Cryptography and Network Security Unit-3 (First Half)

Authentication Applications

- will consider authentication functions
- developed to support application-level authentication & digital signatures
- will consider Kerberos – a private-key authentication service
- then X.509 - a public-key directory authentication service

Kerberos

- trusted key server system from MIT
- provides centralised private-key third-party authentication in a distributed network
  - allows users access to services distributed through network
  - without needing to trust all workstations
  - rather all trust a central authentication server
- two versions in use: 4 & 5

Kerberos Requirements

- its first report identified requirements as:
  - secure
  - reliable
  - transparent
  - scalable
- implemented using an authentication protocol based on Needham-Schroeder

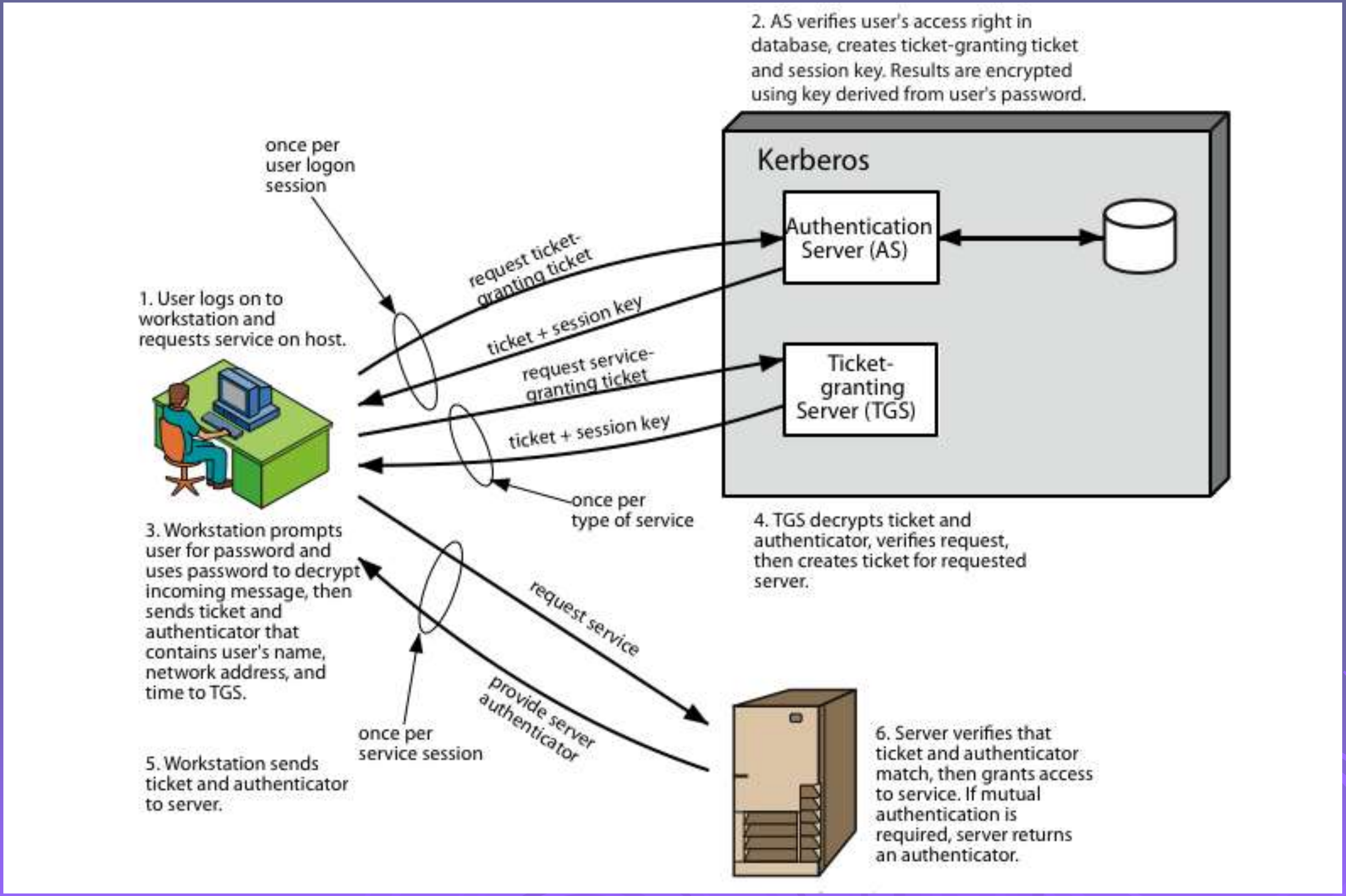
Kerberos v4 Overview

- a basic third-party authentication scheme
- have an Authentication Server (AS)
  - users initially negotiate with AS to identify self
  - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
- have a Ticket Granting server (TGS)
  - users subsequently request access to other services from TGS on basis of users TGT

Kerberos v4 Dialogue

1. obtain ticket granting ticket from AS
   - once per session
2. obtain service granting ticket from TGT
   - for each distinct service required
3. client/server exchange to obtain service
   - on every service request

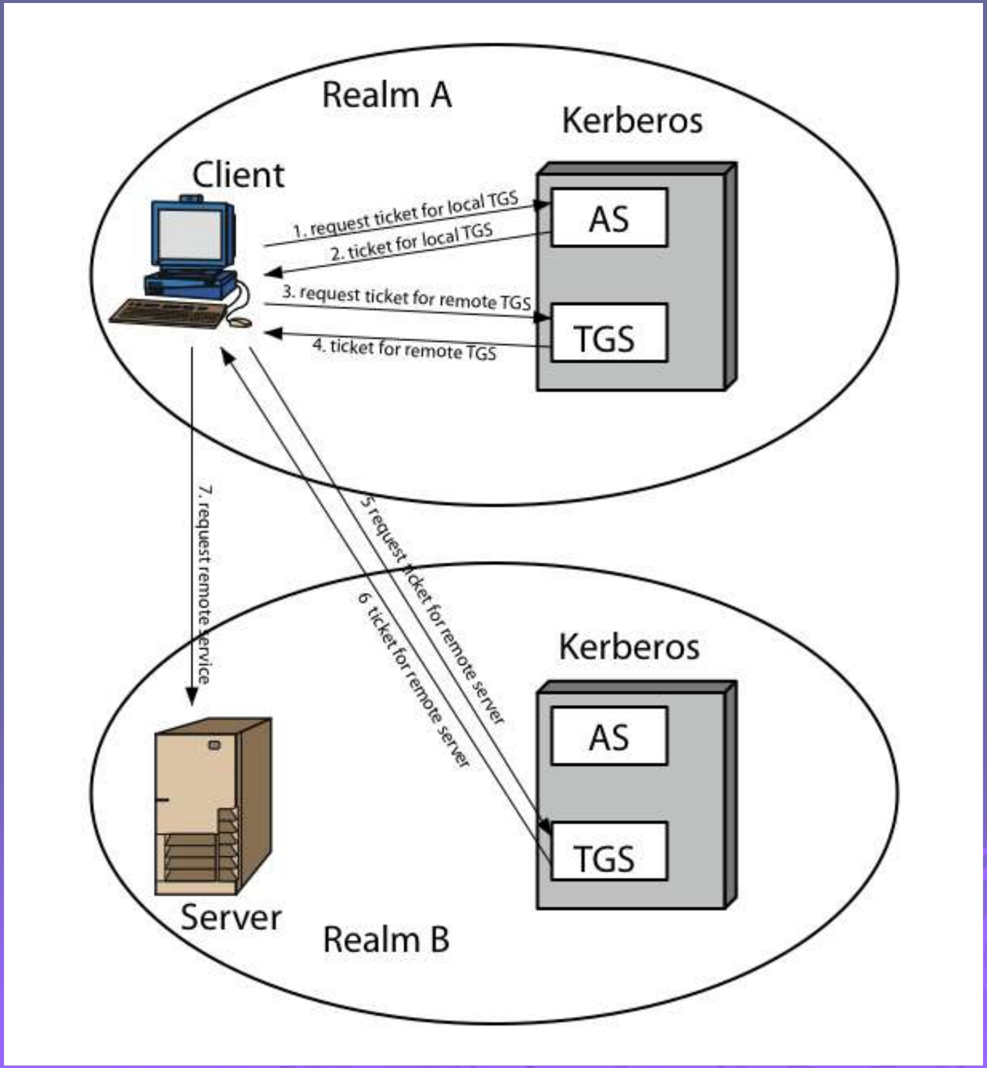
Kerberos 4 Overview


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
1. User logs on to workstation and requests service on host.
once per type of service
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Kerberos Realms

- a Kerberos environment consists of:
  - a Kerberos server
  - a number of clients, all registered with server
  - application servers, sharing keys with server
- this is termed a realm
  - typically a single administrative domain
- if have multiple realms, their Kerberos servers must share keys and trust

Kerberos Realms


Realm A
Kerberos
Client
AS
TGS
1. request ticket for local TGS
2. ticket for local TGS
3. request ticket for remote TGS
4. ticket for remote TGS
5 request ticket for remote server
6 ticket for remote server
7. request remote service
Kerberos
AS
TGS
Server
Realm B

Kerberos Version 5

- developed in mid 1990’s
- specified as Internet standard RFC 1510
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption alg, network protocol, byte order, ticket lifetime, authentication forwarding, interrealm auth
  - and technical deficiencies
    - double encryption, non-std mode of use, session keys, password attacks

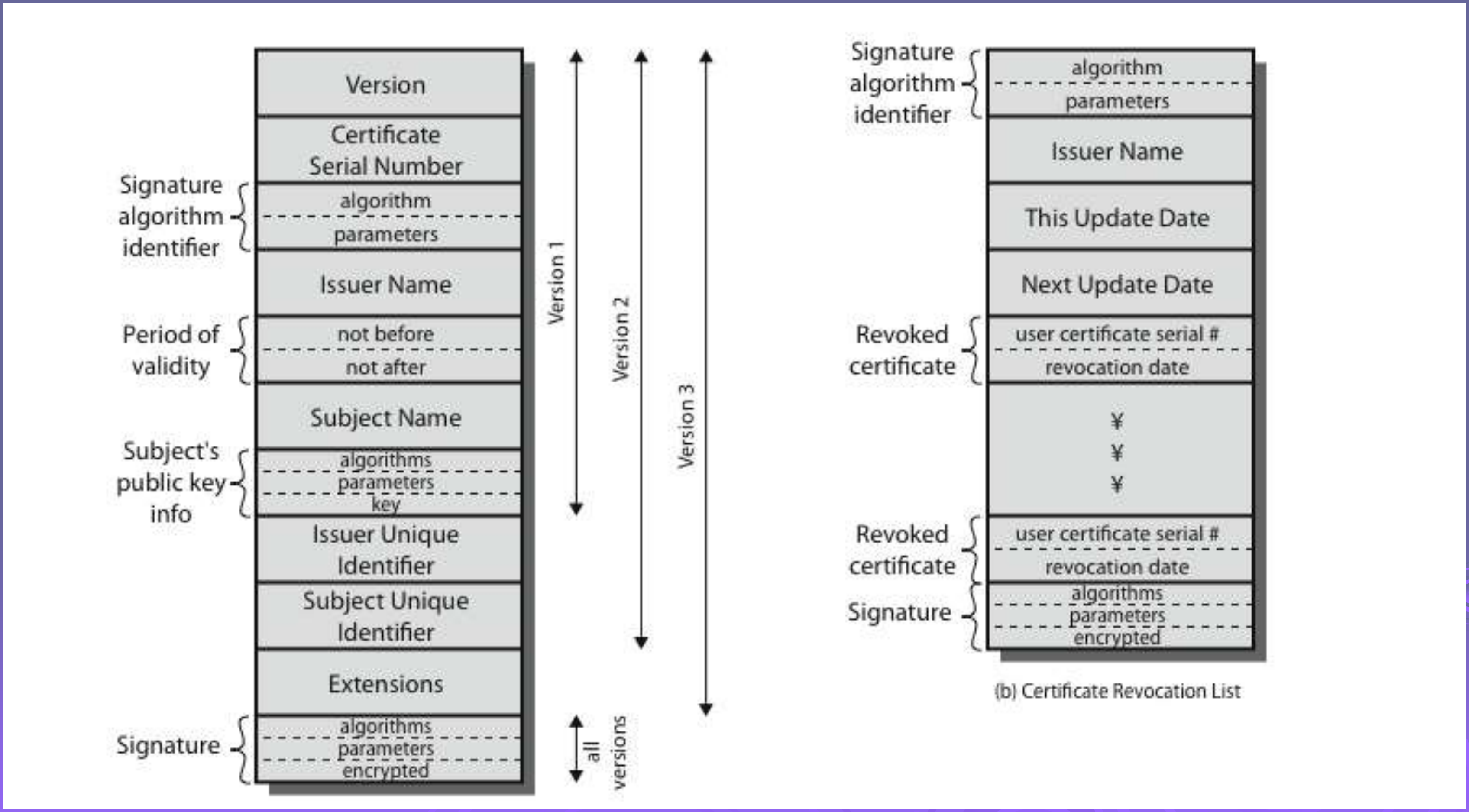
X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining user info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended
- X.509 certificates are widely used

X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - 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)
- notation `CA<<A>>` denotes certificate for A signed by CA

X.509 Certificates



(b) Certificate Revocation List

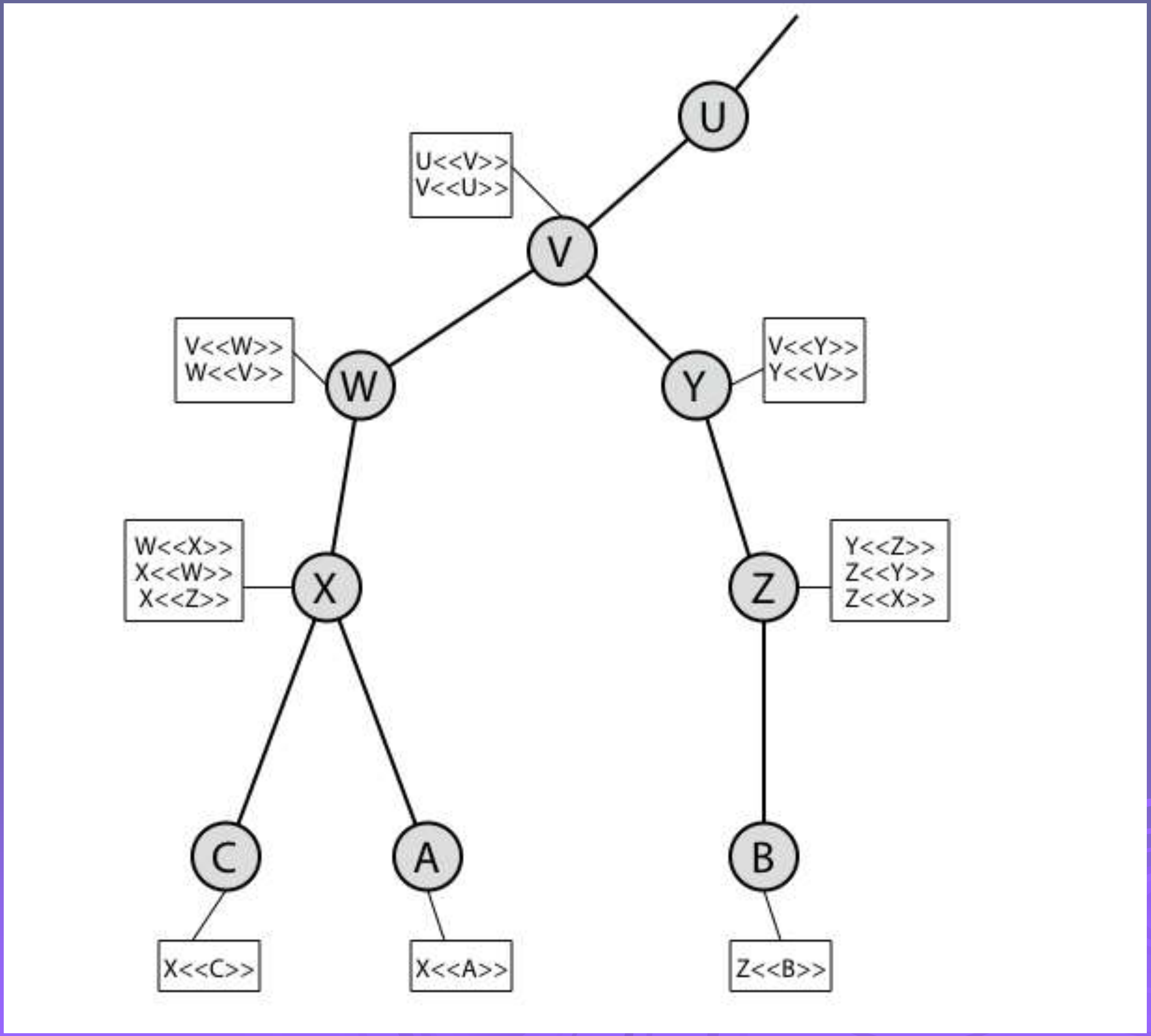
Obtaining a Certificate

- any user with access to CA can get any certificate from it
- only the CA can modify a certificate
- because cannot be forged, certificates can be placed in a public directory

CA Hierarchy

- if both users share a common CA then they are assumed to know its public key
- otherwise CA's must form a hierarchy
- 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
- enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA Hierarchy Use



Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
- CA’s maintain list of revoked certificates
  - the Certificate Revocation List (CRL)
- users should check certificates with CA’s CRL

Authentication Procedures

- X.509 includes three alternative authentication procedures:
- One-Way Authentication
- Two-Way Authentication
- Three-Way Authentication
- all use public-key signatures

One-Way Authentication

- 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
- message must include timestamp, nonce, B's identity and is signed by A
- may include additional info for B
  - eg session key

Two-Way Authentication

- 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
- reply includes original nonce from A, also timestamp and nonce from B
- may include additional info for A

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from A back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

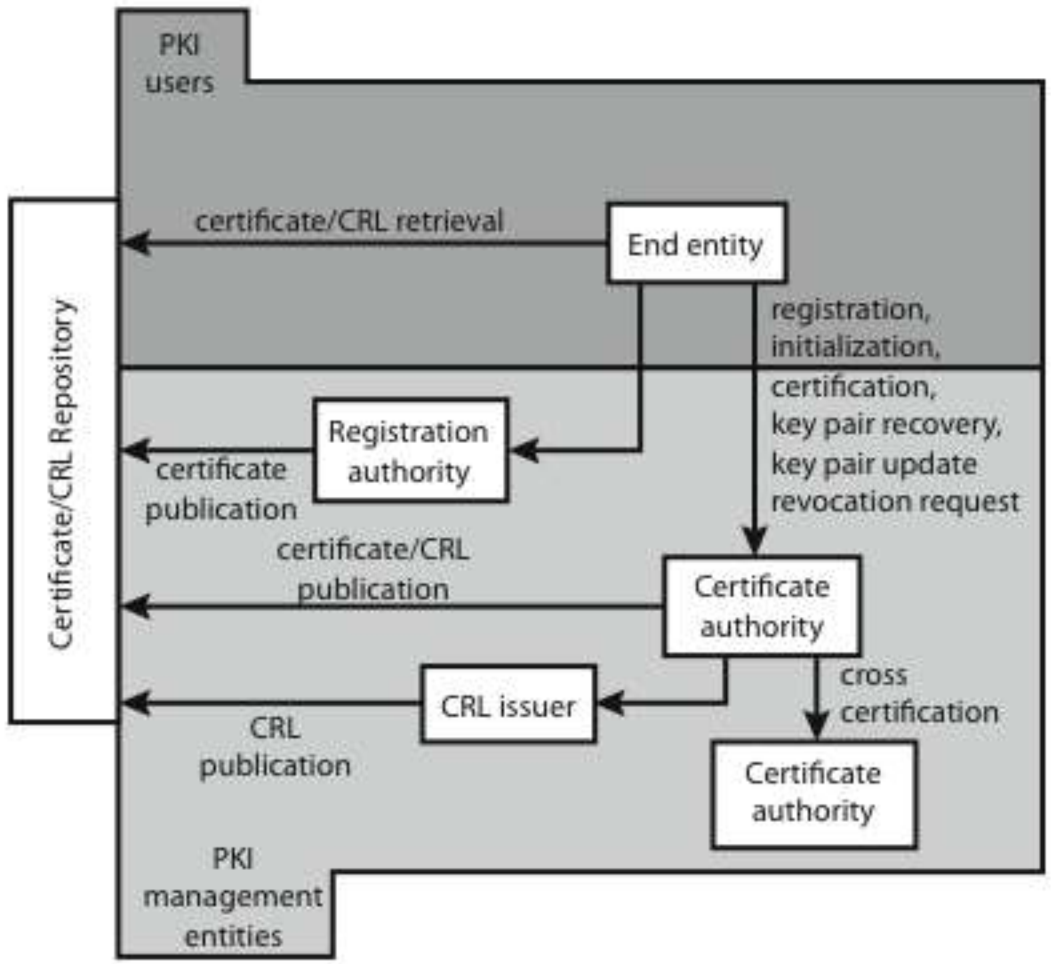
X.509 Version 3

- has been recognised that additional information is needed in a certificate
  - email/URL, policy details, usage constraints
- rather than explicitly naming new fields defined a general extension method
- extensions consist of:
  - extension identifier
  - criticality indicator
  - extension value

Certificate Extensions

- key and policy information
  - convey info about subject & issuer keys, plus indicators of certificate policy
- certificate subject and issuer attributes
  - support alternative names, in alternative formats for certificate subject and/or issuer
- certificate path constraints
  - allow constraints on use of certificates by other CA's

Public Key Infrastructure


PKI users
certificate/CRL retrieval
End entity
registration, initialization, certification, key pair recovery, key pair update revocation request
Certificate/CRL Repository
Registration authority
certificate publication
certificate/CRL publication
Certificate authority
CRL issuer
cross certification
CRL publication
Certificate authority
PKI management entities

Summary

- have considered:
  - Kerberos trusted key server system
  - X.509 authentication and certificates

2. **Higher Level:** Lower layer functions are used to create a protocol that enables a receiver to verify the authenticity of message

The different types of functions that may be used to produce an authenticator are as follows:

1. **Message encryption:** The cipher text of the entire message serves as its authenticator.

2. **Message Authentication Code (MAC):** A public function of the message and a secret key that produces a fixed length value serves as the authenticator.

3. **Hash function:** A public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

4.9.KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

Motivation

A distributed architecture consists of dedicated user workstations (clients) and distributed or centralized servers. In this environment, there are three approaches to security:

- Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
- Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
- Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

The following are the **requirements for Kerberos**:

- **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 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.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on Needham and Schroeder.

It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, and then the authentication service is secure if the Kerberos server itself is secure.

Two versions of Kerberos are in common use. **Version 4** and **Version 5**

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service

1.A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. To counter this threat, servers must be able to confirm the identities of clients who request service. But in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. The simple authentication dialogue is as follows:

1. C >> AS: $ID_c||P_c||ID_v$

2. AS >> C: Ticket

3. C >> V: $ID_c||$Ticket

Ticket= $EK_v(ID_c||ADc||ID_v)$

C	: Client,
AS	: Authentication Server,
V	: Server, ID_c : ID of the client,
P_c	: Password of the client,
AD_c	: Address of client, ID_v : ID of the server,
K_v	: secret key shared by AS and V,
||	: concatenation.

2.A More Secure Authentication Dialogue

There are two major problems associated with the previous approach:

- Plaintext transmission of the password.
- Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, and a new server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

Once per user logon session:-

1. C >> AS: $ID_c||ID_{tgs}$
2. AS >> C: Ek_c ($Ticket_{tgs}$)

Once per type of service:

3. C >> TGS: $ID_{c|}|ID_v||Ticket_{tgs}$
4. TGS >> C: $ticket_v$

Once per service session:

5. C >> V: $ID_c||$ $Ticket_v$

$Ticket_{tgs}$= $Ekt_{gs}(ID_c||AD_c||IDt_{gs}||TS_1||Lifetime_1)$

$Ticket_v$= $Ek_v(ID_c||AD_c||ID_v||TS2||Lifetime_2)$

C: Client, AS: Authentication Server, V: Server,
IDc : ID of the client, Pc:Password of the client, ADc: Address of client,
IDv : ID of the server, Kv: secret key shared by AS and V,
|| : concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime: lifetime of the ticket.

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket.

Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested.

Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password.

When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message.

If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket
4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server.

Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password.

Note that the ticket is encrypted with a secret key (K_v) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

Kerberos V4 Authentication Dialogue Message Exchange

Two additional problems remain in the more secure authentication dialogue:

- Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

- Requirement for the servers to authenticate themselves to users.

The actual Kerberos protocol version 4 is as follows
:

- A basic third-party authentication scheme
- Have an Authentication Server (AS)
    - Users initially negotiate with AS to identify self
    - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)

- Have a Ticket Granting
    - Users subsequently request access to other services from TGS on basis of users TGT

(a) Authentication service exchange: to obtain ticket granting ticket
(1) C → AS : ID_C II ID_{tgs} II TS_1
(2) AS → C : EKc [$K_{c,tgs}$ II ID_{tgs} II TS_2 II $Lifetime_2$ II $Ticket_{tgs}$]
(b) Ticket-Granting Service Exchange: to obtain service-granting ticket

(3) C → TGS: ID_v II $Ticket_{tgs}$ II $Authenticator_c$

(4) TGS → C: $EK_{c,tgs}[K_{c,y}$ II ID_v II TS_4 II $Ticket_v]$

$Ticket_{tgs} = E_{K,tgs}[K_{c,tgs}$ II ID_C II AD_C II ID_{tgs} IITS_2 II $Lifetime_2]$

$Ticket_v = E_{Kv}[K_{c,v}$ II ID_C II AD_C II ID_v IITS_4 II $Lifetime_4]$

$Authenticator_C = E_{Ktgs}$ [ID_C II AD_C II TS_3]

(c) Client/Server Authentication Exchange: to obtain service

(5) C → V : $Ticket_v$ II $Authenticator_c$

(6) V → C: $E_{kc,v}[TS_5 +1]$

$Ticket_v = EK_v[K_{c,v}$ II ID_C II AD_C II Id_v II TS_4 II $Lifetime_4]$

$Authenticator_c = EK_{tgs}$ [ID_C II AD_C II TS_3]

Kerberos 4 Overview

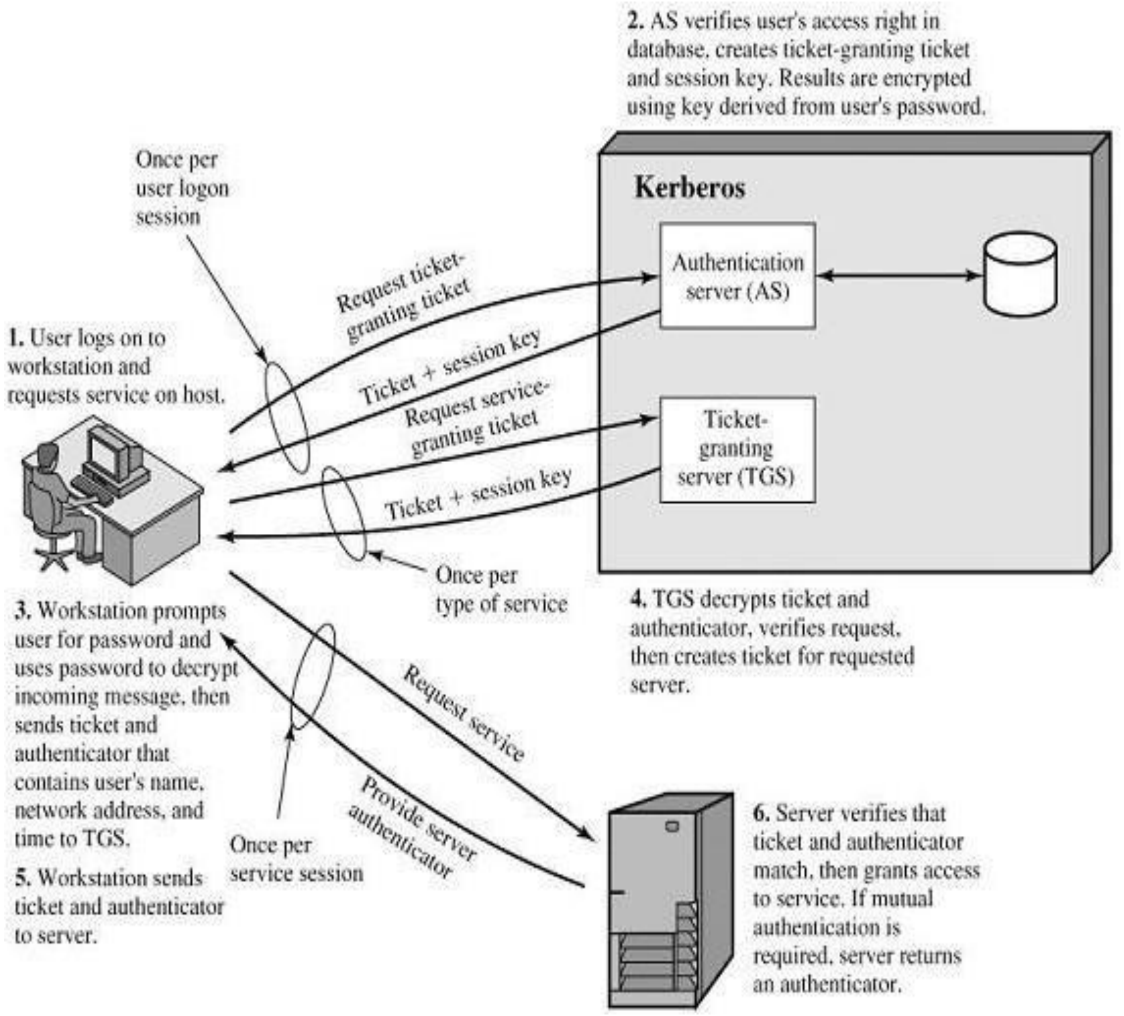

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication server (AS)
Ticket-granting server (TGS)
Once per user logon session
Request ticket-granting ticket
Ticket + session key
Request service-granting ticket
Ticket + session key
1. User logs on to workstation and requests service on host.
Once per type of service
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
Request service
Provide server authenticator
Once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Fig 4.1 Overview of Kerberos 4

Kerberos Realms and Multiple Kerberi

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

4. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
5. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm**

The concept of *realm* can be explained as follows.

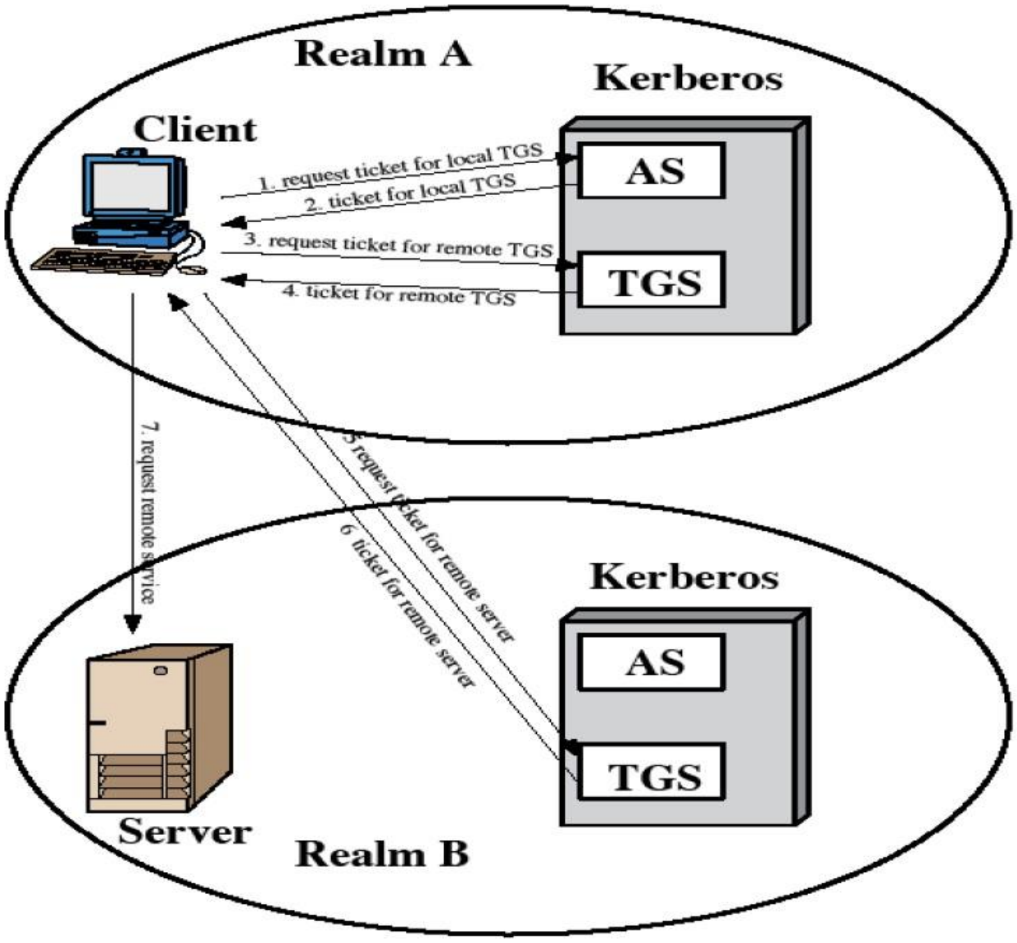

Realm A
Kerberos
Client
AS
TGS
1. request ticket for local TGS
2. ticket for local TGS
3. request ticket for remote TGS
4. ticket for remote TGS
7. request remote service
5 request ticket for remote server
6 ticket for remote server
Kerberos
AS
TGS
Server
Realm B

Fig .Request for service in another Realm

A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos computer systems.

However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a Kerberos principal, which is a service or user that is known to the Kerberos system.

Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name. Networks of clients and servers under different administrative organizations typically constitute different realms.

That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere.

However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such inter realm authentication. For two realms to support inter realm authentication, a third requirement is added:

6. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

The details of the exchanges illustrated in Fig 2are as follows:

$C \rightarrow AS$:ID_C II ID_{tgs} II TS_1

$AS \rightarrow C$:$EK_c[K_{c,tgs}$ ii ID_{tgs} II TS_2 II $Lifetime_2$ II $Ticket_{tgs}$

$C \rightarrow TGS$:ID_{tgsrem} II $Ticket_{tgs}$ II $Authenticator_c$

$TGS \rightarrow C$:$E\ K_{c,tgs[}K_{c,tgsrem}$ II ID_{tgsrem} II TS_4 II $Ticket_{tgsrem}$

$C \rightarrow TGS_{rem}$:ID_{vrem} II $Ticket_{tgsrem}$ II $Authenticator_c$

$TGS_{rem} \rightarrow C$:$EK_{c,tgsrem}$ [$K_{c,vrem}$ II ID_{vrem} II TS_6 II $Ticket_{vrem}$:

$C \rightarrow V_{rem}$:$Ticket_{vrem}$ $_{II}$$Authenticator_c$

Differences between Versions 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies.

Environmental shortcomings:

7. **Encryption system dependence:**
   Version 4 requires the use of DES. In version 5, ciphertext is tagged with an encryption type identifier so that any encryption technique may be used.

8. **Internet protocol dependence:**
   Version 4 requires the use of Internet Protocol (IP) addresses. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

9. **Message byte ordering:**
   In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

10. **Ticket lifetime:**
    Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

11. **Authentication forwarding:**
    Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. Version 5 provides this capability.

Technical deficiencies in the version 4 protocol:

- Double encryption
- PCBC encryption
- Session keys
- Password attacks

The Version 5 Authentication Dialogue

(a) Authentication Service Exchange: to obtain ticket-granting ticket
(1) $C \rightarrow AS$: Options II ID_c II $Realm_c$ II Times II $Nonce_1$
(2) $AS \rightarrow C$: $Realm_c$ II ID_c II $Ticket_{tgs}$ II EK_c [$K_{c,tgs}$ II Times II $Nonce_1$ II $Realm_{tgs}$ II ID_{tgs}]
$Ticket_{tgs} = EK_{tgs}$ [Flags II $K_{c,tgs}$ II $Realm_c$ II ID_c II AD_c II Times]
(b) Ticket – Granting Service Exchange:to obtain service-granting ticket
(3) $C \rightarrow TGS$: Optionns II ID_v II Times II $Nonce_1$
(4) $TGS \rightarrow C$: $Realm_c$ II ID_c II $Ticket_v$ II $EK_{c,tgs}$[$K_{c,v}$ II Times II $Nonce_2$ II $Realm_v$ II ID_v]
$Ticket_{tgs} = EK_{tgs}$[Flags II $K_{c,tgs}$ II $Realm_c$ II ID_c II AD_c II Times]
$Ticket_v = Ek_v$[[Flags II $K_{c,v}$ II $Realm_c$ II ID_c II AD_c II Times]

Authenticator$_c$ = E$K_{c,tgs}$[ID$_c$ II Realm$_c$ II TS$_1$]

(c) Client/Server AUTHENTICATION Exchange: to obtain service

(5) C → V : Options II Ticket$_v$ II Authenticator$_c$
(6) V → C : E$K_{c,v}$ [TS$_2$ II subkey II Seq #]
Ticketv = EK_v[Flags II $K_{c,v}$ II Realm$_c$ II ID$_c$ II AD$_c$ II Times]
Authenticator$_c$ = $E_{Kc,v}$[ID$_c$ II Realm$_c$ II TS$_2$ II Subkey II Seq#]

First, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. It includes the ID of the user and the TGS.

The following new elements are added:

- Realm: Indicates realm of user
- Options: Used to request that certain flags be set in the returned ticket
- Times: Used by the client to request the following time settings in the ticket:
    - from : the desired start time for the requested ticket
    - till : the requested expiration time for the requested ticket
    - r_{time} : requested renew-till time

Nonce: A random value to be repeated in message (2) to assure that the response is fresh and has not been replaced by an opponent .

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, times specified in message (1), the nonce from message (1), and TGS identifying information.

The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options.

Let us now compare the ticket-granting service exchange for versions 4 and 5.

We see that message (3) for both versions include 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.

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. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

- **Subkey**: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket (Kc,v) is used.

- **Sequence number**: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys.

X.509 AUTHENTICATION SERVICES

X.509 defines a framework for authentication services by the X.500 directory to its users. The directory consists of public-key certificates.

Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority.

X.509 defines authentication protocols based on public key certificates. X.509 standard certificate format used in S/MIME, IP Security and SSL/TLS and SET.

Certificates

The certificates are created and stored in the directory by the trusted Certification Authority (CA). The directory server not having certification functions and not create public key. But the user obtains the certificate from some easily accessible location

The general format of the certificate as shown below Fig 4.3

The elements of the certificates are

1. **Version(V):** The default version is 1. The issuer and subject unique identifier are present in version 2. If one or more extensions are present in version 3.
2. **Serial Number (SN):** Unique integer value issued by CA
3. **Signature Algorithm Identifier (AI):** This algorithm is used to sign the certificate with some parameters
4. **Issuer Name (CA):** The name of the CA that created and signed this certificate
5. **Period of validity (T_A):** The first and last on which the certificate is valid
6. **Subject Name (A):** The name of the user to whom this certificate refers
7. **Subject's Public Key Information (AP):** The public key of the subject plus identifier of the algorithm for which this key is to be used, with associated parameters.

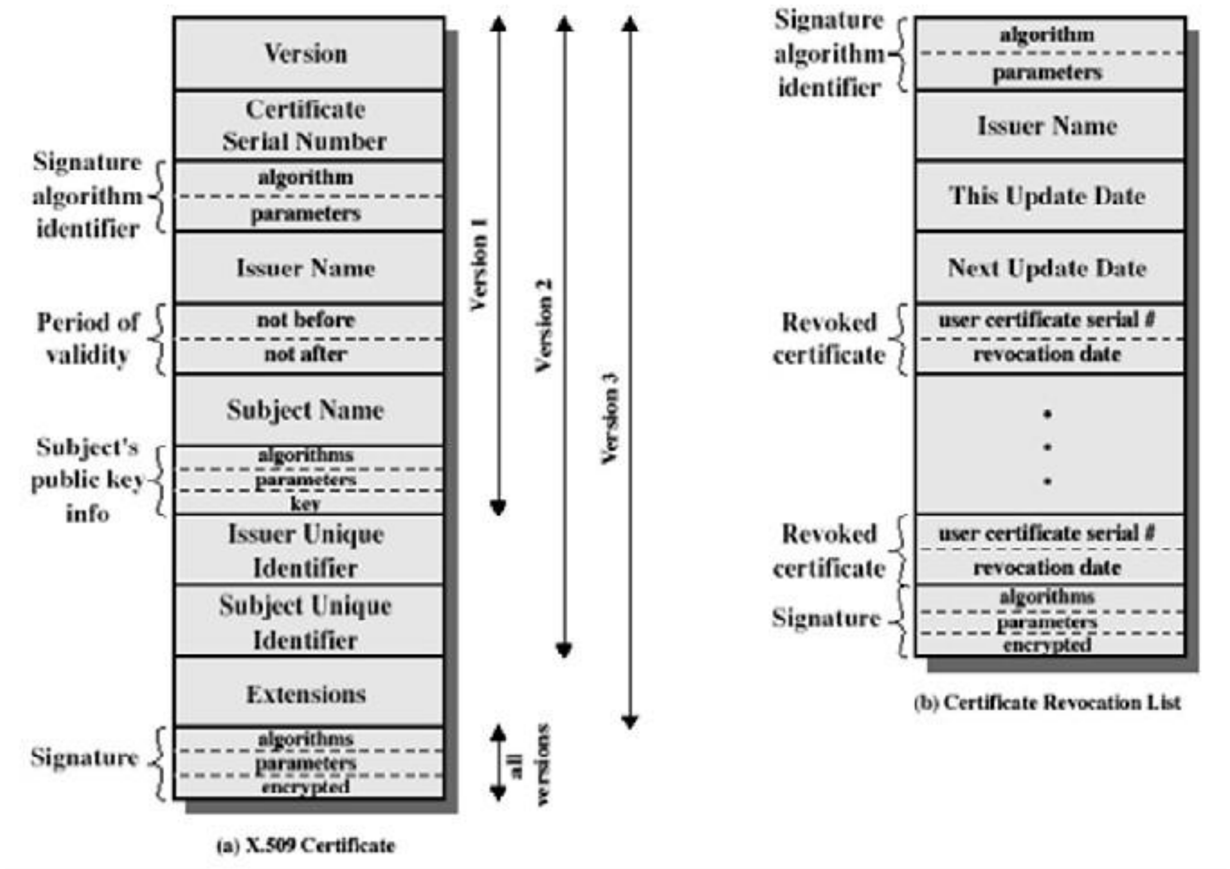


Fig X.509 AUTHENTICATION SERVICES

Issuer Unique Identifier: It is used to identify uniquely the issuing CA

8. **Subject Unique Identifier:** It is used to identify uniquely the subject
9. **Extensions:** A set of one or more extension fields
10. **Signature:** Covers all of the other fields of certificate; it contains hash code of other fields, encrypted with the CA‟s private key.

[**Note:** Unique identifier is used to avoid reuse of subject and issuer names over time]

Notation to define a certificate

CA<<A>> = CA {V, SN, AI, CA, TA, 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.

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

Generation and usage of certificate by a user

The user certificates generated by CA have the following characteristics:

11. Any user with access to the public key of the CA can verify the user public key that was certified.
12. No party other than the Certification Authority (CA) can modify the certificate without this being detected.
13. Certificates are unforgeable,

If all users belong to the same CA, the certificates can be placed in the directory for access by all users. If the number of users increased, single CA could not be satisfied all user requirements.

For example, User A has obtained the certificate from certificate authority x_1 and user B from x_2. Now the two CAs (x_1 and x2) securities exchange their own public keys in the form of certificates. That is x_1 must hold x_2‟s certificate and x_2 holds x_1‟s certificate

Now A want s to access B‟s public key, it follows the following chain to obtain B‟s public key.

x_1<< x_2>> x_2<<B>>

i.e., first A gets x_2‟s certificate from x_1‟s directory to obtain x_2‟s public key. Then using x_2‟s public key to obtain B‟s certificate from x_2‟s directory to obtain„s public key.

In the same method, B can obtain A‟s public key with the reverse chain

x_2<< x_1>> x_1<<A>>

Hierarchy of CAs

To obtain public key certificate of user efficiently, more than one CAs can be arranged in a hierarchy, so that navigation in easy.

The connected circles indicate the hierarchical(Fig 4.4) relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. 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

User A can acquire the following certificates from the directory to establish a certification path to B:

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

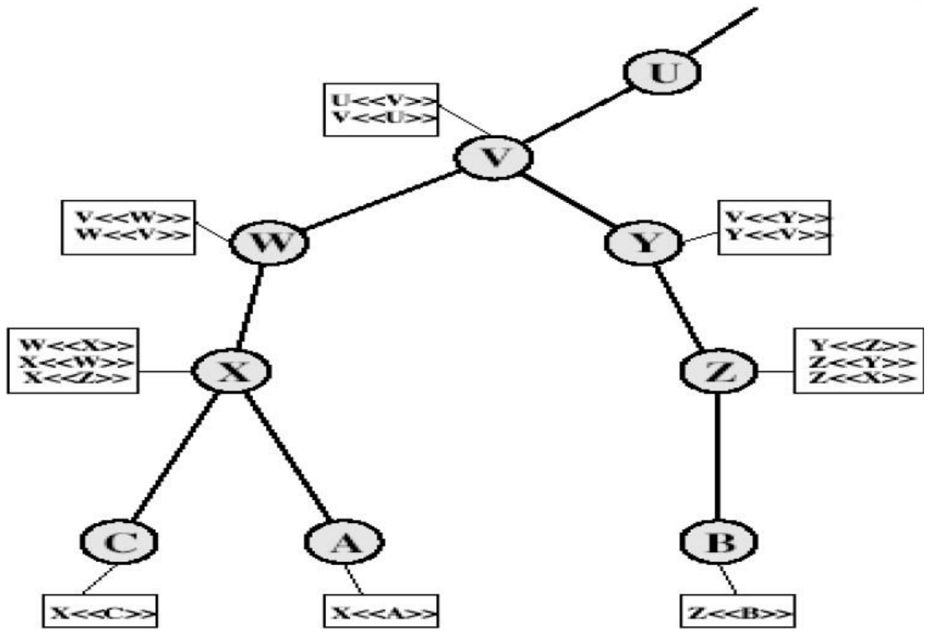


Fig4.4 : Hierarchy of X.509

Revocation of certificates

- Certificates have a period of validity
- May need to revoke before expiry, for the following reasons eg:
  - ✓ User's private key is compromised
  - ✓ User is no longer certified by this CA
  - ✓ CA's certificate is compromised
- CA‟s maintain list of revoked certificates
  - ✓ The Certificate Revocation List (CRL)
- Users should check Certificates with CA‟s CRL

Authentication Procedures

X.509 includes three alternative authentication procedures:

- One-Way Authentication
- Two-Way Authentication
- Three-Way Authentication

One-Way Authentication

One 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

Message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

Two 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

Reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

Three messages (A→B, B→A, A→B) which enables above authentication without synchronized clocks (Fig 4.5)

- Has reply from A back to B containing signed copy of nonce from B
- Means that timestamps need not be checked or relied upon

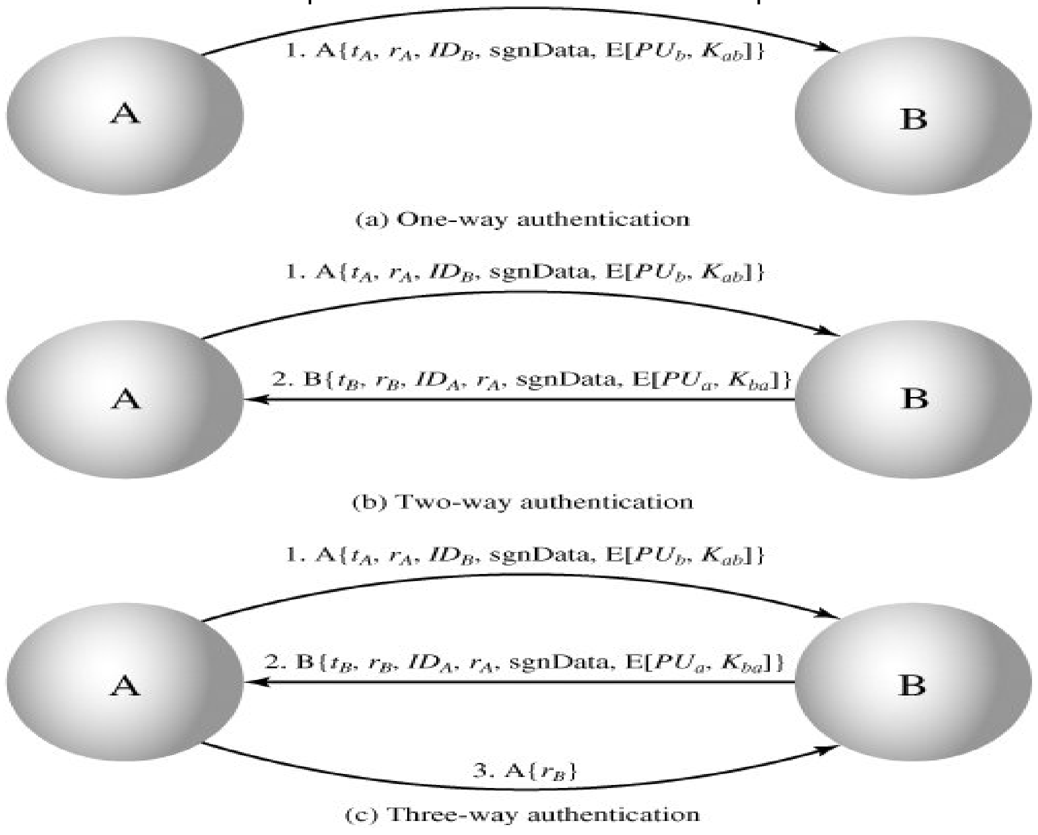

1. A{t_A, r_A, ID_B, sgnData, E[PU_b, K_{ab}]}
A
B
(a) One-way authentication
1. A{t_A, r_A, ID_B, sgnData, E[PU_b, K_{ab}]}
2. B{t_B, r_B, ID_A, r_A, sgnData, E[PU_a, K_{ba}]}
A
B
(b) Two-way authentication
1. A{t_A, r_A, ID_B, sgnData, E[PU_b, K_{ab}]}
2. B{t_B, r_B, ID_A, r_A, sgnData, E[PU_a, K_{ba}]}
3. A{r_B}
A
B
(c) Three-way authentication

Fig: X509 Strong Authentication Procedure

X.509 Version 3

The following requirements not satisfied by version 2:

14. The Subject field is inadequate to convey the identity of a key owner to a public-key user.
15. 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.
16. There is a need to indicate security policy information. 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.
17. It is important to be able to identify different keys used by the same owner at different times.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

Authority key identifier: Identifies the public key to be used to verify the signature on this certificate or CRL.
Subject key identifier: Identifies the public key being certified.
Key usage: Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.
Private-key usage period: Indicates the period of use of the private key corresponding to the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
Certificate policies: Certificates may be used in environments where multiple policies apply.
Policy mappings: Used only in certificates for CAs issued by other CAs.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include the following:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms

- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The extension fields in this area include the following:

- **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
- **Name constraints**: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
- **Policy constraints**: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

14.3. Public-Key Infrastructure

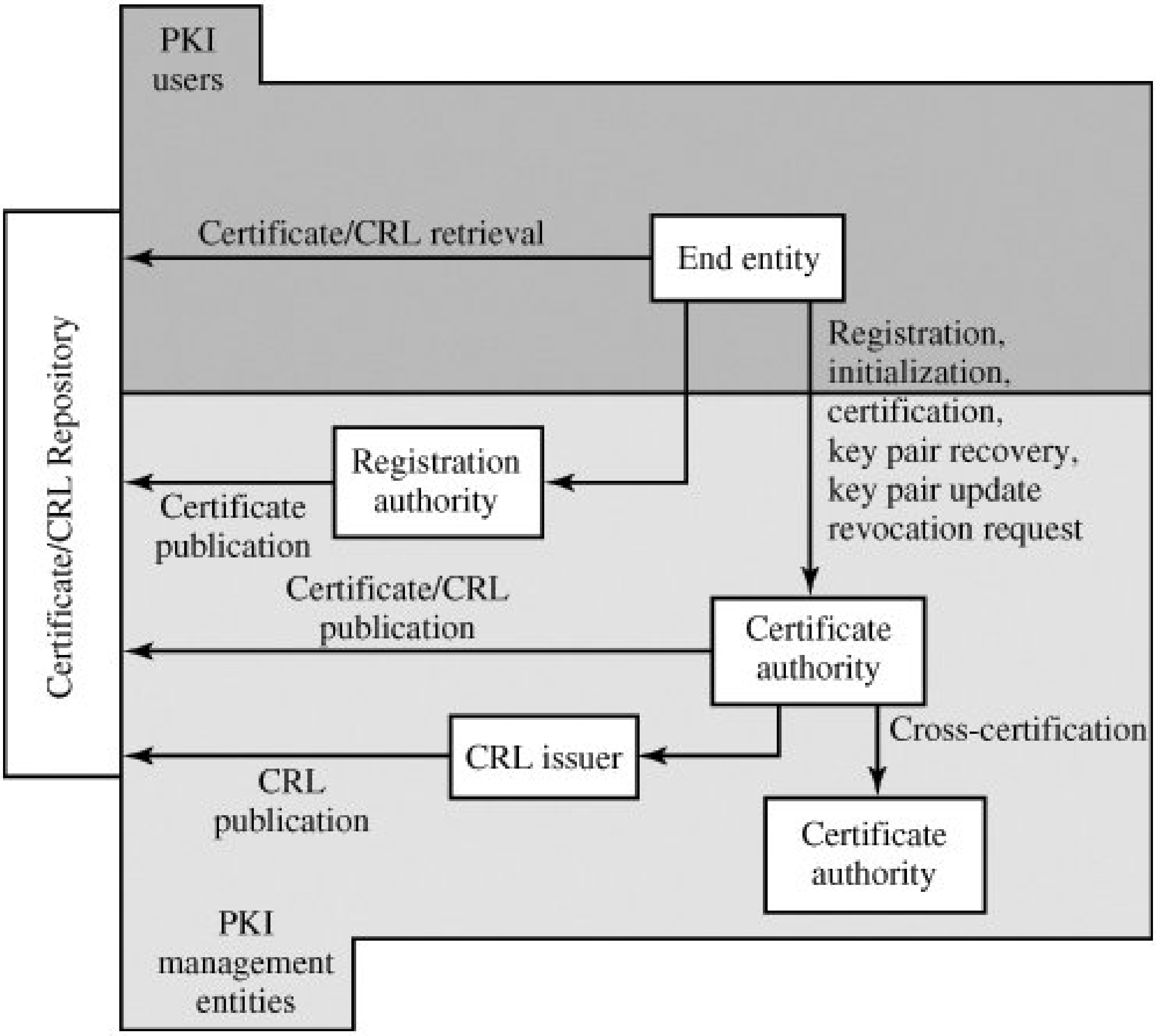
RFC 2822 (*Internet Security Glossary*) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

Figure 14.7 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. End entities typically consume and/or support PKI-related services.

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

Figure 14.7. PKIX Architectural Model


PKI users
Certificate/CRL retrieval
End entity
Registration, initialization, certification, key pair recovery, key pair update revocation request
Certificate/CRL Repository
Registration authority
Certificate publication
Certificate/CRL publication
Certificate authority
Cross-certification
CRL issuer
CRL publication
Certificate authority
PKI management entities

PKIX Management Functions

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 14.7 and include the following:

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

- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key, and returns that certificate to the user's client system and/or posts that certificate in a repository.
- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is

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

- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

PKIX Management Protocols

The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed in the preceding subsection. 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, 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.

PREV NEXT