Chapter 4

Authentication Applications

Outline

- Security Concerns
- Kerberos
- X.509 Authentication Service

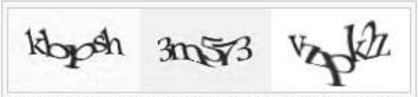
Security Concerns

- key concerns are **confidentiality** and **timeliness**
- to provide confidentiality must encrypt identification and session key info
- which requires the use of previously shared private or public keys
- need timeliness to prevent **replay attacks**
- provided by using sequence numbers or timestamps or challenge/response
  - A CAPTCHA (Completely Automated Public Turing Test to tell Computers and Humans Apart)

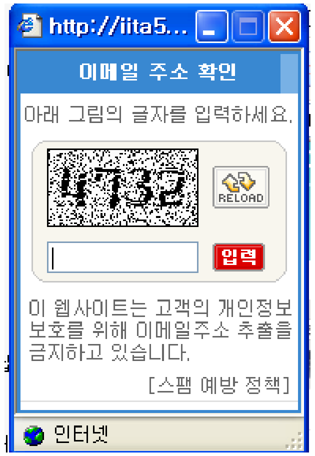
Early CAPTCHAs such as these, generated by the EZ-Gimpy program, were used on Yahoo. However, technology was developed to read this type of CAPTCHA [1].

following
finding

A modern CAPTCHA, rather than attempting to create a distorted background and high levels of warping on the text, might focus on making segmentation difficult by adding an angled line.



Another way to make segmentation difficult is to crowd symbols together. This can be read by humans but can't be segmented by bots



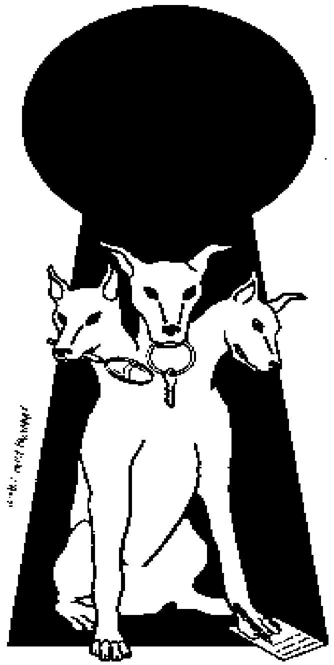
이메일 주소 확인

아래 그림의 글자를 입력하세요.

이 웹사이트는 고객의 개인정보 보호를 위해 이메일주소 추출을 금지하고 있습니다.

[스팸 예방 정책]

KERBEROS



In Greek mythology, a many headed dog, the guardian of the entrance of Hades

KERBEROS

- An authentication service developed as part of project Athena in MIT
- Kerberos assumes :
  - An open distributed environment for users at workstations to access services on servers distributed through the network.
  - No trust on the identification of users by workstations (WS)
- Kerberos wants servers to be able to :
  - restrict access to authorized users
  - authenticate requests for service

KERBEROS

- Users wish to access services on servers.
- Three threats exist:
  - An opponent pretends to be another user operating on the workstation.
  - An opponent alters the network address of a workstation.
  - An opponent eavesdrops on exchanges and uses a replay attack.

KERBEROS

- Provides a centralized authentication server to authenticate users to servers and servers to users.
- Relies on conventional encryption, making no use of public-key encryption
- Two versions: version 4 and 5
- Version 4 makes use of DES
- Version 5
  - Encryption: AES128-CTS-HMAC-SHA1-96, DES-CBC-MD5, DES3-CBC-SHA1-KD
  - Checksums: DES-MD5, HMAC-SHA1-DES3-KD, HMAC-SHA1-96-AES128

Motivation of Kerberos

- Today, more common is a distributed architecture
  - consisting of dedicated user WSs (clients) and distributed or centralized servers.
- Three approaches envisioned for security
  1. Each WS assures the identity of its user and each server enforces a security policy based on user ID
  2. Client systems authenticate themselves to servers, but servers trust Client systems concerning the identity of its user
  3. The Client proves user’s identity for each service invoked and the servers prove its identity to the clients
- Kerberos supports this third approach.

Motivation of Kerberos

- The requirements of Kerberos :
  - Secure : A network eavesdropper can't obtain the necessary info. to impersonate a user
  - Reliable : a distributed server architecture should be employed with one system to back up another
  - Transparent : users don't know the authentication process beyond entering a password.
  - Scalable : The system should support large number of clients and severs (i.e. a modular, distributed architecture)
- The overall scheme of Kerberos is a trusted third-party authentication service

Kerberos Version 4

- Terms:
  - C = Client
  - AS = authentication server
  - V = server
  - ID_c = identifier of user on C
  - ID_v = identifier of V
  - P_c = password of user on C
  - AD_c = network address of C
  - K_v = secret encryption key shared by AS an V
  - TS = timestamp
  - $||$ = concatenation

A Simple Authentication Dialogue

(1) $C \rightarrow AS$: $ID_c \| P_c \| ID_v$

(2) $AS \rightarrow C$: Ticket

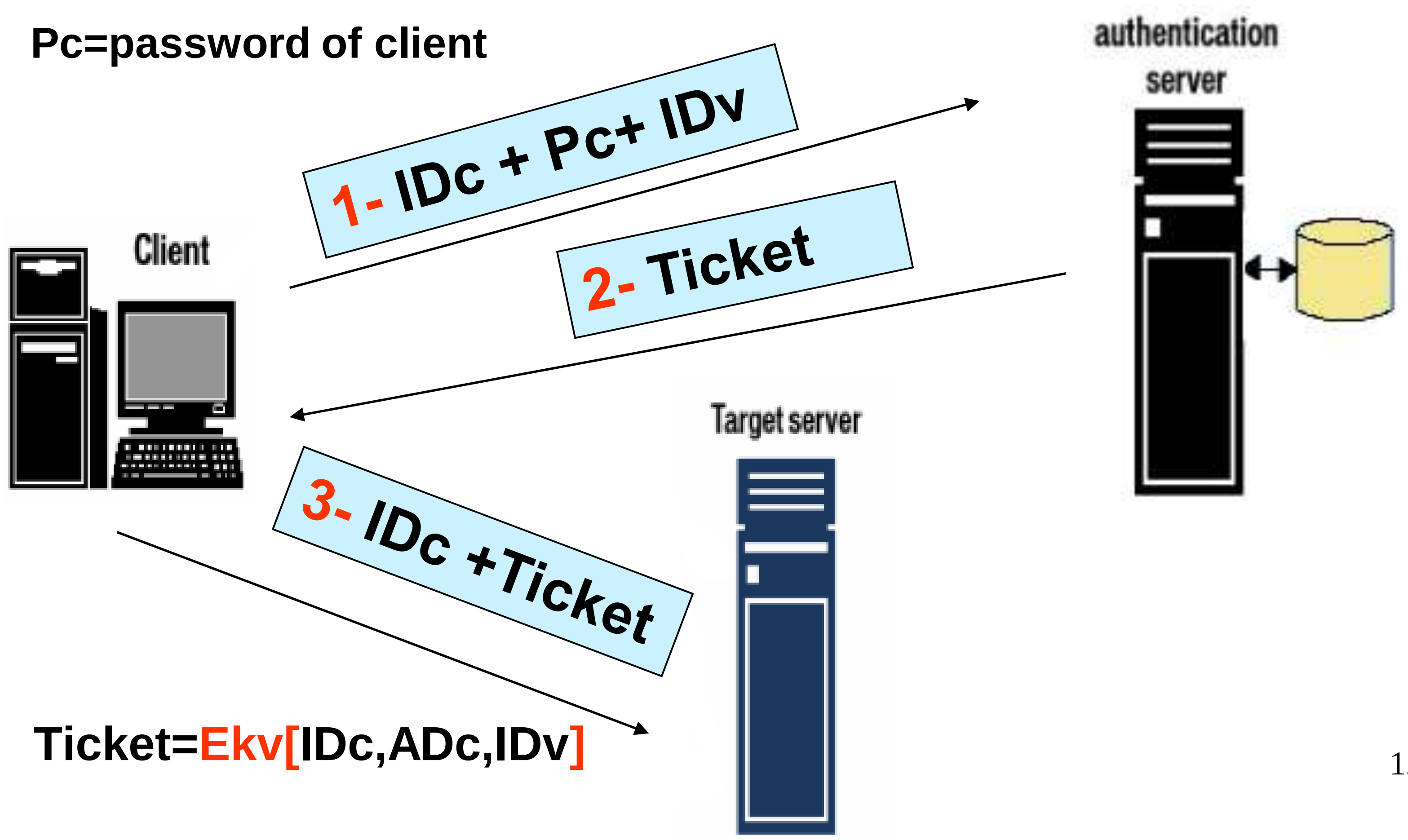
(3) $C \rightarrow V$: $ID_c \|$ Ticket

Ticket = $\mathbf{E}_{\mathbf{Kv}}[ID_c \| \mathbf{AD}_c \| ID_v]$

- The user logs on to a WS and requests access to server V
- The client module C requests user's password
- Then C sends message(1) to AS
- AS send a ticket to convince V of the user's authenticity

A Simple Authentication Dialogue

Pc=password of client



Ticket=Ekv[IDc,ADc,IDv]

A Simple Authentication Dialogue

- Frequent requests to enter user’s password
  - Suppose each ticket can be used only once

    -> A user enters a password to get a ticket each time the user wants access to V
  - Suppose the tickets are reusable to improve the matters

    -> A user needs a new ticket for every different service and hence be required to enter a password
- A plaintext transmission of password in message(1)
  - An opponent could capture the password and use any service accessible to the victim

A More Secure Dialogue

- **This introduces a scheme for avoiding plaintext passwords and a new server, ticket-granting server (TGS)**
- **TGS issues tickets for services to users who have been authenticated to AS**
  - Thus, the user first requests a ticket-granting ticket (TGT) to AS
  - TGT is saved in the client module of WS and used to authenticate the user itself to TGS for each access to a new service
  - The service-granting ticket (SGT) issued by TGS is saved and used to authenticate its user to a server for a particular service

A More Secure Dialogue

Once per user logon session:

(1) $C \rightarrow AS$: $ID_C \| ID_{tgs}$

(2) $AS \rightarrow C$: $E(K_C, Ticket_{tgs})$

Once per a type of service:

(3) $C \rightarrow TGS$: $ID_C \| ID_V \| Ticket_{tgs}$

(4) $TGS \rightarrow C$: $Ticket_v$

Once per a service session:

(5) $C \rightarrow V$: $ID_C \| Ticket_v$

$Ticket_{tgs} = E(K_{tgs}, [Idc \| Adc \| Id_{tgs} \| TS_1 \| Lifetime_1])$

$Ticket_v = E(K_v, [Idc \| Adc \| Idv \| TS_2 \| Lifetime_2])$

A More Secure Dialogue

Once per user logon session:

(1) $C \rightarrow AS$: $ID_C \| ID_{tgs}$

(2) $AS \rightarrow C$: $E(K_C, Ticket_{tgs})$

$Ticket_{tgs} = E(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$

- **The client requests a TGT by sending msg(1) to AS**
- **The AS responds with a ticket encrypted with a key derived from user's password**
- **The client prompts the user to enter a password when receiving the response from the AS and generates a key**
- **If the correct password is supplied, the ticket is successfully recovered**

A More Secure Dialogue

Once per type of service:

(3) C → TGS: ID_C ll ID_V ll $Ticket_{tgs}$

(4) TGS → C: $Ticket_v$

$Ticket_v$ = E (K_v, [IDc ll ADc ll IDv ll TS_2 ll $Lifetime_2$])

- **The client requests a service-granting ticket (SGT) for the user with a message(3) including the TGT**
- **The TGS issues a SGT when the user has been authenticated by the content of the TGT**
- **The SGT has the same structure as the TGT because both authenticate clients**

A More Secure Dialogue

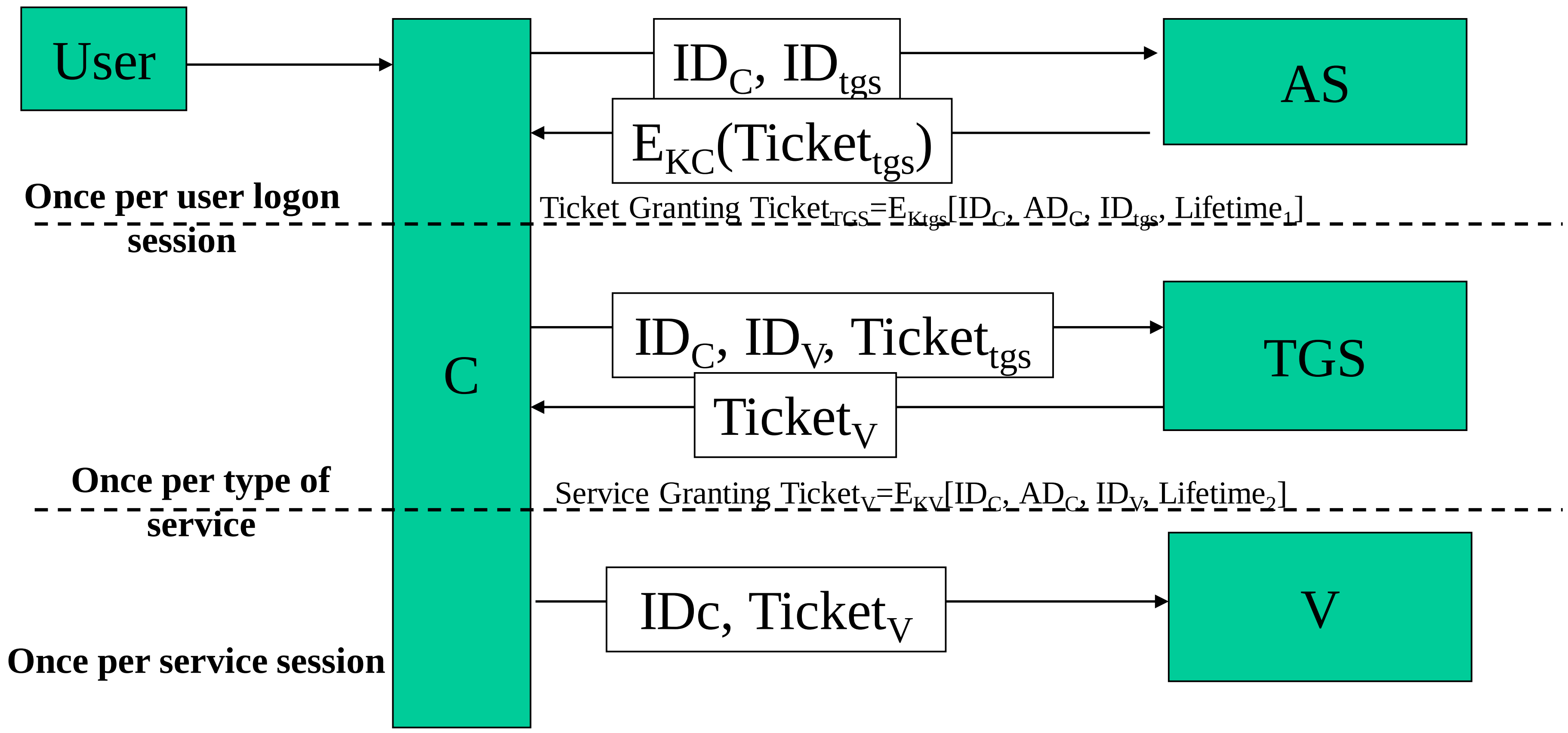
Once per type of service:

(5) $C \rightarrow V$: ID_C ll $Ticket_v$

$Ticket_v = E(K_v, [Idc \text{ ll } Adc \text{ ll } Idv \text{ ll } TS_2 \text{ ll } Lifetime_2])$

- **The client requests access to a server for the user with message(5)**
- **The server authenticates by using the contents of the SGT**
- **The scenario satisfies the two requirements:**
  - Only one password query
  - No transmission of the user password in plaintext

Authentication Dialogue - Summary


User
C
AS
TGS
V
ID_C, ID_{tgs}
$E_{KC}(Ticket_{tgs})$
Once per user logon session
Ticket Granting $Ticket_{TGS}=E_{Ktgs}[ID_C, AD_C, ID_{tgs}, Lifetime_1]$
ID_C, ID_V, $Ticket_{tgs}$
$Ticket_V$
Once per type of service
Service Granting $Ticket_V=E_{KV}[ID_C, AD_C, ID_V, Lifetime_2]$
IDc, $Ticket_V$
Once per service session

TGS : Ticket Granting Server

A More Secure Dialogue

- **Two additional problems:**

1. **The lifetime associated with the TGT**
   - **Too short → frequent prompts for entering the password**
   - **Too long → replay attack after capturing the ticket**

     (similar with the SGT)
   - **TGS or AS must prove that the person using the ticket is the same person to whom that ticket was issued.**
2. **The requirement for servers to authenticate themselves to users.**
   - **The impersonated server could deny the true service to the user**

Version 4 Authentication Dialogue

(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

Summary of Kerberos Version 4 Message Exchanges

Version 4 Authentication Dialogue

(1) $C \rightarrow AS$ $ID_c \| ID_{tgs} \| TS_1$

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

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

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

The client requests a TGT to AS with message(1)

To handle the problem of captured TGT and the genuiness of ticket presenter,

- the AS provides both the TGS and the client with a secret information, called a session key, in a secure manner through message(2)
- then the key is used to prove the identity of the client to TGS

Version 4 Authentication Dialogue

(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 SGT

- C transmits an authenticator (A) used only once with very short lifetime in message(3)
  - Replay attack is encountered.
- The TGS decrypts the A and the ticket with keys,
  - The contents from the both are checked if those match
  - The ticket is a way to distribute keys securely
  - The A proves the client's identity.
- Reply from TGS includes a session key shared b/w C and V.
  - It says that the key can be used by only C and V for authentication.

Version 4 Authentication Dialogue

(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

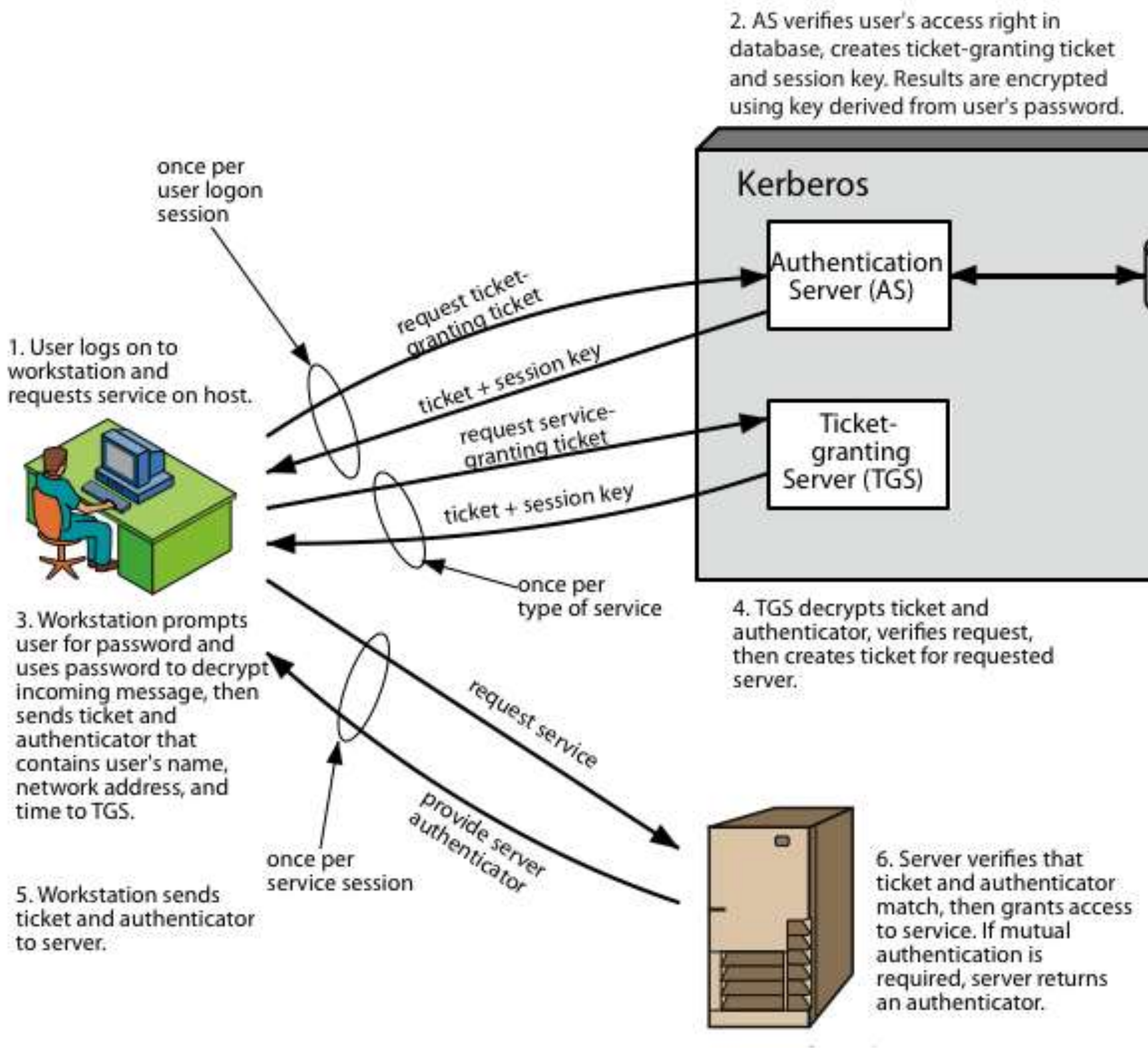
The message(5) is similar to message(3)

- V examines the contents of A and the ticket if the ticket presenter is genuine

The mutual authentication is done with message(6)

- The value of timestamp from the A is incremented by 1 and encrypted by the session key.
- The contents of the message assures C that this is not a replay
- The session key is used to encrypt future messages b/w the two or to exchange a new random session key for that purpose

Version 4 Authentication Dialogue



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
once per type of service
1. User logs on to workstation and requests service on host.
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.

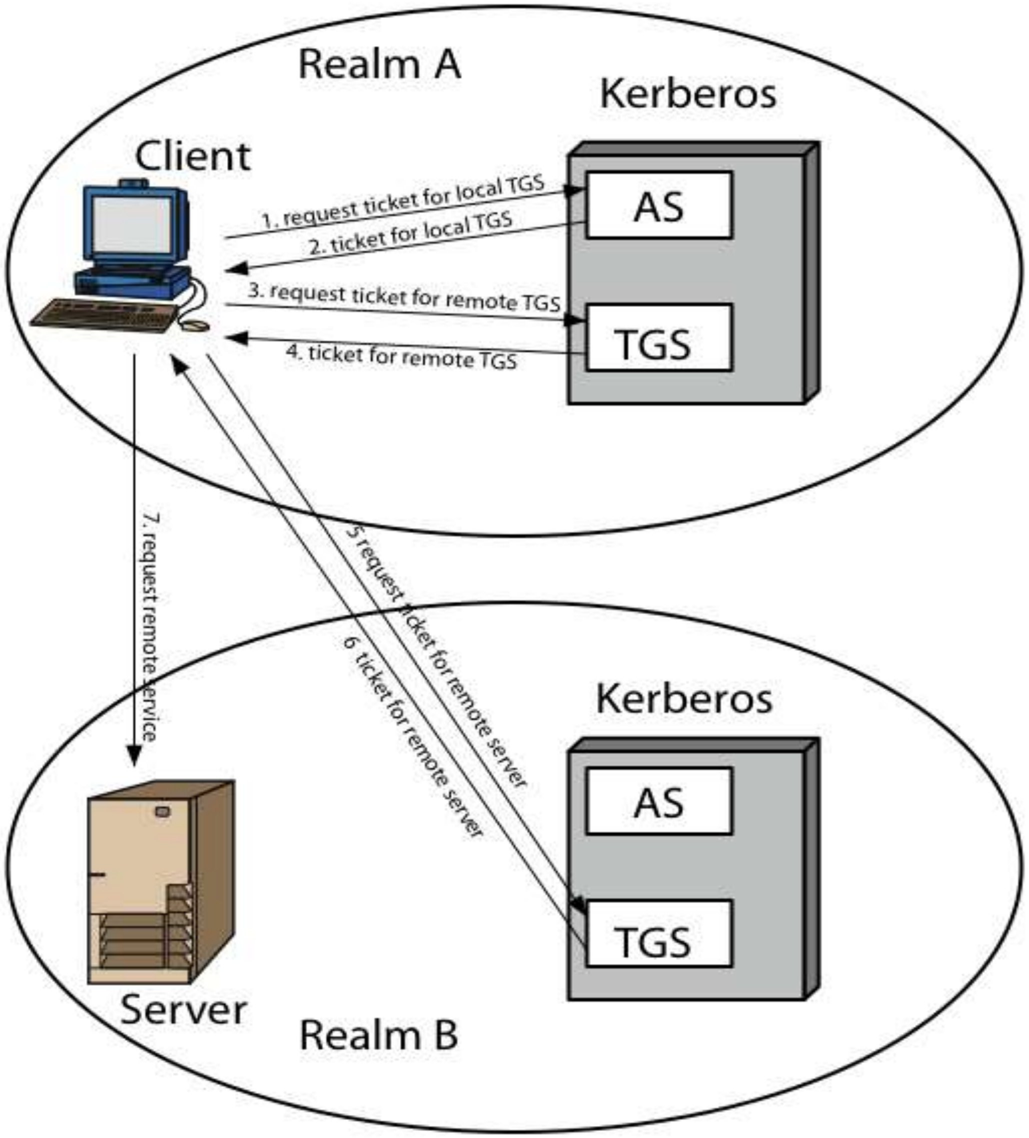
Kerberos Realm

- A Kerberos realms is a set of managed nodes that share the same Kerberos DB.
- A Kerberos realm consists of :
  - A Kerberos server, with all user IDs and their passwords in its DB
  - A number of clients, registered with the Kerberos server
  - A number of application servers, sharing a key and registered with the Kerberos server
- Networks of clients and servers under different administrative organizations constitute typically different realms.

Kerberos Realms with Multiple Kerberi

- For two realms to support interrealm auth,
  - The Kerberos server in one realm shares a secret key with the sever in the other realm. The two Kerberos servers are registered with each other
  - The participating servers in the second realm must trust the Kerberos server in the first realm
- One problem with above approach :
  - It does not scale well to many realms
  - It requires N(N-1)/2 secure key exchanges for interoperation of all realms

Request for Service in Another Realm


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

Interrealm Authentication Message Exchanges

(1) $C \rightarrow AS$: $ID_c \| ID_{tgs} \| TS_1$

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

(3) $C \rightarrow TGS$: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$

(4) $TGS \rightarrow C$: $E(K_{c,tgs} [K_{c,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$

(5) $C \rightarrow TGS_{rem}$: $ID_{vrem} \| Ticket_{tgsrem} \| Authenticator_c$

(6) $TGS_{rem} \rightarrow C$: $E(K_{c,tgsrem}, [K_{c,vrem} \| ID_{vrem} \| TS_6 \| Ticket_{vrem}])$

(7) $C \rightarrow V_{rem}$: $Ticket_{vrem} \| Authenticator_c$

Kerberos Version 5

- Version 5 is intended to address the limitations of Version 4 in two areas :
  - Environment shortcomings, due to development for use within the Project Athena environment, not for general purpose
  - Technical deficiencies in the version 4 protocol itself
- First, examine the differences b/w version 4 and 5
- Then, look at the version 5 protocol

Environment Shortcomings of Version 4

1. Encryption system dependence : the use of DES only on Version 4
   - Any encryption technique may be used
   - Encryption type identifier is tagged with ciphertext
   - Encryption Keys are tagged with type and a length to be used in different algorithms

2. Internet protocol dependence : the use of IP address only
   - Network address is tagged with type and length (e.g. ISO)

Environment Shortcomings of Version 4

3. Message byte ordering : least or most significant byte ordering chosen by the sender of a message
   - use of ASN(Abstract Syntax Notation).1 and BER (basic ending rules) for unambiguous ordering

4. Ticket lifetime : an 8-bit quantity in units of 5 min (max = $2^8 \times 5 = 1280$ min)
   - use of explicit start and end time for arbitrary lifetime

Environment Shortcomings of Version 4

5. Authentication forwarding : no forwarding

- Version 5 allows credentials issued to one client to be forward to some other host and used by some other client
- For example, a client issues a request to a print server that then accesses the client's file from file server, using the client's credentials for access

6. Interrealm authentication : N^2 Kerberos-to-Kerberos relationships

- Version 5 supports a method with fewer relationships

Technical Deficiencies of Version 4

1. Double encryption : the tickets encrypted twice in messages 2 & 4
   - No double encryption on tickets in Version 5

2. PCBC encryption : use of nonstandard PCBC mode of DES
   - its vulnerability has been demonstrated
   - PCBC was intended to provide and integrity check as part of encryption operation
   - Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode for encryption

PCBC Mode

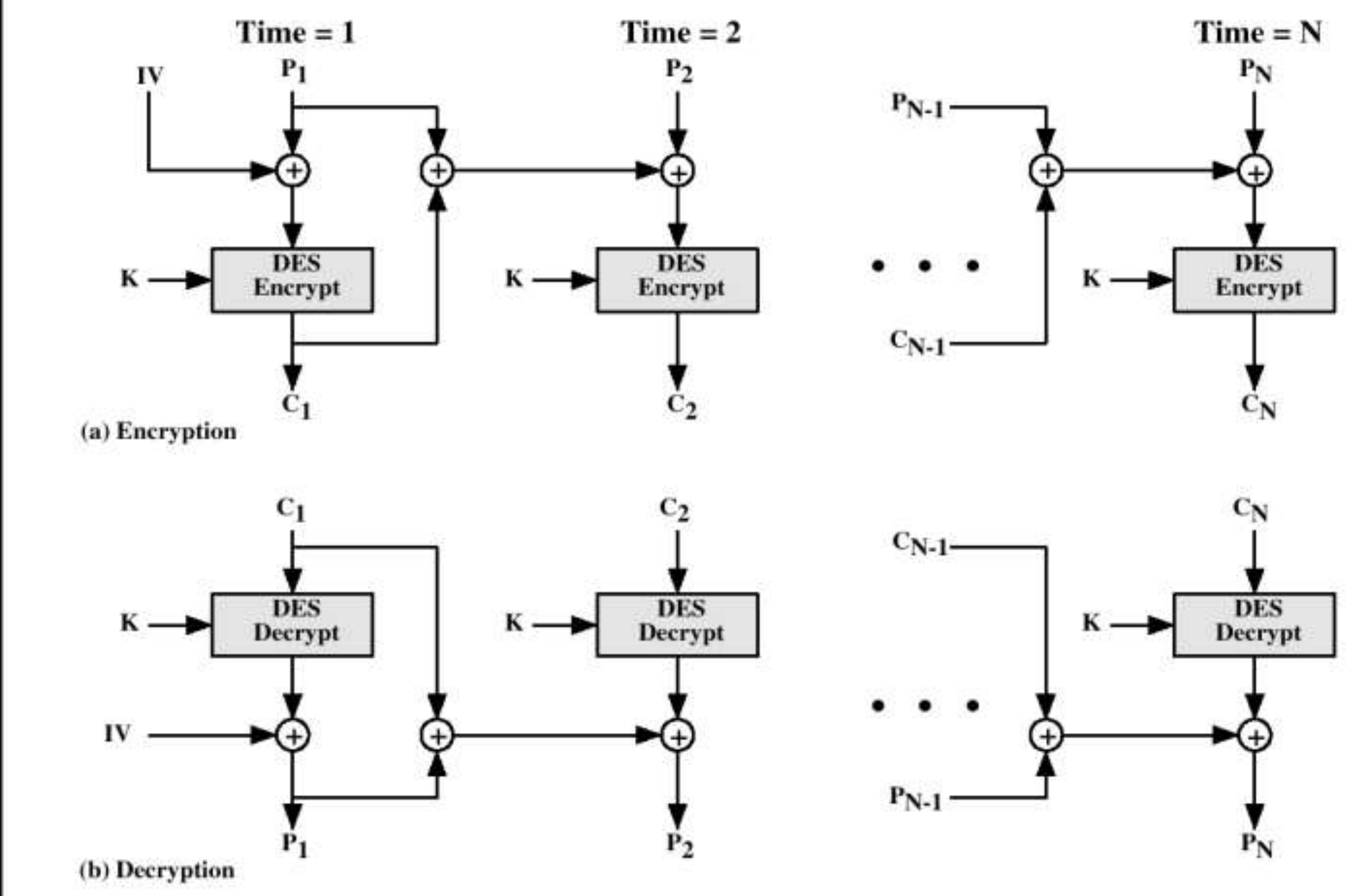


Figure 4.7 Propagating Cipher Block Chaining (PCBC) Mode

Used in Kerberos v4, cf : Fig.2.9

Technical Deficiencies of Version 4

3. Session keys : possibility of replay attack by repeated uses of the same ticket
   - A subsession key for C and V is allowed to be used only for that connection

4. Password attack : both versions are weak to this attack
   - The key is generated based on user’s password
   - The password is limited to characters in a 7-bit ASCII
   - An opponent attempts to decrypt a message by trying various passwords
   - Version 5 provides a mechanism “preauthentication “ to make the attack more difficult (but not preventing it)

Version 5 Authentication Dialogue

(1) $C \rightarrow AS$ Options ll ID_c || $Realm_c$ || ID_{tgs} ll Times ll $Nonce_1$

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

$Ticket_{tgs} = E(K_{tgs},[Flags$ ll $K_{c,tgs}$ || $Realm_c$ ll ID_c ll AD_c ll $Times])$

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

(3) $C \rightarrow TGS$ Options ll ID_v ll times ll $Nonce_2$ ll $Ticket_{tgs}$ ll $Authenticator_c$

(4) $TGS \rightarrow C$ $Realm_c$ ll ID_c ll $Ticket_v$ ll $E(K_{c,tgs}, [K_{c,v}$ ll Times ll $Nonce_2$ ll $Realm_v$ ll $ID_v])$

$Ticket_{tgs} = E(K_{tgs},[Flags$ ll $K_{c,tgs}$ ll $Realm_c$ ll ID_c ll AD_c ll $Times])$
$Ticket_v = E(K_v,[Flags$ ll $K_{c,v}$ ll $Realm_c$ ll ID_c ll AD_c ll $Times])$
$Authenticator_c = E(K_{c,tgs}\ [ID_c$ ll $Realm_c$ ll $TS_1])$

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

(5) $C \rightarrow V$ Options ll $Ticket_v$ ll $Authenticator_c$

(6) $V \rightarrow C$ $E_{Kc,v}$, [TS_2 ll Subkey ll Seq#]

$Ticket_v = E(K_v,[Flags$ ll $K_{c,v}$ ll $Realm_c$ ll ID_c ll AD_c ll $Times])$
$Authenticator_c = E(K_{c,v}, [ID_c$ ll $Realm_c$ ll TS_2 ll Subkey ll Seq#])

(c) Client/Server Authentication Exchange to obtain service

Summary of Kerberos Version 5 Message Exchanges

Version 5 Authentication Dialogue

(1) C $\rightarrow$ AS Options ll ID_c || $Realm_c$ || ID_{tgs} ll Times ll $Nonce_1$
(2) AS $\rightarrow$ C $Realm_c$ ll ID_c ll $Ticket_{tgs}$ $E(K_c,[K_{c,tgs}$ ll Times ll $Nonce_1$ ll $Realm_{tgs}$ ll $ID_{tgs}])$

$Ticket_{tgs} = E(K_{tgs},[$Flags ll $K_{c,tgs}$ || $Realm_c$ ll ID_c ll AD_cll Times])

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

- Message (1) is a client request for a TGT
  - Options : used to request for certain flags to be set in returned ticket
  - Nonce : is a random value to be repeated in msg(2) to counter replay attack
- Message (2) returns a TGT
  - Flags : reflect the status of this ticket and requested options with new functionality added to Version 5

Version 5 Authentication Dialogue

(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

- Message (3) is similar to both versions
  - Version 5 includes requested time, options for ticket and a nonce
- Message (4) has the same structure as message (2)

Version 5 Authentication Dialogue

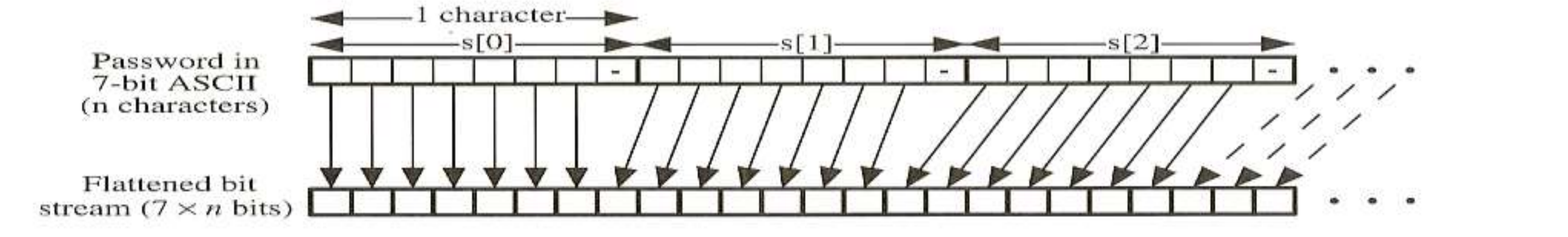
(5) C $\rightarrow$ V Options ll $Ticket_v$ ll $Authenticator_c$
(6) V $\rightarrow$ C $E_{K_{c,v}}$, [TS_2 ll Subkey ll Seq#]

$Ticket_v$ = E(K_v,[Flags ll $K_{c,v}$ ll $Realm_c$ ll ID_c ll AD_c ll Times])
$Authenticator_c$ = E($K_{c,v}$, [ID_c ll $Realm_c$ ll TS_2 ll Subkey ll Seq#])

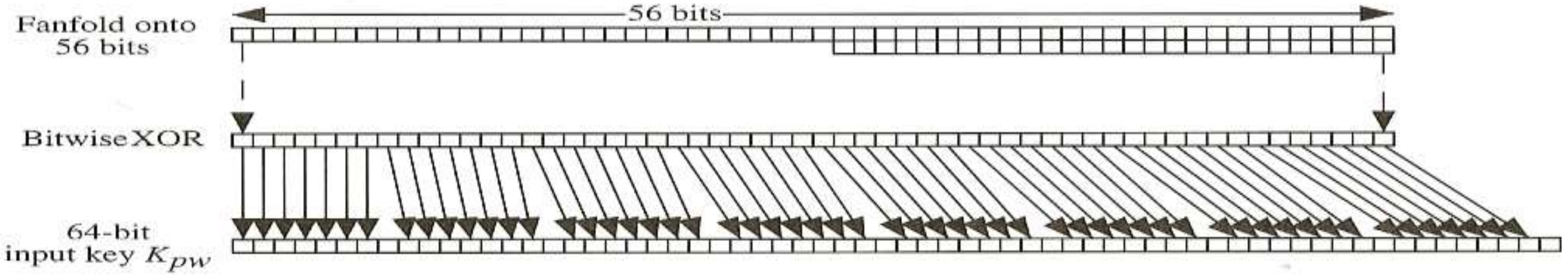
(c) Client/Server Authentication Exchange to obtain service

- Message(5) may include a request as an option for mutual authentication
  - Subkey : if omitted, the session key is used
  - Sequence number : may be included in message to detect replay
- Message (6) is sent by server when mutual authentication is required

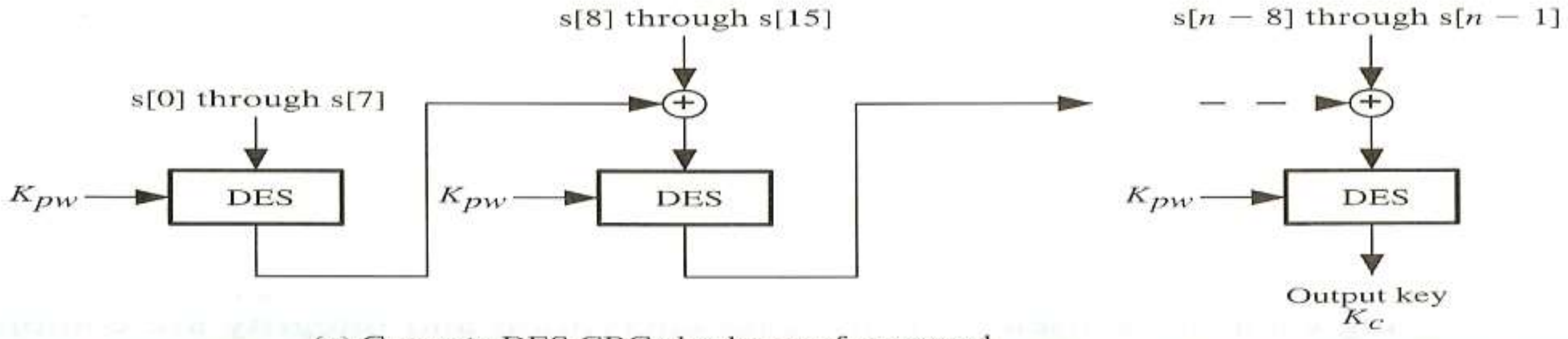
Password-to-key Transformation


1 character
s[0]
s[1]
s[2]
Password in 7-bit ASCII (n characters)
Flattened bit stream ($7 \times n$ bits)

(a) Convert password to bit stream



(b) Convert bit stream to input key



(c) Generate DES CBC checksum of password

X.509 Authentication Service

- ITU-T Rec. X.509 is part of X.500 series that define a directory service
- The directory :
  - is, in effect, a server or distributed set of servers to maintain a DB of information about users.
  - may serve as a repository of public-key certificates
- X.509 defines a framework for authentication services

  - using the certificate format and authentication

    protocols defined in X.509

X.509 Authentication Service

- X.509 authentication protocols :

  - use the directory service provided by X.500

  - use certificates, PKC, and digital signature

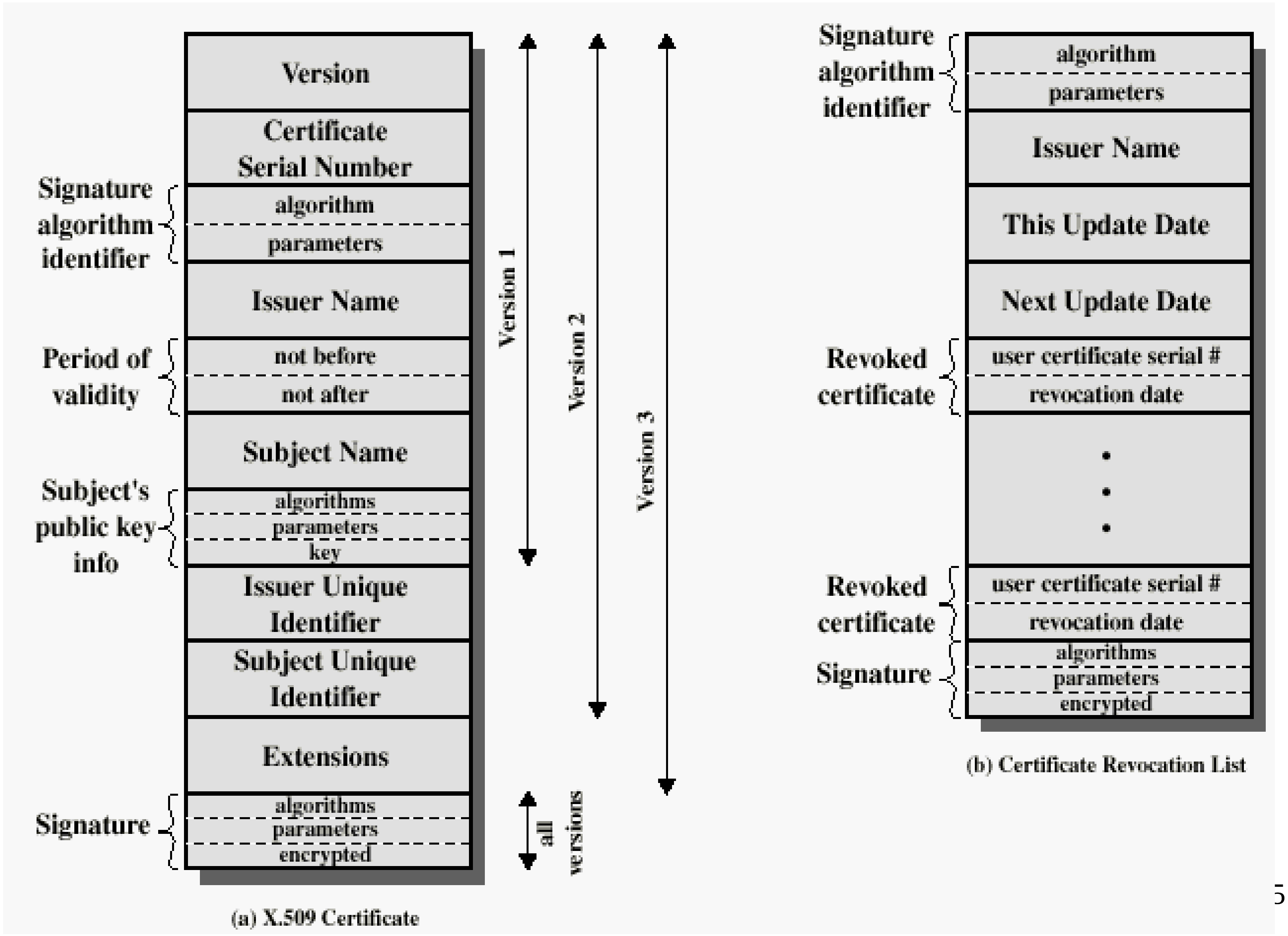
  - dictate no specific encryption algorithm, but recommend RSA

  - use a hash function for the digital signature scheme, but dictate no specific hash algorithm

X.509 Certificates

- The heart of the X.509 scheme is the public-key certificate :
  - created by some trusted CA and
  - placed in the directory by the CA or by the user
- The directory server merely provides an easily accessible location for users to obtain certificates.
- The user certificate can be verified by using the KU of CA known to the user.

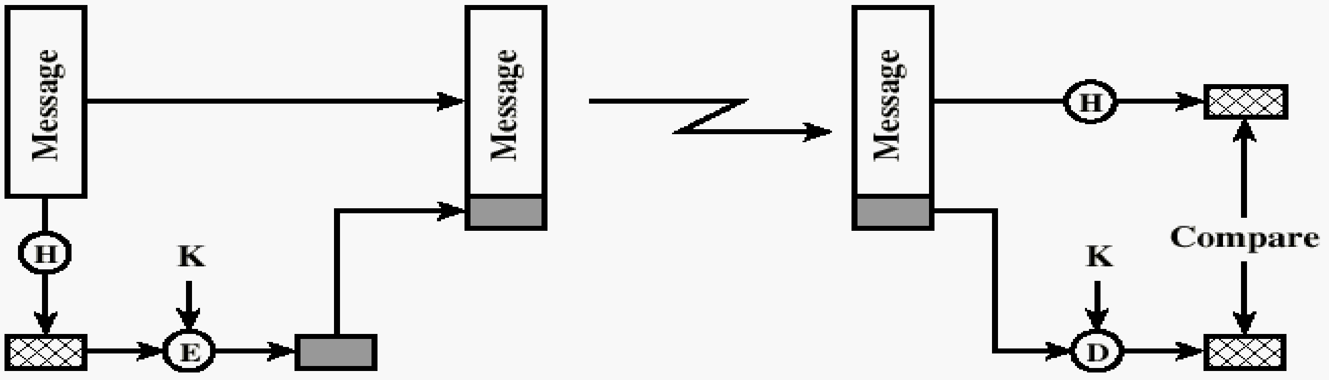
X.509 Certificate Format



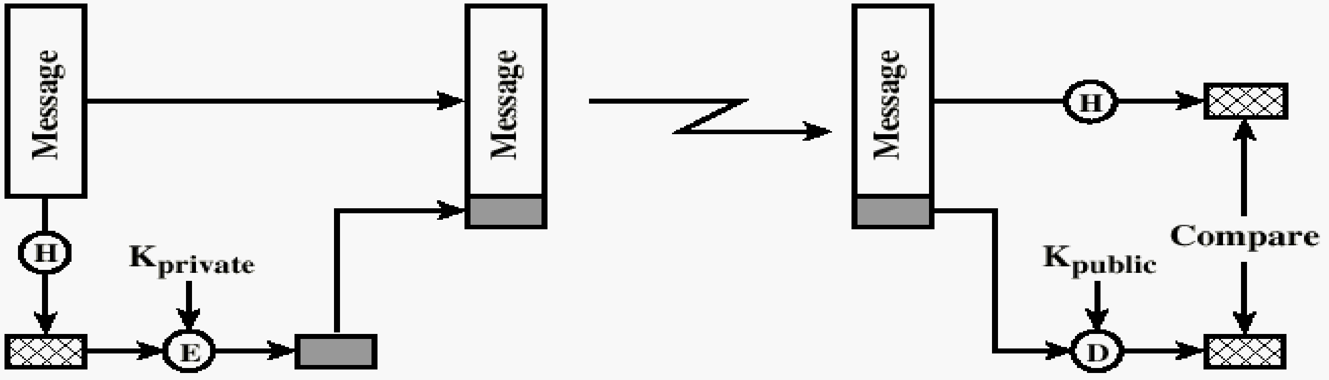
(a) X.509 Certificate

(b) Certificate Revocation List

Typical Digital Signature Approach



(a) Using conventional encryption



(b) Using public-key encryption

Obtaining a User's certificate

- Characteristics of certificates generated by CA:
  - Any user with CA's KU can verify the user public key that was certified by CA's KR.
  - No part other than the CA can modify the certificate without this being detected.
- So, certificates can be placed in a directory without special protection for the directory

Obtaining a User's certificate

- **If all users share a common CA, then they are assumed to know CA's KU.**
- **For a large community of users, it is not practical for all users to share a CA.**
  - **CA's KU should be provided to all users in absolutely secure way**
  - **A number of CAs may be required so that each CA provides its public key to some fraction of users.**
  - **CAs need to exchange their own KUs in a secure way so that a user can verify certificates of users in the other community**

- **Standard notation to define a certificate**

  - CA<<A>> = CA {V, SN, AI, CA, T_A, A, A_p }

  Y<<X>> = Certificate of user X issued by CA Y

  Y{I} = the signing of I by Y ; I + an encrypted hash code

Obtaining a User's certificate

- **When users** A & B **belong to different CAs** X1 **&** X2**, A can verify B's KU if the two CAs have securely exchanged their own KUs :**
  - A obtains the certificate of X2 signed by X1 from the directory
  - A then goes back to the directory and obtains the certificate of B signed by X2 because A has a trusted copy of X2's KU
  - A has used a chain of certificates to obtain B's KU. In the same way, B can obtain A's KU.

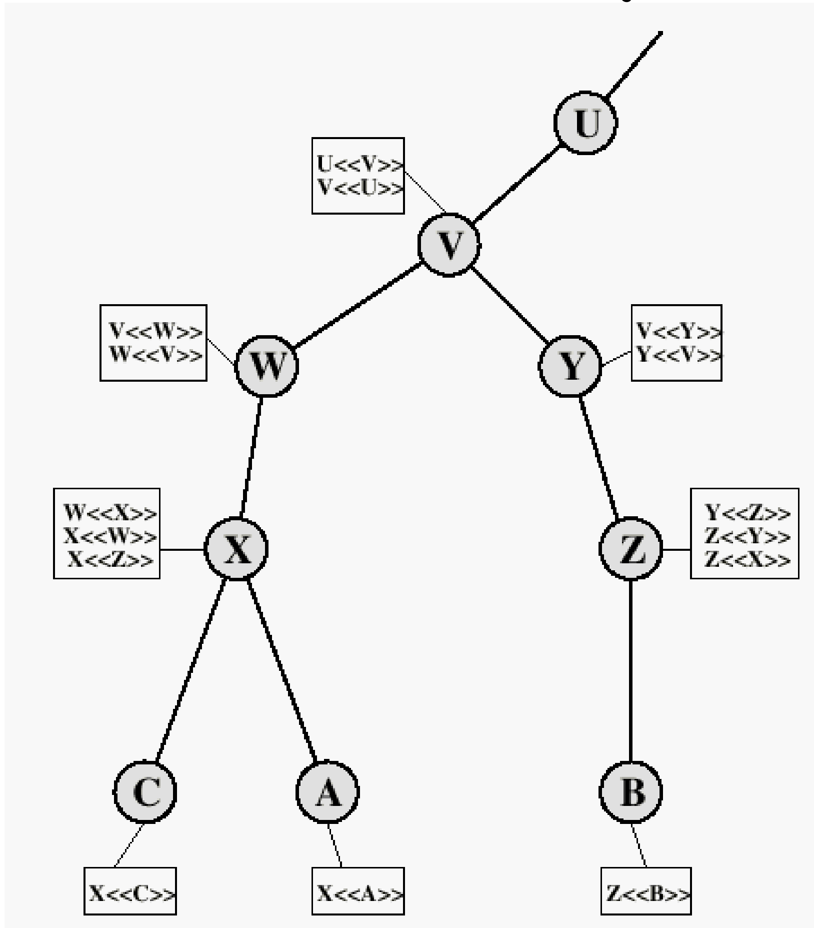
X1<<X2>>X2<<B>>,   X2<<X1>>X1<<A>>

- **This scheme can be applied to an arbitrarily long path of CAs to produce a chain**

Obtaining a User's certificate

- **All these certificates of CAs by CAs need to appear in the directory**
  - Each pair of CAs in the chain must have created certificates for each other
  - Then the user knows how they find the path to another user's KU.
  - CAs need to be arranged in a hierarchy for simple navigation of certificates
- **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

Obtaining a User's certificate (CA hierarchy)



- The path for user A to obtain the certificate of user B, X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>

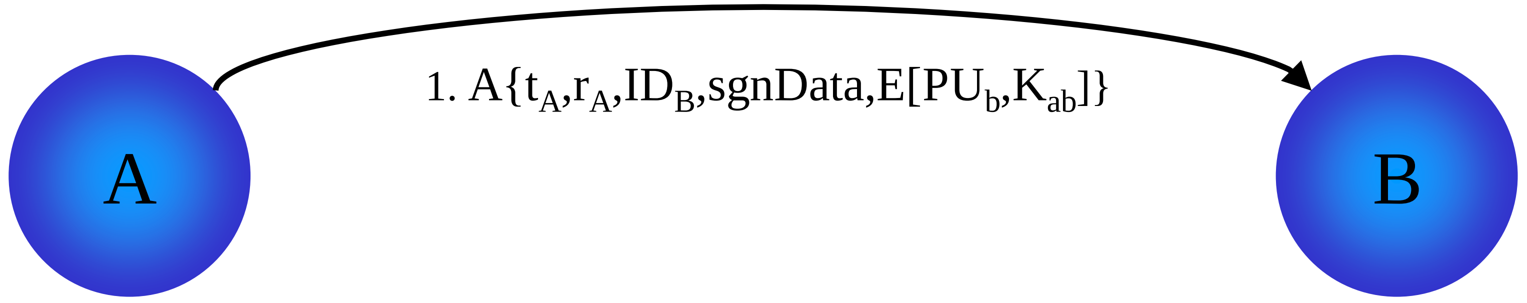
Revocation of Certificates

- Reasons for revocation before it expires
  - The user’s KR 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 of revoked certificates issued to users or other CAs
  - Each certificate revocation list (CRL) posted to the directory is signed by the issuer.
- Users must check certificates with CA’s CRL

Authentication Procedures

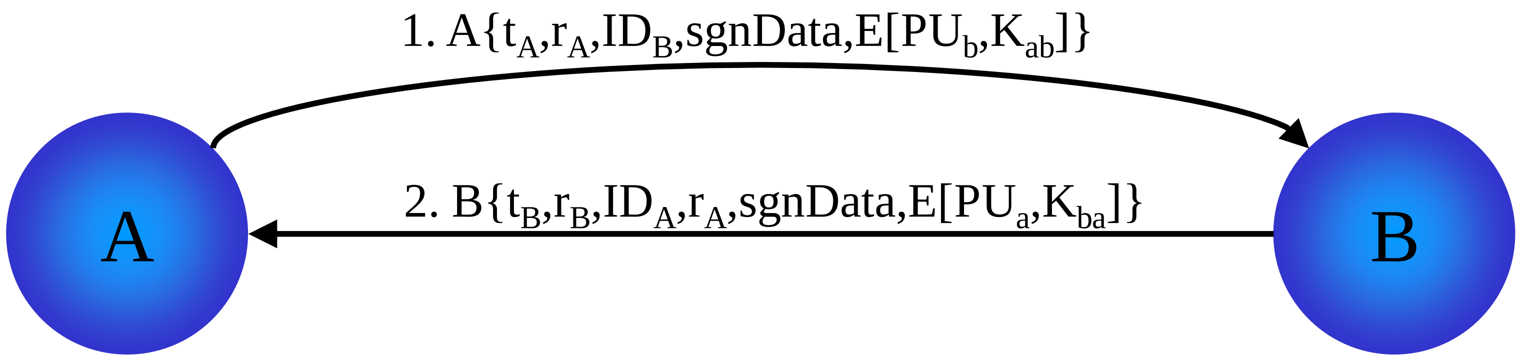
- X.509 includes three alternative authentication procedures
  - intended for use across a variety of applications
  - to use public-key signature
  - assumed for the two parties to know each other's KU
- The three procedures :
  - One-way authentication
  - Two-way authentication
  - Three-way authentication

One-Way Authentication



- A single transfer of information from A to B
- Verification of the followings:
  1. A's identity and the message generated by A
  2. the message intended for B
  3. the integrity and originality of the message
- At minimum : TS t_A, nonce r_A, B's identity, A's signature

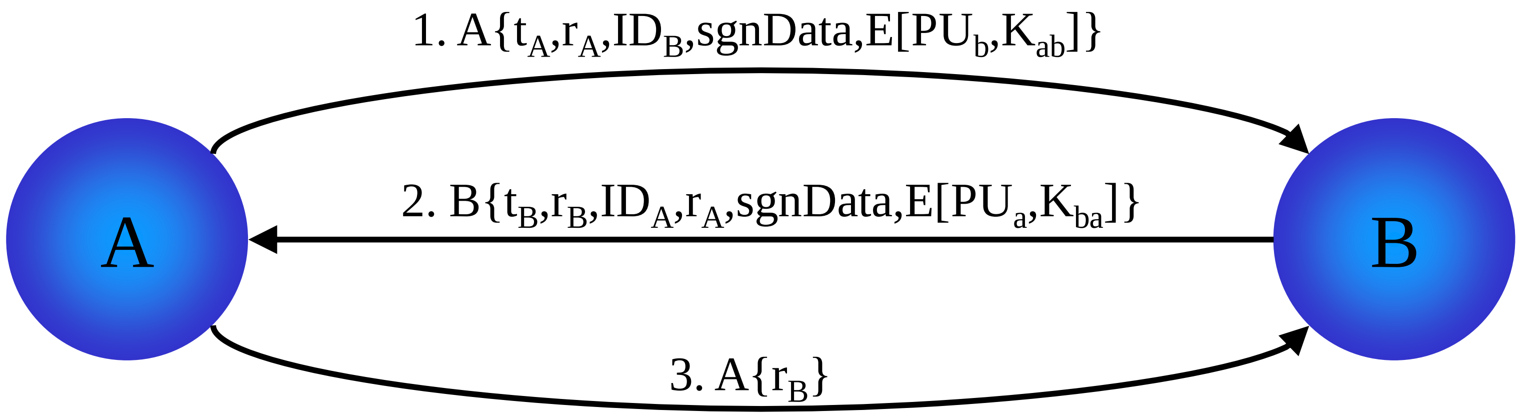
Two-Way Authentication


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}]}
A
B

- Verification of the followings:
  4. B’s identity and the message generated by B
  5. the message intended for A
  6. the integrity and originality of the reply

- Permission of verification for both parties
- Reply includes A’s nonce, TS and nonce from B

Three-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

- The signed copy of the nonce r_B in the final MSG
  - TS need not to be checked
  - Replay attack can be detected by nonces echoed to each other

X.509 Version 3

- Has been recognized that additional information is needed in a certificate
  - e-mail/URL, policy details, usage constraints
- Include a number of optional extensions added to version 2 format
  - rather than continue to add fields to a fixed format
- Each extension consists of extension identifier, criticality indicator, extension value

Certificate Extensions

- Key and policy information
  - Convey additional info. about subject & issuer keys, plus indicators of certificate policy
  - A certificate policy is a named set of rules for the applicability of a certificate to a particular community and/or class of application
- Certificate subject and issuer attributes
  - Support alternative names, in alternative format, for a certificate subject or certificate issuer
- Certification path constraints
  - Allow constraint specifications to be included in certificates issued for CAs by other CAs

Public-Key Infrastructure (PKI)

- The set of hardware, software, people, policies, and procedures needed to create, store, distribute, and revoke digital certificates
- The principal objective is to enable secure, convenient, and efficient acquisition of public keys.
- The PKI X.509 (PKIX) model is a formal model suitable for deploying a X.509 certificate-based architecture on the Internet

PKIX of IETF

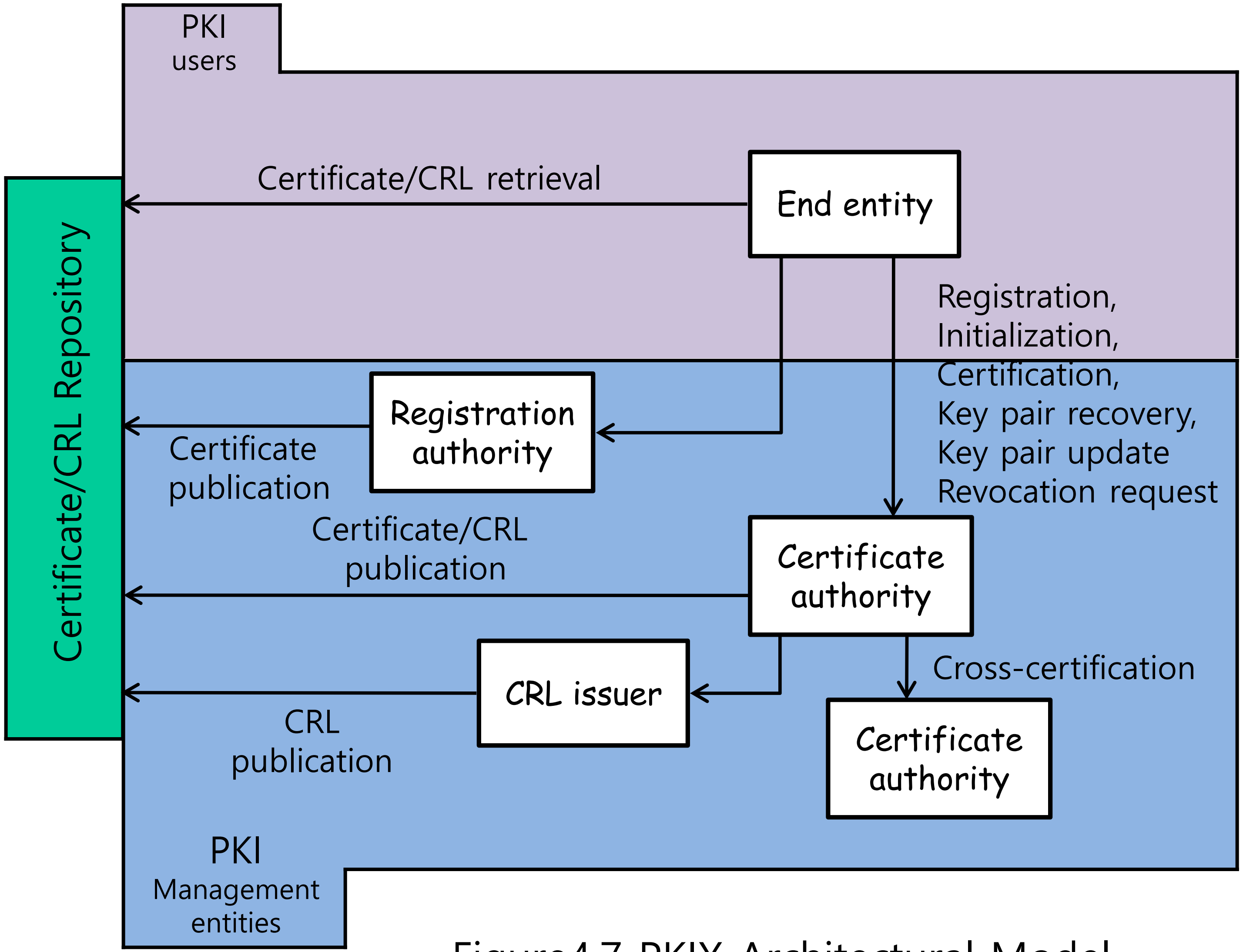

PKI users
Certificate/CRL retrieval
End entity
Registration, Initialization, Certification, Key pair recovery, Key pair update Revocation request
Certificate/CRL Repository
Registration authority
Certificate publication
Certificate/CRL publication
Certificate authority
Cross-certification
CRL issuer
CRL publication
Certificate authority
PKI Management entities

Figure4.7 PKIX Architectural Model

PKIX Management Protocols

- Two alternative management protocols has been defined between PKIX entities to support the management functions
- Certificate Management Protocols (CMP)
  - Within CMP, each of the mgmt functions is explicitly identified by specific protocol exchanges
- Certificate Management Message over CMS (CMC)
  - CMS : cryptographic message syntax
  - Is built on earlier work and intended to leverage existing implementations
  - The functions do not all map into specific protocol exchanges

Summary

- A survey of the most important authentication specifications in current use
  - Kerberos
    - . authentication protocol based on conventional encryption that has received widespread support
  - X.509
    - . Specifying an authentication algorithm and define a certificate facility
    - . Enables users to obtain certificates of the public keys so that a community of users can have confidence in the validity of the public keys