1 and includes several functions like SHA-224, SHA-256, SHA-384, and SHA-512. SHA-256, specifically, generates a 256-bit hash value and is known for its security and performance. It is part of the SHA-2 family and was also designed by the NSA. SHA-2 remains secure and is widely used in various applications, including SSL certificates and cryptocurrency transactions.

SHA-3: SHA-3 is the latest member of the SHA family, formerly known as Keccak. It was selected through a public competition and supports the same hash lengths as SHA-2. SHA-3's internal structure is significantly different from the rest of the SHA family, offering an alternative to SHA-2 with comparable security levels.

SHA-512 Cryptographic Hash Algorithm

SHA-512 (Secure Hash Algorithm 512) is a cryptographic hash function that converts text of any length into a fixed-size string of 512 bits (64 bytes). It is commonly used for email addresses hashing, password hashing, and digital record verification. SHA-512 ensures data integrity, verifies message authenticity, and protects against data tampering.

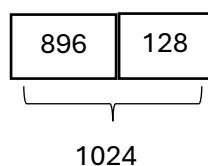
- SHA-512 produces message digest of size 512-bits

SHA-512 algorithm contains the following stages:

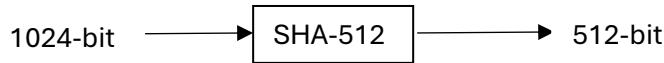
- **Input formatting**
- **Hash buffer initialization**
- **Message Processing**
- **Output**

Input Formatting

Input message to SHA-512 is a block of size 1024-bit. Original message is always 128-bit less than 1024-bit. 128-bit is used to represent the length of the original message.



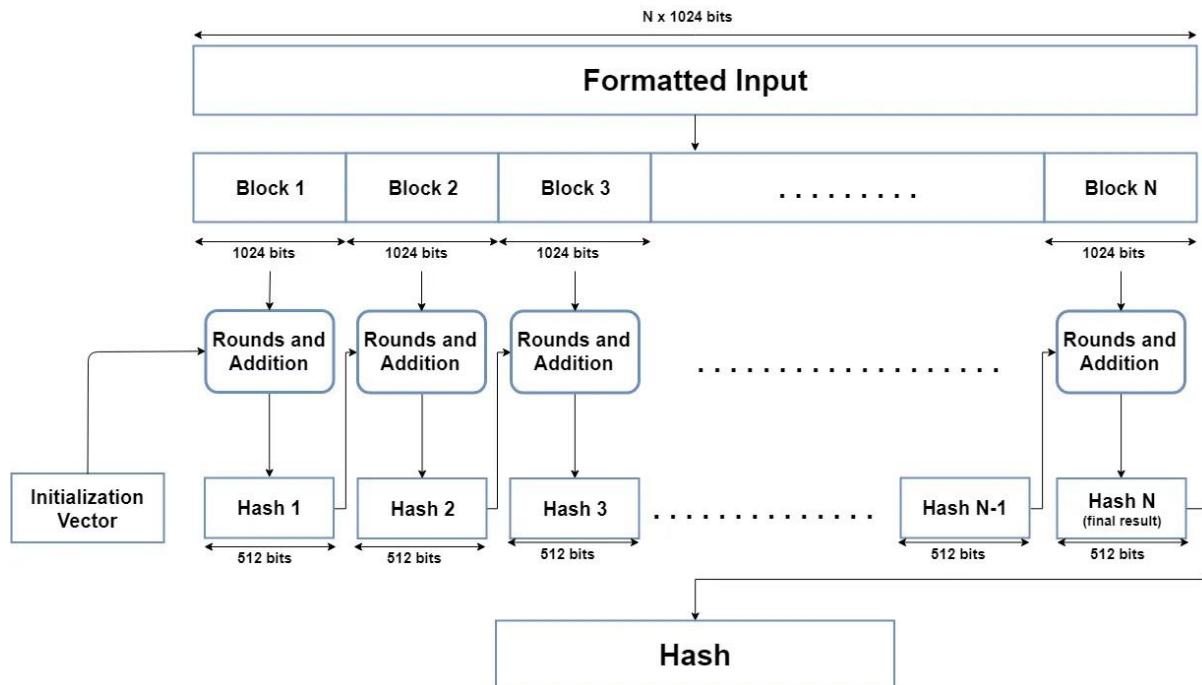
If the original message is less than 896-bits, pad the message with padding bits with the most significant bit as ‘1’ and the remaining bits as ‘0’ depending on the requirement.



After formatting the input with padding and length of message, the complete message contain multiples of 1024.



Structure of SHA-512



Hash buffer initialization (Initialization Vector)

8 hash buffers (a, b, c, d, e, f, g, h) are used to store the intermediate results of SHA-512 algorithm. These intermediate results are given as input to next block hash function.

The algorithm works in a way where each block of 1024 bits are processed with the hash value of the previous block. So previous hash value is stored in 8 hash buffers. But for the First block, 8 hash buffers are filled with default constant values.

Default values are obtained by taking the **first 64 bits of the fractional parts of the square roots of the first 8 prime numbers** (2, 3, 5, 7, 11, 13, 17, 19).

8 hash buffers are named as Register a, Register b, Register c, Register d, Register e, Register f, Register g, Register h.

Default/Initial values are:

a=6A09E667F3BCC908

b=BB67AE8584CAA73B

c=3C6EF372FE94F82B

d=A54FF53A5F1D36F1

$e=510E527FADE682D1$
 $f=9B05688C2B3E6C1F$
 $g=1F83D9ABFB41BD6B$
 $h=5BE0CD19137E2179$

Message Processing

Message processing is done on the formatted input by taking one block of 1024 bits at a time. The actual processing takes three things: 1) The 1024-bit block, and 2) the result of 8 hash buffers from the previous processing, 3) An additive Constant.

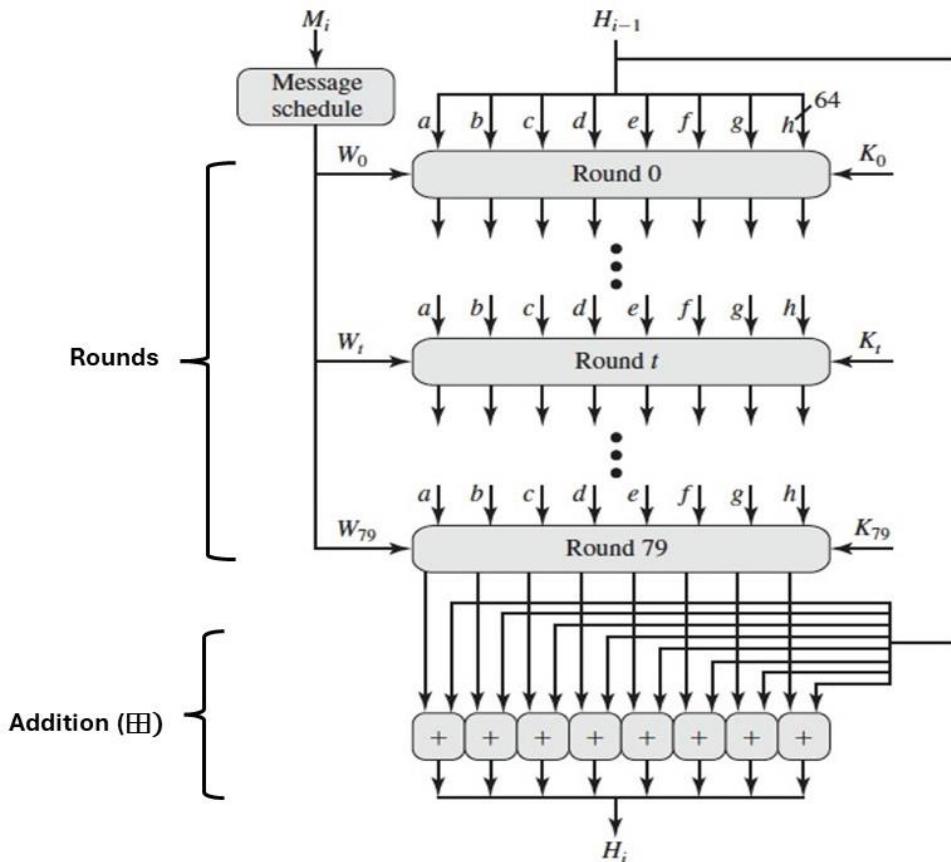
SHA-512 algorithm consists of several ‘**Rounds**’ and an **addition (\oplus by 2^{64})** operation. The Message block (1024 bit) is expanded out into ‘Words’ using a ‘message sequencer’. Eighty Words to be precise, each of them having a size of 64 bits.

So, **SHA-512 contains 80 Rounds**.

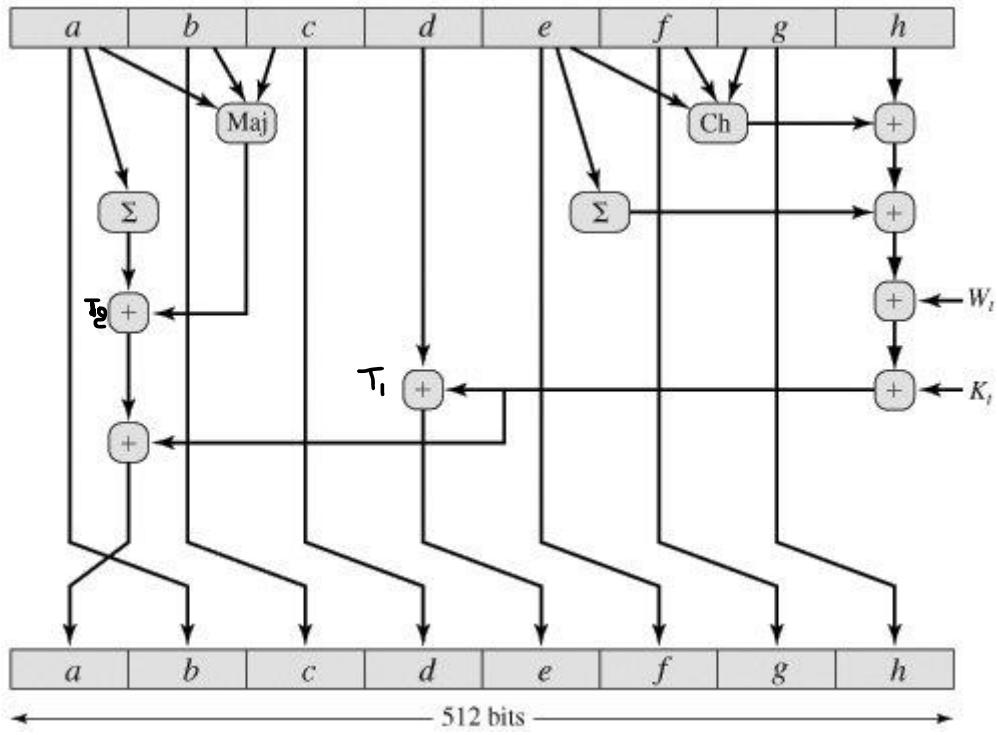
Inputs to each round are:

- 64bit message word (80 message words of 64-bit size are created from 1024-bit block)
 - W_t : W_1 to W_{79}
- An additive constant K_i (These are the **first 64 bits from the fractional part of the cube roots of the first 80 prime numbers**).
 - K_t : K_1 to K_{79}
- **a, b, c, d, e, f, g, h** hash buffers with initial values.

Structure of Message Processing:



Structure of Each Round:



$$T_1 = h \boxplus \text{Ch}(e, f, g) \boxplus \left(\sum_1^{512} e \right) \boxplus W_t \boxplus K_t$$

$$T_2 = \left(\sum_0^{512} a \right) \boxplus \text{Maj}(a, b, c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d \boxplus T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 \boxplus T_2$$

where

t = step number; $0 \leq t \leq 79$

$\text{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$

the conditional function: If e then f else g

$\text{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$

the function is true only if the majority (two or three) of the arguments are true

$\left(\sum_0^{512} a \right) = \text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$

$\left(\sum_1^{512} e \right) = \text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

W_t = a 64-bit word derived from the current 512-bit input block

K_t = a 64-bit additive constant

\boxplus = addition modulo 2^{64}

- Six of the eight hash buffers (b, c, d, f, g, h) of the round function involve simply permutation by means of rotation.
- Only two of the output words (a, e) are generated by substitution.
 - Buffer e is a function of input variables (d, e, f, g, h), as well as the round word W_t and the constant K_t .
 - Buffer a is a function of all of the input variables except d , as well as the round word W_t and the constant K_t .

80 Words creation in SHA-512 Algorithm

1024-bit plaintext message is used to generate 80 words of size 64-bit using the below process.

First 16 words: W_0 to W_{15} directly taken from the 1024-bit, remaining W_{16} to W_{79} are created by using the below Steps:

$$W_t = W_{t-16} \oplus (W_{t-15})\sigma_0^{512} \oplus W_{t-7} \oplus (W_{t-2})\sigma_1^{512}$$

Where,

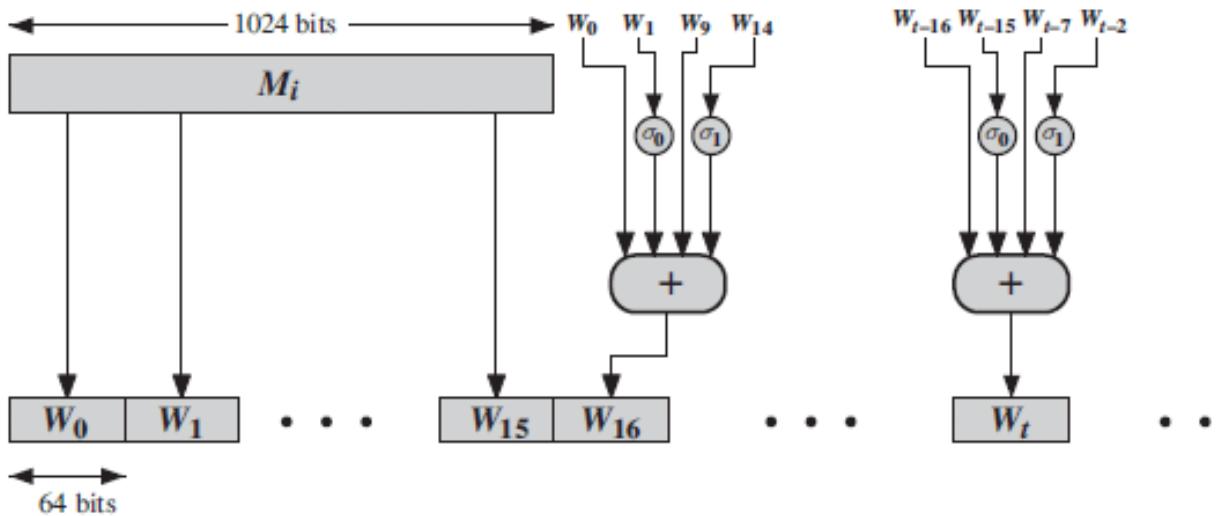
$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$

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

$\text{ROTR}^n(x)$ = Circular right shift of the 64-bit argument x by n bits

$\text{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits

\oplus = Addition modulo by 2^{64}



4.4 Message Authentication

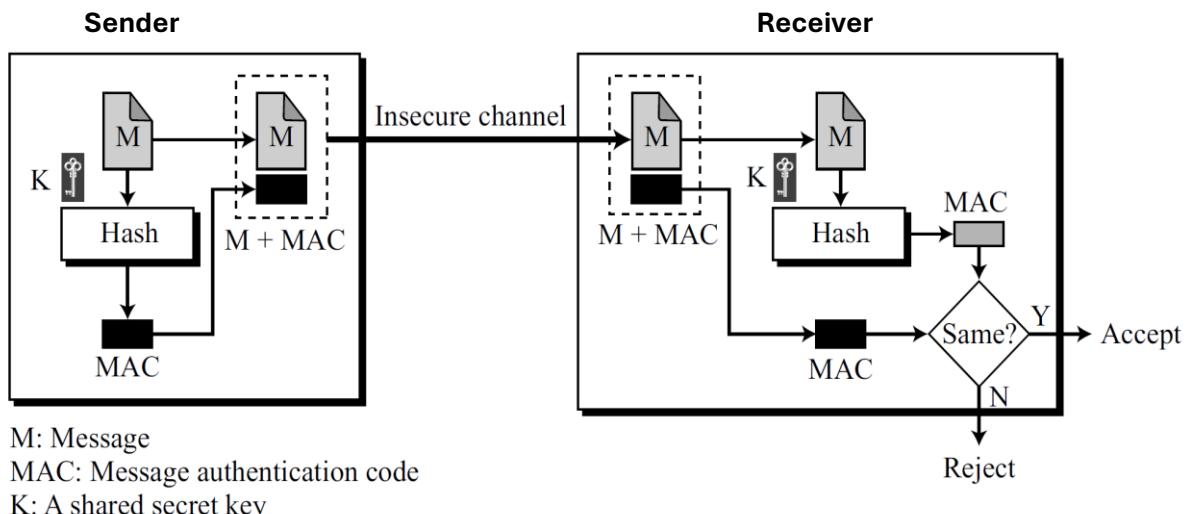
A message digest guarantees the integrity of a message. It guarantees that the message has not been changed. However, message digest does not authenticate the sender of the message.

To provide message authentication, Sender needs to provide a proof that it is he that is sending the message, not an impostor.

Message Authentication Code (MAC): It ensures the integrity of the message and the data origin authentication.

MAC includes a secret information between sender and receiver. For example: a secret key that the adversary does not possess.

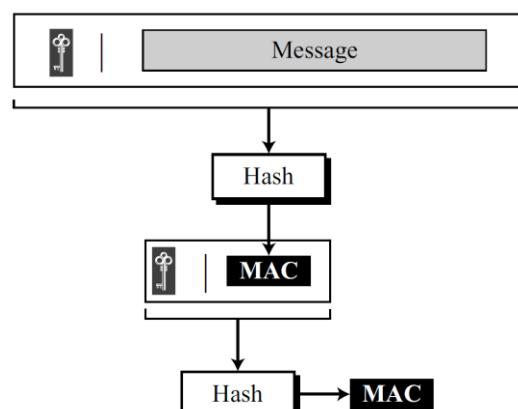
MAC Process:



Sender uses a hash function to create a MAC from the concatenation of the secret key and the message, $h(K|M)$, and sends the message and the MAC to receiver over the insecure channel. After receiving it, receiver separates the message from the MAC. He then makes a new MAC from the concatenation of the message and the secret key, 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.

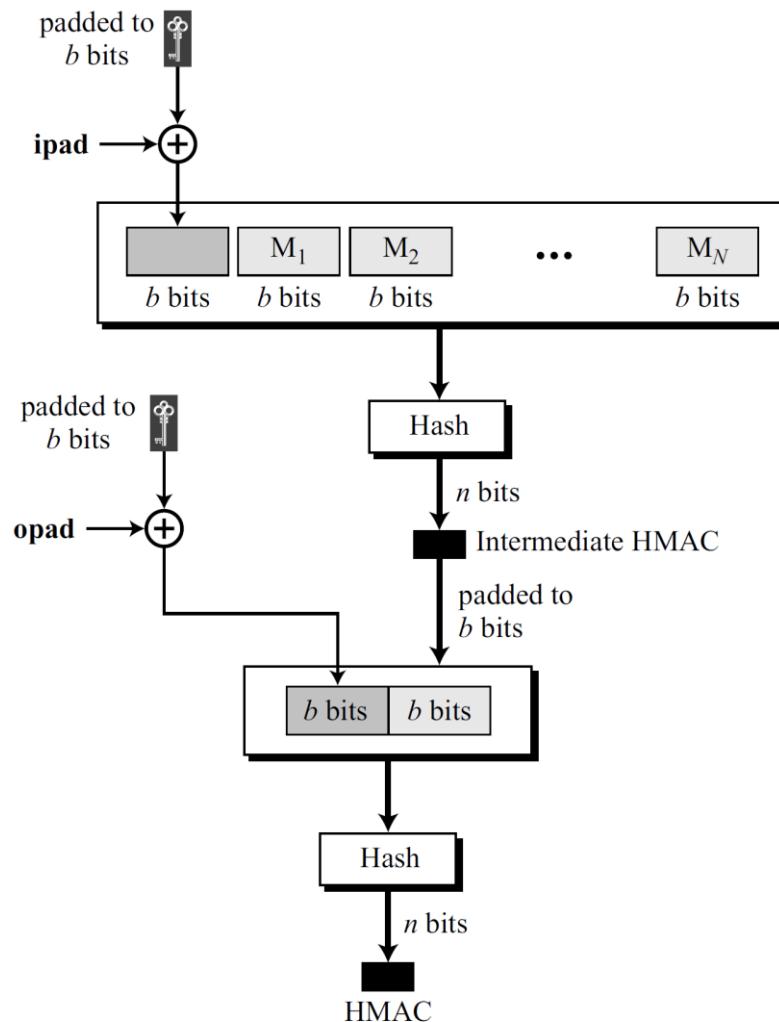
Nested MAC

To improve the security of a MAC, nested MACs were designed in which hashing is done in two steps. In the first step, the key is concatenated with the message and is hashed to create an intermediate digest. In the second step, the key is concatenated with the intermediate digest to create the final digest. General Idea is shown in the below figure.



HMAC

NIST has issued a standard (FIPS 198) for a nested MAC that is often referred to as HMAC (hashed MAC). The implementation of HMAC is much more complex than the simplified nested MAC shown in the Figure:



Steps of HMAC:

1. The message is divided into N blocks, each of b bits. (example: 512-bits)
2. The secret key is left-padded with 0's to create a b -bit key. Note that it is recommended that the secret key (before padding) be longer than n bits, where ' n ' is the size of the HMAC (hash code).
3. The result of step 2 is exclusive-ored (XOR) with a constant called **ipad** (input pad) to create a b -bit block. The value of **ipad** is 00110110 (36 in hexadecimal), $b/8$ repetition of the sequence.
4. The resulting block is prepended to the N -block message. The result is $N + 1$ blocks.
5. The result of step 4 is hashed to create an n -bit digest. We call the digest the intermediate HMAC.
6. The intermediate n -bit HMAC is left padded with 0s to make a b -bit block.

7. Steps 2 and 3 are repeated by a different constant **opad** (output pad). The value of opad is 01011100 (5C in hexadecimal), b/8 repetition of the sequence.

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

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

4.5 Digital Signature

A person signs a document to show that it originated from him/her or approved by him/her. The signature is proof to the recipient that the document comes from the correct entity. Therefore, a signature on a document, when verified, is a sign of authentication.

This kind of Signature is called as Digital Signature.

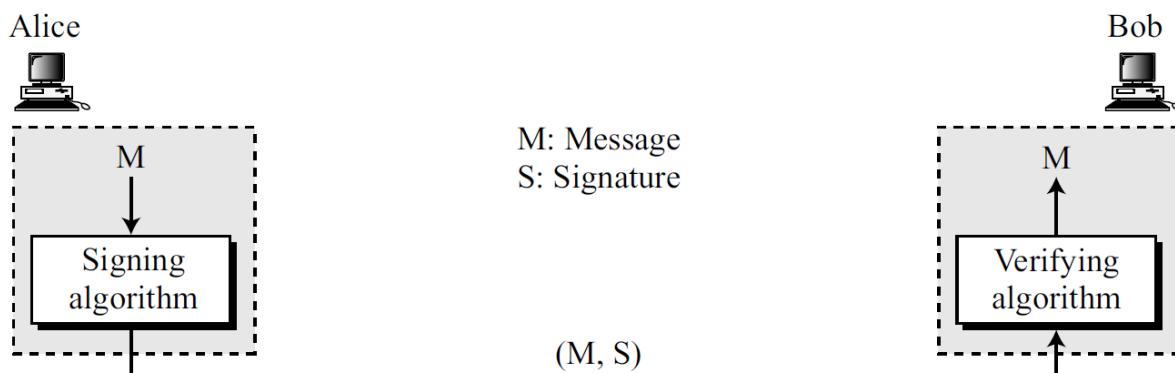
The objective of digital signatures is to authenticate and verify documents and data. This is necessary to avoid tampering and digital modification or forgery during the transmission of official documents.

Differences between conventional signatures and digital signatures

	Conventional Signature	Digital Signature
Inclusion	Signature is included in the document, it is part of the document.	It is sent as a separate document. The sender sends two documents, the message, and the signature.
Verification Method	Received signature is compared with the existing signature.	Receiver receives the message and the signature, needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.
Relationship	One-to-Many relationship between a signature and documents.	One-to-One relationship between a signature and a message. Each message has its own signature.
Duplicity	A copy of the signed document can be distinguished from the original one on file.	There is no such distinction unless there is a factor of time (or a timestamp) on the document.

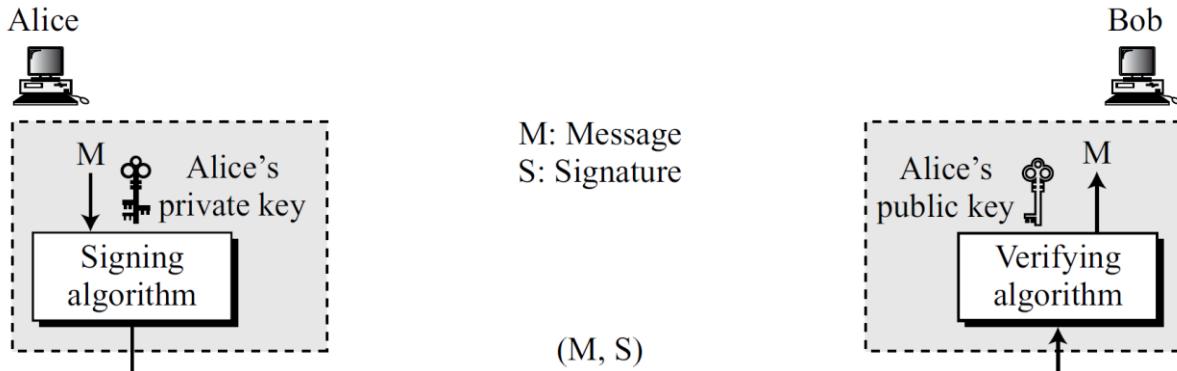
Process:

The sender uses a signing algorithm to sign the message. The message and the signature are sent to the receiver. The receiver receives the message and the signature and applies the verifying algorithm to the combination. If the result is true, the message is accepted; otherwise, it is rejected.



Need for Keys

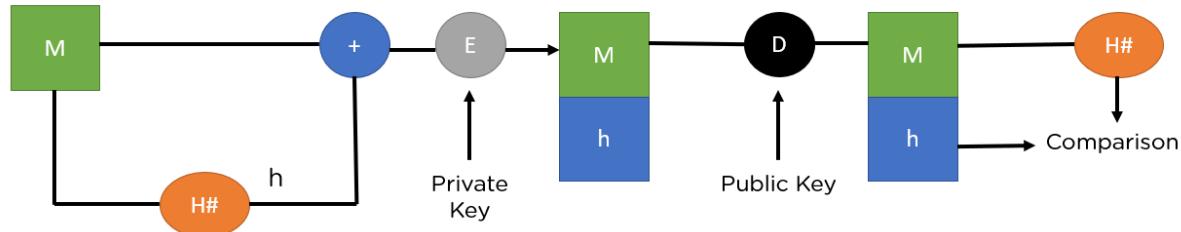
In a digital signature, the signer uses her private key, applied to a signing algorithm, to sign the document. The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document.



Note: A digital signature needs a public-key system. The signer signs with her private key; the verifier verifies with the signer's public key.

Block Diagram of Digital Signature

To implement both Authentication and Integrity check, the following procedure is used.



M - Plaintext

H - Hash function

h - Hash digest

'+' - Bundle both plaintext and digest

E - Encryption

D - Decryption

The above figure shows the entire process of signing and verification. And the steps are:

- Step 1: M , the original message is first passed to a hash function denoted by $H\#$ to create a digest.
- Step 2: Next, it bundles the message together with the hash digest ' h ' and encrypts it using the sender's private key.
- Step 3: Sender sends the encrypted bundle to the receiver, who can decrypt it using the sender's public key.
- Step 4: After decryption, the message is passed through the same hash function ($H\#$), to generate a similar digest.
- Step 5: Receiver compares the newly generated hash with the bundled hash value received along with the message. If they match, it verifies data integrity.

Services

A digital signature can directly provide message authentication, message integrity, and nonrepudiation services.

- **Message Authentication:** A secure digital signature scheme helps the receiver to know the origin of the received message.
- **Message Integrity:** The digital signature schemes today use a hash function in the signing and verifying algorithms that preserve the integrity of the message.
- **Nonrepudiation:** If sender signs a message and then denies it later, Bob later proves that sender signed it by employing a trusted third party.
- **Confidentiality:** A digital signature does not provide confidentiality directly. If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem.
- **Provides Security:** Digital signatures use encryption algorithms to protect the data from unauthorized access and tampering. The cryptographic techniques used by digital signatures also protect the data from being changed or manipulated during transmission.
- **Improves Efficiency:** Digital signatures can reduce the time and money spent on paperwork, printing, scanning, and mailing documents.

Attacks on Digital Signature

Some attacks on digital signatures and the types of forgery.

Attack Types

Key-Only Attack: In this attack, adversary has access only to the public information released by sender. To forge a message, adversary needs to create sender's signature to convince receiver that the message is coming from sender. This is the same as the ciphertext-only attack.

Known-Message Attack: In this attack, adversary has access to one or more message-signature pairs. He tries to create another message and forge sender's signature on it. This is similar to the known-plaintext attack.

Chosen-Message Attack: In this attack, adversary somehow makes sender's sign one or more messages for her. He now has a chosen message/signature pair. Later creates another message, with the content she wants, and forges sender's signature on it. This is similar to the chosen-plaintext attack.

Forgery Types

If the attack is successful, the result is a forgery. We can have two types of forgery: Existential and Selective.

Existential Forgery: In this forgery, adversary may be able to create a valid message-signature pair, but not one that she can really use.

Selective Forgery: In this forgery, adversary may be able to forge sender's signature on a message with the content selectively chosen by him. Although this is beneficial to him, and may be very detrimental to sender, the probability of such forgery is low, but not negligible.

4.5.1 Digital Signature Schemes

There are two industry-standard for creating digital Signature using the above methodology.

- **RSA Digital Signature Scheme**
- **Digital Signature Standard (DSS) Scheme**

Both the algorithms serve the same purpose, but the encryption and decryption functions differ quite a bit.

RSA Digital Signature Scheme

RSA Public-Key Cryptosystem idea can also be used for signing and verifying a message. So, this scheme is called as RA digital signature scheme. It only provides message authentication.

- Private and Public keys of the sender are only used in signing a document.
- Sender uses his/her own private key to sign the document. Receiver uses the sender's public key to verify it.

RSA Signature on the Message

This technique is only used to check whether the message is altered or not with sender authentication without creating message digest.

General idea behind the RSA digital signature scheme

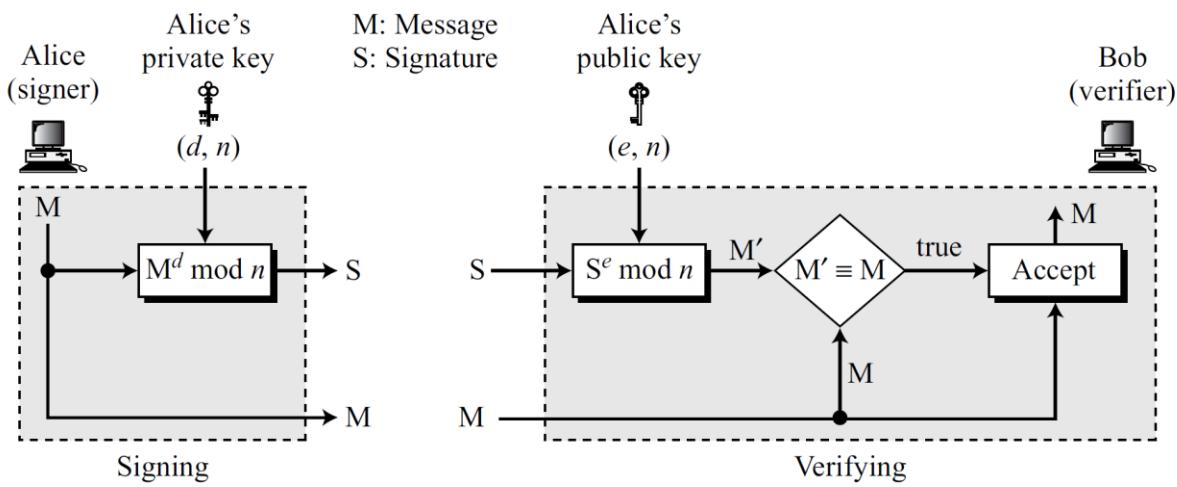
The signing and verifying sites use the same function, but with different parameters. The verifier compares the message and the output of the function for congruence. If the result is true, the message is accepted.

M: Message

S: Signature

(e, n): Alice's public key

(d, n): Alice's private key



Key Generation

Key generation in the RSA digital signature scheme is exactly the same as key generation in the RSA cryptosystem.

Alice chooses two large primes 'p' and 'q' and calculates $n = p \times q$. Alice calculates $\phi(n) = (p - 1)(q - 1)$. She then chooses 'e', the public exponent, and calculates 'd', the private exponent such that $e \times d = 1 \bmod \phi(n)$. Alice keeps d , she publicly announces n and e .

Signing and Verifying

As specified in the above figure.

Signing procedure: Alice creates a signature out of the message using her private exponent, $S = M^d \bmod n$ and sends the message and the signature to Bob.

Verifying Bob receives M and S . Bob applies Alice's public exponent to the signature to create a copy of the message $M' = S^e \bmod n$. Bob compares the value of M' with the value of M . If the two values are congruent, Bob accepts the message.

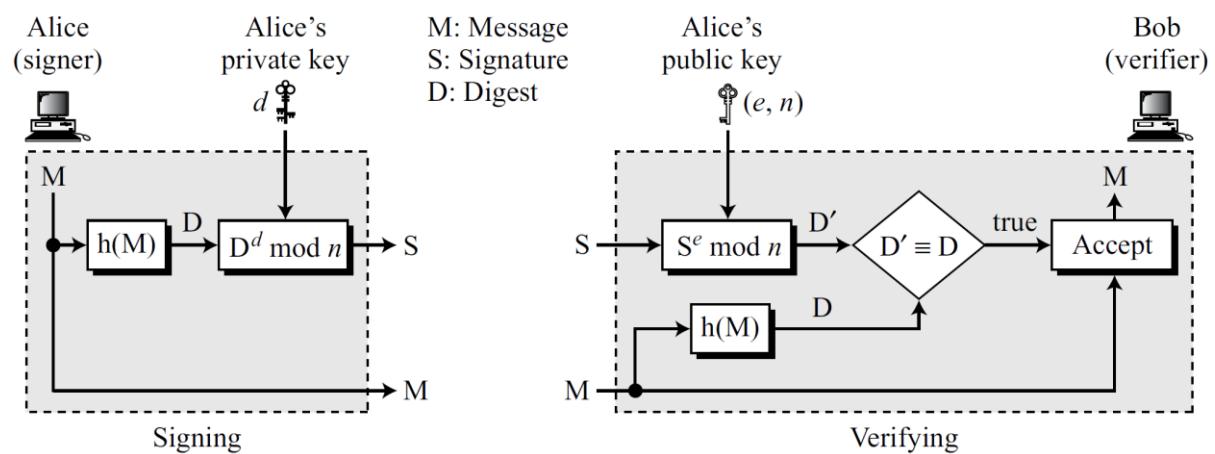
RSA Signature on the Message Digest

This technique provides both authentication and integrity.

Signing a message digest using a strong hash algorithm has several advantages. The use of a strong cryptographic hashing function with RSA digital signature makes the attack on the signature much more difficult.

Procedure:

Alice, the signer, first uses an agreed-upon hash function to create a digest from the message, $D = h(M)$. She then signs the digest, $S = D^d \text{ mod } n$. The message and the signature are sent to Bob. Bob, the verifier, receives the message and the signature. He first uses Alice's public exponent to retrieve the digest, $D' = S^e \text{ mod } n$. He then applies the hash algorithm to the message received to obtain $D = h(M)$. Bob now compares the two digests, D and D' . If they are congruent to modulo n , he accepts the message.



Attacks on RSA Signature

There are some attacks that Eve can apply to the RSA digital signature scheme to forge Alice's signature.

- **Key-Only Attack:** Eve has access only to Alice's public key. Eve intercepts the pair (M, S) and tries to create another message M' such that $M' \equiv S^e \pmod{n}$.
- **Known-Message Attack:** Here Eve uses the multiplicative property of RSA. Assume that Eve has intercepted two message-signature pairs (M_1, S_1) and (M_2, S_2) that have been created using the same private key.
- **Chosen-Message Attack:** This attack also uses the multiplicative property of RSA. Eve can somehow ask Alice to sign two legitimate messages, M_1 and M_2 , for her and later creates a new message $M = M_1 \times M_2$.

Digital Signature Standard (DSS) Scheme

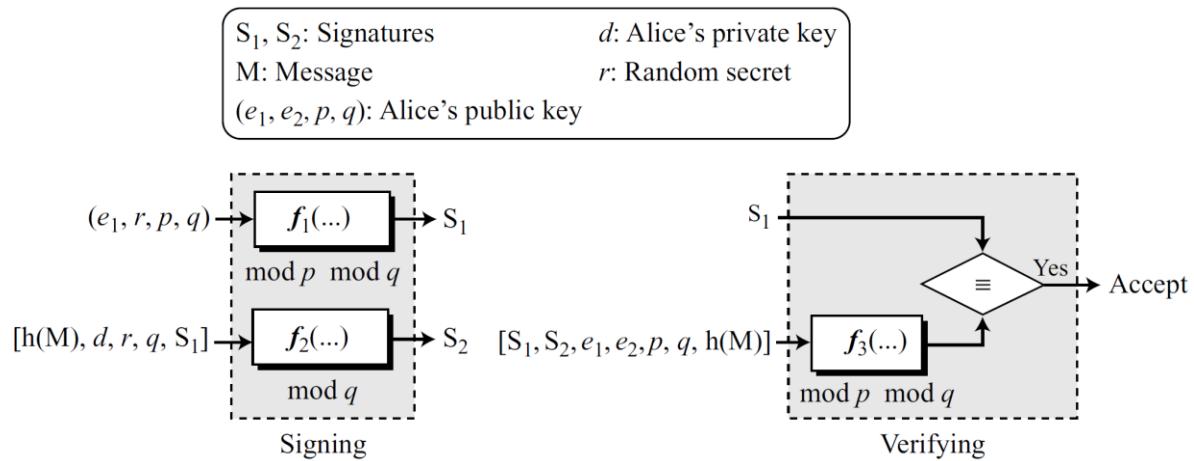
Digital Signature Standard (DSS) was adopted by the National Institute of Standards and Technology (NIST) in 1994. NIST published DSS as FIPS 186.

DSS uses a **Digital Signature Algorithm (DSA)** based on the ElGamal scheme with some ideas from the Schnorr scheme.

General idea of DSS scheme

- Two functions $f_1()$, $f_2()$ are used to create two signatures S_1 , S_2 in the signing process.

- In verifying process, function $f_3()$'s output is compared with S_1 for verification.
- This scheme uses the message digest as part of inputs to functions $f_1()$ and $f_3()$. Two public moduli 'p' and 'q' are used in $f_1()$, $f_3()$ and $f_2()$ uses only 'q'.



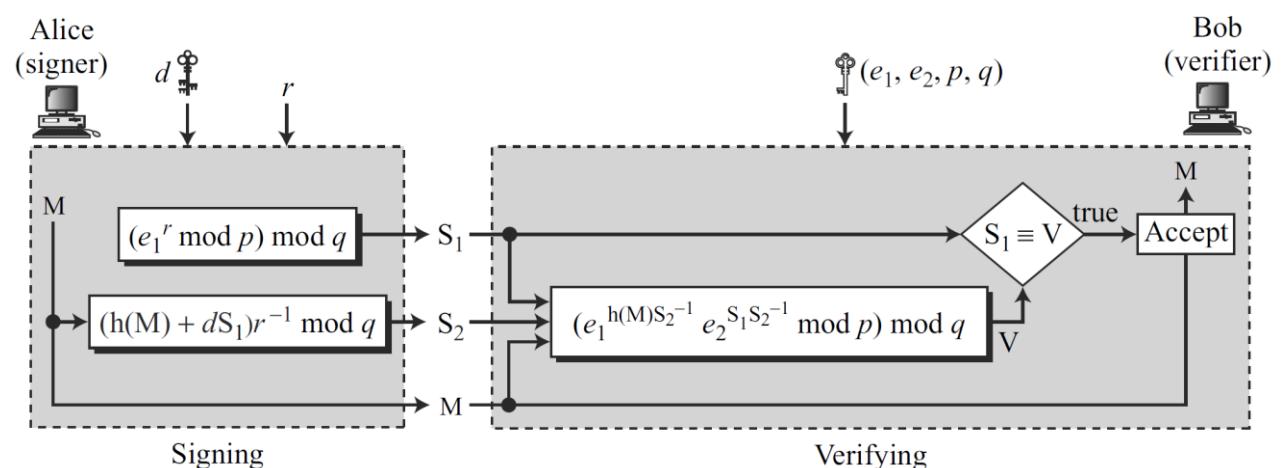
Key Generation Process for DSA

Before signing a message to any entity, Sender needs to generate keys and announce the public key to the public.

1. Alice chooses a prime '**p**', between 512 and 1024 bits in length. The number of bits in '**p**' must be a multiple of 64.
2. Alice chooses a 160-bit prime '**q**' in such a way that '**q** divides (**p** - 1).
3. Alice uses two multiplication groups $\langle \mathbb{Z}_p^*, \times \rangle$ and $\langle \mathbb{Z}_q^*, \times \rangle$ the second is a subgroup of the first.
4. Alice creates e_1 to be the q^{th} root of 1 modulo **p** ($e_1^p = 1 \pmod{p}$). To do so, Alice chooses a primitive element in \mathbb{Z}_p , e_0 , and calculates $e_1 = e_0^{(p-1)/q} \pmod{p}$.
5. Alice chooses **d** as the private key and calculates $e_2 = e_1^d \pmod{p}$.
6. Alice's public key is (e_1, e_2, p, q) ; her private key is (d) .

Signature creation and Verification Process

M: Message	r: Random secret	$h(M)$: Message digest
S_1, S_2 : Signatures	d : Alice's private key	
V: Verification	(e_1, e_2, p, q) : Alice's public key	



Signing

The following shows the steps to sign the message:

1. Alice chooses a random number '**r**' ($1 \leq r \leq q$). Note that although public and private keys can be chosen once and used to sign many messages, Alice needs to select a new '**r**' each time she needs to sign a new message.
2. Alice calculates the first signature $\mathbf{S}_1 = (e_1^r \bmod p) \bmod q$. Note that the value of the first signature does not depend on **M**, the message.
3. Alice creates a digest for the message **M** using $h(\mathbf{M})$.
4. Alice calculates the second signature $\mathbf{S}_2 = (h(\mathbf{M}) + d * S_1)r^{-1} \bmod q$. Note that the calculation of \mathbf{S}_2 is done in modulo '**q**' arithmetic.
5. Alice sends **M**, \mathbf{S}_1 , and \mathbf{S}_2 to Bob.

Verifying

Following are the steps used to verify the message when **M**, S_1 , and S_2 are received:

1. Bob checks to see if $0 < S_1 < q$.
2. Bob checks to see if $0 < S_2 < q$.
3. Bob calculates a digest of **M** (i.e. $h(M)$) using the same hash algorithm used by Alice.
4. Bob calculates $V = [(e_1^{h(M)S_2^{-1}} e_2^{S_1S_2^{-1}}) \bmod p] \bmod q$
5. If S_1 is congruent to V , the message is accepted; otherwise, it is rejected.

4.6 Key Management

Distribution and maintenance of secret keys in symmetric-key cryptography, and public keys in asymmetric-key cryptography is called as key management.

Symmetric-Key Distribution

Symmetric-key cryptography is more efficient than asymmetric-key cryptography for enciphering large messages. Symmetric-key cryptography requires sharing secret key between two communicating parties.

Example:

- If Alice needs to exchange confidential messages with N number of people, he/she needs N different secret keys. If N people want to communicate with each other, then a total of $N(N - 1)$ keys are required.
- If two communicating parties decided to use the same secret key for bi-directional communication, then N people require $N(N-1)/2$ secret keys.

So, each person will have almost N key in this cryptosystem.

Major problems faced in this cryptosystem regarding key management are:

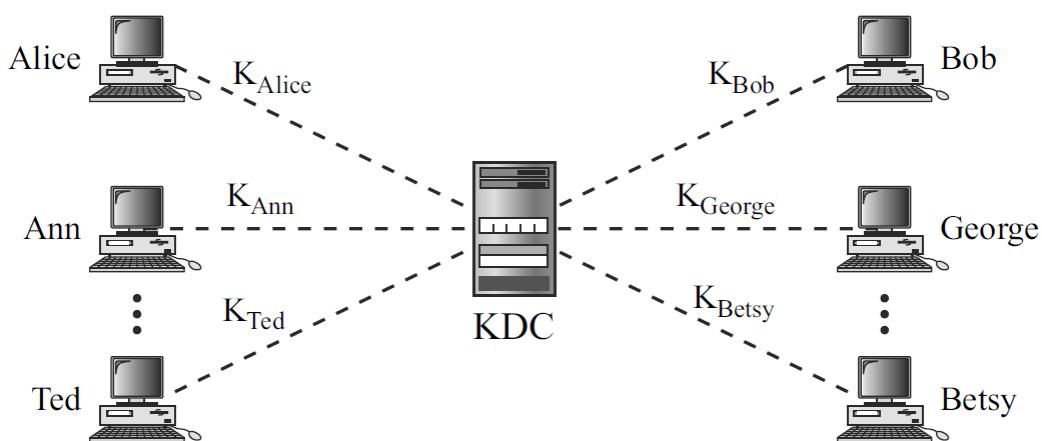
- Number of keys
- Distribution of keys
- The way/process to exchange keys.

So, there is a need of a trusted third party to exchange keys in symmetric-key cryptosystem. This third party is referred as Key-Distribution Center (KDC).

Key-Distribution Center (KDC)

General Model

KDC is the practical solution for the above-mentioned problems. To reduce the number of keys, each person establishes a shared secret key with the KDC as shown in the below figure.



All the participants established their secret key with KDC by using key exchange techniques.

The procedure to send information from Alice to Bob confidentially is:

1. Alice sends a request to the KDC stating that she needs a session (temporary) secret key between herself and Bob.

2. The KDC informs Bob about Alice's request.
3. If Bob agrees, a session key is created between the two.

The secret key between Alice and Bob that is established with the KDC is used to authenticate Alice and Bob to the KDC and to prevent Eve from impersonating either of them.

Flat Multiple KDCs

When the number of people using a KDC increases, the system becomes unmanageable. To solve the problem, we need to have multiple KDCs. We can divide the world into domains. Each domain can have one or more KDCs (for redundancy in case of failure).

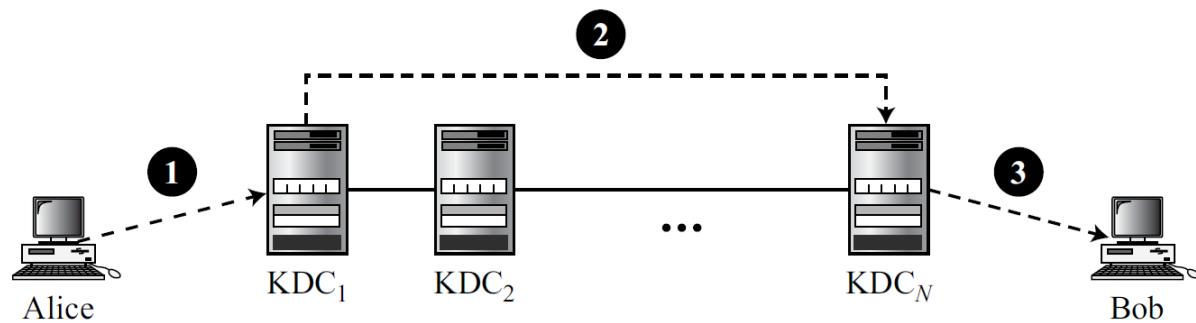
If Alice wants to send a confidential message to Bob, who belongs to another domain:

Step1: Alice contacts her KDC,

Step2: Which in turn contacts the KDC in Bob's domain.

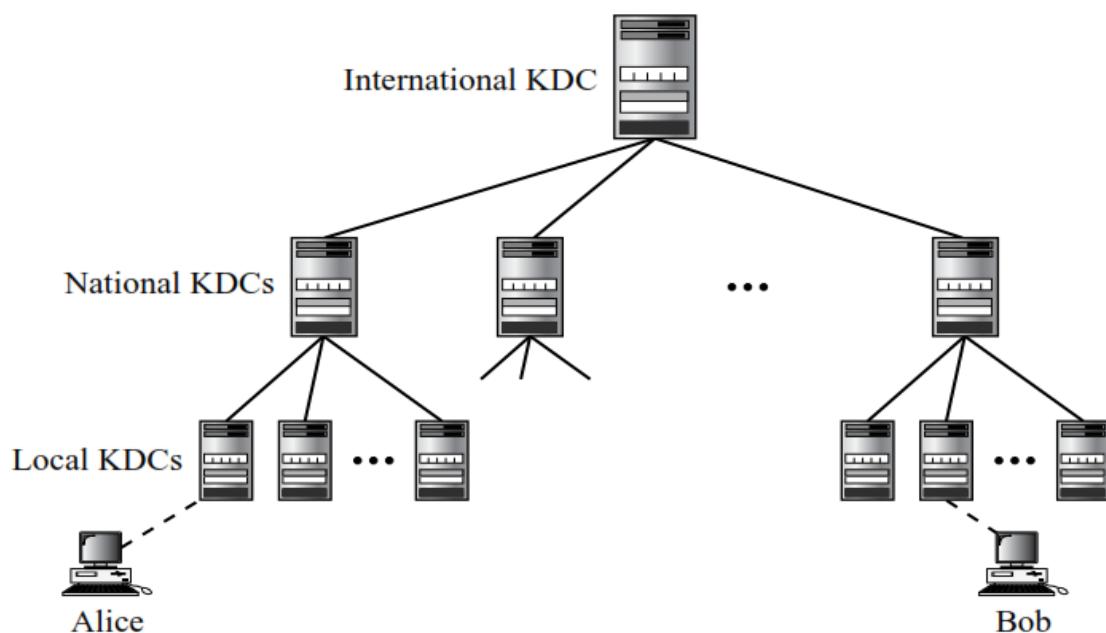
Step3: KDC of Bob domain contacts Bob.

The two KDCs can create a secret key between Alice and Bob.



Hierarchical Multiple KDCs

The concept of flat multiple KDCs can be extended to a hierarchical system of KDCs, with one or more KDCs at the top of the hierarchy. For example, there can be local KDCs, national KDCs, and international KDCs.



A Simple Protocol Used by KDC:

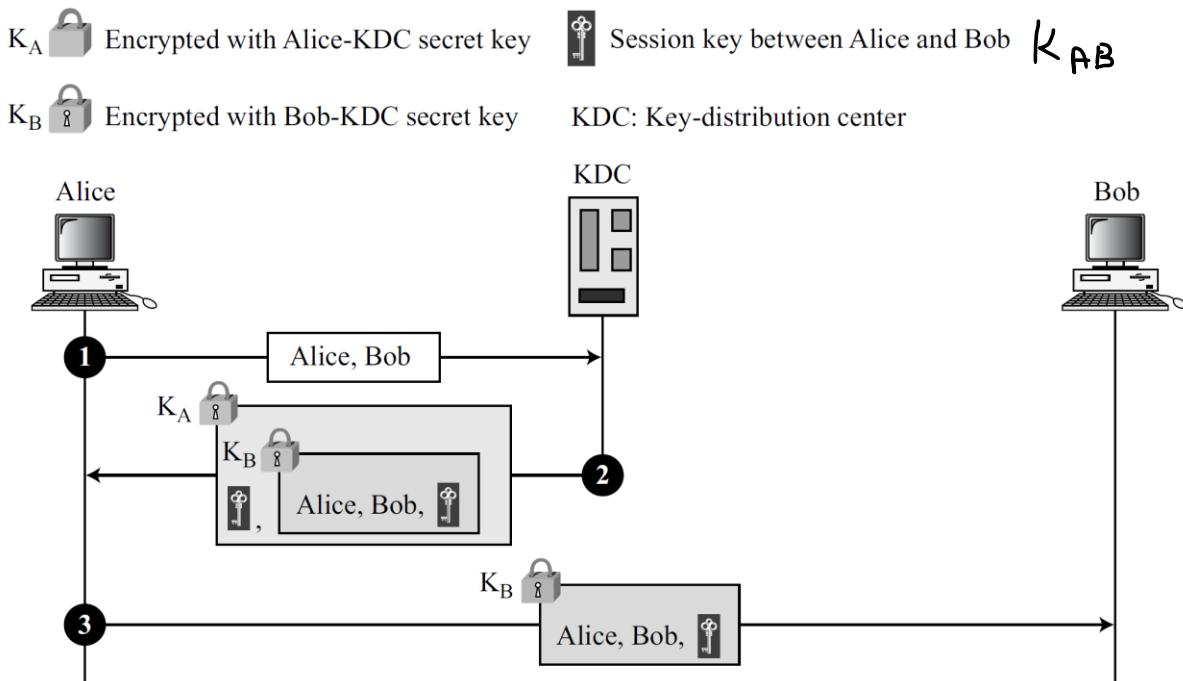
KDC creates a session key K_{AB} between Alice and Bob using the following steps.

Step1: Alice sends a plaintext message to the KDC to obtain a symmetric session key between Bob and herself. The message contains her registered identity (the word Alice in the figure) and the identity of Bob (the word Bob in the figure). This message is not encrypted, it is public.

Step2: After KDC receives the request from Alice, KDC creates a ticket, which contains, Alice and Bob's Identity and Session key (K_{AB}). Then the ticket is encrypted with Bob's secret key K_B . Then, encrypted ticket and the session key (K_{AB}) is again encrypted with Alice Secret key (K_A), sent to Alice.

Step3: After receiving the encrypted data from KDC, Alice decrypts the received message from KDC, separates, K_{AB} and ticket. Keeps K_{AB} securely and sends the ticket to Bob.

Step4: After receiving the ticket from Alice, Bob decrypts the ticket and knows that Alice is interested in sending message using K_{AB} as Session Key.



Unfortunately, this simple protocol has a flaw. Eve can use the replay attack. i.e., Eve can save the message in step 3 and replay it later.

To overcome the above problem, include a timestamp called as nonces (R_A, R_B) of Alice and Bob.

Approaches which include nonces are **Needham-Schroeder Protocol**, **Otway-Rees Protocol**.

Kerberos

Kerberos is an authentication protocol, and at the same time a KDC, that has become very popular. Originally designed at MIT. Several versions are available, most popular version is 4 and the latest one is version 5.

Servers

Three servers involved in the Kerberos protocol.

- 1) Authentication server (AS)
- 2) Ticket-granting server (TGS)
- 3) Real (data) server that provides services to others.

Authentication Server (AS)

The authentication server (AS) is called as Key Distribution Center (KDC) in the Kerberos protocol. Each user registers with the AS and gets a user identity and a password from AS (i.e., K_u). The AS maintains a database with these identities and the corresponding passwords.

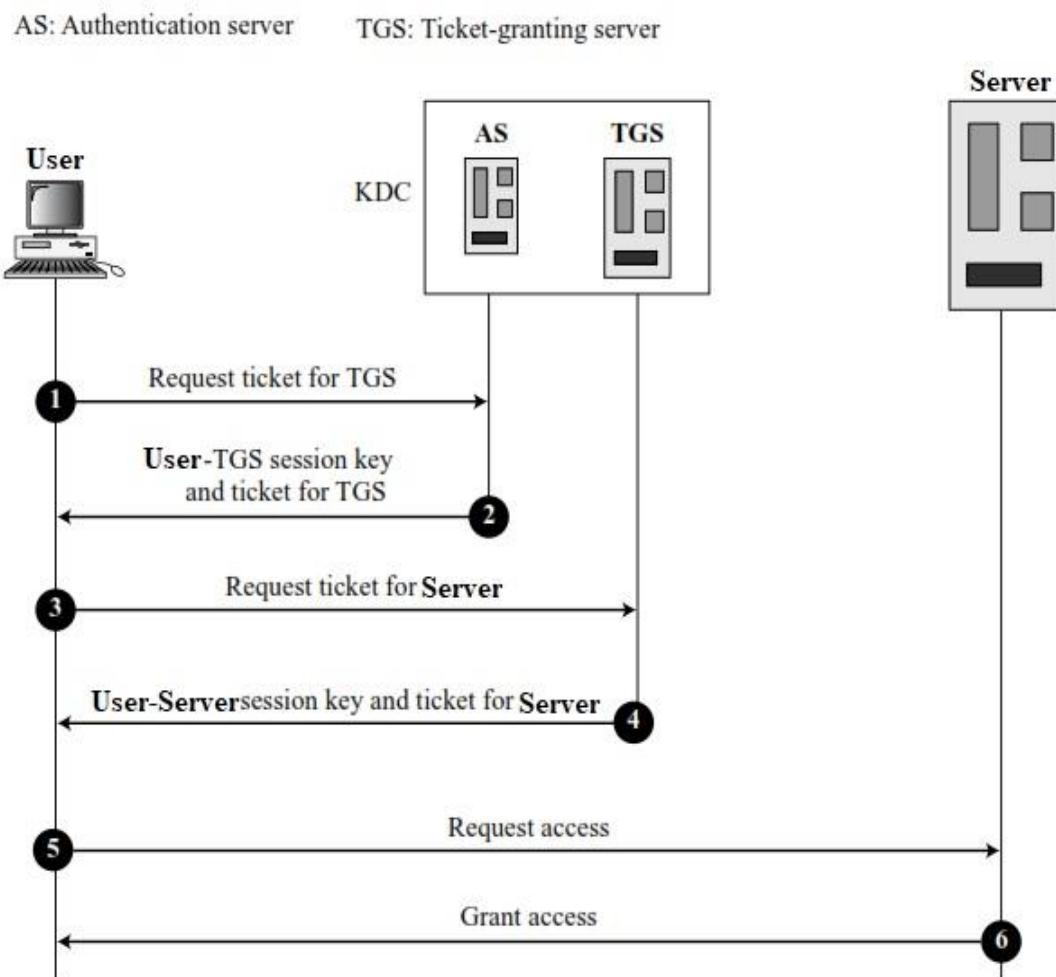
The AS verifies the user, issues a session key to be used between sender and the TGS, and a ticket for the TGS.

Ticket-Granting Server (TGS)

The ticket-granting server (TGS) issues a ticket for the real server. It also provides the session key (KAB) between Sender and Receiver. Kerberos has separated user authentication mechanism before issuing ticket.

Real Server

The real server provides services for the user. Kerberos is designed for a client-server program, such as FTP, in which a user uses the client process to access the server process. Kerberos is not used for person-to-person authentication.



Steps to access a service running on Server

A client/user can access a process running on the real server using the following six steps.

1. User/Client sends his/her request to the AS in plain text using her registered identity.
 2. The AS sends a message encrypted with User/Client permanent symmetric key, K_{U-AS} . The message contains two items: **A Session key**, K_{U-TGS} , which will be used by the user to contact the TGS, and a **ticket** for the TGS that is encrypted with the TGS symmetric key.
- Note: K_{U-TGS} and the ticket are extracted by user after decryption.
3. User now sends three items to the TGS. The first is the **ticket** which was received from the AS. The second is the name of the **real server**, the third is a **timestamp** that is encrypted by K_{U-TGS} . The timestamp prevents a replay by Eve.
 4. TGS sends two tickets, each containing the session key between User and Server, K_{U-S} . The ticket for user is encrypted with K_{U-TGS} , the ticket for Server is encrypted with Server's key, K_{TGS-S} .
 5. User sends Server's ticket with the timestamp encrypted by K_{U-S} .
 6. Server confirms the receipt by adding 1 to the timestamp. The message is encrypted with K_{U-S} and sent to User.

K_{U-AS} : Users Symmetric Key

K_{U-TGS} : Session Key for user and TGS

K_{TGS-S} : Session Key for TGS and Server

K_{AS-TGS} : Session Key for AS and TGS

K_{U-S} : Session Key for User and Server

Ticket for TGS: $K_{AS-TGS}(User-id, K_{U-TGS})$

Ticket for User: $K_{U-TGS}(Server-id, K_{U-S})$

Ticket for Server: $K_{TGS-S}(User-id, K_{U-S})$

Symmetric-Key Agreement

Sender and Receiver can create a session key between themselves without using a KDC. This method of session-key creation is referred to as the symmetric-key agreement. Although there are several ways to accomplish this task.

Two common methods:

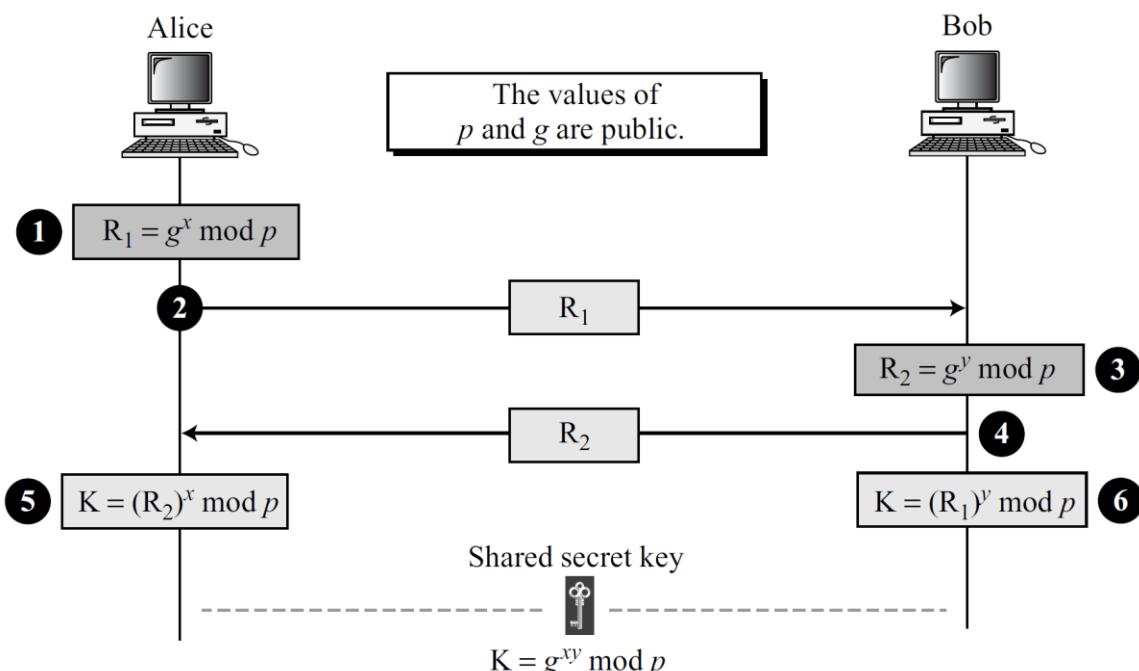
1. Diffie Hellman
2. Station-to-station

Diffie-Hellman Key Agreement

In the Diffie-Hellman protocol two parties create a symmetric session key without the need of a KDC. Before establishing a symmetric key, the two parties need to choose two numbers ‘ p ’ and ‘ g ’. The first number, ‘ p ’, is a large prime number on the order of 300 decimal digits (1024 bits). The second number, ‘ g ’, is a generator of order ‘ $p - 1$ ’ in the group $\langle Z_p^*, \times \rangle$ (i.e. $(p-1)/q$). These two (group and generator) do not need to be confidential.

The steps are as follows:

1. Alice chooses a large random number ‘ x ’ such that $0 \leq x \leq p - 1$ and calculates $R_1 = g^x \text{ mod } p$.
2. Bob chooses another large random number ‘ y ’ such that $0 \leq y \leq p - 1$ and calculate $R_2 = g^y \text{ mod } p$.
3. Alice sends R_1 to Bob. (Note that Alice does not send the value of x , she sends only R_1).
4. Bob sends R_2 to Alice. (Note that Bob does not send the value of y , she sends only R_2).
5. Alice calculates $K = (R_2)^x \text{ mod } p$.
6. Bob also calculates $K = (R_1)^y \text{ mod } p$.



Finally, K is the symmetric key for the session.

Station-to-Station Key Agreement

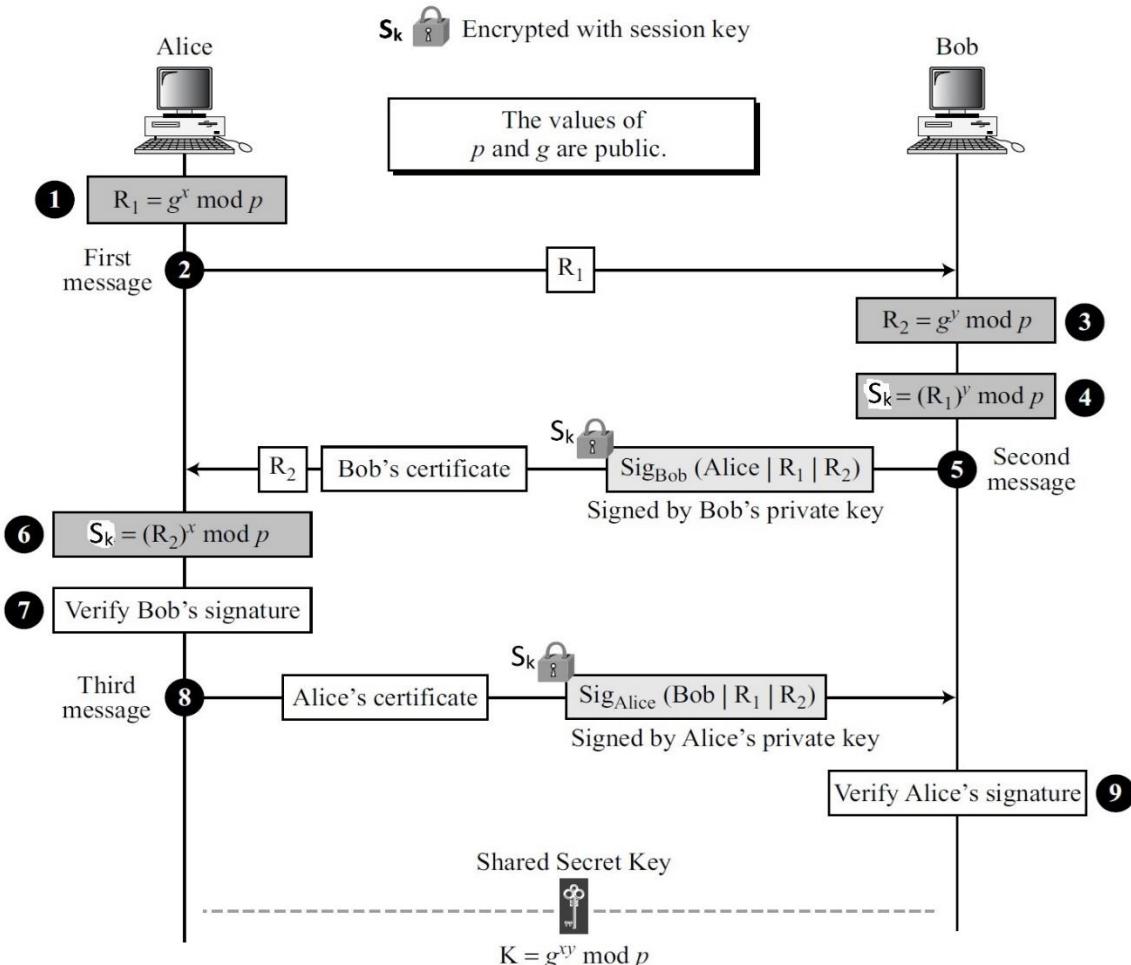
The station-to-station protocol is a method based on Diffie-Hellman. It uses digital signatures with public-key certificates to establish a session key between Sender(Alice) and Receiver(Bob).

Steps:

Two parties need to choose two numbers ‘ p ’ and ‘ g ’. The first number, ‘ p ’, is a large prime number on the order of 300 decimal digits (1024 bits). The second number, ‘ g ’, is a generator of order ‘ $p - 1$ ’ in the group $\langle Z_p^*, \times \rangle$ (i.e. $(p-1)/q$).

1. Alice chooses a large random number ‘ x ’ such that $0 \leq x \leq p - 1$ and calculates $R_1 = g^x \text{ mod } p$. and sends R_1 to Bob.
2. Bob chooses a large random number ‘ y ’ such that $0 \leq y \leq p - 1$ and calculate $R_2 = g^y \text{ mod } p$ and calculates session key $S_k = (R_1)^y \text{ mod } p$. Then Bob concatenates, **Alice ID**, R_1 , R_2 and signs it with his private key, it is called as signature. Then signature is encrypted with session key S_k . Bob now sends R_2 , Signature, Bob’s public-key certificate to Alice.
3. After receiving R_2 , Alice calculates session key $S_k = (R_2)^x \text{ mod } p$, verifies Bob’s signature using S_k . If Bob’s signature is verified, then Alice concatenates **Bob’s ID**, R_1 , R_2 and signs it with his private key, then encrypts it with his session key. Then Alice sends R_2 , Signature, Alice public-key certificate to Bob.
4. Bob verifies Alice signature, if signature is verified, then Bob keeps session key.

Above are shown in the below figure.



Public-KEY Distribution

In public-key cryptography, everyone has access to everyone's public key, i.e., public keys are publicly available to all. So, like secret keys or session keys, public keys need to be distributed as well.

The following are the ways to distribute public keys.

1. **Public Announcement**
2. **Trusted Center**
3. **Controlled Trusted Center**
4. **Certification Authority**
 - a. X-509 certificate format
5. **Public Key Infrastructure (PKI)**

1. Public Announcement:

The naive approach to announce public keys publicly is to keep public key on their website or announce it in a local or national newspaper. When someone wants to send a confidential message to him/her, he/she can obtain his/her public key from his/her website or from the newspaper and sends encrypted message.

This approach, however, is not secure, because it is subject to forgery. For example, adversary may also make such a public announcement with someone else names.

2. Trusted Center:

Another approach is to have a trusted center which maintains a directory of public keys just like a dynamically updatable telephone directory.

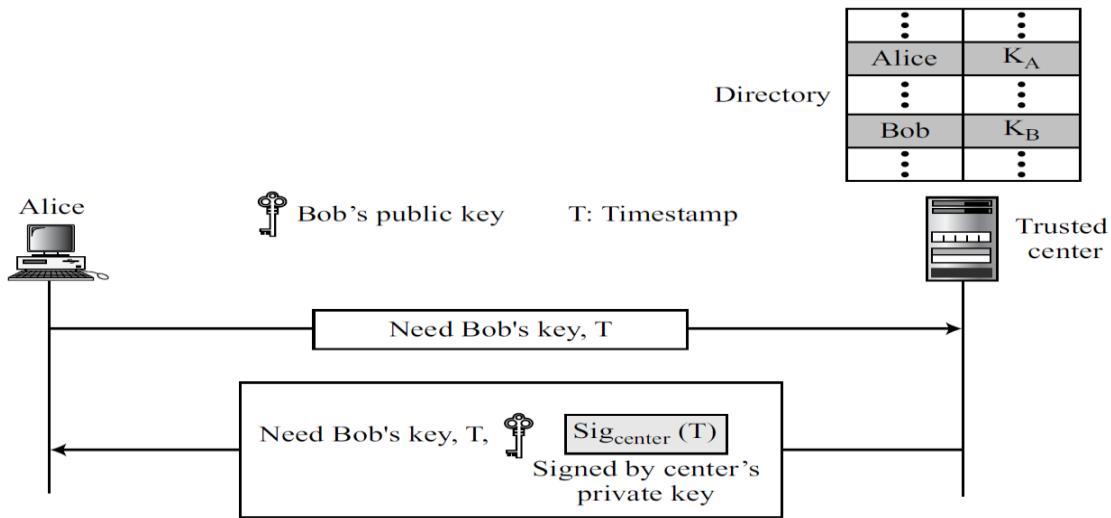
- Each user must register with the center and prove his or her identity.
- Each user creates private and public key, keep the private key, and deliver the public key for insertion into the directory.
- Then the directory is publicly advertised by the trusted center. The center can also respond to any inquiry about a public key.

⋮	⋮
Alice	K_A
⋮	⋮
Bob	K_B
⋮	⋮

3. Controlled Trusted Center

In this approach, more level of security is introduced by including timestamp and centers' public and private keys. So, the public-key announcements can include a timestamp and signed by center to prevent interception and modification of the response.

Example: If Alice wants Bob's public key, he/she can send a request to the center including Bob's name and a timestamp. The center responds with Bob's public key, and the timestamp signed with the private key of the center. Alice uses the public key of the center, to verify the timestamp. If the timestamp is verified, he/she extracts Bob's public key.



4. Certification Authority

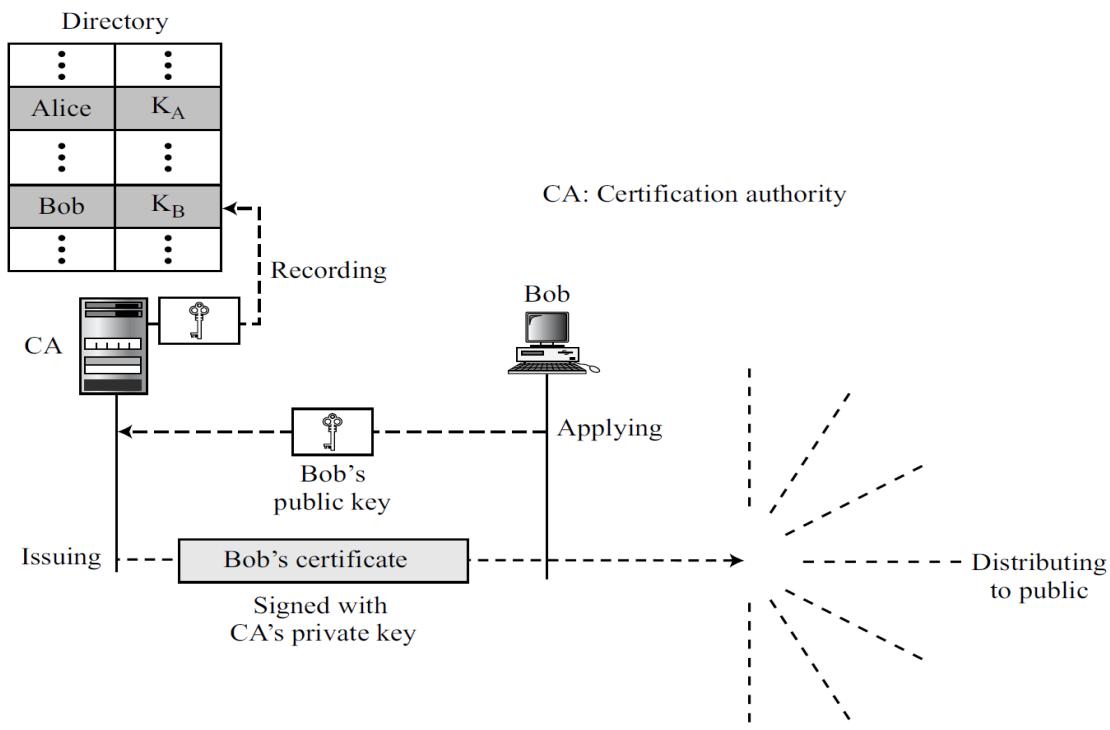
Controlled center approach increases burden on the center if the number of requests increases.

The alternative approach is to create public-key certificates.

Certification authority (CA) is a federal or state organization that binds a public key to an entity and issues a certificate (Digital Certificates). The CA has a well-known public key for itself that cannot be forged.

Anyone who wants to create his public key certificate, contacts CA, CA verifies the identity of the requester, then ask for requester public key, writes it on the certificate and signs the certificate with centers' private key to prevent it from being forged. CA maintains the certificates.

Example: Bob request CA for creation of certificate. After Bob is verified, CA asks Bob for her public key, and writes it on certificate and sign the certificate with CA's private key. If Alice wants the public key of Bob, Alice downloads Bob's certificate from CA and uses centers' public key to extract Bob's public key.

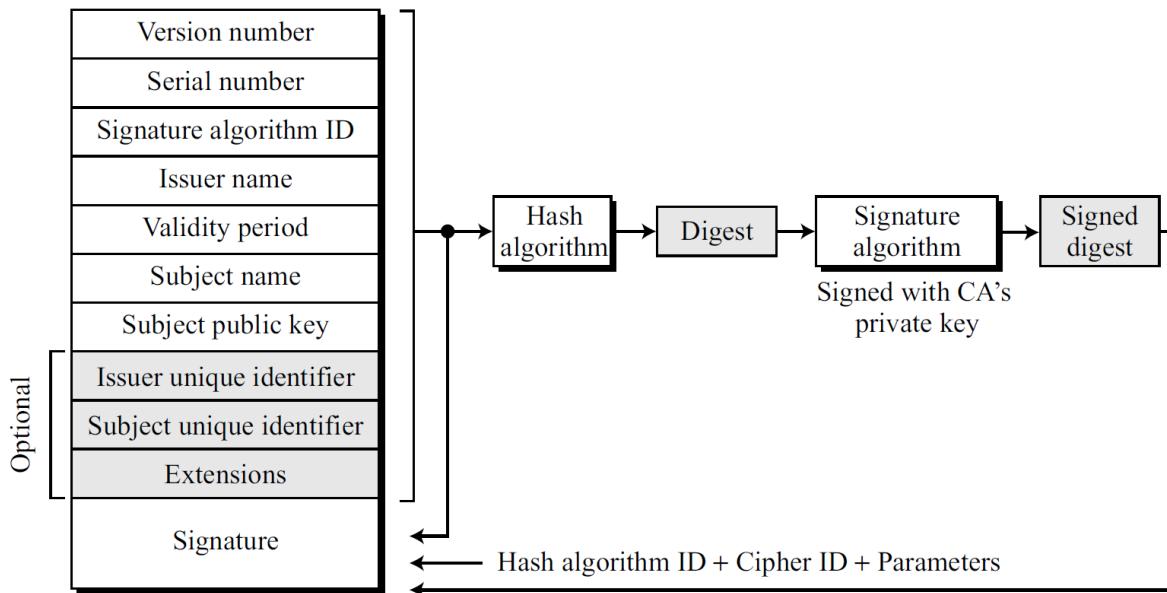


X.509

Certification Authority has solved the problem of public-key fraud, but it has created a side-effect. Each certificate may have a different format.

- Different formats of certificates create difficulty in retrieving the public key from them, because people use automated program for retrieving.
- So, we need a standard universal format for certificates. UTI has designed X.509, standard certificate structure.

X.509 Certificate Format



X.509 Certificate has the following fields:

Version number: This field defines the version of X.509. The version number started at 0, the current version (third version) is 2.

- **Serial number:** This field defines certificate number. The value is unique for each certificate issuer.
- **Signature algorithm ID:** This field identifies the algorithm used to sign the certificate.
- **Issuer name:** This field identifies the certification authority that issued the certificate.
- **Validity Period:** This field defines validity period of the certificate. (Start and ending date).
- **Subject name:** This field defines the owner's name to which the public key belongs.
- **Subject public key:** This field defines the owner's public key, the heart of the certificate. Also specifies the algorithm name used to create the key.
- **Issuer unique identifier:** This optional field contains the issuer identity. Two issuers can have the same issuer field value, if the issuer unique identifiers are different.
- **Subject unique identifier:** This optional field contains subjects' identity. Two different subjects can have the same subject field value, if the subject unique identifiers are different.
- **Extensions:** This optional field allows issuers to add more private information to the certificate.
- **Signature:** This field is made of three sections. The first section contains all other fields in the certificate. The second section contains the digest of the first section encrypted with the CA's public key. The third section contains the algorithm identifier used to create the digest.

Certificate Renewal

Each certificate has a period of validity. If there is no problem with the certificate, the CA issues a new certificate before the old one expires.

Certificate Revocation

In some cases, a certificate must be revoked before its expiration. (i.e., certificates are to be modified are recreated). The reasons for the revocation are:

- The user's (subject's) private key (corresponding to the public key listed in the certificate) might have been comprised.
- The CA is no longer willing to certify the user.
- The CA's private key, which can verify certificates, may have been compromised.

The revocation is done by periodically issuing a certificate revocation list (CRL). The list contains all revoked certificates that are not expired on the date the CRL is issued. When a user wants to download a certificate, he/he first needs to check the directory of the CA for the latest certificate revocation list.

PKI (Public-Key Infrastructure)

Public-Key Infrastructure (PKI) is a model for creating, distributing, and revoking certificates based on the X.509 format. The Internet Engineering Task Force (IETE) has created the Public-Key Infrastructure X.509 (PKIX).

PKI Duties

Several duties have been defined for a PKI. The most important ones are:

- **Certificates' issuing, renewal, and revocation:** These are duties defined in the X.509.
- **Keys' storage and update:** A PKI should be a storage place for private keys of those members that need to hold their private keys somewhere safe. In addition, a PKI is responsible for updating these keys on members' demands.
- **Providing services to other protocols:** Some Internet security protocols, such as IPSec and TLS, are relying on the services by a PKI.
- **Providing access control:** A PKI can provide different levels of access to the information stored in its database.

PKI Models (Trust Models)

- **Hierarchical Model**
- **Mesh Model**
- **Web of Trust Model**