UNIT-3

Public key cryptography principles, public key cryptography algorithms, digital signatures, digital Certificates, Certificate Authority and key management Kerberos, X.509 Directory Authentication Service.

Public Key Cryptography:

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. It is *asymmetric*, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. Public key schemes are neither more nor less secure than private key (security depends on the key size for both). Public-key cryptography *complements rather than replaces* symmetric cryptography. Both also have issues with key distribution, requiring the use of some suitable protocol.

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:

1.) ***key distribution*** – how to have secure communications in general without having to trust a KDC with your key
2.) ***digital signatures*** – how to verify a message comes intact from the claimed sender

Public-key/two-key/asymmetric cryptography involves the use of **two** keys:

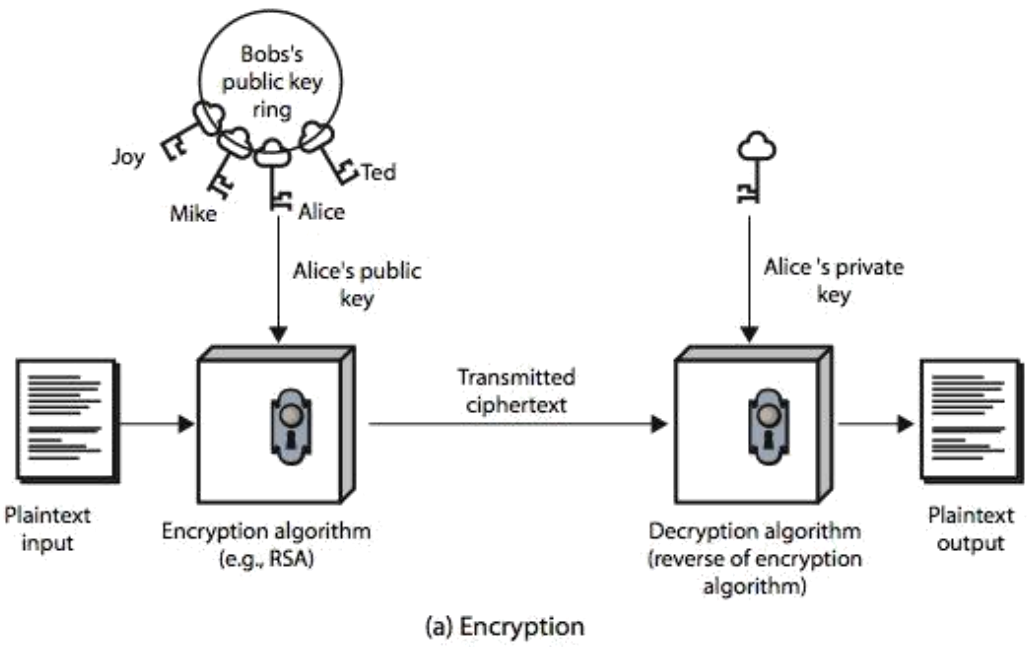
- a *public-key*, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
- a *private-key*, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**.
- is **asymmetric** because those who encrypt messages or verify signatures cannot decrypt messages or create signatures

Public-Key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- it is computationally infeasible to find decryption key knowing only algorithm & encryption key
- it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known

- either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms like RSA)

The following figure illustrates public-key encryption process and shows that a public-key encryption scheme has six ingredients: plaintext, encryption algorithm, public & private keys, ciphertext & decryption algorithm.



(a) Encryption

The essential steps involved in a public-key encryption scheme are given below:

1.) Each user generates a pair of keys to be used for encryption and decryption.

2.) Each user places one of the two keys in a public register and the other key is kept private.

3.) If B wants to send a confidential message to A, B encrypts the message using A's public key.

4.) When A receives the message, she decrypts it using her private key. Nobody else can decrypt the message because that can only be done using A's private key (Deducing a private key should be infeasible).

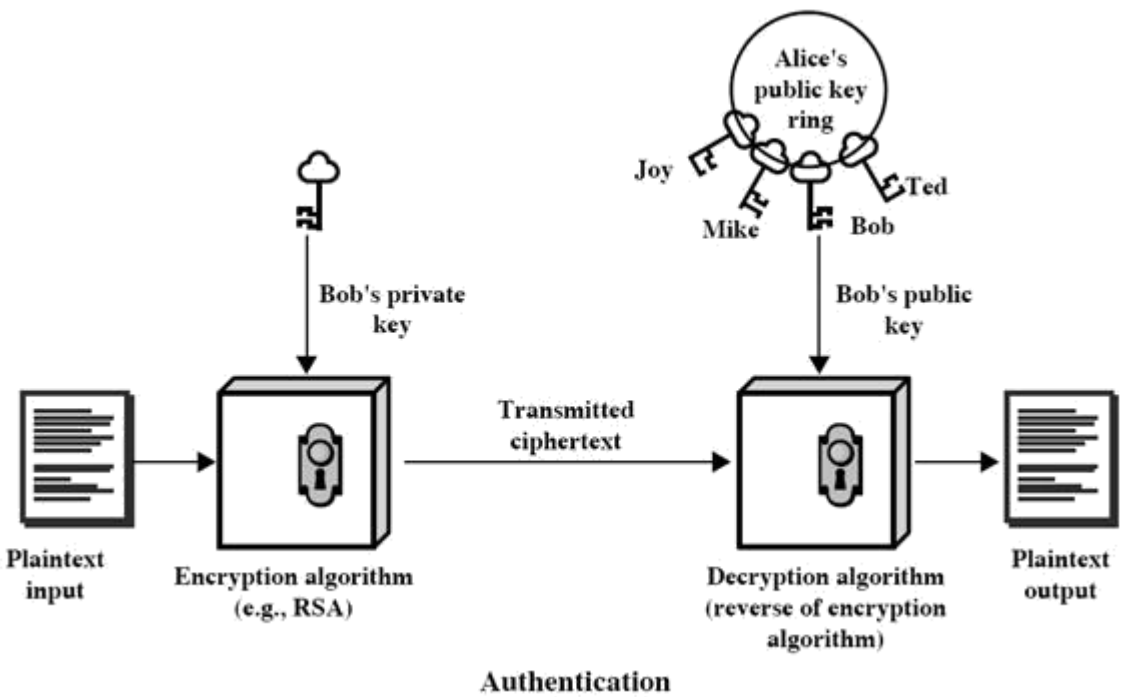
5.) If a user wishes to change his keys –generate another pair of keys and publish the public one: no interaction with other users is needed.

Notations used in Public-key cryptography:

- The public key of user A will be denoted $\mathbf{KU_A}$.
- The private key of user A will be denoted $\mathbf{KR_A}$.
- Encryption method will be a function E.

- Decryption method will be a function D.
- If B wishes to send a plain message X to A, then he sends the cryptotext $Y=E(KU_A,X)$
- The intended receiver A will decrypt the message: $D(KR_A,Y)=X$

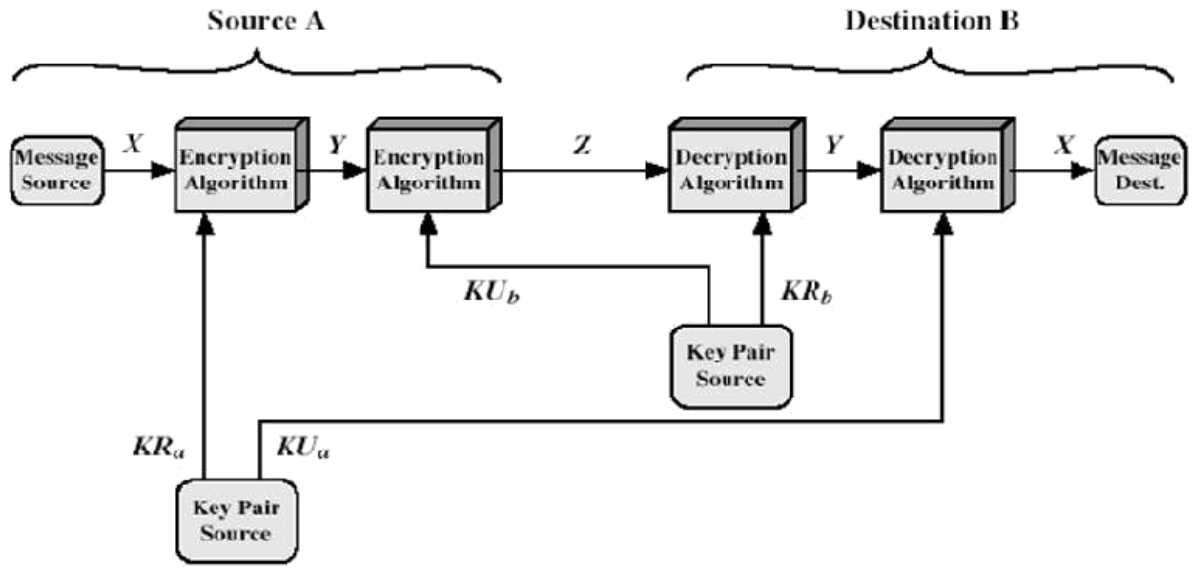
The first attack on Public-key Cryptography is the attack on Authenticity. **An attacker may impersonate user B**: he sends a message $E(KU_A,X)$ and claims in the message to be B –A has no guarantee this is so. To overcome this, B will encrypt the message using his private key: $Y=E(KR_B,X)$. Receiver decrypts using B's public key KR_B. This shows the authenticity of the sender because (supposedly) he is the only one who knows the private key. The entire encrypted message serves as a digital signature. This scheme is depicted in the following figure:



Authentication

But, a drawback still exists. Anybody can decrypt the message using B's public key. So, secrecy or confidentiality is being compromised.

One can provide both *authentication and confidentiality* using the public-key scheme twice:



Public-Key Cryptosystem: Secrecy and Authentication

- B encrypts X with his private key: $Y=E(KR_B,X)$
- B encrypts Y with A's public key: $Z=E(KU_A,Y)$
- A will decrypt Z (and she is the only one capable of doing it): $Y=D(KR_A,Z)$
- A can now get the plaintext and ensure that it comes from B (he is the only one who knows his private key): decrypt Y using B's public key: $X=E(KU_B,Y)$.

Applications for public-key cryptosystems:

1.) **Encryption/decryption**: sender encrypts the message with the receiver's public key.

2.) **Digital signature**: sender "signs" the message (or a representative part of the message) using his private key

3.) **Key exchange**: two sides cooperate to exchange a secret key for later use in a secret-key cryptosystem.

The main requirements of Public-key cryptography are:

1. Computationally easy for a party B to generate a pair (public key KUb, private key KR_b).
2. Easy for sender A to generate ciphertext: $$C = E_{KUb}(M)$$
3. Easy for the receiver B to decrypt ciphertect using private key: $$M = D_{KRb}(C) = D_{KRb}[E_{KUb}(M)]$$
4. Computationally infeasible to determine private key (KR_b) knowing public key (KU_b)
5. Computationally infeasible to recover message M, knowing KU_b and ciphertext C
6. Either of the two keys can be used for encryption, with the other used for decryption:

$$M = D_{KRb}[E_{KUb}(M)] = D_{KUb}[E_{KRb}(M)]$$

Easy is defined to mean a problem that can be solved in polynomial time as a function of input length. A problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. Public-key cryptosystems usually rely on difficult math functions rather than S-P networks as classical cryptosystems. **One-way function** is one, easy to calculate in one direction, infeasible to calculate in the other direction (i.e., the inverse is infeasible to compute). **Trap-door function** is a difficult function that becomes easy if some extra information is known. Our aim to find a ***trap-door one-way function***, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known.

Security of Public-key schemes:

- Like private key schemes brute force **exhaustive search** attack is always theoretically possible. But keys used are too large (>512bits).
- Security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalyse) problems. More generally the **hard** problem is known, its just made too hard to do in practise.
- Requires the use of **very large numbers,** hence is **slow** compared to private key schemes

RSA algorithm

RSA is the best known, and by far the most widely used general public key encryption algorithm, and was first published by Rivest, Shamir & Adleman of MIT in 1978 [RIVE78]. Since that time RSA has reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption. The RSA scheme is a block cipher in which the plaintext and the ciphertext are integers between 0 and n-1 for some fixed n and typical size for n is 1024 bits (or 309 decimal digits). It is based on

exponentiation in a finite (Galois) field over integers modulo a prime, using large integers (eg. 1024 bits). Its security is due to the cost of factoring large numbers.

RSA involves a public-key and a private-key where the public key is known to all and is used to encrypt data or message. The data or message which has been encrypted using a public key can only be decryted by using its corresponding private-key. Each user generates a key pair i.e. public and private key using the following steps:

- *each user selects two large primes at random - p, q*
- *compute their system modulus n=p.q*
- *calculate ø(n), where **ø(n)=(p-1)(q-1)***
- *selecting at random the encryption key e, where 1<e<ø(n),and **gcd(e,ø(n))=1***
- *solve following equation to find decryption key d: **e.d=1 mod ø(n)** and 0≤d≤n*
- *publish their public encryption key: **KU={e,n}***
- *keep secret private decryption key**: KR={d,n}***

Both the sender and receiver must know the values of n and e, and only the receiver knows the value of d. Encryption and Decryption are done using the following equations.

To encrypt a message M the sender:

- obtains **public key** of recipient *KU={e,n}*
- computes: $\mathbf{C=M^e \bmod n}$, where 0≤M<n

To decrypt the ciphertext C the owner:

- uses their private key *KR={d,n}*
- computes: $\mathbf{M=C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n}$

For this algorithm to be satisfactory, the following requirements are to be met.

a) Its possible to find values of e, d, n such that $\mathbf{M^{ed} = M \bmod n}$ for all M<n
b) It is relatively easy to calculate M^e and C for all values of M < n.
c) It is impossible to determine d given e and n

The way RSA works is based on Number theory:

Fermat's little theorem: if **p** is prime and **a** is positive integer not divisible by **p**, then

$$\mathbf{a^{p-1} \equiv 1 \bmod p.}$$

Corollary: For any positive integer **a** and prime **p**, $\mathbf{a^p \equiv a \bmod p.}$

Fermat's theorem, as useful as will turn out to be does not provide us with integers d,e we are looking for –Euler's theorem (a refinement of Fermat's) does. Euler's function associates to any positive integer **n,** a number ϕ**(n)**: the number of positive integers smaller than **n** and relatively prime to **n.** For example, ϕ**(37) = 36** i.e. ϕ**(p) = p-1** for any prime **p.** For any two primes **p,q**, ϕ**(pq)=(p-1)(q-1).**

Euler's theorem: for any relatively prime integers a,n we have $\mathbf{a^{\phi(n)} \equiv 1 \bmod n.}$
Corollary: For any integers a,n we have $\mathbf{a^{\phi(n)+1} \equiv a \bmod n}$
Corollary: Let p,q be two odd primes and n=pq. Then:

ϕ(n)=(p-1)(q-1)

For any integer m with 0<m<n, $m^{(p-1)(q-1)+1} \equiv m \bmod n$ For any integers k,m with 0<m<n, $m^{k(p-1)(q-1)+1} \equiv m \bmod n$

Euler's theorem provides us the numbers d, e such that M^{ed}=M mod n. We have to choose d,e such that ed=kϕ(n)+1, or equivalently, $d \equiv e^{-1} \bmod \phi(n)$

An example of RSA can be given as,

- Select primes: *p*=17 & *q*=11
- Compute *n* = *pq* =17×11=187
- Compute ø(*n*)=(*p*–1)(*q*-1)=16×10=160
- Select e *:* gcd(e,160)=1; choose *e*=7
- Determine d*: de*=1 mod 160 and *d* < 160 Value is d=23 since 23×7=161= 10×160+1
- Publish public key KU={7,187}
- Keep secret private key KR={23,187}
- Now, given message M = 88 (nb. 88<187)
- encryption: $C = 88^7 \bmod 187 = 11$
- decryption: $M = 11^{23} \bmod 187 = 88$

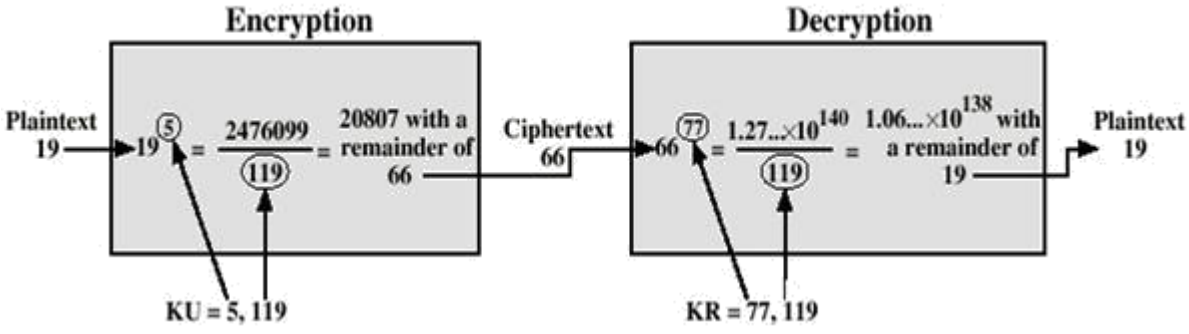
Another example of RSA is given as,
Let **p = 11, q = 13, e = 11, m = 7**

n = pq i.e. n= 11*13 = 143

ø(n)= (p-1)(q-1) i.e. (11-1)(13-1) = 120

e.d=1 mod ø(n) i.e. 11d mod 120 = 1 i.e. (11*11) mod 120=1; so d = 11
public key :{11,143} and private key: {11,143}
C=M^{e} mod n, so ciphertext = 7^{11}mod143 = 727833 mod 143; i.e. C = 106
M=C^{d} mod n, plaintext = 106^{11} mod 143 = 1008 mod 143; i.e. M = 7

Another example is:



For RSA key generation,

- users of RSA must:
  - determine two primes at random - p, q
  - select either e or d and compute the other
- primes p,q must not be easily derived from modulus N=p.q
  - means must be sufficiently large
  - typically guess and use probabilistic test
- exponents e, d are inverses, so use Inverse algorithm to compute the other

Security of RSA

There are three main approaches of attacking RSA algorithm.

Brute force key search (infeasible given size of numbers)

As explained before, involves trying all possible private keys. Best defence is using large keys.

Mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)

There are several approaches, all equivalent in effect to factoring the product of two primes. Some of them are given as:

- factor N=p.q, hence find ø(N) and then d
- determine ø(N) directly and find d
- find d directly

The possible defense would be using large keys and also choosing large numbers for p and q, which should differ only by a few bits and are also on the order of magnitude 10^{75} to 10^{100}. And gcd (p-1, q-1) should be small.

Timing attacks (on running of decryption)

These attacks involve determination of a private key by keeping track of how long a computer takes to decipher a message (ciphertext-only attack) –this is essentially an attack on the fast exponentiation algorithm but can be adapted for any other algorithm. Though these attacks are a quite serious threat, there are some simple countermeasures that can be used. They are explained below:

Constant exponentiation time:- Ensure that all exponentiations take the same time before returning a result: degrade performance of the algorithm.

Random Delay:- A random delay can be added to exponentiation algorithm to confuse the timing attack. If there is not enough noise added by defenders, the attackers can succeed.

Blinding:- Multiply the ciphertext by a **random** number before performing exponentiation – in this way the attacker does not know the input to the exponentiation algorithm.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation $M = C^d \bmod n$ is implemented as follows:

- Generate a secret random number r between 0 and n-1
- Compute $C'=C(r^e) \bmod n$ where e is the public exponent
- Compute $M'=(C'^d) \bmod n$ with the ordinary exponentiation
- Compute $M=M'r^{-1} \bmod n$
- Reported performance penalty: 2 to 10%

Key Management

One of the major roles of public-key encryption has been to address the problem of key distribution. Two distinct aspects to use of public key encryption are present.

- The distribution of public keys.
- Use of public-key encryption to distribute secret keys.

Distribution of Public Keys

The most general schemes for distribution of public keys are given below

PUBLIC ANNOUNCEMENT OF PUBLIC KEYS

Here any participant can send his or her public key to any other participant or broadcast the key to the community at large. For example, many PGP users have adopted the practice of appending their public key to messages that they send to public forums.

Uncontrolled Public-Key Distribution

Though this approach seems convenient, it has a major drawback. Anyone can forge such a public announcement. Some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until the time when A discovers about the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

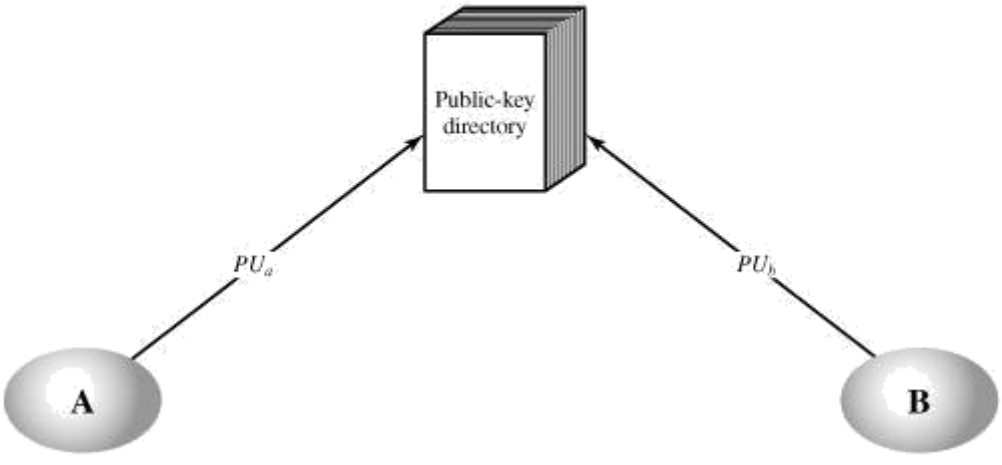
PUBLICLY AVAILABLE DIRECTORY

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization. It includes the following elements:

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.

Public-Key Publication



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.

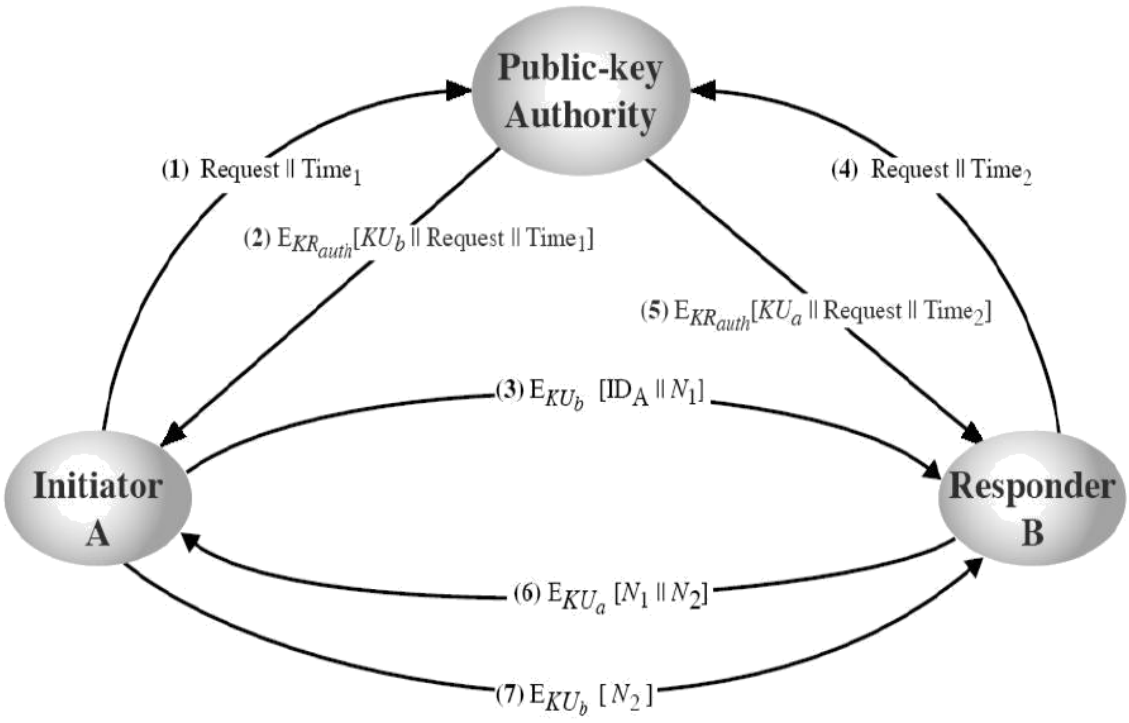
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme has still got some vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Or else, the adversary may tamper with the records kept by the authority.

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. This scenario assumes the existence of a public authority (whoever that may be) that maintains a dynamic directory of public keys of all users. The public authority has its own (private key, public key) that it is using to communicate to users. Each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key.

For example, consider that Alice and Bob wish to communicate with each other and the following steps take place and are also shown in the figure below:



1.) Alice sends a **timestamped** message to the central authority with a request for Bob's public key (the time stamp is to mark the moment of the request)

2.) The authority sends back a message encrypted with its private key (for authentication) –message contains Bob's public key and the original message of Alice –this way Alice knows this is not a reply to an old request;

3.) Alice starts the communication to Bob by sending him an encrypted message containing her identity ID_A and a nonce N_1 (to identify uniquely this transaction)

4.) Bob requests Alice's public key in the same way (step 1)

5.) Bob acquires Alice's public key in the same way as Alice did. (Step-2)

6.) Bob replies to Alice by sending an encrypted message with N_1 plus a new generated nonce N_2 (to identify uniquely the transaction)

7.) Alice replies once more encrypting Bob's nonce N_2 to assure bob that its correspondent is Alice

Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can save the other's public key for future use, a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

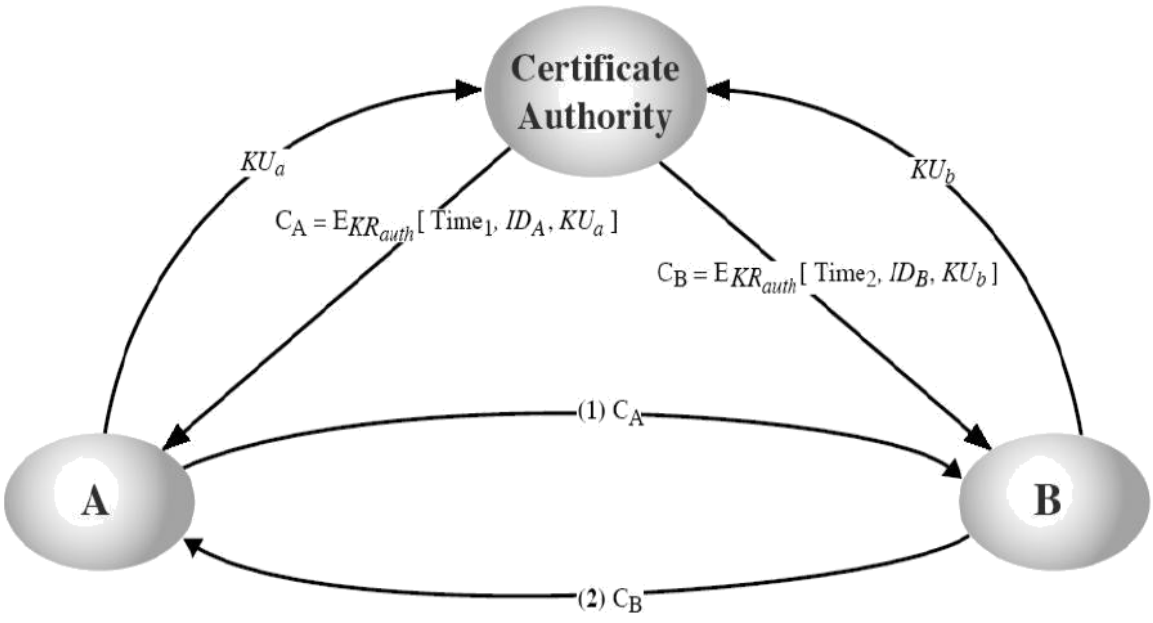
PUBLIC-KEY CERTIFICATES

The above technique looks attractive, but still has some drawbacks. For any communication between any two users, the central authority must be consulted by both users to get the newest public keys i.e. the central authority must be online 24 hours/day. If the central authority goes offline, all secure communications get to a halt. This clearly leads to an undesirable bottleneck.

A further improvement is to use certificates, which can be used 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. A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA). 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 needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority.

This certificate issuing scheme does have the following requirements:

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.



Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

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

where PR_{auth} 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:

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

The recipient uses the authority's public key, PU_{auth} 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 ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages. In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

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, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

Public Key Distribution of Secret Keys

Public-key encryption is usually viewed as a vehicle for the distribution of secret keys to be used for conventional encryption and the main reason for this is the relatively slow data rates associated with public-key encryption.

Simple Secret Key Distribution:

If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

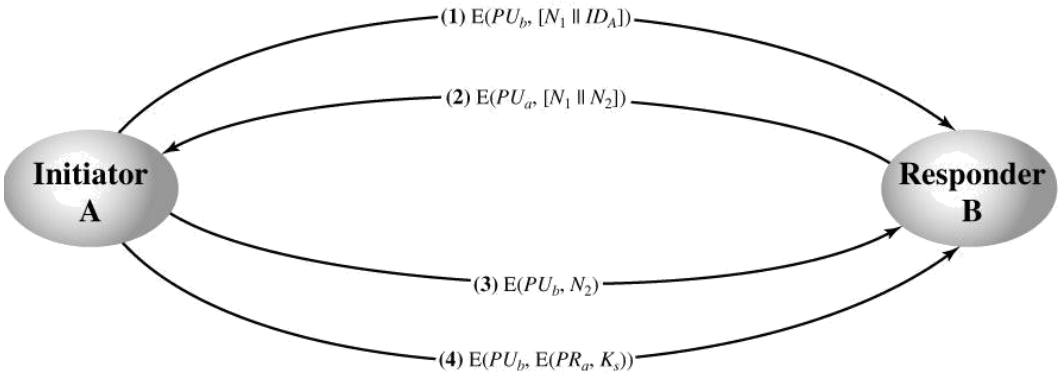
In this case, if an adversary, E, has control of the intervening communication channel, then E can compromise the communication in the following fashion without being detected:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e||ID_A$ to B.
3. B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.
4. E intercepts the message, and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.
5. E transmits $E(PU_a, K_s)$ to A.

The result is that both A and B know K_s and are unaware that K_s has also been revealed to E. A and B can now exchange messages using K_s E no longer actively interferes with the communications channel but simply eavesdrops. Knowing K_s E can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

It is assumed that A and B have exchanged public keys by one of the schemes described earlier. Then the following steps occur:



1. A uses B's public key to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. 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 (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2 encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_S and sends M = E(PU_b, E(PR_a, K_S)) to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes D(PU_a, D(PR_b, M)) to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

Diffie-Hellman Key Exchange

Diffie-Hellman key exchange (**D-H**) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. The D-H algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.

First, a primitive root of a prime number p, can be defined as one whose powers generate all the integers from 1 to p-1. If a is a primitive root of the prime number p, then the

numbers, ***a mod p, a^2 mod p,..., a^{p-1} mod p***, are distinct and consist of the integers from 1 through p 1 in some permutation.

For any integer ***b*** and a primitive root ***a*** of prime number ***p***, we can find a unique exponent **i** such that $b \equiv a^i \pmod{p}$ where $0 \leq i \leq (p\ 1)$.The exponent i is referred to as the discrete logarithm of b for the base a, mod p. We express this value as $\boldsymbol{dlog_{a,p}\ (b)}$**.** The algorithm is summarized below:

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

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

Generation of Secret Key by User B

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

For this scheme, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_A < q$ and computes $Y_B = \alpha^{XB} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{XA} \bmod q$ and user B computes the key as $K = (Y_A)^{XB} \bmod q$. These two calculations produce identical results.

Discrete Log Problem

The (discrete) exponentiation problem is as follows: Given a base a, an exponent b and a modulus p, calculate c such that $a^b \equiv c \pmod p$ and $0 \le c < p$. It turns out that this problem is fairly easy and can be calculated "quickly" using fast-exponentiation.

The discrete log problem is the inverse problem:

Given a base a, a result c ($0 \le c < p$) and a modulus p, calculate the exponent b such that

$$a^b \equiv c \pmod p.$$

It turns out that no one has found a quick way to solve this problem

With DLP, if P had 300 digits, X_a and X_b have more than 100 digits, it would take longer than the life of the universe to crack the method.

Examples for D-H key distribution scheme:

1) Let p = 37 and g = 13.

Let Alice pick a = 10. Alice calculates $13^{10} \pmod{37}$ which is 4 and sends that to Bob.
Let Bob pick b = 7. Bob calculates $13^7 \pmod{37}$ which is 32 and sends that to Alice.
(Note: 6 and 7 are secret to Alice and Bob, respectively, but both 4 and 32 are known by all.)

- Alice receives 32 and calculates $32^{10} \pmod{37}$ which is 30, the secret key.
- Bob receives 4 and calculates $4^7 \pmod{37}$ which is 30, the same secret key.

2) Let p = 47 and g = 5.

Let Alice pick a = 18. Alice calculates $5^{18} \pmod{47}$ which is 2 and sends that to Bob.
Let Bob pick b = 22. Bob calculates $5^{22} \pmod{47}$ which is 28 and sends that to Alice.

- Alice receives 28 and calculates $28^{18} \pmod{47}$ which is 24, the secret key.
- Bob receives 2 and calculates $2^{22} \pmod{47}$ which is 24, the same secret key

Man-in-the-Middle Attack on D-H protocol

Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X}{}_{D2} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X}{}_{E} \bmod q$.
5. Bob transmits X_A to Alice.
6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X}{}_{D1} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X}{}_{A} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key K1 and Alice and Darth share secret key K2. All future communication between Bob and Alice is compromised in the following way:

1. Alice sends an encrypted message M: E(K2, M).
2. Darth intercepts the encrypted message and decrypts it, to recover M.
3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The use of elliptic curves in cryptography was suggested independently by Neal Koblitz and Victor S. Miller in 1985. The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing the processing overhead.

Elliptic Curve over GF(p)

Let GF(p) be a finite field, p > 3, and let a, b $\in$ GF(p) are constant such that $4a^3 + 27b^2 \equiv 0 \pmod p$.

An elliptic curve, $E_{(a,b)}(GF(p))$, is defined as the set of points (x,y) $\in$ GF(p) * GF(p) which satisfy the equation $y^2 \equiv x^3 + ax + b \pmod p$, together with a special point, O, *called the point at infinity*.

Let P and Q be two points on $E_{(a,b)}(GF(p))$ and O is the point at infinity.

- P+O = O+P = P
- If P = (x_1,y_1) then -P = $(x_1, -y_1)$ and P + (-P) = O.
- If P = (x_1,y_1) and Q = (x_2,y_2), and P and Q are not O.

  then P +Q = (x_3, y_3) where

  $x_3 = \lambda^2 - x_1 - x_2$

  $y_3 = \lambda(x_1 - x_3) - y_1$

  and $\lambda = (y_2-y_1)/(x_2-x_1)$ if P ≠ Q

  $\lambda = (3x_1^2+a)/ 2y_1$ if P = Q

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & for\_ x_1 \neq x_2 \\ \dfrac{3x_1^2 + a}{2y_1} & for\_ x_1 = x_2 \end{cases}$$

An elliptic curve may be defined over any finite field GF(q). For GF(2^m), the curve has a different form:- $y^2 + xy = x^3 + ax^2 + b$, *where* $b \neq 0$.

Cryptography with Elliptic Curves

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, some kind of hard problem such as discrete logarithm or factorization of prime numbers is needed. Considering the equation, Q=kP, where Q,P are points in an elliptic curve, it is "easy" to compute Q given k,P , but "hard" to find k given Q,P. This is known as the elliptic curve logarithm problem. k could be so large as to make brute-force fail.

ECC Key Exchange

Pick a prime number p= 2^{180} and elliptic curve parameters a and b for the equation $y^2 \equiv x^3 + ax + b \pmod p$ which defines the elliptic group of points $E_p(a,b)$. Select generator point G=(x_1,y_1) in $E_p(a,b)$ such that the smallest value for which nG=O be a very large prime number. $E_p(a,b)$ and G are parameters of the cryptosystem known to all participants. The following steps take place:

- **A & B select private keys $n_A<n$, $n_B<n$**
- **compute public keys: $P_A=n_A \times G$, $P_B=n_B \times G$**
- **compute shared key: $K=n_A \times P_B$, $K=n_B \times P_A$ {same since $K=n_A \times n_B \times G$ }**

ECC Encryption/Decryption

As with key exchange system, an encryption/decryption system requires a point G and and elliptic group $E_p(a,b)$ as parameters. First thing to be done is to encode the plaintext message m to be sent as an x-y point $\mathbf{P_m}$. Each user chooses private key $n_A<n$ and computes public key $P_A=n_A \times G$. To encrypt and send a message to P_m to B, A chooses a random positive integer k and produces the ciphertext $\mathbf{C_m}$ consisting of the pair of points $\mathbf{C_m=\{kG, P_m+kP_b\}}$. here, A uses B's public key. To decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point

$$\mathbf{P_m + kP_b - n_B(kG) = P_m + k(n_B G) - n_B(kG) = P_m}$$

A has masked the message $\mathbf{P_m}$ by adding $k\mathbf{P_b}$ to it. Nobody but A knows the value of k, so even though $\mathbf{P_b}$ is a public key, nobody can remove the mask $k\mathbf{P_b}$. For an attacker to recover the message, he has to compute k given G and kG, which is assumed hard.

Security of ECC

To **protect** a 128 bit AES key it would take a RSA Key Size of 3072 bits whereas an ECC Key Size of 256 bits.

Computational Effort for Cryptanalysis of Elliptic Curve Cryptography Compared to RSA

Key Size	MIPS-Years
150	3.8×10^{10}
205	7.1×10^{18}
234	1.6×10^{28}

(a) Elliptic Curve Logarithms using the Pollard rho Method

Key Size	MIPS-Years
512	3×10^{4}
768	2×10^{8}
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

(b) Integer Factorization using the General Number Field Sieve

Hence for similar security ECC offers significant computational advantages.

Applications of ECC:

- Wireless communication devices
- Smart cards
- Web servers that need to handle many encryption sessions
- Any application where security is needed but lacks the power, storage and computational power that is necessary for our current cryptosystems

Digital Signature

The most important development from the work on public-key cryptography is the digital signature. Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. A digital signature is analogous to the handwritten signature, and provides a set of security capabilities that would be difficult to implement in any other way. It must have the following properties:

- It must verify the author and the date and time of the signature
- It must to authenticate the contents at the time of the signature
- It must be verifiable by third parties, to resolve disputes

Thus, the digital signature function includes the authentication function. A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender's private key, or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by further encrypting the entire

message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message and its signature. But these approaches are dependent on the security of the sender's private-key. Will have problems if it is lost/stolen and signatures forged. Need time-stamps and timely key revocation.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter, in a variety of possible arrangements. The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. These schemes can be implemented with either private or public-key algorithms, and the arbiter may or may not see the actual message contents.

Using Conventional encryption

$X \rightarrow A : M \,||\, E(K_{xa}, [ID_x \,||\, H(M)])$

$A \rightarrow Y : E(K_{ay}, [ID_x \,||\, M \,||\, E(K_{xa}, [ID_x \,||\, H(M)])] \,||\, T])$

- It is assumed that the sender X and the arbiter A share a secret key K_{xa} and that A and Y share secret key K_{ay}. X constructs a message M and computes its hash value H(m) . Then X transmits the message plus a signature to A. the signature consists of an identifier ID_X of X plus the hash value, all encrypted using K_{xa}.
- A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with K_{ay}. The message includes ID_X, the original message from X, the signature, and a timestamp.
- Arbiter sees message
- Problem : the arbiter could form an alliance with sender to deny a signed message, or with the receiver to forge the sender's signature.

Using Public Key Encryption

$A : ID_X \,||\, E(PR_X, [ID_X \,||\, E(PU_Y, E(PR_X, M))])$

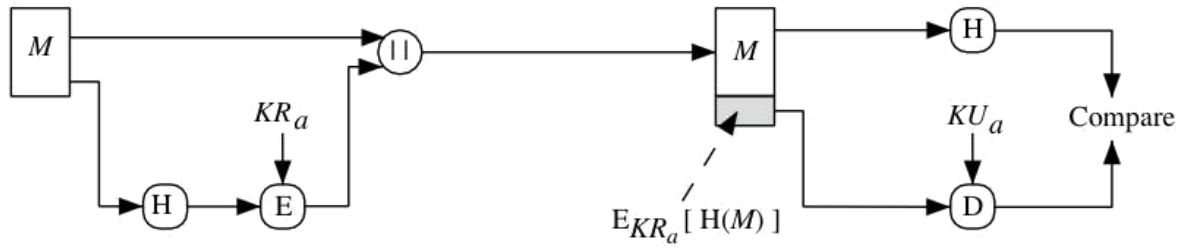
$A \rightarrow Y : E(PR_a, [ID_X \,||\, E(PU_Y, E(PR_X, M)) \,||\, T])$

- X double encrypts a message M first with X's private key, PR_X, and then with Y's public key, PU_Y. This is a signed, secret version of the message. This signed message, together with X's identifier , is encrypted again with PR_X and, together with ID_X, is sent to A. The inner, double encrypted message is secure from the arbiter (and everyone else except Y)
- A can decrypt the outer encryption to assure that the message must have come from X (because only X has PR_X). Then A transmits a message to Y, encrypted with PR_a. The message includes ID_X, the double encrypted message, and a timestamp.
- Arbiter does not see message

Digital Signature Standard (DSS)

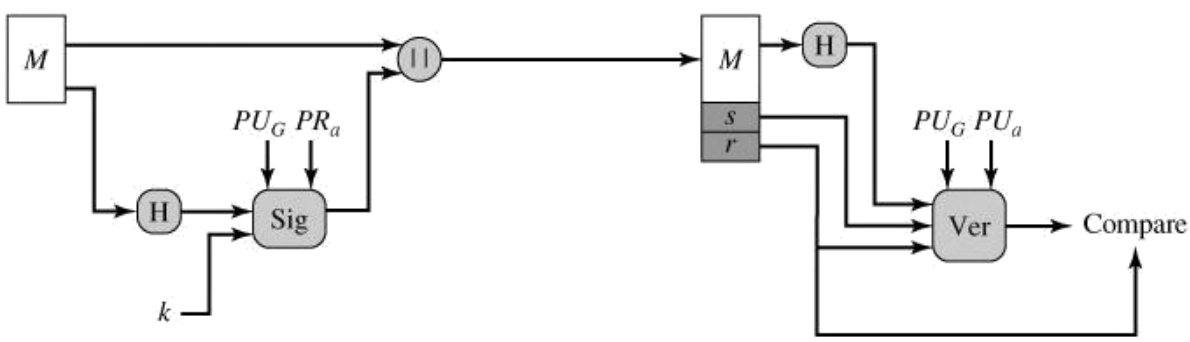
The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS uses an algorithm that is designed to provide only the digital signature function and cannot be used for encryption or key exchange, unlike RSA.

The RSA approach is shown below. The message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted.



The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G).The result is a signature consisting of two components, labeled s and r.



(b) DSS approach

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's

private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. Some of these threats are:

2 A user may gain access to a particular workstation and pretend to be another user operating from that workstation.

3 A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.

4 A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

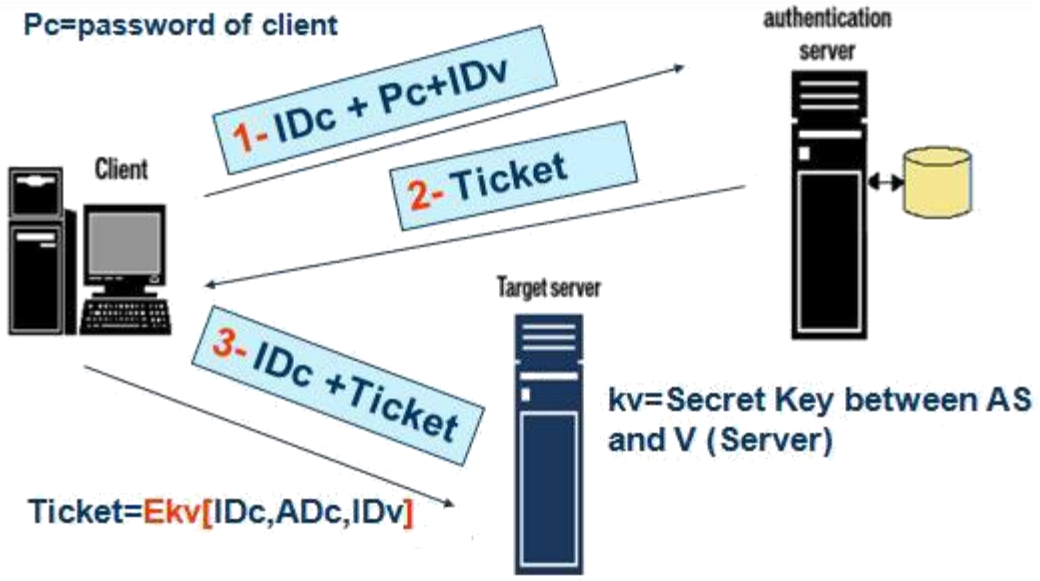
Two versions of Kerberos are in current use: Version-4 and Version-5. The first published report on Kerberos listed the following requirements:

- *Secure*: A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- *Reliable*: For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- *Transparent*: Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- *Scalable:* The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture

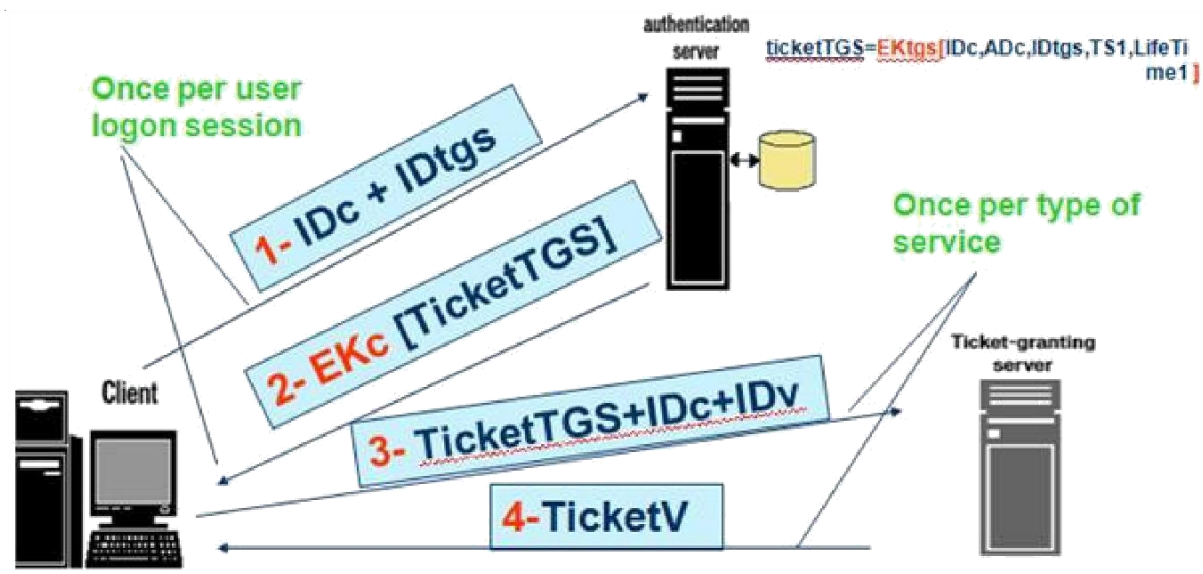
Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Version 4. Version 5 has been issued as a draft Internet Standard (RFC 1510)

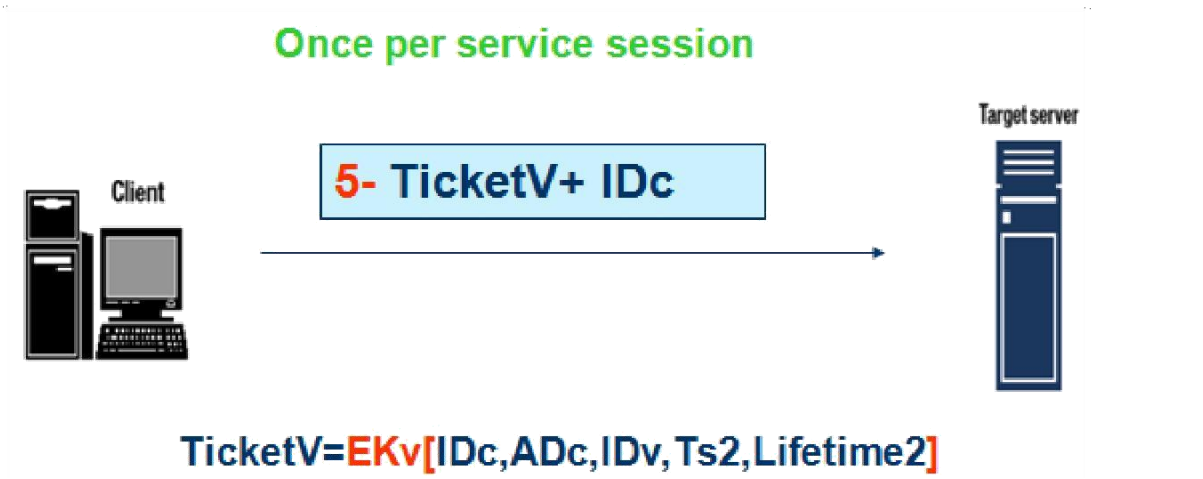
Kerberos Version 4

1.) **Simple dialogue**:



More Secure Dialogue





The Version 4 Authentication Dialogue

The full Kerberos v4 authentication dialogue is shown here divided into 3 phases.

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$$

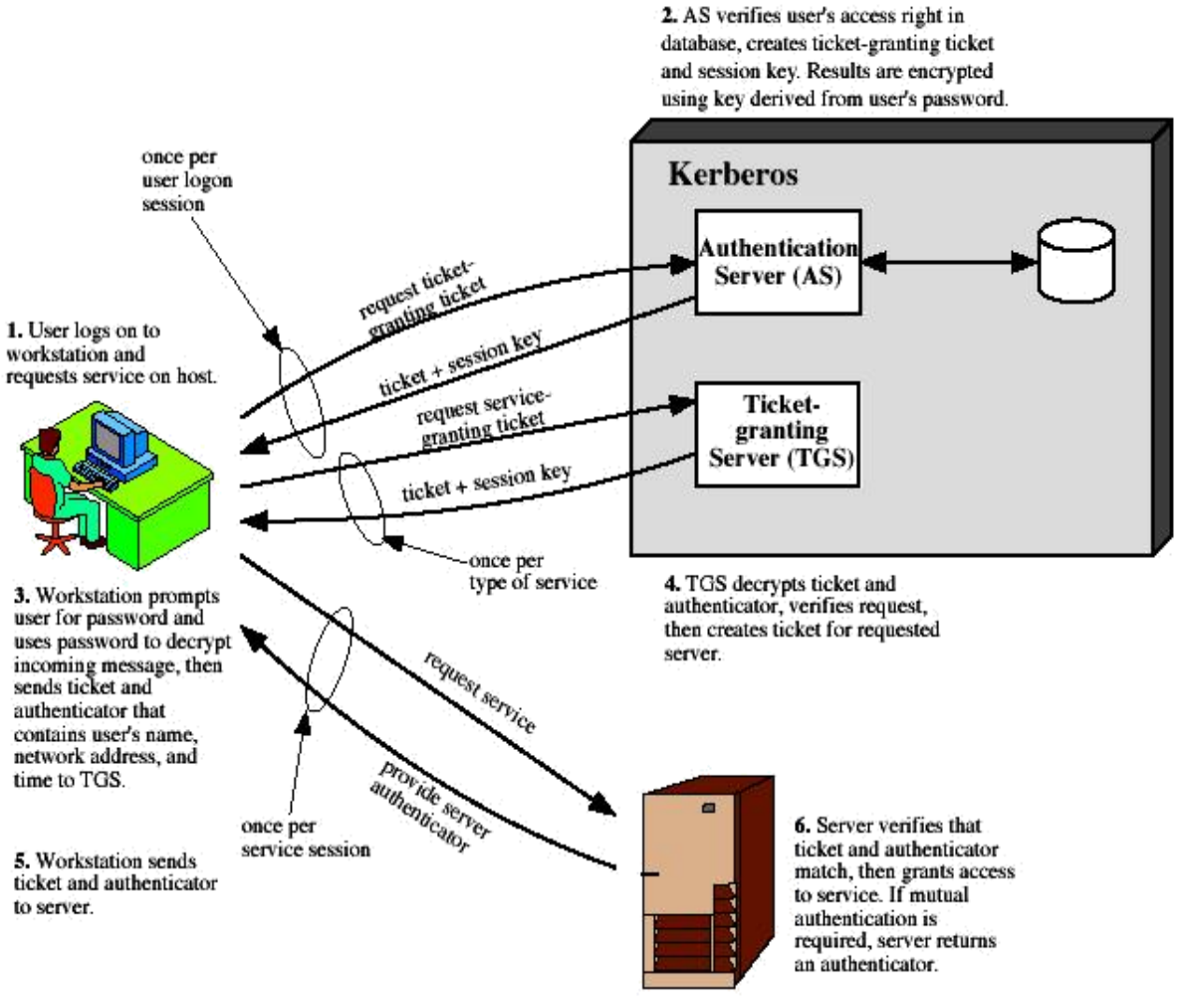
(c) Client/Server Authentication Exchange to obtain service

There is a problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information.

Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time. Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. C then sends the TGS a message that includes the ticket plus the ID of the requested service (message 3). The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator.

The server can decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6).

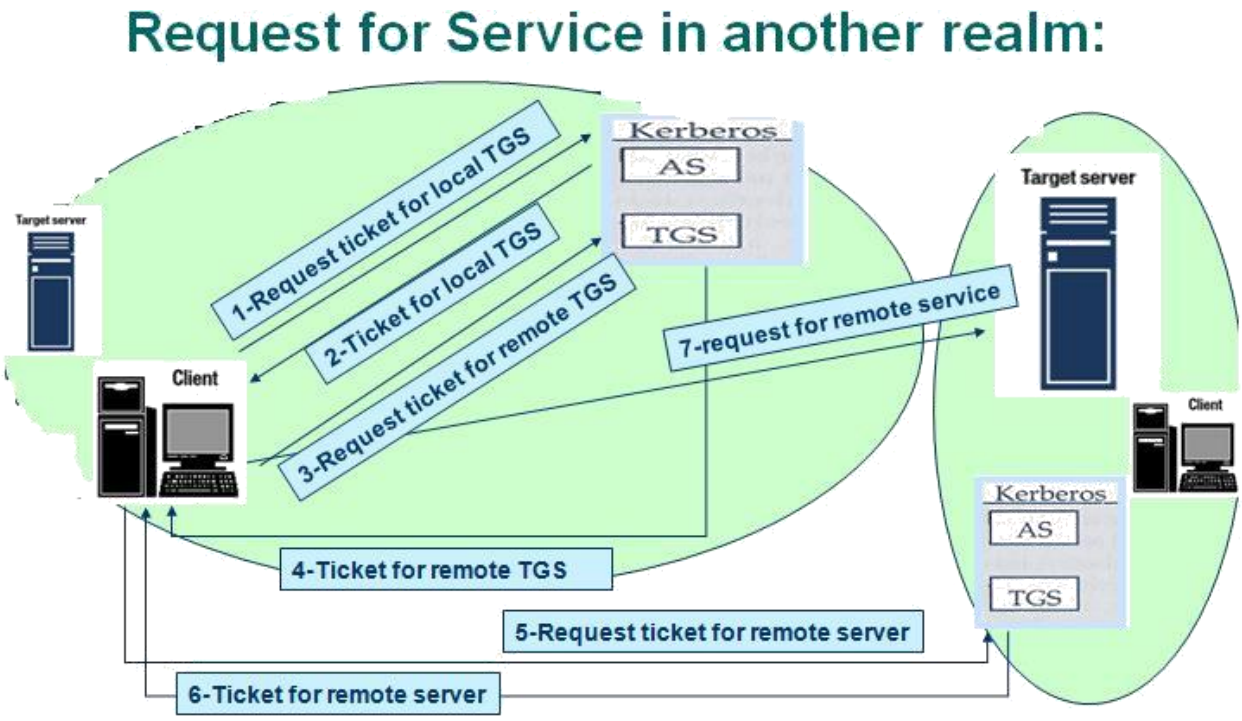
Overview of Kerberos



Kerberos Realms

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database, and are part of the same administrative domain. If have multiple realms, their Kerberos servers must share keys and trust each other.

The following figure shows the authentication messages where service is being requested from another domain. The ticket presented to the remote server indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request. One problem presented by the foregoing approach is that it does not scale well to many realms, as each pair of realms need to share a key.



The limitations of Kerberos version-4 are categorised into two types:

- Environmental shortcomings of Version 4:
  - Encryption system dependence: DES
  - Internet protocol dependence
  - Ticket lifetime
  - Authentication forwarding
  - Inter-realm authentication
- Technical deficiencies of Version 4:
  - Double encryption
  - Session Keys
  - Password attack

Kerberos version 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 in the areas of environmental shortcomings and technical deficiencies. It includes some new elements such as:

- Realm: Indicates realm of the user
- Options
- Times
  - From: the desired start time for the ticket
  - Till: the requested expiration time
  - Rtime: requested renew-till time
- Nonce: A random value to assure the response is fresh

The basic Kerberos version 5 authentication dialogue is shown here First, consider the **authentication service exchange.**

(1) C → AS Options ‖ ID_c ‖ $Realm_c$ ‖ ID_{tgs} ‖ $Times$ ‖ $Nonce_1$

(2) AS → C $Realm_c$ ‖ ID_C ‖ $Ticket_{tgs}$ ‖ $E(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$

$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS Options ‖ ID_v ‖ $Times$ ‖ ‖ $Nonce_2$ ‖ $Ticket_{tgs}$ ‖ $Authenticator_c$

(4) TGS → C $Realm_c$ ‖ ID_C ‖ $Ticket_v$ ‖ $E(K_{c,tgs}, [K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$

$Ticket_{tgs} = E(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$

$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$

$Authenticator_c = E(K_{c,tgs}, [ID_C \| Realm_c \| TS_1])$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V Options ‖ $Ticket_v$ ‖ $Authenticator_c$

(6) V → C $E_{K_{C,V}}$ [TS_2 ‖ Subkey ‖ Seq#]

$Ticket_v = E(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$

$Authenticator_c = E(K_{c,v}, [ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#])$

(c) Client/Server Authentication Exchange to obtain service

Message (1) is a client request for a ticket-granting ticket.

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS.

Now compare the **ticket-granting service** exchange for versions 4 and 5. See that message

(3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4. Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS. Finally, for the client/server authentication exchange, several new features appear in version 5, such as a request for mutual authentication. If required, the server responds with message (6) that includes the timestamp from the authenticator. The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

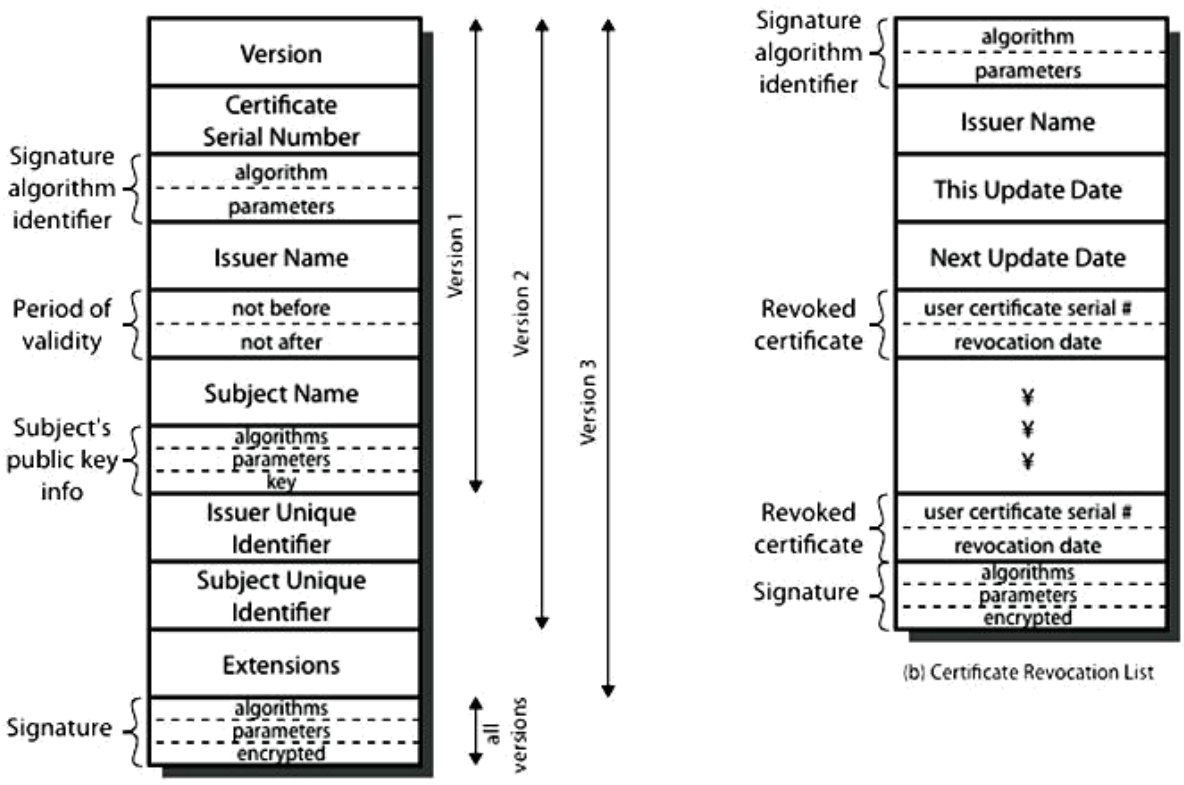
Advantages of Kerberos:

- User's passwords are never sent across the network, encrypted or in plain text
- Secret keys are *only* passed across the network in encrypted form
- Client and server systems mutually authenticate
- It limits the duration of their users' authentication.
- Authentications are **reusable** and **durable**
- Kerberos has been scrutinized by many of the top programmers, cryptologists and security experts in the industry

X.509 Authentication Service

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users. X.509 is based on the use of public-key cryptography and digital signatures.

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.



(b) Certificate Revocation List

The general format of a certificate is shown above, which includes the following elements:

- version 1, 2, or 3
- serial number (unique within CA) identifying certificate
- signature algorithm identifier
- issuer X.500 name (CA)
- period of validity (from - to dates)
- subject X.500 name (name of owner)
- subject public-key info (algorithm, parameters, key)
- issuer unique identifier (v2+)
- subject unique identifier (v2+)
- extension fields (v3)
- signature (of hash of all fields in certificate)

The standard uses the following notation to define a certificate:

$$\mathbf{CA{<<}A{>>} = CA\ \{V, SN, AI, CA, T_A, A, Ap\}}$$

Where,

Y <<X>> = the certificate of user X issued by certification authority Y

Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

User certificates generated by a CA have the following characteristics:

- Any user with CA's public key can verify the user public key that was certified
- No party other than the CA can modify the certificate without being detected
- because they cannot be forged, certificates can be placed in a public directory

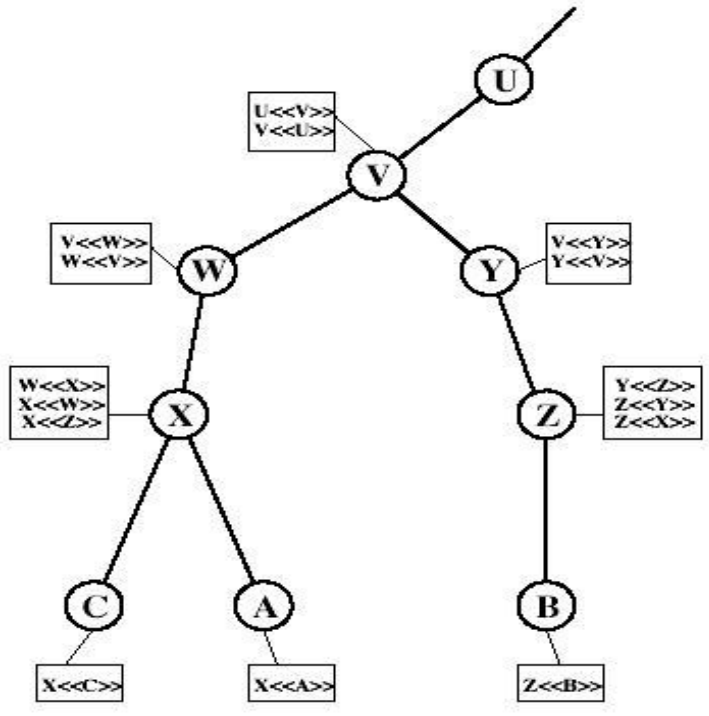
Scenario: Obtaining a User Certificate

If both users share a common CA then they are assumed to know its public key. Otherwise CA's must form a hierarchy and use certificates linking members of hierarchy to validate other CA's. Each CA has certificates for clients (forward) and parent (backward). Each client trusts parents certificates. It enables verification of any certificate from one CA by users of all other CAs in hierarchy.

A has obtained a certificate from the CA X1. B has obtained a certificate from the CA X2. A can read the B's certificate but cannot verify it. In order to solve the problem ,the Solution:

X1<<X2> X2<<B>>. A obtain the certificate of X2 signed by X1 from directory. → obtain X2's public key. A goes back to directory and obtain the certificate of B signed by X2. → obtain B's public key securely. The directory entry for each CA includes two types of certificates: Forward certificates: Certificates of X generated by other CAs Reverse certificates: Certificates generated by X that are the certificates of other CAs

X.509 CA Hierarchy



A acquires B certificate using chain:
X<<W>>W<<V>>V<<Y>>Y<<Z>> Z<<B>>

B acquires A certificate using chain:
Z<<Y>>Y<<V>>V<<W>>W<<X>> X<<A>>

Revocation of Certificates

Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

♌♏■♎♋♦♒	The user's private key is assumed to be compromised.
♌♏■♎♋♦♒	The user is no longer certified by this CA.
♌♏■♎♋♦♒	The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each **certificate revocation list (CRL) posted** to the directory is signed by the issuer and includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

Authentication Procedures

X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side.

1. One-Way Authentication: One way authentication involves a single transfer of information from one user (A) to another (B), and establishes the details shown above. Note that only the identity of the initiating entity is verified in this process, not that of the responding entity. At a minimum, the message includes a timestamp ,a nonce, and the identity of B and is signed with A's private key. The message may also include information to be conveyed, such as a session key for B.

- 1 message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message

2. Two-Way Authentication: Two-way authentication thus permits both parties in a communication to verify the identity of the other, thus additionally establishing the above details. The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B, and possible additional information for A.

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply

3. Three-Way Authentication: Three-Way Authentication includes a final message from A to B, which contains a signed copy of the nonce, so that timestamps need not be checked, for use when synchronized clocks are not available.

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed.

\endash The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.

\endash The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.

\endash There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.

\endash There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.

\endash It is important to be able to identify different keys used by the same owner at different

times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. X.509 version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored or not.