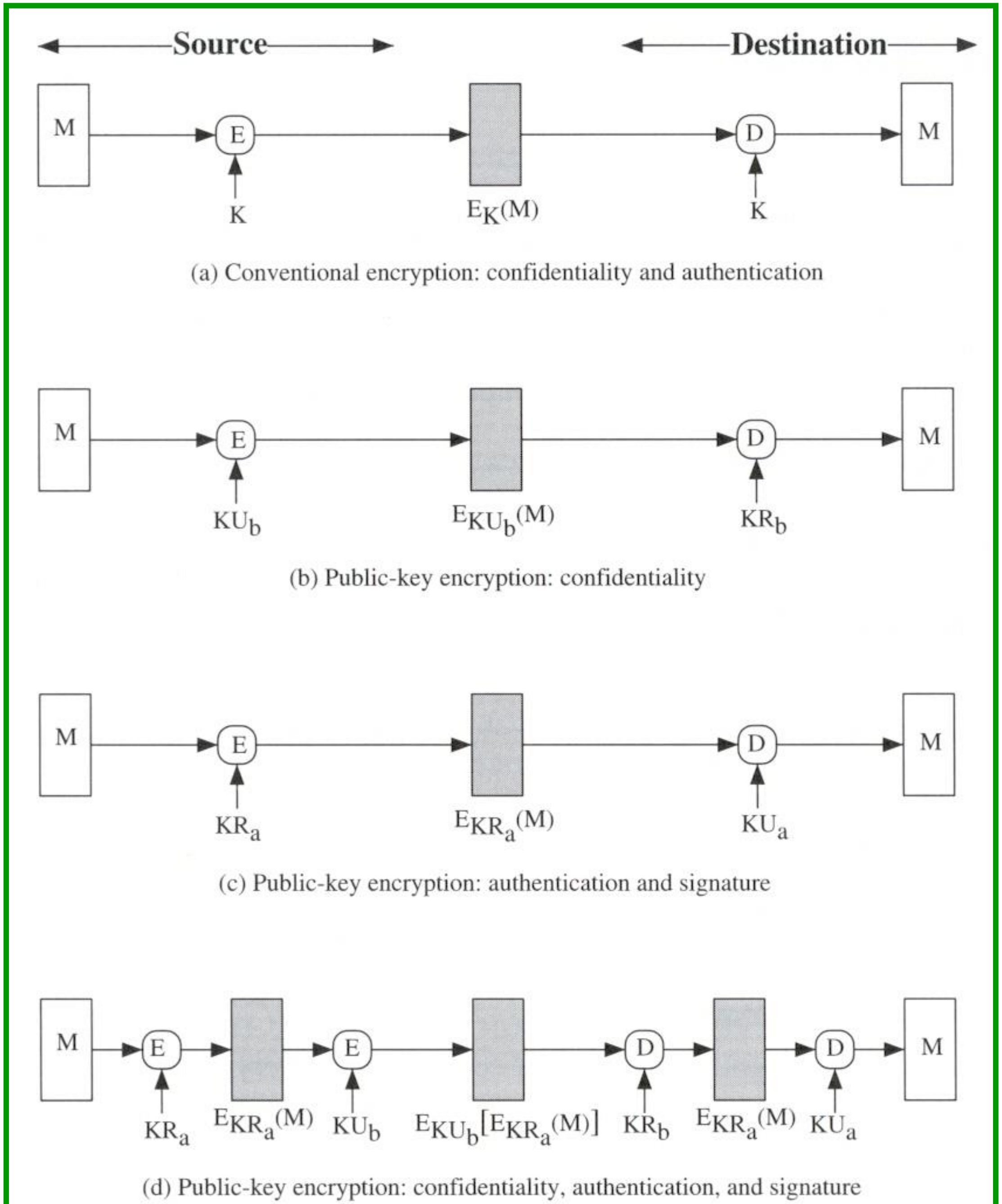
Chap. 4 Key Distribution and User Authentication

- Authentication
- Kerberos
- Public Key Infrastructure

Message Authentication

- Message authentication is a service that
  - allows receivers of a messages to identify its origin
  - makes it difficult for third parties to masquerade as someone else
- Methods to provide authentication of a message
  - Encryption: only the holder of a secret key could have generated the message
  - Hash functions: generating another message that matches the hash is difficult
  - Message Authentication Codes (MAC): encrypted hash value using a secret key

Authentication Using Encryption



Authentication Using Encryption

- All things decrypt! -> even garbage text sent by a malicious foe will decrypt to *something*
- We need to reduce the chances of garbage decrypting to something useful
- Before encrypting, compute a *frame-check sequence* (CRC codes, …)
- Encrypt both the plaintext and its FCS
- Garbage has little chance of decrypting to a correct FCS
- It's costly for both Alice and Bob when authenticating a large message: the whole message must be encrypted !!!

Hash Functions

- Hash H is a *one-way* function that operates on arbitrary length msg. m, and returns a fixed-length value h

$$h = H(m)$$

Given a message m, it is easy to compute H(m)

- Given h, it is impossible to compute m such that $H(m)=h$
- Given specific m, it is impossible to find another msg. m', such that $H(m) = H(m')$. => weak collision resistance
- Given a large set M, it's difficult to find any pair (m_i, m_j) that hash to the same value => strong collision resistance

- Hashes provide a fingerprint of m

Digital Signatures

- Real signatures provide a number of features
  - Signature provides authenticity for a documents
  - Signatures are hard to forge
  - Signatures, as parts of the document, aren't reusable
  - Signatures are unalterable or erasable
  - Signatures can't be repudiated

Signing with Hash Functions

1. Alice produces a one-way hash of the document D

$$A: h = H(D)$$

2. Alice encrypts the hash with her private key (Digital Signature)

$$A: \boxed{DS = E_{KRa}(h)} \qquad ; \text{KRa : private key of Alice}$$

3. Alice sends the document and the signed hash to Bob

$$A\text{->}B: [\ D \,||\, \boxed{DS}\ ]$$

4. Bob verifies the received document [D' || DS'] as follows:

$$\text{Bob: check if } \boxed{E_{KUa}(DS') == H(D')}$$

Signing Documents with Private Keys

- Digital signatures
  - The *DS* is authentic (the hashes match)
  - The *DS* is unforgeable (only the sender has the private key to create *DS*)
  - The *DS* is not reusable (it's a function of the document)
  - The signed document is unalterable (the hashes wouldn't match)
  - The document can't be repudiated (only the sender can create *DS*)

Bob attacks the notary

- Weak collision resistance: given a *m*, it is hard to find another message *m'*, such that $H(m) = H(m')$
  - Alice is a notary for Bob's documents
  - For $5, Alice signs Bob's document and sends her
  - Bob has one "valid" document
  - Bob tries to look for another version of the document (with fraudulent info) that hashes to the same value
  - Bob places Alice's DS for the valid document with the fraudulent one
  - Bob has to search through 2^n messages (2^{n-1} on average); n is the length of H(m)

Birthday Attack

- Strong collision resistance: given a large set M, it's difficult to find any pair (m_i, m_j) that hash to the same value
  - Alice is a notary for Bob's documents
  - For $5, Alice signs Bob's document and sends her
  - Bob has many versions of the same "valid" document
  - Creates many versions of the fraudulent doc → there is a good chance that one pair will match up
  - Bob places Alice's encryption of the hash for the valid document with the fraudulent one
  - Bob has to search through $2^{n/2}$ messages on the average → Birthday attack

Birthday Paradox

- What is the min value of k such that the prob. is greater than 0.5 that at least 2 people in a group of k people have the same birthday?
- Pr(365, k): the prob. that there are duplicates in the group
- Pr(365, k) = 1 – Q(365, k), where Q(365, k) is the probability that there are no duplicates in the group
- N: the # of different ways that can have k values without duplicates
  - $N = 365 \times 364 \times \ldots \times (365-k+1) = \dfrac{365!}{(365-k)!}$
- $Q(365, k) = N / (365)^k$
- $Pr(365, k) = 1 - N / (365)^k > 0.5 \quad \approx \quad 2^{n/2} \quad -> \quad k = 23$

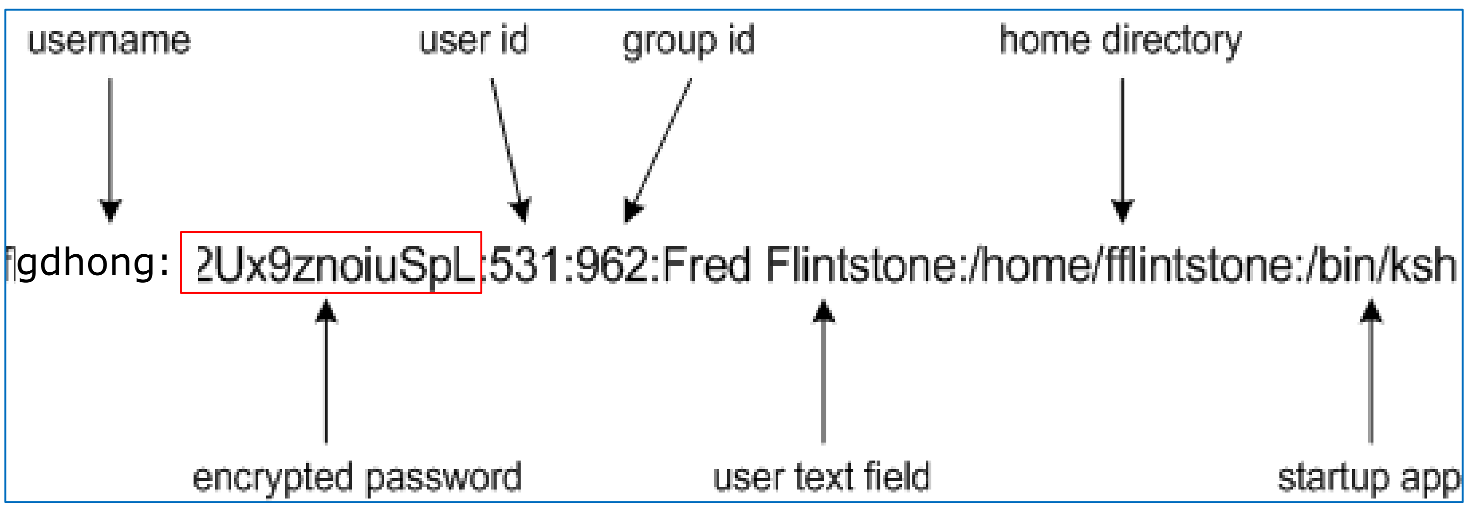
n: number of bits of the hash value

Authentication

- Password based authentication
  - The password file could be stolen
  - An eavesdropper can sniff the password off the network
- Authentication based on the source address
  - IP spoofing is possible
- Authentication based on biometrics : thumb prints, retinal scans
- Authentication using symmetric keys : Kerberos
- Authentication using asymmetric cryptosystem

Protecting the Password File

- Actual password is not stored -> one-way hashes of passwords are kept and are used for comparison
- Other users can only read /etc/passwd
- Only the "root" user can read the password hashes in the /etc/shadow password file



Dictionary Attacks

- Finding a password for a specific account is hard
- Instead, get the password file (even if it’s hashed passwords)
- Compute the one-way hashes of all the words in the dictionary, including common names for pets, and compare

- You won’t get every password, but you’ll get *some, probably about 40%*

Protecting Passwords over the network

- If Alice just sends the password, anyone can read it
- In *promiscuous* mode, Ethernet cards will pass to the operating system all received IP packets
- Attackers can use a "packet-sniffer", like wireshark or tcpdump on Unix, to read all packets across your network
- Such programs require root privileges

Authentication Using OTPs

Challenge-response method

- Ahead of time, Alice and Bob agree upon a secret, shared key (or a secret function F)
- Alice requests a log-in challenge from Bob (the remote computer) : A->B: *request*
- Bob sends Alice a *nonce N* (challenge): B->A: N
  - A nonce is a random string used only once ever
- Alice responds (response) : A->B: $E_{Kab}(N)$ (or A->B: F(N))
- Changing nonces for each access prevents replay attack

Authentication Using OTPs

S/Key method

- Ahead of time, Alice and Bob generate a list of passwords from a chosen pass-phrase, and share the list
- When accessing Bob, Alice uses each password in the list only once one-by-one
- Bob checks the password using the password list
- Changing passwords for each access prevents replay attack

Key Exchange Protocols: Needham-Schroeder

T is the Key Distribution Center (KDC)

A is Alice, B is Bob, M is Mallony

K_{TA} (K_{TB}) : master key between KDC and A (B)

N_1 and N_2 are nonces

K_{AB} is the session key the KDC generates

- A->T: A, B, N_1
- T->A: $\{K_{AB}, B, N_1, \{K_{AB}, A\}K_{TB}\}K_{TA}$
- A->B: $\{K_{AB}, A\}K_{TB}$
- B->A: $\{N_2\}K_{AB}$
- A->B: $\{N_2+1\}K_{AB}$

<Message exchanges using K_{AB}>

$\{M\}K \rightarrow E_K(M)$

M stores the entire session and works on the key, and catches K_{AB}. Later...

M->B: $\{K_{AB}, A\}K_{TB}$

B->A: $\{N_3\}K_{AB}$

M->B: $\{N_3+1\}K_{AB}$

Using Timestamp

t is a timestamp(current time), everyone has synchronized clocks.

- A-> T: A,B
- T->A: $\{K_{AB}, B, t, \{K_{AB}, A, t\}K_{TB}\}K_{TA}$
- A->B: $\{K_{AB}, A, t\}K_{TB}$
- B->A: $\{N_2\}K_{AB}$
- A->B: $\{N_2+1\}K_{AB}$

M stores the entire session and works on the key, and catches K_{AB}. Later...

M->B: $\{K_{AB}, A, t\}K_{TB}$

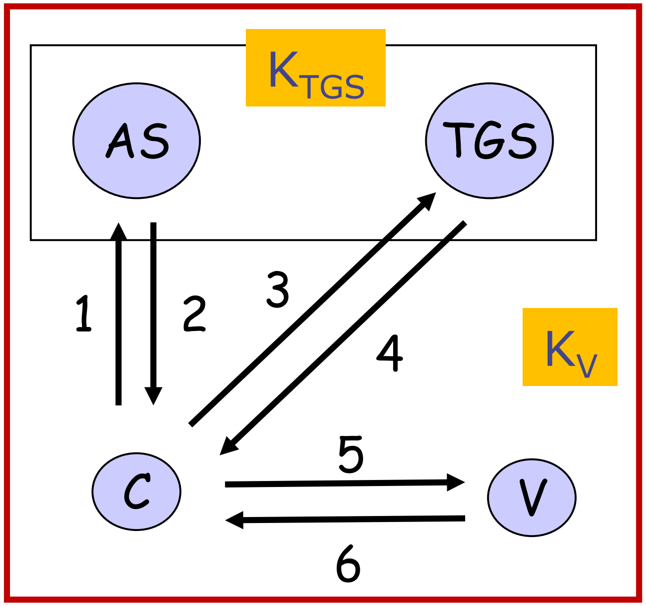
B rejects because t is old !!!

KERBEROS

- Users wish to access services in servers
- Three threats exist:
  - User pretend to be another user
  - User alter the network address of a workstation
  - User eavesdrop on exchanges and use a replay attack
- Kerberos
  - Provides a centralized authentication server to authenticate b/w users and servers
  - Relies on conventional encryption
  - Two versions: version 4 and 5 → V4 makes use of DES; V5 allows other encryption alg.

KERBEROS

- Kerberos environment
  - AS – authenticates users; issues ticket-granting-ticket
  - TGS – issues a ticket to access servers



Kerberos domain

AS: authentication server
TGS: ticket granting server
V: application server

1,2 : when a user logins in
3,4 : when a user wants to access an application server
5,6 : while a user is accessing an application server

Kerberos V4

- Terms:
  - C = Client
  - AS = authentication server
  - TGS : Ticket granting Server
  - V = application server
  - IDc = identifier of user on C
  - IDv = identifier of V
  - K_c = master key of IDc (generated from the password of user IDc)
  - ADc = network address of C
  - K_{TGS} = secret encryption key shared by AS and TGS
  - K_v = secret encryption key shared by TGS and V
  - TS = timestamp
  - || = concatenation

Kerberos V4 Authentication Dialogue

- Authentication Service Exchange
  - To obtain Ticket-Granting Ticket
  - Once when a user logins in

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

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

$Ticket_{tgs} = E_{Ktgs}[K_{c,tgs} \,||\, ID_c \,||\, AD_c \,||\, ID_{tgs} \,||\, TS_2 \,||\, Lifetime_2\,]$

Kerberos V4 Authentication Dialogue

- Ticket-Granting Service Exchange
  - To obtain Service-Granting Ticket to access a server
  - Every time when a user wants to access a new server

(3) $C \rightarrow TGS$: $ID_v \,||\, Ticket_{tgs} \,||\, Authenticator_c$

(4) $TGS \rightarrow C$: $E_{Kc,tgs}\,[K_{c,v} \,||\, ID_v \,||\, TS_4 \,||\, Lifetime_4 \,||\, Ticket_v]$

$Ticket_{tgs} = E_{Ktgs}[K_{c,tgs} \,||\, ID_c \,||\, AD_c \,||\, ID_{tgs} \,||\, TS_2 \,||\, Lifetime_2\,]$

$Authenticator_c = E_{Kc,tgs}[ID_c \,||\, AD_c \,||\, TS_3]$

$Ticket_v = E_{Kv}[K_{c,v} \,||\, ID_c \,||\, AD_c \,||\, ID_v \,||\, TS_4 \,||\, Lifetime_4\,]$

Kerberos V4 Authentication Dialogue

- Client/Server Authentication Exhange
  - To obtain Service
  - Every time when a user wants to access a server

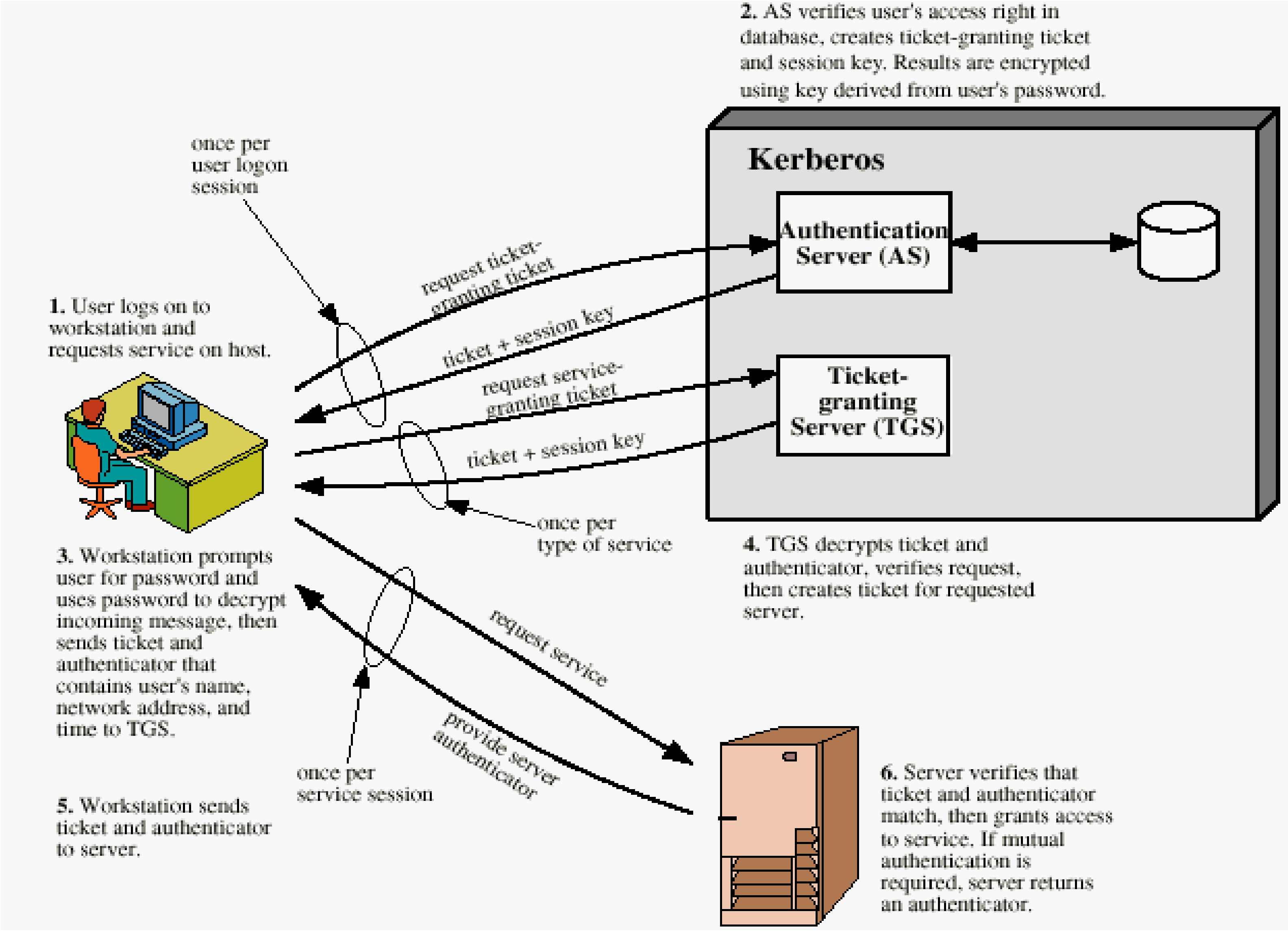
(5) C → V: $Ticket_v$ || $Authenticator_c$

(6) V → C: $E_{Kc,v}[TS_5 + 1]$

$Ticket_v = E_{Kv}[K_{c,v} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime_4]$

$Authenticator_c = E_{Kc,v}[ID_c \| AD_c \| TS_5]$

Kerberos

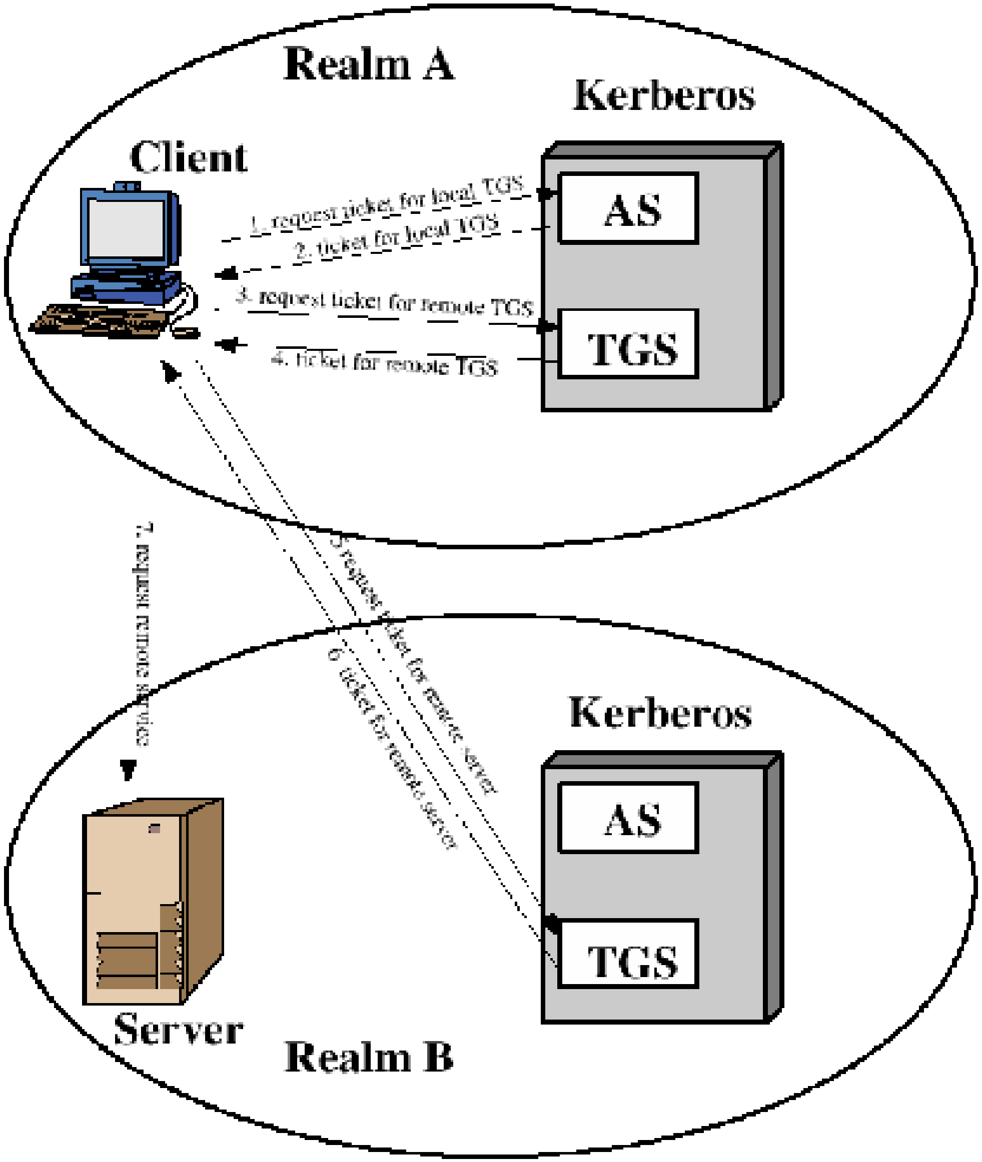


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
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

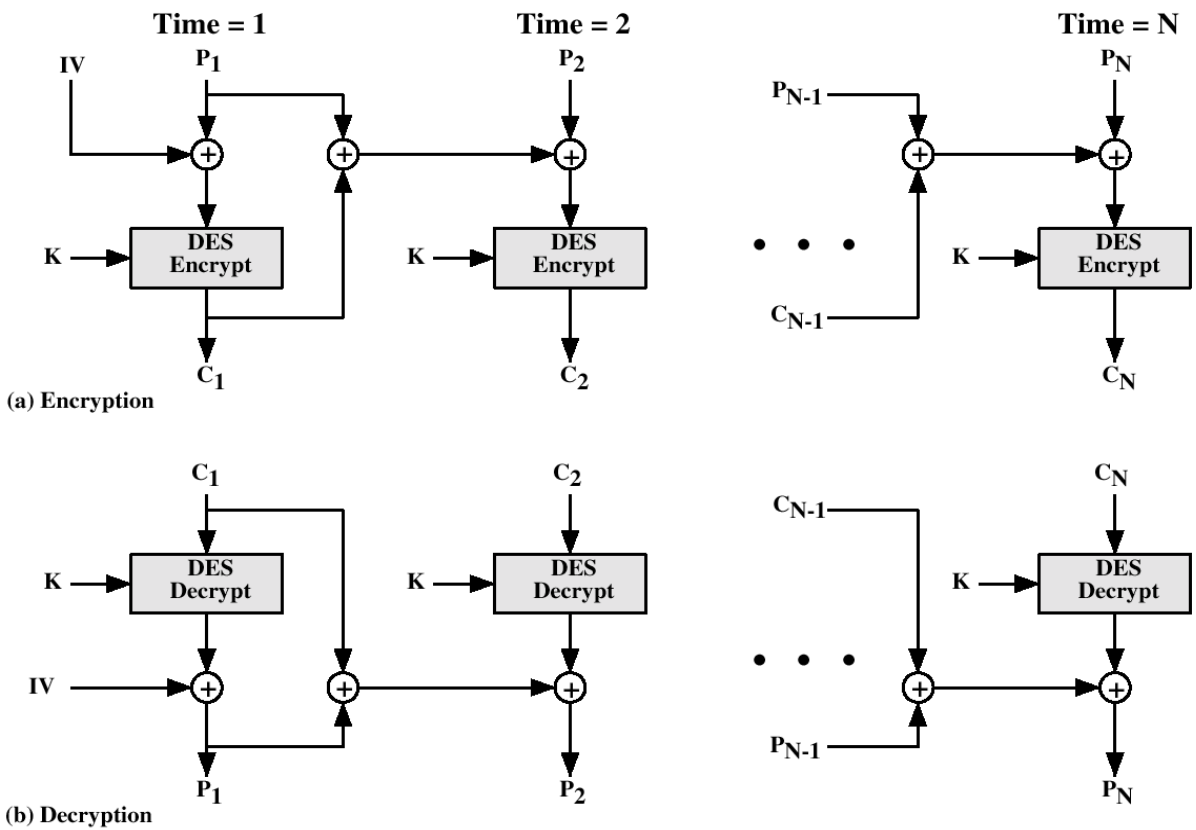
- Request for service in another Realm

1. Client gets ticket for local TGS from AS
2. Client gets ticket for remote TGS from local TGS
3. Client gets ticket for appl server from remote TGS



Kerberos Encryption

- Kerveros V4 uses DES PCBC Mode
  - Vulnerable to an attack involving the interchange of cypertext blocks



Kerberos - in practice

- **C**urrently have two Kerberos versions:
  - V4 : restricted to a single realm
  - V5 : allows inter-realm authentication
  - Kerberos v5 is an Internet standard (RFC1510)
- To use Kerberos:
  - need to have a KDC (AS and TGS) on your network
  - need modification to applications running on all participating systems
  - major problem – US export restrictions (Kerberos cannot be directly distributed outside the US in source format)

X.509 Authentication Service

- X.500 directory service
  - directory – distributed set of servers that maintains a database about users
- X.509 Authentication Service
  - Defines a framework for providing authentication services using X.500 directory to users
  - directory – a repository of public-key certificates of users
  - Each certificate contains the public key of a user and is signed with the private key of a CA
  - Is used in S/MIME, IP Security, SSL/TLS and SET
  - RSA is recommended to use as a public-key algorithm

Key Exchange with Public-Keys

1. Alice gets Bob's public key from a Key Distribution Center (KDC), T

   $$T\text{->}A: \; K_{\mathbf{KUb}}$$

2. Alice generates a random session key, encrypts it using Bob's public key, and sends it to Bob

   $$A\text{->}B: \{K_{\mathbf{AB}}\}K_{\mathbf{KUb}}$$

3. Bob decrypts the message using his private key

   $$B: K_{\mathbf{AB}}$$

4. The session begin, both Alice and Bob using the same session key

=> this has serious man-in-the-middle attack problems

Man-in-the-Middle Attack

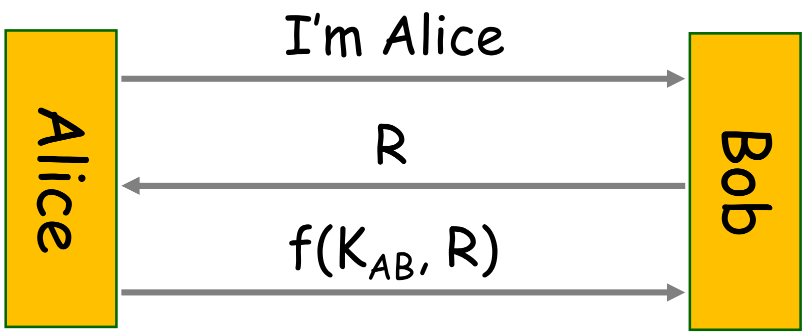
1. Alice sends Bob her public Key; Mallory intercepts and sends Bob his own public key
2. Mallory also intercepts when Bob sends his key to Alice
3. When Alice sends a message encrypted with "Bob's" key, Mallory intercepts, decrypts, alters, and re-encrypts with the correct key

-> needs a secure way of obtaining other's public keys

More : One Way Authentication

- Designing secure authentication protocols is hard
- One way authentication using a shared secret

<Proto A-1>



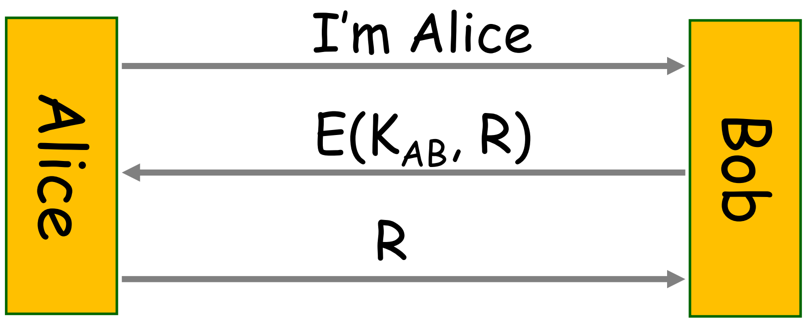
K_{AB} : shared secret b/w Alice and Bob

  - Authentication is not mutual
  - Someone who reads the database at Bob can impersonate Alice

- Suspicious party should generate a challenge

One Way Authentication

- Variant of <Proto A-1>



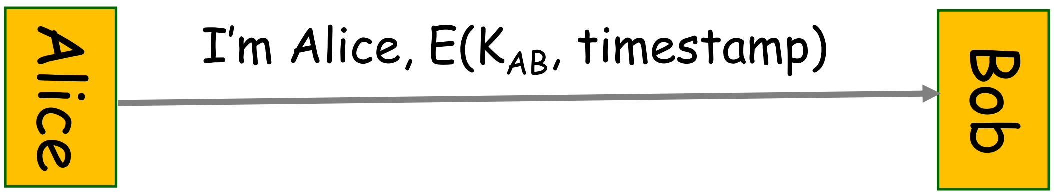
<Proto A-2>

- The cryptography must be reversible
- If R is a recognizable quantity (e.g. includes timestamp), Alice can authenticate Bob, i.e. Bob knows a shared secret

One Way Authentication

- Variant of <Proto A-1>

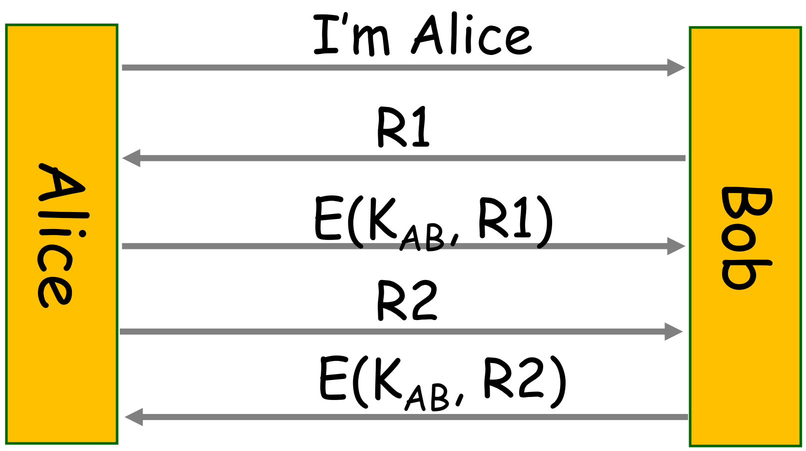
<Proto A-3>



  - More efficient: needs short messages and Bob does not need to keep any volatile state
  - Bob and Alice must have reasonably synchronized clocks
  - An eavesdropper can use $E(K_{AB}$, timestamp) to impersonate Alice, if done within the acceptable clock skew

Mutual Authentication

- Mutual authentication based on a shared secret

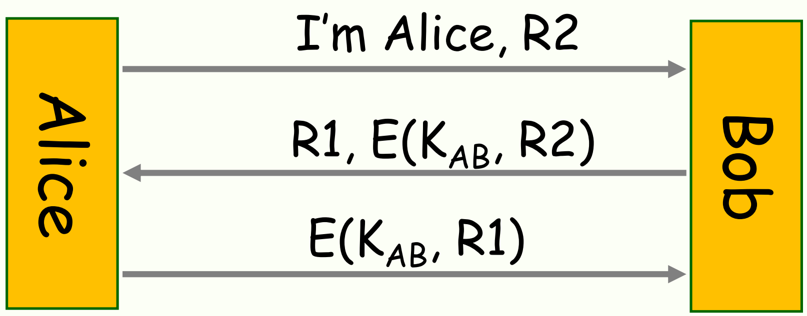


<Proto A-4>

- The initiator should be the first to prove its identity
- Inefficient: 5 message exchanges for authentication

Mutual Authentication

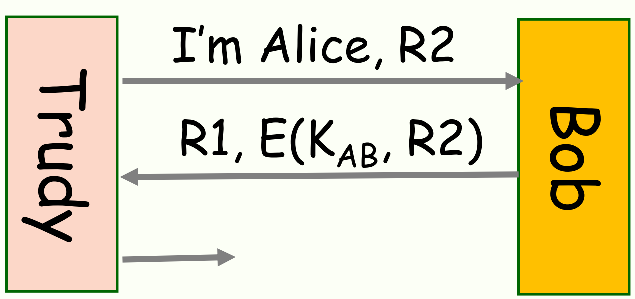
Variant of <Proto A-4>

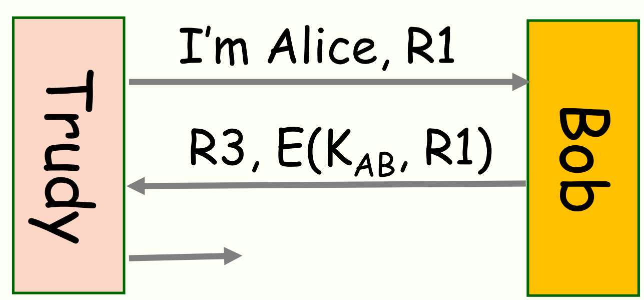


<Proto A-5>

- Reflection attack: Trudy initiates a first session (<Fig 1>), and initiates a new second session using R1 (<Fig 2>) -> uses the reply from the second session and completes the first session

<Fig 1>





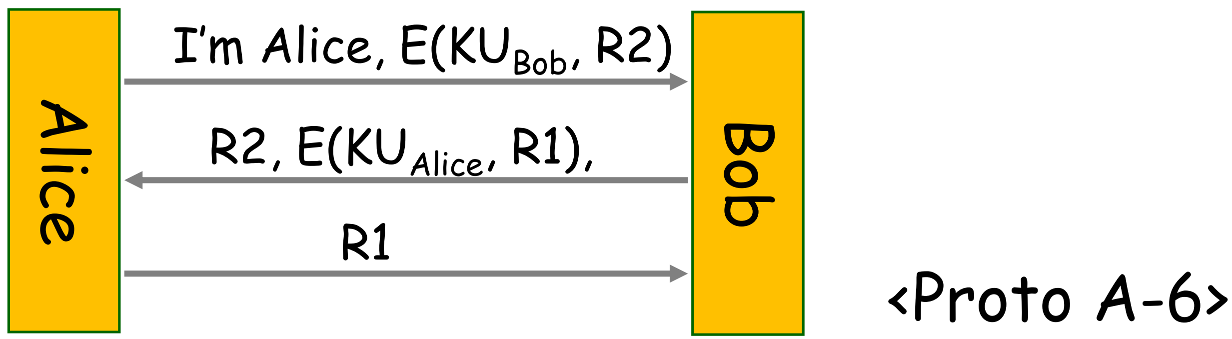
<Fig 2>

Mutual Authentication

- Fixing the problem of reflection attack
  - Using different keys hardens the attack: (e.g.) Alice -> Bob: K_{AB} and Bob -> Alice: ($K_{AB} \oplus F0F0F0F0F0F0F0F0$)
  - Using different types of challenges between from Bob to Alice and from Alice to Bob makes the attack difficult: (e.g.) odd number from Bob to Alice and even number from Alice to Bob

Mutual Authentication

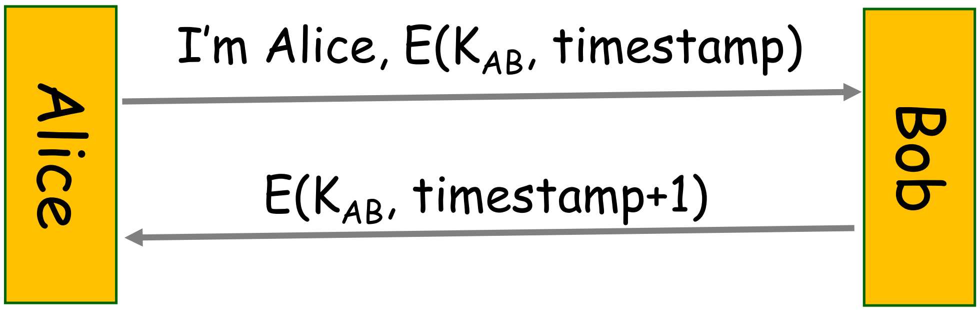
- Mutual authentication using public keys



  - How does each one obtain the other's public key securely?
  - Trudy can intervene the public key exchange: man-in-the-middle attack
  - Needs a secure way of obtaining peer's public key: PKI (public key infrastructure)

Mutual Authentication

- Mutual authentication using timestamps

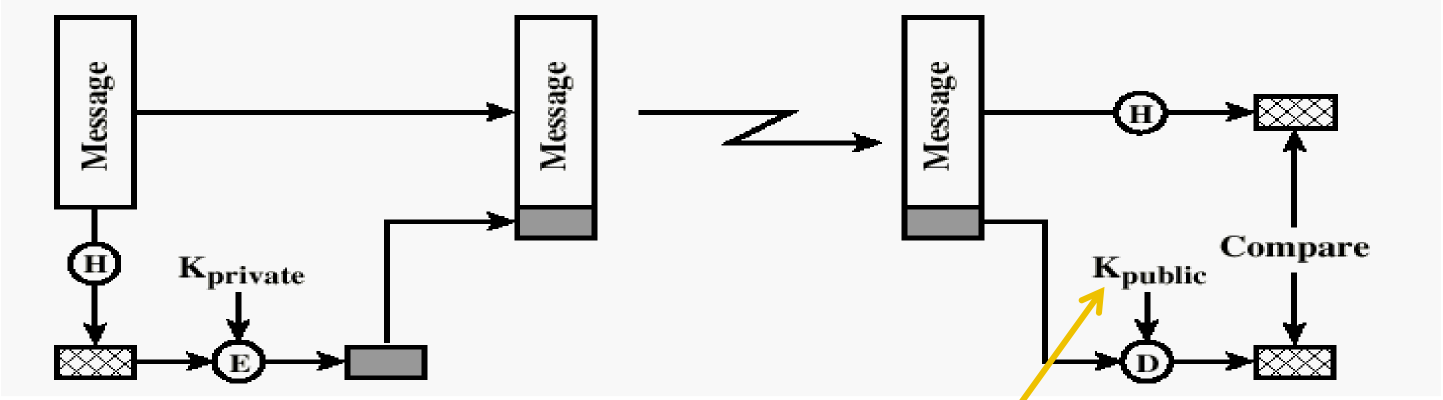


<Proto A-7>

  - Efficient: needs short messages and easy to add to existing protocols (request/response paradigm)
  - Trudy impersonate Alice by eavesdropping E(K_{AB}, timestamp+1)

X.509 Authentication Service

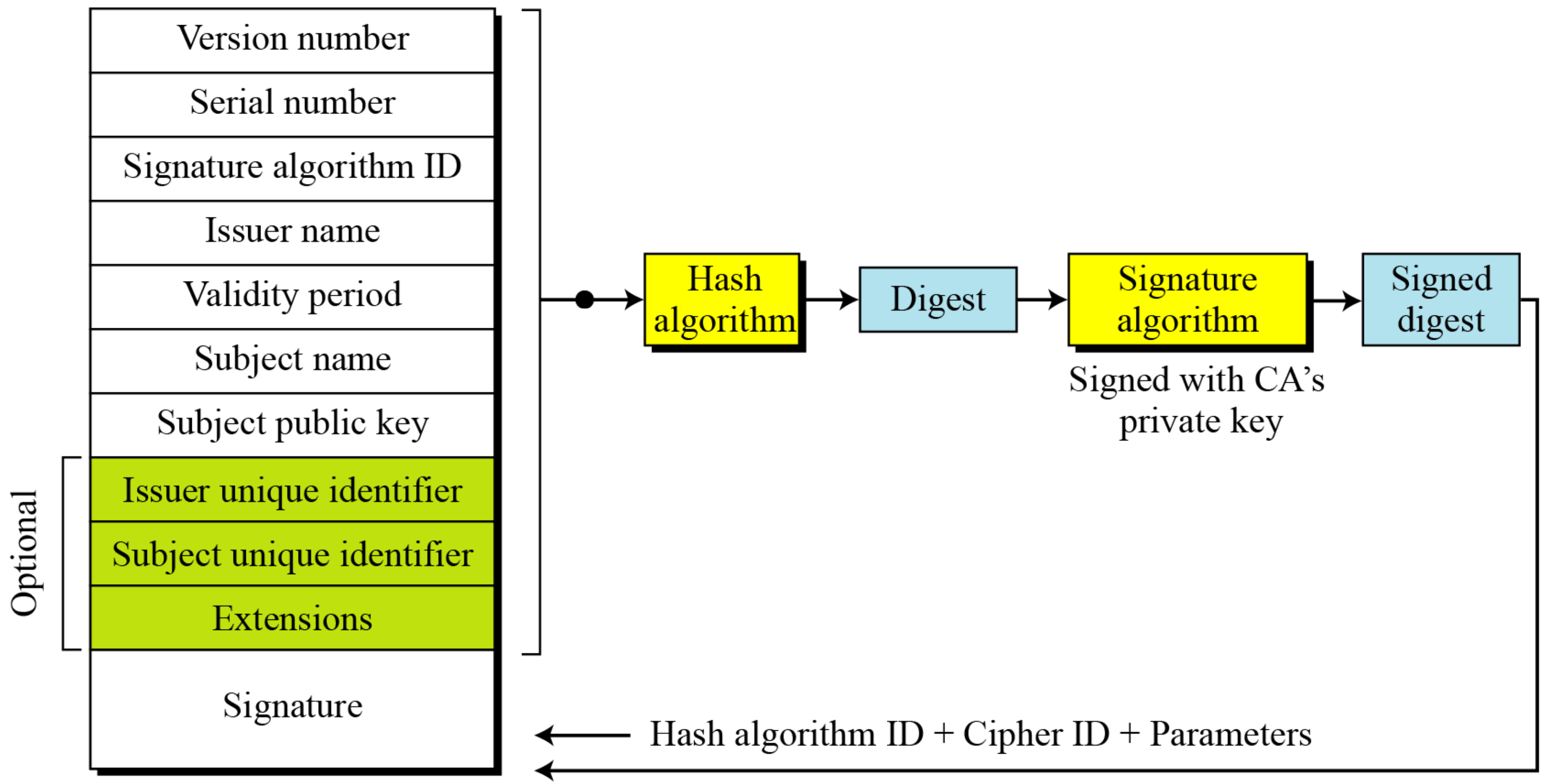
- Certification Authority (CA)
  - Issues a certificate for a user
  - Each certificate is signed by the CA



We need a correct
peer’s public key. How?

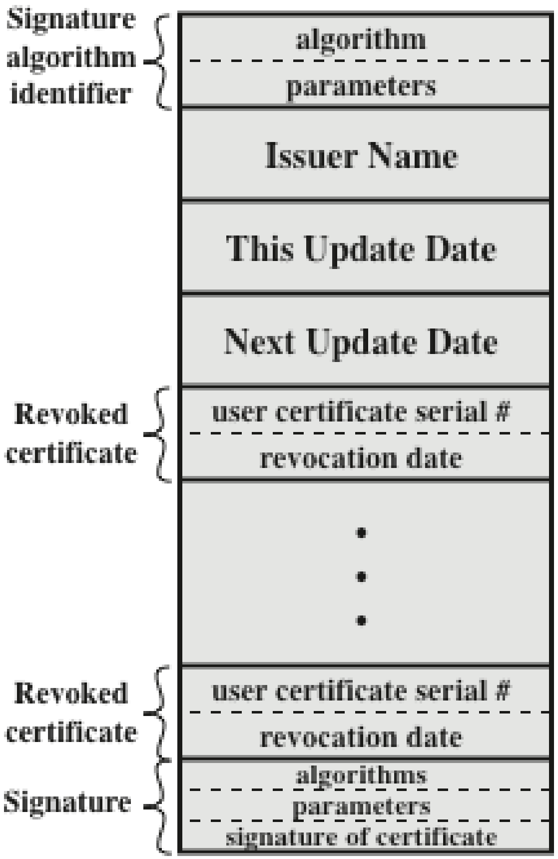
X.509 Authentication Service

- X.509 certificate format



Revocation of Certificates

- Reasons for revocation:
  - The users private key is assumed to be compromised
  - The user is no longer certified by this CA
- CRL (Certification Revocation List)
  - Each CA keeps CRL and updates CRL periodically
  - Checks certificate's validity from CRL
- Delta Revocation
  - To make revocation more efficient, the delta CRL has been introduced

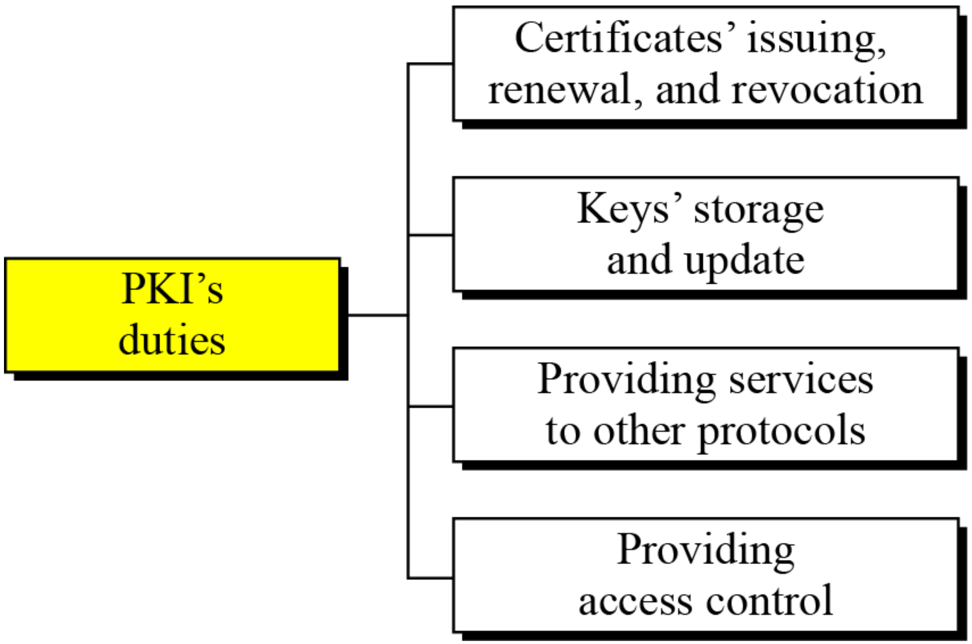


(b) Certificate Revocation List

Public-Key Infrastructure

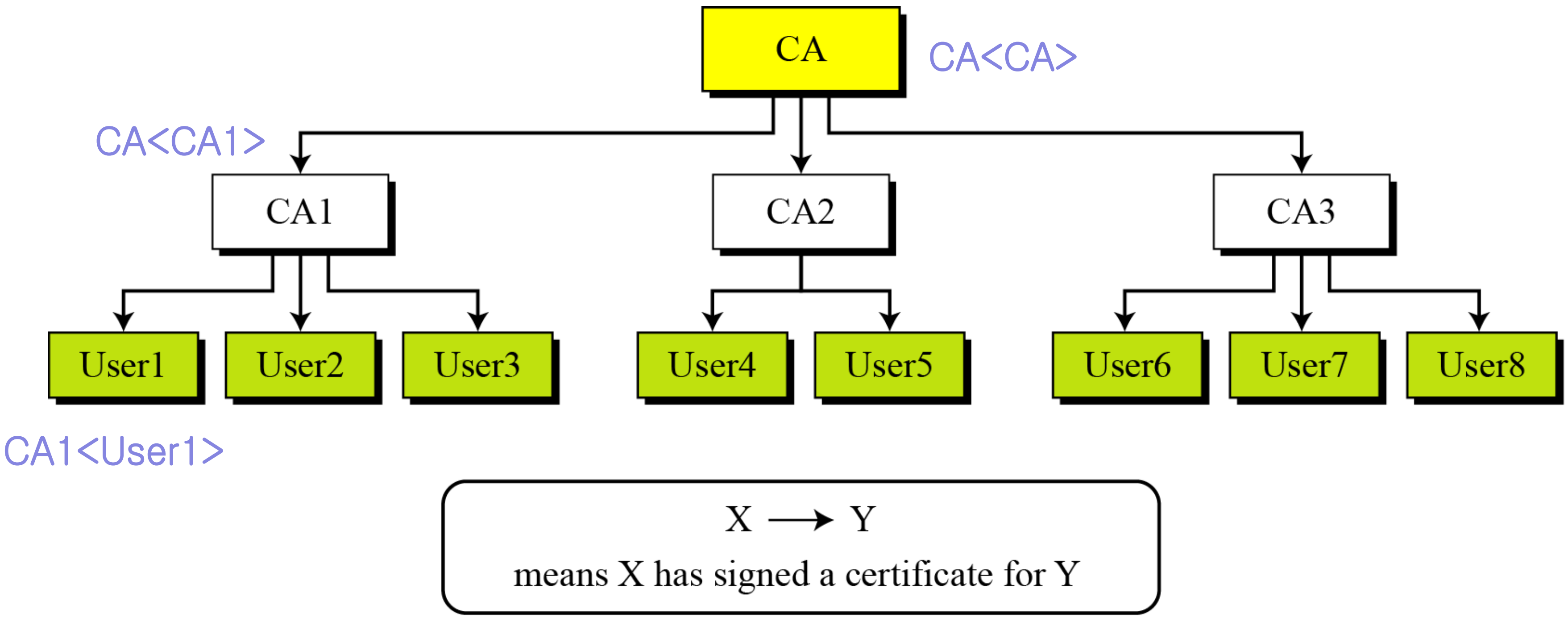
Public-Key Infrastructure (PKI)

- An intra-structure to enable users to get correct public keys of others



Public-Key Infrastructure

- PKI trust model
  - Hierarchical model



Public-Key Infrastructure

- Example
  - Show how can User 1 obtain a verified copy of User 5's public key?
- Verification
  - User 1 has the certificate of root CA: CA<CA>
  - User 5 sends a chain of certificates, CA<CA2> and CA2<User5> to User1

1. User1 validates CA<CA2> using the public key of CA
2. User1 extracts the public key of CA2 from CA<CA2>
3. User1 validates CA2<User5> using the public key of CA2
4. User1 extracts the public key of User 5 from CA2<User5>

Public-Key Infrastructure

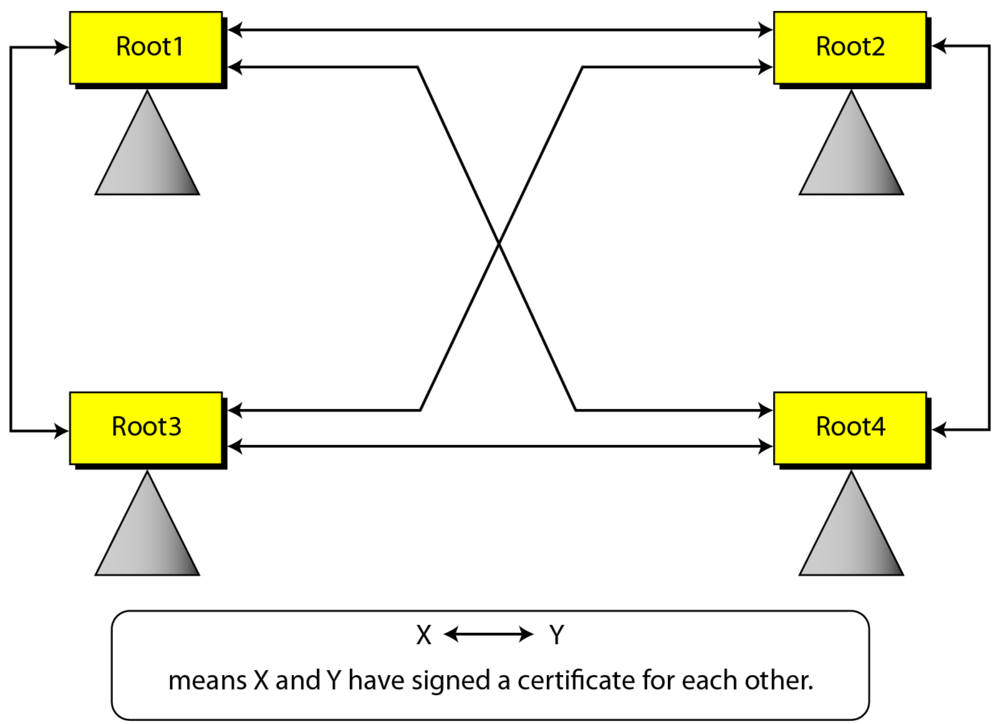
Cross-certification

- Some Web browsers include a set of certificates from multiple independent roots
- Internet Explorer : 제어판/인터넷옵션/내용/인증서/신뢰된 루트 인증기관

Roo1<Root3>

Needs cross-certification among root CAs

Roo3<Root1>



Public-Key Infrastructure

- Example
  - Alice is under the authority Root1; Bob is under the authority Root4
  - How can Alice obtain Bob’s verified public key ?
- Verification
  - Bob sends a chain of certificates from Root4 to Bob to Alice
  - Alice gets Root1<Root4> from Root1

1. Alice validates Roo1<Root4> using the public key of Root1
2. Alice extracts the public key of Root4 from Root1< Root4>
3. Alice validates and extracts public keys step-by-step the chain of certificates from Root4 to Bob using the public key of Root4

Public-Key Infrastructure