- The key distribution concept can be deployed in a number of ways.
- A typical scenario is illustrated in Figure.
- The scenario assumes that each user shares a unique master key with the key distribution center (KDC).
- Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key.
- The following steps occur.
  1. A issues a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N_1 (**nonce)**. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs with each request.
  2. The KDC responds with a message encrypted using K_a. Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:
     - The one-time session key, K_s, to be used for the session
     - The original request message, including the nonce, to enable A to match this response with the appropriate request

     Thus, A can verify that its original request. Also, the message includes two items intended for B:

     - The one-time session key, K_s, to be used for the session
     - An identifier of A ID_A,

     These last two items are encrypted with K_b (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.
  3. A stores the session key and forwards the $E(K_b, [K_s||ID_A])$ to B. Because this information is encrypted with K_b, it is protected from eavesdropping. B now knows the session key K_s, knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_b).
  4. Using the newly minted session key for encryption, B sends a nonce, N_2, to A.
  5. Also, using K_s, A responds with $f(N_2)$, where f is a function that performs some transformation on N_2.
- Note that the actual key distribution involves only steps 1 through 3, but that steps 4 and 5, as well as step 3, perform an authentication function.

Hierarchical Key Control

- It is not necessary to limit the key distribution function to a single KDC.
- Indeed, for very large networks, a hierarchy of KDCs can be established.
- For communication among entities within the same local domain, the local KDC is responsible for key distribution.
- If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC.
- In this case, any one of the three KDCs involved can actually select the key.
- The hierarchical concept can be extended to three or even more layers.

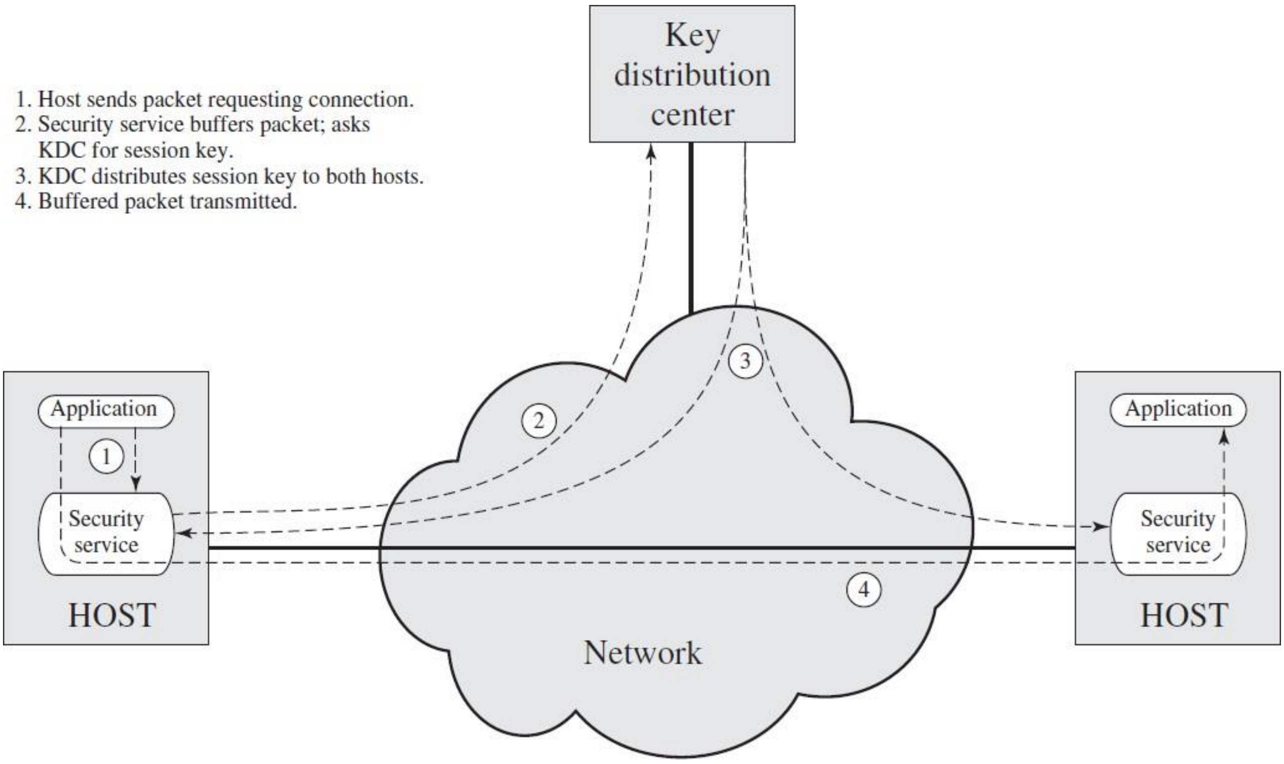
Session Key Lifetime

- The more frequently session keys are exchanged, the more secure they are, because the opponent has less cipher text to work with for any given session key.
- On the other hand, the distribution of session keys delays the start of any exchange and places a burden on network capacity.

- A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.
- For connection-oriented protocols, one obvious choice is to use the same session key for the length of time that the connection is open.
- If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically.
- For a connectionless protocol, it is not obvious how often one needs to change the session key.
- The most secure approach is to use a new session key for each exchange.
- However, this negates one of the principal benefits of connectionless protocols, which is minimum overhead and delay for each transaction.
- A better strategy is to use a given session key for a certain fixed period only or for a certain number of transactions.

A Transparent Key Control Scheme

- The scheme is useful for providing end-to-end encryption at a network or transport level in a way that is transparent to the end users.

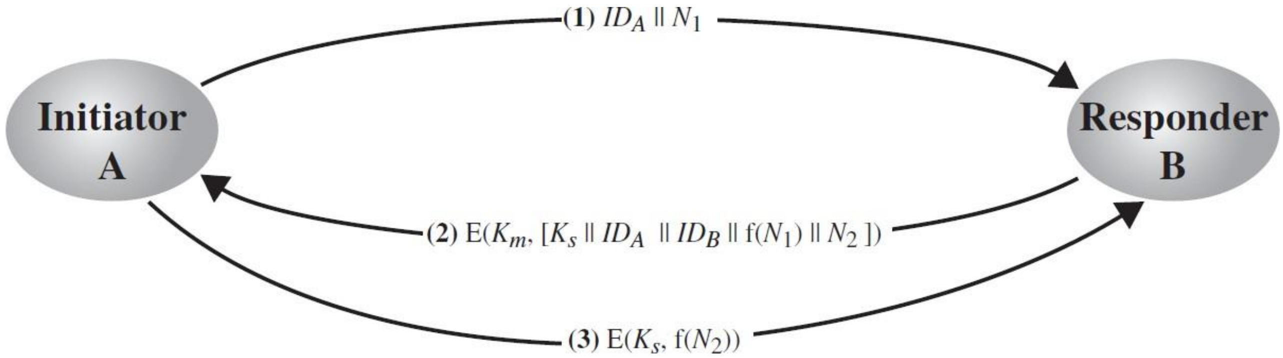


- The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as TCP.
- The steps involved in establishing a connection are shown in Figure.
- When one host wishes to set up a connection to another host, it transmits a connection request packet (step 1).
- The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2).
- The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC.
- If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3).

- The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4).
- All user data exchanged between the two end systems are encrypted by their respective SSMs using the onetime session key.

Decentralized Key Control

- The use of a KDC imposes the requirement that the KDC be trusted and be protected.
- This requirement can be avoided if key distribution is fully decentralized.
- Although full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context.
- A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution.
- Thus, there may need to be as many as master keys for a configuration with end systems.



- A session key may be established with the following sequence of steps.
  1. A issues a request to B for a session key and includes a nonce, N_1.
  2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value $f(N_1)$, and another nonce, N_2.
  3. Using the new session key, A returns $f(N_2)$ to B.

Controlling Key Usage

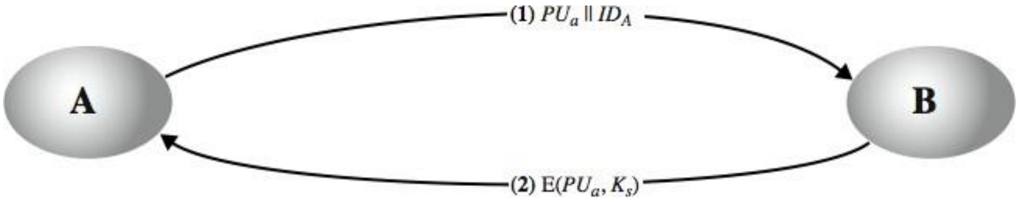
- The concept of a key hierarchy and the use of automated key distribution techniques greatly reduce the number of keys that must be manually managed and distributed.
- It also may be desirable to impose some control on the way in which automatically distributed keys are used.
- For example, in addition to separating master keys from session keys, we may wish to define different types of session keys on the basis of use, such as
  - Data-encrypting key, for general communication across a network
  - PIN-encrypting key, for personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications
  - File-encrypting key, for encrypting files stored in publicly accessible locations

Symmetric Key Distribution Using Asymmetric Encryption

One of the most important uses of a public-key cryptosystem is to encrypt secret keys for distribution.

Simple Secret Key Distribution

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



1. A generates a public/private key pair 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, which is 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.

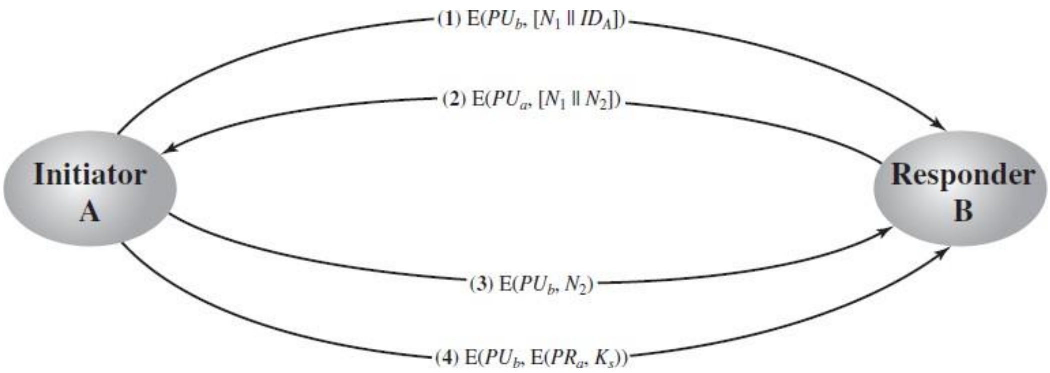
- A and B can now securely communicate using conventional encryption and the session key K_s.
- At the completion of the exchange, both A and B discard K_s.
- No keys exist before and after the communication.
- The protocol depicted is insecure against man-in-the-middle attack.

man-in-the-middle attack

- 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.

Secret Key Distribution with Confidentiality and Authentication

- We assumed that A and B have exchanged public key.



- 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.

- Scheme ensures both confidentiality and authentication in the exchange of a secret key.

A Hybrid Scheme

- Yet another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes.
- This scheme retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key.
- A public key scheme is used to distribute the master keys.
- The following rationale is provided for using this three-level approach:
  - ✓ **Performance**
  - ✓ **Backward compatibility**

Distribution of Public Keys

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

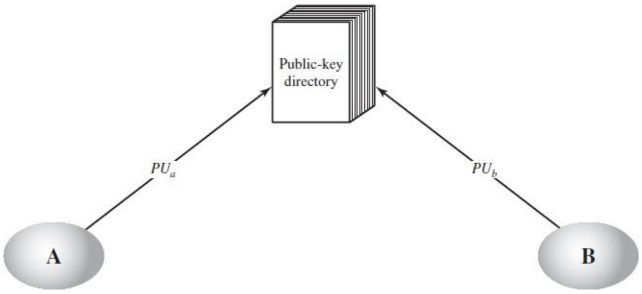
Public Announcement of Public Keys

- If there is some broadly accepted public-key algorithm, such as RSA, any participant can broadcast the key to the community at large.

- Although this approach is convenient, it has a major weakness.
- Some user could pretend to be user A and send a public key to another participant or broadcast such a public key.
- Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

Publicly Available Directory

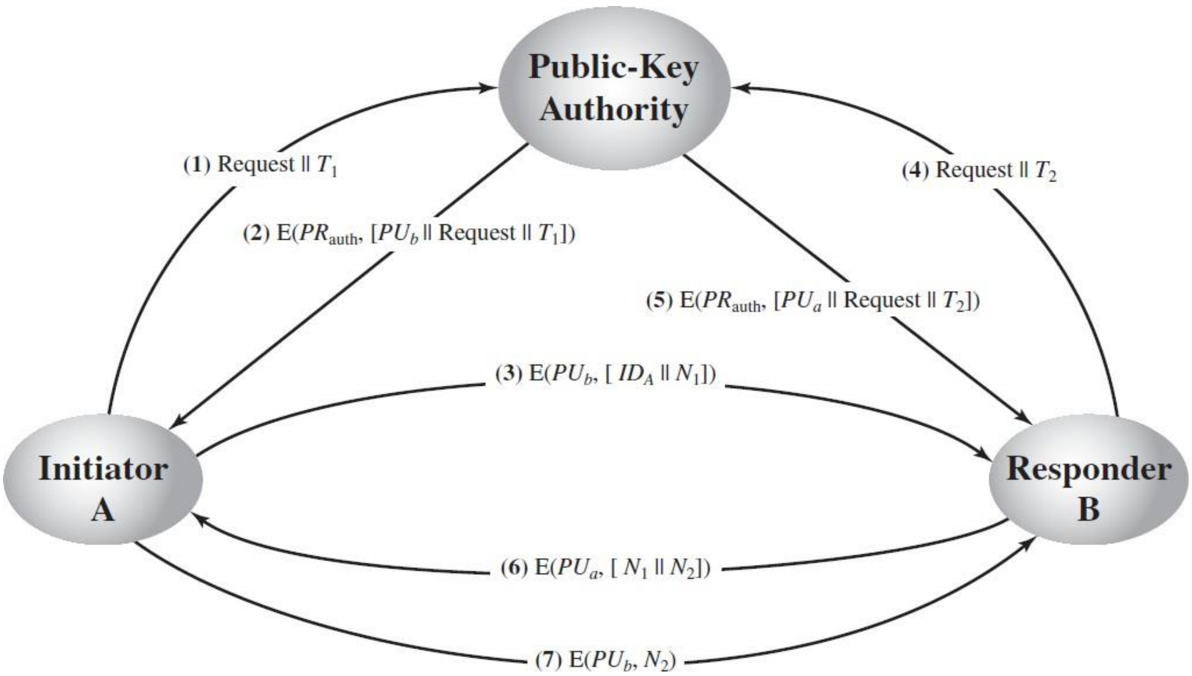
- A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys.
- Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization.



- Such a scheme would include 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.
  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 is clearly more secure than individual public announcements but still has vulnerabilities.
- If anyone succeeds in obtaining the private key of the directory authority then he can pass false public key.
- Another way to achieve the same end is for the adversary to 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.
- A typical scenario is illustrated in Figure.

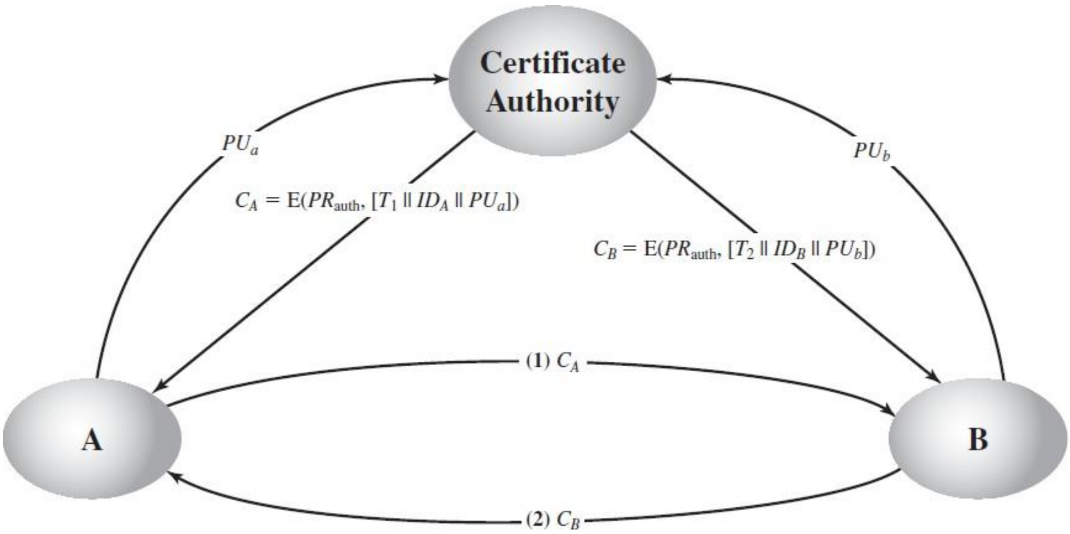


- Assumes that a central authority maintains a dynamic directory of public keys of all participants.
- Each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key.
- The following steps occur.
  1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.

2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth}.Thus,A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:
   - B's public key, PU_b, which A can use to encrypt messages destined for B
   - The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
   - The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key
3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
4. -Step 4 and 5 are similar to 2 and 3.
5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.
6. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (3), the presence of in message (6) assures A that the correspondent is B.
7. A returns N_2, which is encrypted using B's public key, to assure B that its correspondent is A.

Public-Key Certificates

- The directory of names and public keys maintained by the authority is vulnerable to tampering.
- An alternative approach, first suggested by Kohnfelder, is to use certificates.
- In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party.
- Typically, the third party is a certificate authority, such as a government agency or a financial institution that is trusted by the user community.
- A user can present his or her public key to the authority in a secure manner and obtain a certificate.
- The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature.
- 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.
- We can place the following requirements on this scheme:

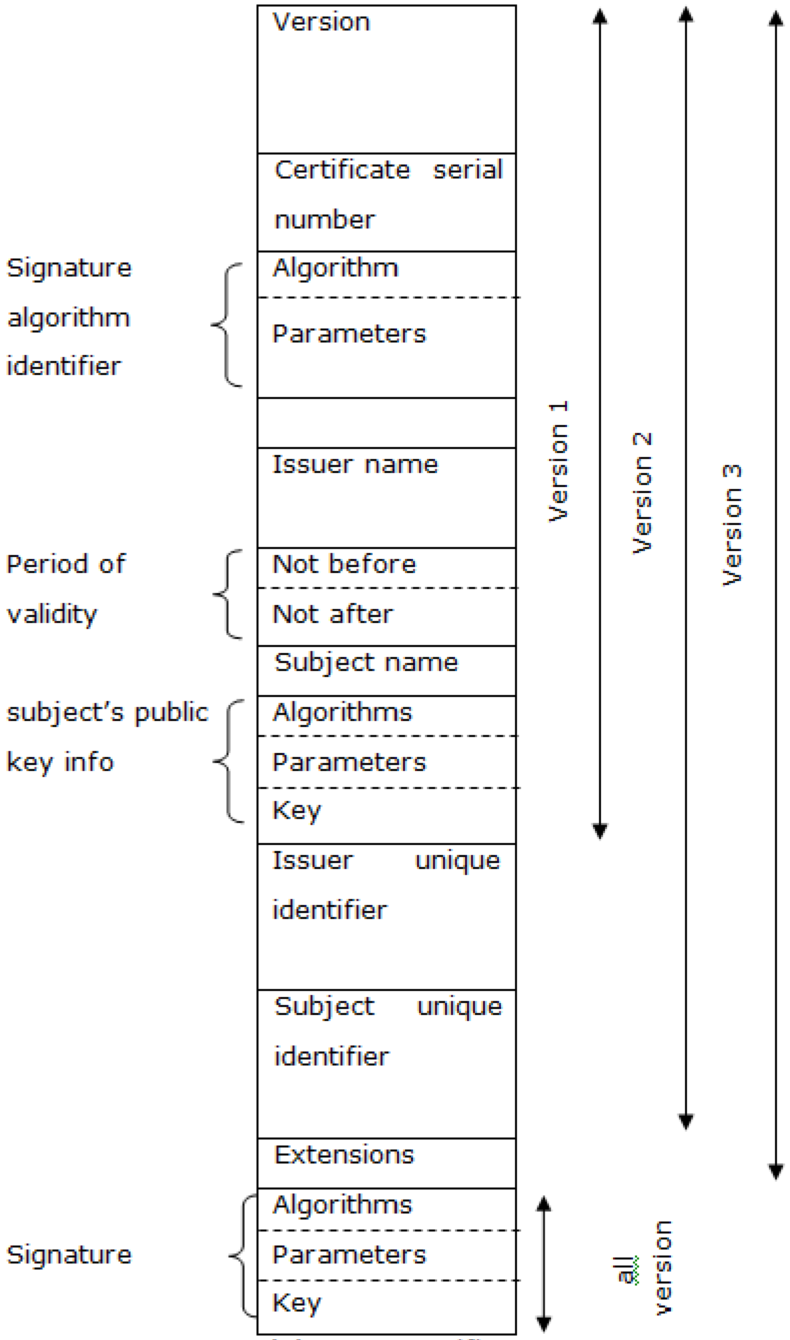


1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any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

X.509 Certificates

- X.509 provides authentication services and defines authentication protocols.
- X.509 uses X.500 directory which contains:
  - Public key certificates
  - Public key of users signed by certification authority
- X.509 certificate format is used in S/MIME, IP Security, and SSL/TLS.
- X.509 is based on the use of public-key cryptography (preferably RSA) and digital signatures.

X.509 includes the following elements.



(a) X.509 Certificate

- **Version**: Differentiates among successive versions of the certificate format; the default is version 1. Two other versions (2 and 3) are also available as shown in the figure.

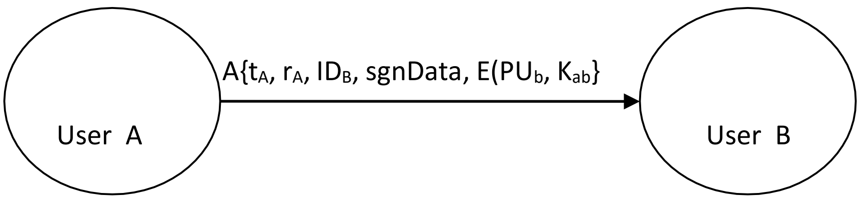
- **Serial number**: An integer value, unique within the issuing CA, different for each certificate.
- **Signature algorithm identifier**: The algorithm used to sign the certificate, together with any associated parameters.
- **Issuer name**: X.500 name of the CA that created and signed this certificate.
- **Period of validity**: Consists of two dates: the first and last on which the certificate is valid.
- **Subject name**: The name of the user to whom this certificate refers.
- **Subject's public-key information**: The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier**: An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier**: An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions**: A set of one or more extension fields.
- **Signature**: Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

Authentication Procedures

- X.509 supports three types of authenticating using public key signatures. The types of authentication are
  1. One-way authentication
  2. Two- way authentication
  3. Three- way authentication

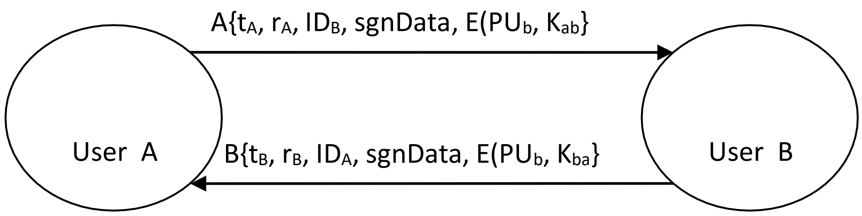
One-way authentication

- It involves single transfer of information from one user (say A) to other (B).
- This method authenticates the identity of A to B and the integrity of message.
- Here, message in the {} is signed by A.
- sgnData is the information that needs to be conveyed.
- $\mathbf{t_A}$ is timestamp and rA is the nonce.



Two- way authentication

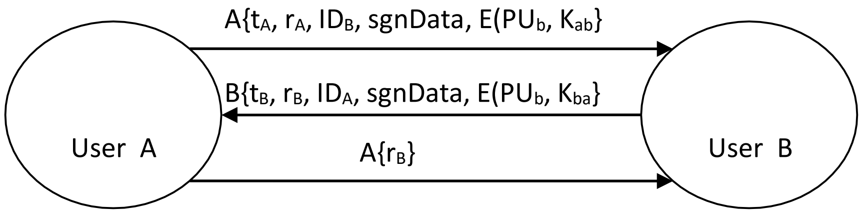
- Two- way authentication allows both parties to communicate and verify the identity of each other.



Three- way authentication

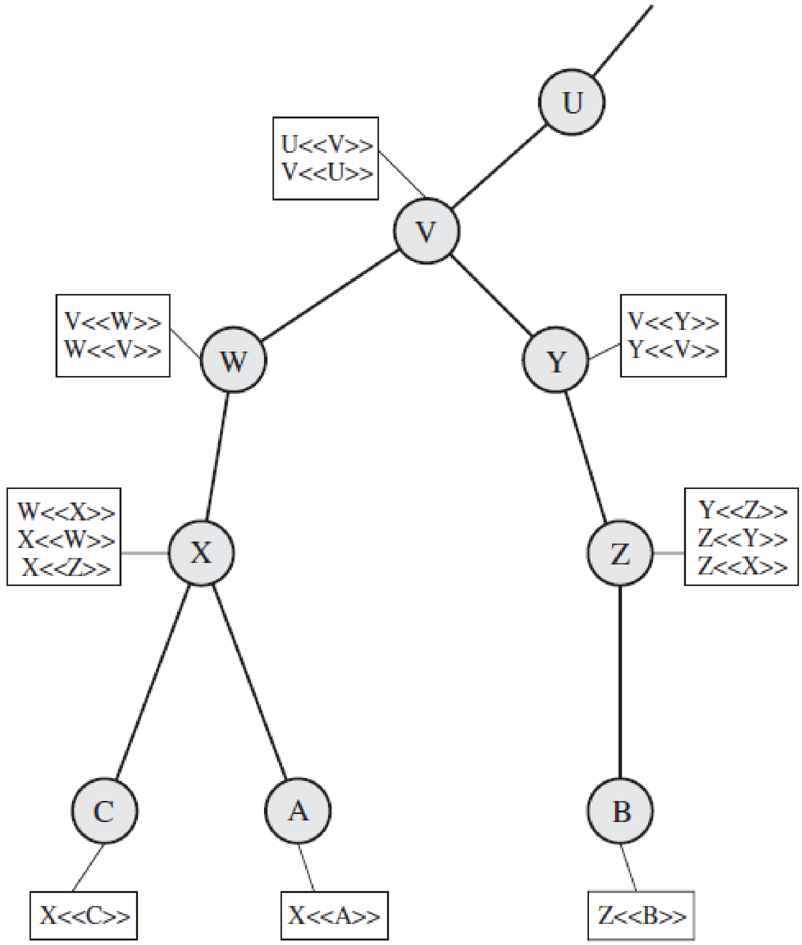
- Three- way authentication is used where synchronized clocks are not available.

- This method includes an additional message from A.



Obtaining Certificates in X.509

- Any user can verify a certificate if he has the public key of the CA that issued the certificate.
- Since certificates are unforgeable, they are simply stored in the directory.
- 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.
- Users subscribed to same CA can obtain certificate from the directory.
- A user may directly send the certificate to the user.
- However, multiple CAs are there and users subscribed to different CAs may want to communicate with each other.



- Suppose, A has obtained a certificate from certification authority X1 and B has obtained a certificate from CA X2.

- If A does not know the public key of X2, then B's certificate, issued by X2, is useless to A because A can read B's certificate, but A cannot verify the signature.
- But if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:
  - A obtains the certificate of X2 signed by X1 from the directory. A securely knows X1's public key, so A can obtain X2's public key from its certificate and verify X1's signature on the certificate.
  - A then obtains the certificate of B signed by X2. A now has a copy of X2's public key, so A can verify the signature and securely obtain B's public key.
  - In this case, A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as:

X1<<X2>> X2 <<B>>

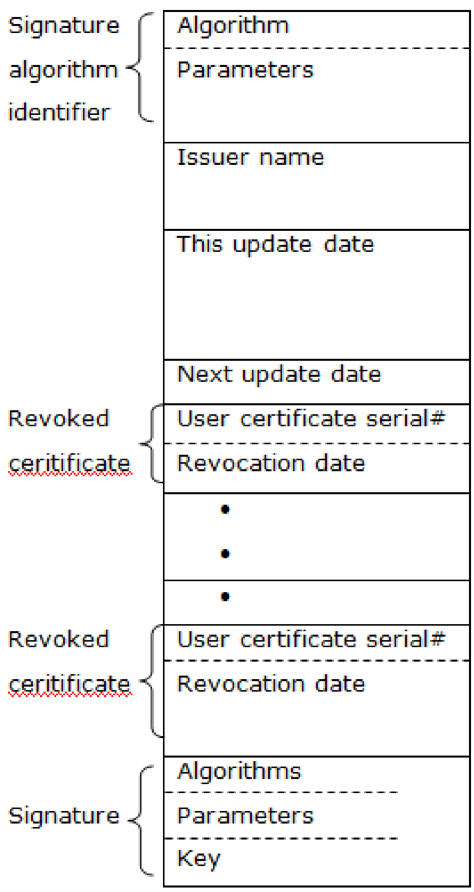
- Any level of hierarchy can be followed to produce a chain in this way. For example, in the figure given below, A can establish a certification path to B in the following way:

X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

- When A has obtained these certificates, it can decrypt the certification path in sequence to recover a copy of B's public key.
- Using this public key, A can send encrypted messages to B.
- If B requires A's public key, it can be obtained in the similar way.

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

Revocation of Certificates

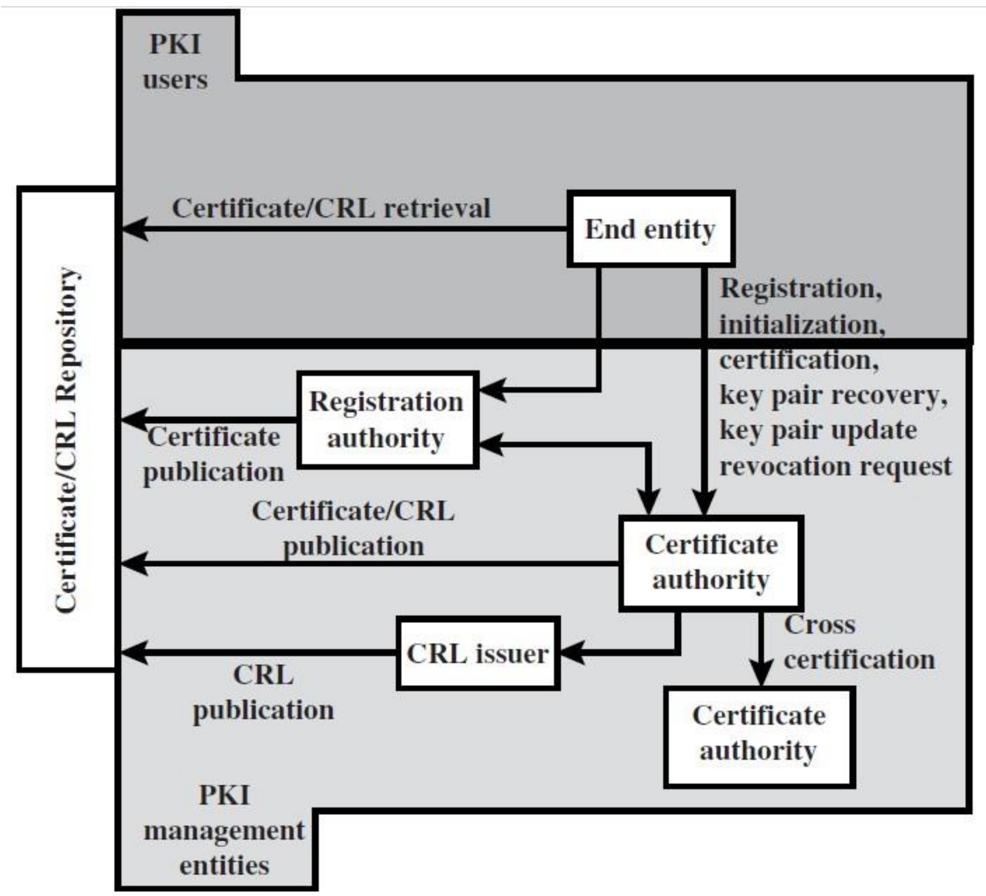


(b) Certificate Revocastion List

- The certificates have an expiry time.
- However, certificates need to be revoked if,
  - The user's private key has been compromised.
  - The user's certificate has been compromised.
  - The user is no longer certified by the CA.
- 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.
- 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.
- The certificate revocation list is shown in the figure.
- Every user must check the CRL before using other user's public key.

Public-Key Infrastructure

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



- These elements are
  - **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate.
  - **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities.
  - **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end entity registration process but can assist in a number of other areas as well.
  - **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
  - **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

PKIX Management Functions

- PKIX identifies a number of management functions that potentially need to be supported by management protocols which are:
  - Registration
  - Initialization
  - Certification
  - Key pair recovery
  - Key pair update
  - Revocation request
  - Cross certification

PKIX Management Protocols

- The PKIX working group has defines two alternative management protocols.
- RFC 2510 defines the certificate management protocols (CMP).
- Within CMP, each of the management functions is explicitly identified by specific protocol exchanges.
- CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models.
- RFC 2797 defines certificate management messages over CMS (CMC).
- Where CMS refers to RFC 2630, and cryptographic message syntax.
- CMC is built on earlier work and is intended to leverage existing implementations.
- Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.