CRYPTOGRAPHY AND NETWORK SECURITY

Ref from: Network Security Essentials

by

William Stallings

About This Course

Textbook:

1. Network Security Essentials: Applications and Standards, 3rd Ed. William Stallings
2. Cryptography and Network Security: Principles and Practices, 4th Ed. William Stallings

Contents:

1. Cryptography
   - Algorithms and protocols
     - Conventional and public key-based encryption, hash func, digital signatures, and key exchange
2. Network security applications
   - Applications and tools
     - Kerberos, X.509v3 certificates, PGP, S/MIME, IP security, SSL/TLS, SET, and SNMPv3
3. System security
   - System-level issues
     - Intruders, viruses, worms, DOS

COURSE EDUCATIONAL OBJECTIVES (CEOs)

The main goal of this course is to provide you with a background, foundation, and insight into the many dimensions of information security. This knowledge will serve as basis for further deeper study into selected areas of the field, or as an important component in your further studies and involvement in computing as a whole.

The primary objectives of the course are to help you:

- Understand information security's importance in our increasingly computer-driven world.
- Master the key concepts of information security and how they "work."
- Develop a "security mindset:" learn how to critically analyze situations of computer and network usage from a security perspective, identifying the salient issues, viewpoints, and trade-offs.
- Clearly and coherently communicate (both verbally and in writing) about complex technical topics.

COURSE OUTCOMES (COs)

After completion of the course, the student will be able to:

- Define the concepts and definition of the information security
- Differentiate between several types of Security attacks, Services and Mechanisms
- Identify the threats to information security
- Show how to protect information recourses
- Show how to maintaining and protecting information system

SYLLABUS

- UNIT-I : Introduction, Symmetric Encryption And Message Authentication, and Message Authentication
- UNIT-II: Public Key Cryptography, and Authentication Applications
- UNIT-III: Email privacy and IP Security
- UNIT-IV : Web Security
- UNIT-V : Intruders, Malicious Software, and Firewalls

Tentative Dates for CRT Classes: 18/06/2018 to 30/06/2018

MID-I Examination: 13/08/2018 to 18/08/2018

UNIT-II

Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender

Public-Key Cryptography

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community

Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **infeasible to determine private key from public**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

A public-key encryption scheme has six ingredients (Figure 3.9a).

•Plaintext: This is the readable message or data that is fed into the algorithm as input.

•Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.

•Public and private key: This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input.

• Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

• Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps are the following:

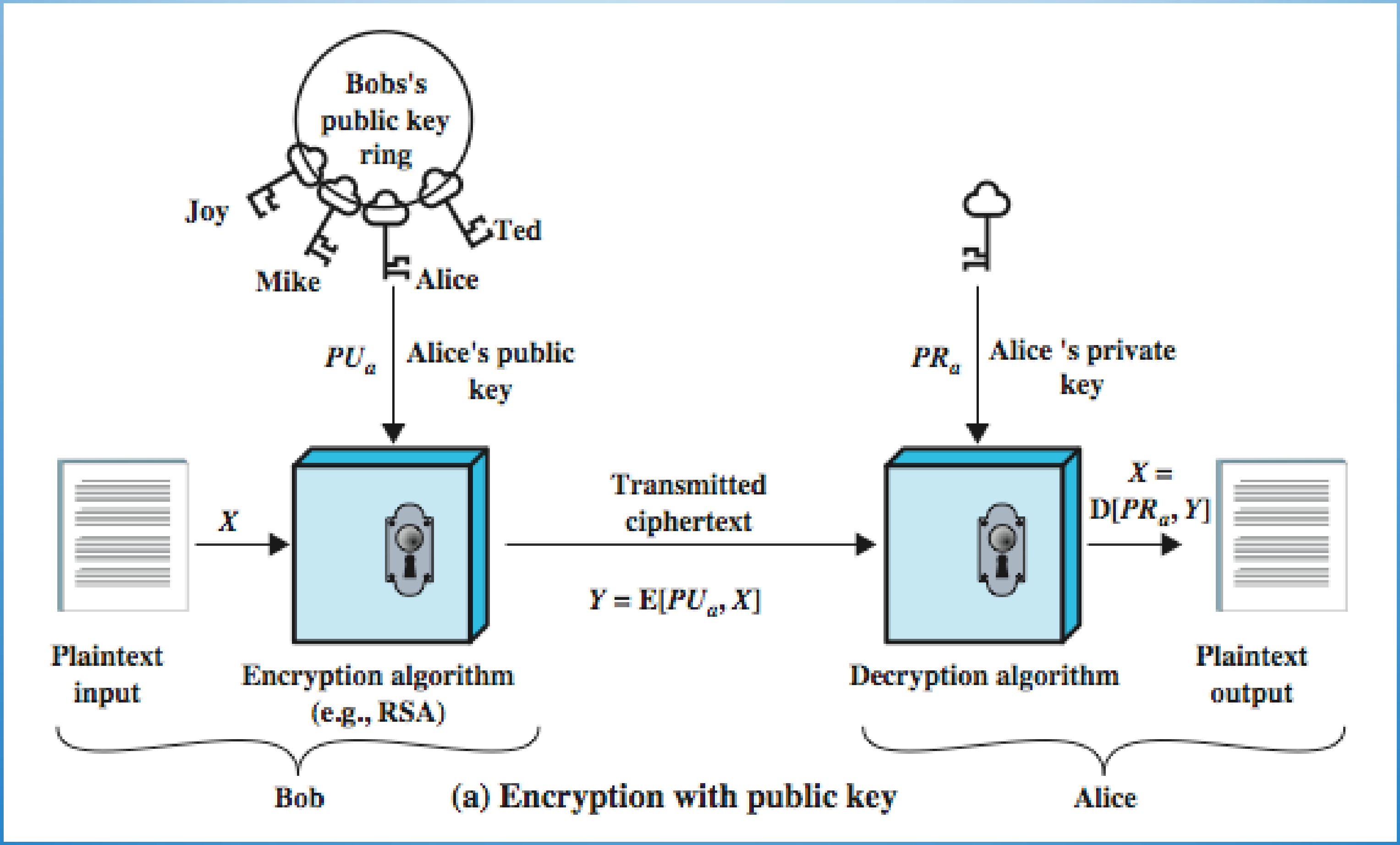
1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.
As Figure 3.9a suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a private message to Alice, Bob encrypts the message using Alice’s public key.

4. When Alice receives the message, she decrypts it using her private key.
No other recipient can decrypt the message because only Alice knows Alice’s private key.

Public-Key Cryptography



(a) Encryption with public key

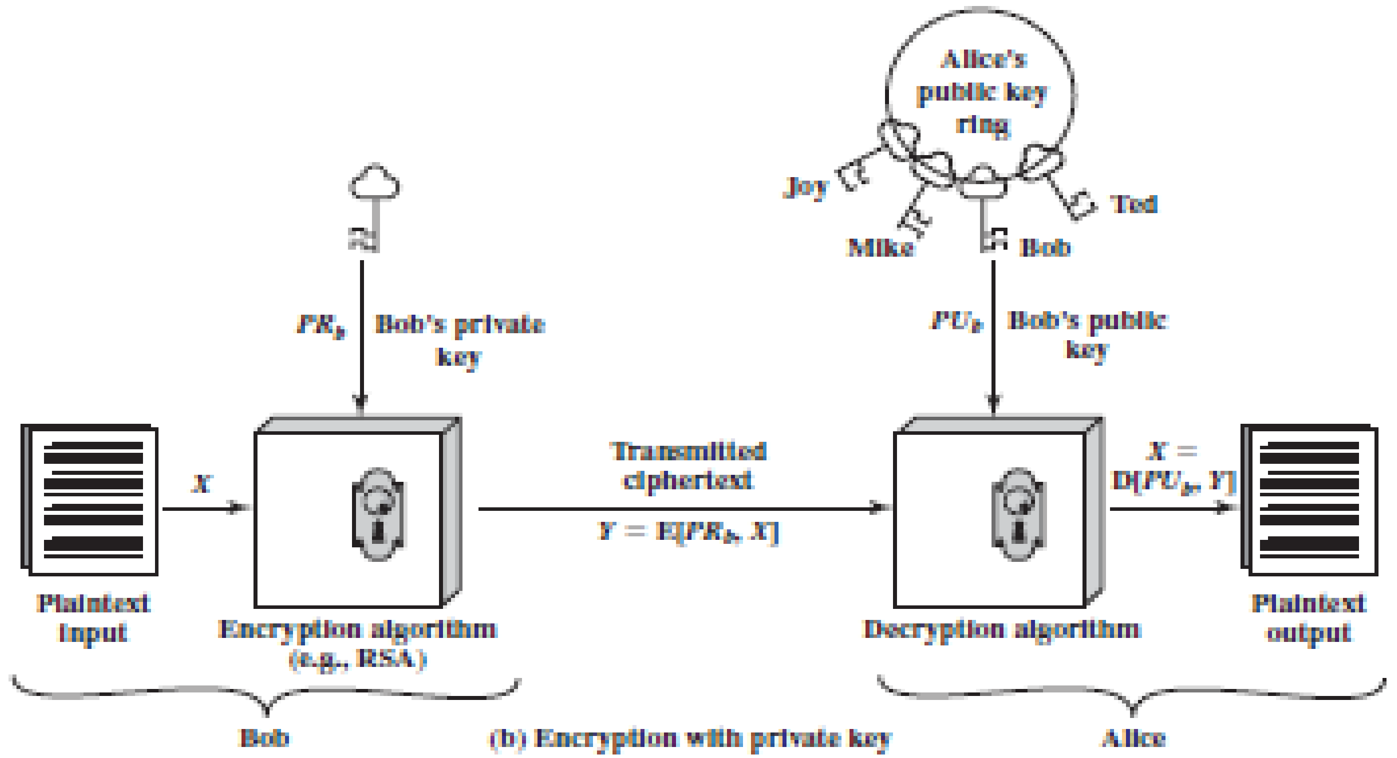
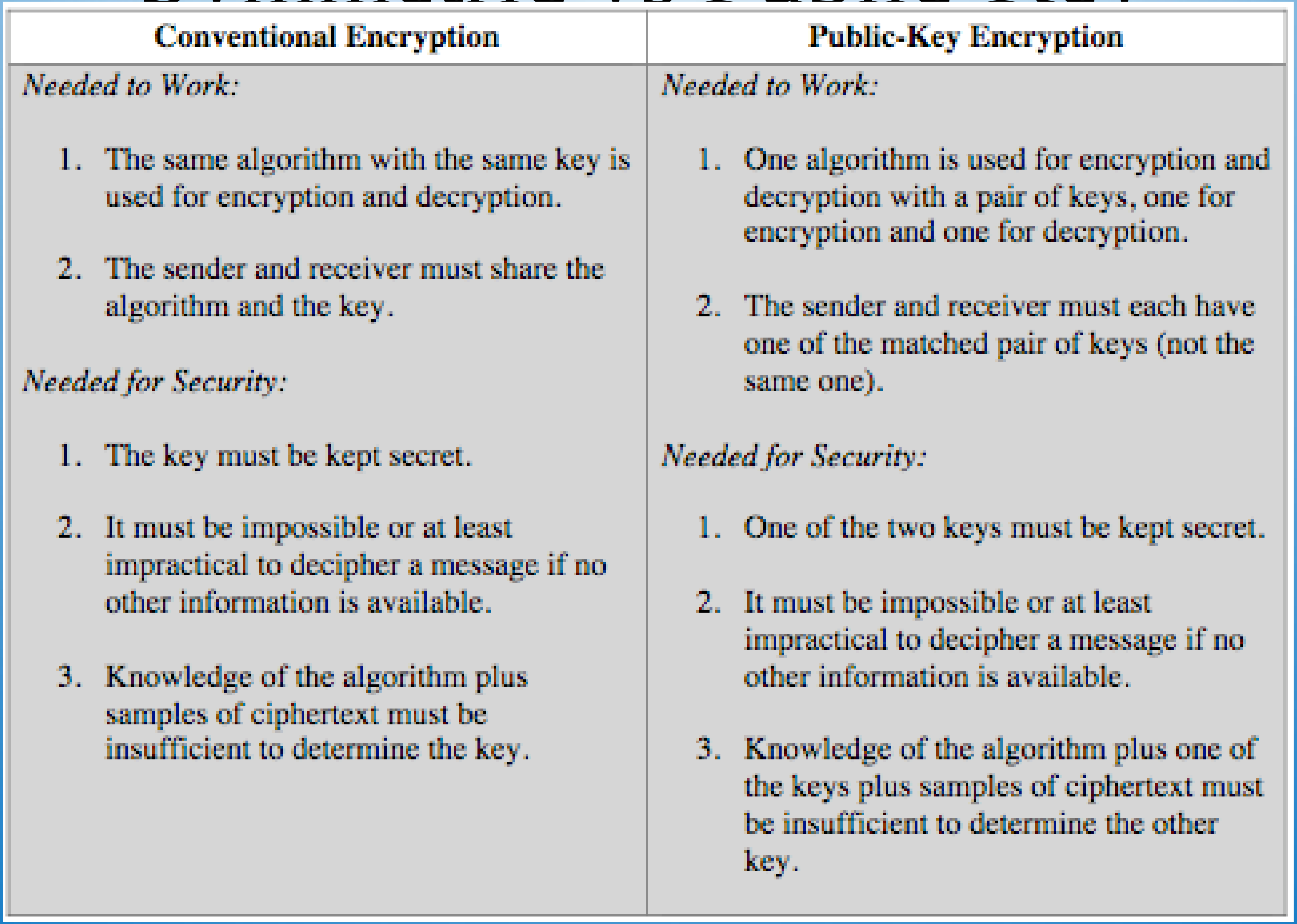


Figure 3.9 Public-Key Cryptography

Symmetric vs Public-Key



Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Applications for Public-Key Cryptosystems

• Encryption/decryption: The sender encrypts a message with the recipient’s public key.

•Digital signature: The sender “signs” a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.

• Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

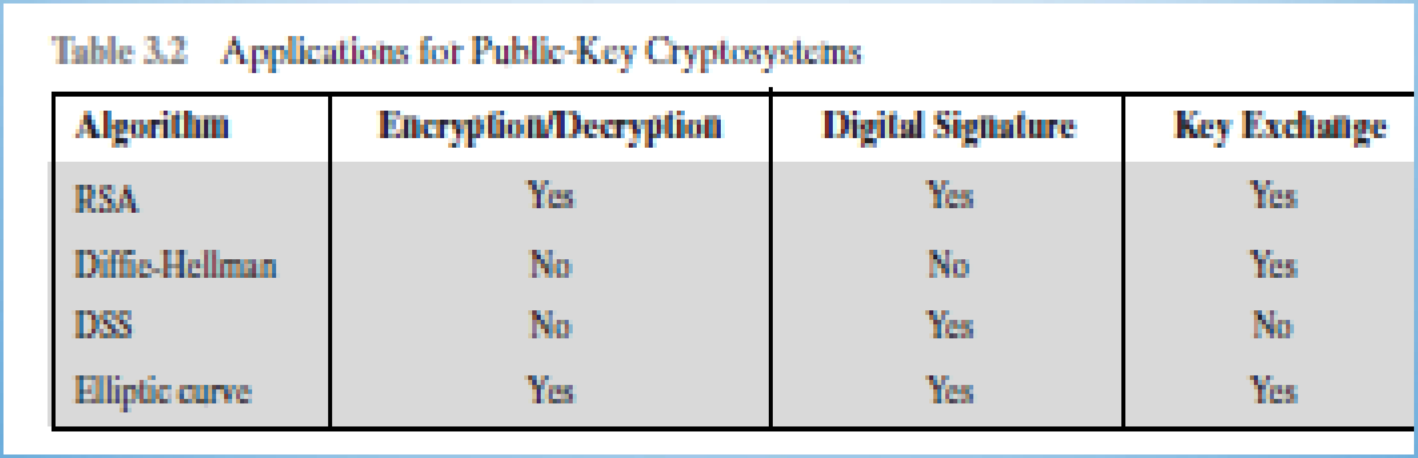


Table 3.2 Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No
Elliptic curve	Yes	Yes	Yes

Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figure 3.9 depends on a cryptographic algorithm based on two related keys.

1. It is computationally easy for a party B to generate a pair (public key *Pub*,private key *PRb)*.

2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted,*M, to generate the corresponding ciphertext: C E(PUb,M)*

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message: *M D(PRb, C) D[PRb, E(PUb,M)]*

4. It is computationally infeasible for an opponent, knowing the public key,*PUb*, *to* determine the private key, *PRb*.

5. It is computationally infeasible for an opponent, knowing the public key, *PUb*, and a ciphertext, *C*, *to recover the original message*, *M*.

We can add a sixth requirement that, although useful, is not necessary for all public-key applications.

6. Either of the two related keys can be used for encryption, with the other used
for decryption. $M = D[PUb, E(PRb,M)] = D[PRb, E(PUb,M)]$

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA En/decryption

- to encrypt a message M the sender:
  - obtains **public key** of recipient `PU={e,n}`
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key `PR={d,n}`
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random: `p, q`
- computing their system modulus `n=p.q`
  - note `ø(n)=(p-1)(q-1)`
- selecting at random the encryption key `e`
  - where `1<e<ø(n), gcd(e,ø(n))=1`
- solve following equation to find decryption key `d`
  - `e.d=1 mod ø(n) and 0≤d≤n`
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

Why RSA Works

- because of Euler's Theorem:
  - $a^{\varnothing(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\varnothing(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \varnothing(n)$
  - hence $e.d=1+k.\varnothing(n)$ for some k
- hence :

$$C^{d} = M^{e.d} = M^{1+k.\varnothing(n)} = M^{1}.(M^{\varnothing(n)})^{k}$$

$$= M^{1}.(1)^{k} = M^{1} = M \bmod n$$

RSA Example - Key Setup

1. Select primes: p=17 & q=11
2. Calculate $n = pq$ =17 x 11=187
3. Calculate $ø(n)=(p-1)(q-1)$=16x10=160
4. Select e: gcd(e,160)=1; choose e=7
5. Determine d: de=1 mod 160 and d < 160
   Value is d=23 since 23x7=161= 10x160+1
6. Publish public key PU={7,187}
7. Keep secret private key PR={23,187}

RSA Example - En/Decryption

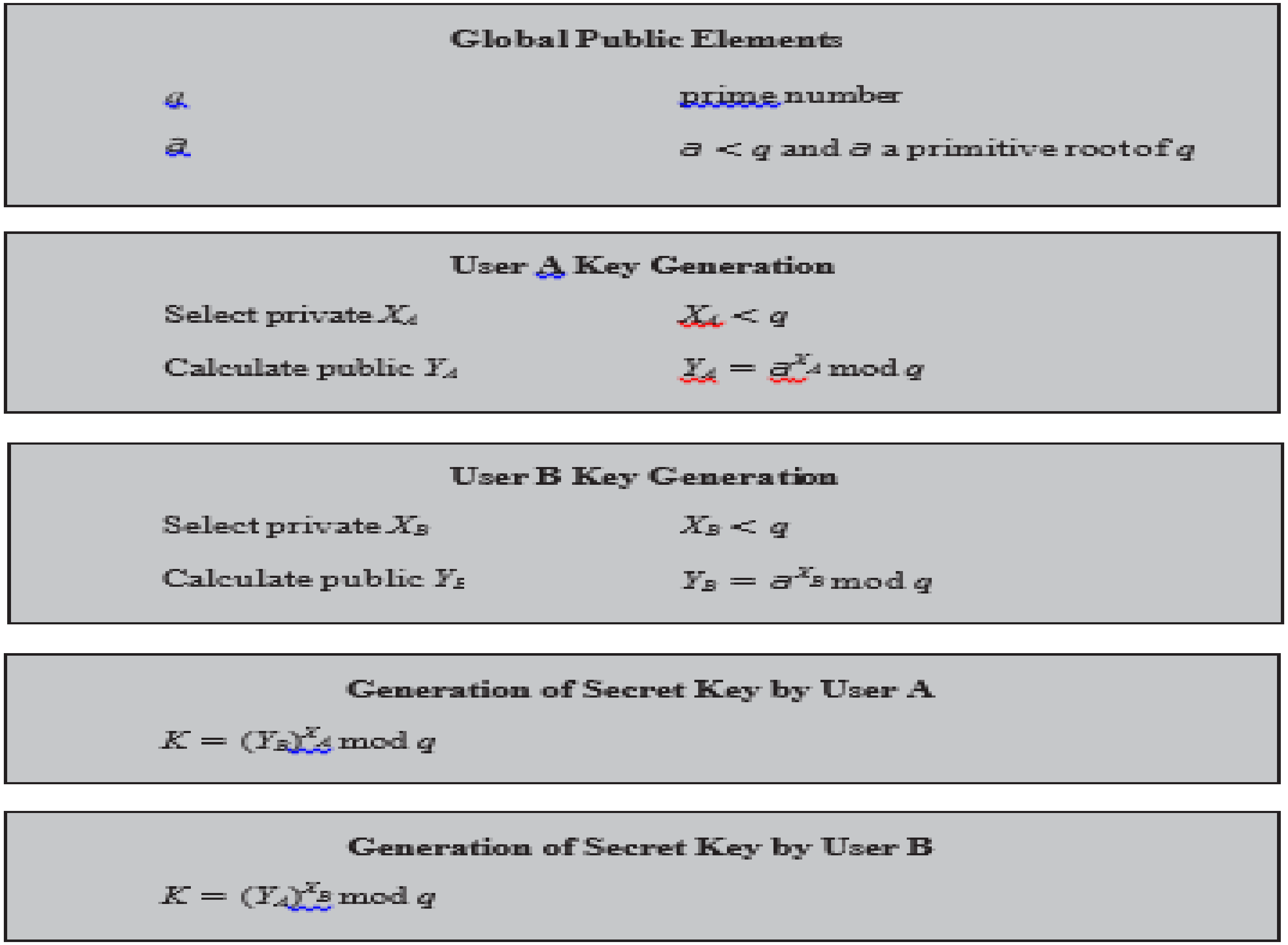
- sample RSA encryption/decryption is:
- given message `M = 88` (nb. `88<187`)
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

  $M = 11^{23} \bmod 187 = 88$

Diffie-Hellman Key Exchange

- The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76] and is generally referred to as the **Diffie-Hellman key exchange**.
- A number of commercial products employ this key exchange technique.
- Purpose- The purpose of the algorithm is to enable two users to exchange a secret key securely that then can be used for subsequent encryption of messages.
- The algorithm itself is limited to the exchange of the keys.
- The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.



Global Public Elements

q	prime number
a	$a < q$ and a a primitive root of q

User A Key Generation

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

User B Key Generation

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

Generation of Secret Key by User A

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

Generation of Secret Key by User B

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

Figure 3.12 The Diffie-Hellman Key Exchange Algorithm

Example

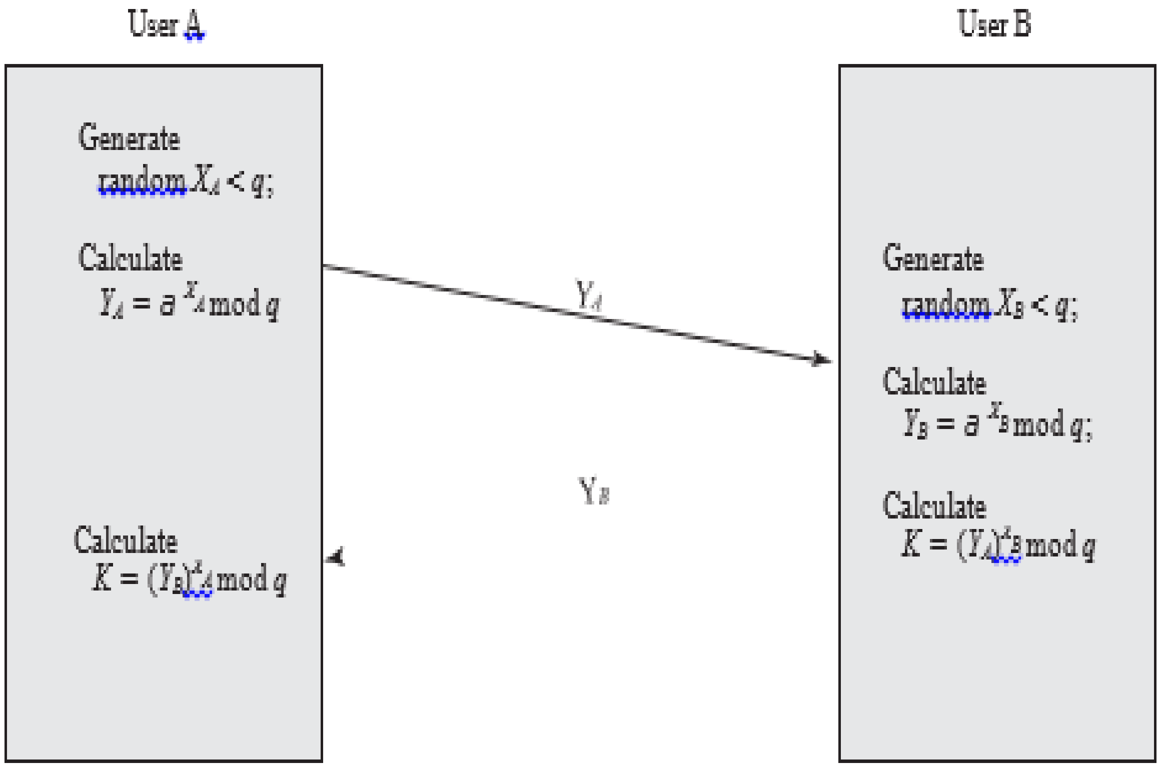
- Key exchange is based on the use of the prime number q = 353 and a primitive root of 353, in this case a = 3.
- A and B select secret keys X_A = 97 and X_B = 233, respectively. Each computes its public key:
- A computes Y_A = 397 mod 353 = 40.
- B computes Y_B = 3233 mod 353 = 248.
- After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)XA$ mod 353 = 24897 mod 353 = 160.

B computes $K = (Y_A)XB$ mod 353 = 40233 mod 353 = 160.

- We assume an attacker would have available the following information:

q = 353; a = 3; Y_A = 40; Y_B = 248



User A
Generate random $X_A < q$;
Calculate $Y_A = a^{X_A} \bmod q$
Calculate $K = (Y_B)^{X_A} \bmod q$
Y_A
Y_B
User B
Generate random $X_B < q$;
Calculate $Y_B = a^{X_B} \bmod q$;
Calculate $K = (Y_A)^{X_B} \bmod q$

Figure 3.13 Diffie-Hellman Key Exchange

MAN IN THE MIDDLE ATTACK

The protocol depicted in Figure 3.13 is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

- Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2}, and then computing the corresponding public keys Y_{D1} and Y_{D2}.
- Alice transmits Y_A to Bob.
- Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates

$$K2 = (Y_A)^{\wedge}XD2 \mod q.$$

- Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \mod q$.
- Bob transmits Y_B to Alice.
- Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates

$$K1 = (Y_B)^{\wedge}XD1 \mod q.$$

- Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \mod q$.

- Bob and Alice think that they share a secret key.
- Instead Bob and Darth share secret key *K*1, and Alice and Darth share secret key *K*2.
- All future communication between Bob and Alice is compromised in the following way:
  - Alice sends an encrypted message *M*: E(*K*2, *M*).
  - Darth intercepts the encrypted message and decrypts it to recover *M*.
  - Darth sends Bob E(*K*1, *M*) or E(*K*1, *M*'), where *M*' is any message.
- In the first case, Darth simply wants to eavesdrop on the communication without altering it.
- In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants.

This vulnerability can be overcome with the use of digital signatures and public-key certificates

Other Public-Key Cryptography Algorithms

- ***DIGITAL SIGNATURE STANDARD* :** The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS PUB 186, known as the **Digital Signature Standard (DSS)**.
- The DSS makes use of the SHA-1 and presents a new digital signature technique, the Digital Signature Algorithm (DSA).
- The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme.
- There was a further minor revision in 1996.
- The DSS uses an algorithm that is designed to provide only the digital signature function.
- Unlike RSA, it cannot be used for encryption or key exchange.

- ***E****LLIPTIC*-***C****URVE* ***C****RYPTOGRAPHY:* The vast majority of the products and standards that use public-key cryptography for encryption and digital signatures use RSA.
- The bit length for secure RSA use has increased over recent years, and this has put a heavier processing load on applications using RSA.
- This burden has ramifications, especially for electronic commerce sites that conduct large numbers of secure transactions.
- Recently, a competing system has begun to challenge RSA: **elliptic curve cryptography (ECC)**.
- Already, ECC is showing up in standardization efforts, including the IEEE P1363 Standard for Public-Key Cryptography.
- The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead.

DIGITAL SIGNATURES

- Public-key encryption can be used in another way, as illustrated in Figure 3.9b.
- Suppose that Bob wants to send a message to Alice, and although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him.
- In this case, Bob uses his own private key to encrypt the message.
- When Alice receives the ciphertext, she finds that she can decrypt it with Bob’s public key, thus proving that the message must have been encrypted by Bob.
- No one else has Bob’s private key, and therefore no one else could have created a ciphertext that could be decrypted with Bob’s public key.

- Therefore, the entire encrypted message serves as a **digital signature**. In addition, it is impossible to alter the message without access to Bob's private key, so the message is authenticated both in terms of source and in terms of data integrity.
- In the preceding scheme, the entire message is encrypted. Although validating both author and contents, this requires a great deal of storage.
- Each document must be kept in plaintext to be used for practical purposes.
- A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute.
- A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document.
- Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator.
- If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing.
- A secure hash code such as SHA-1 can serve this function. Figure 3.2b illustrates this scenario.

- It is important to emphasize that the encryption process just described does not provide confidentiality.
- That is, the message being sent is safe from alteration but not safe from eavesdropping.
- This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear.
- Even in the case of complete encryption, there is no protection of confidentiality because any observer can decrypt the message by using the sender’s public key.

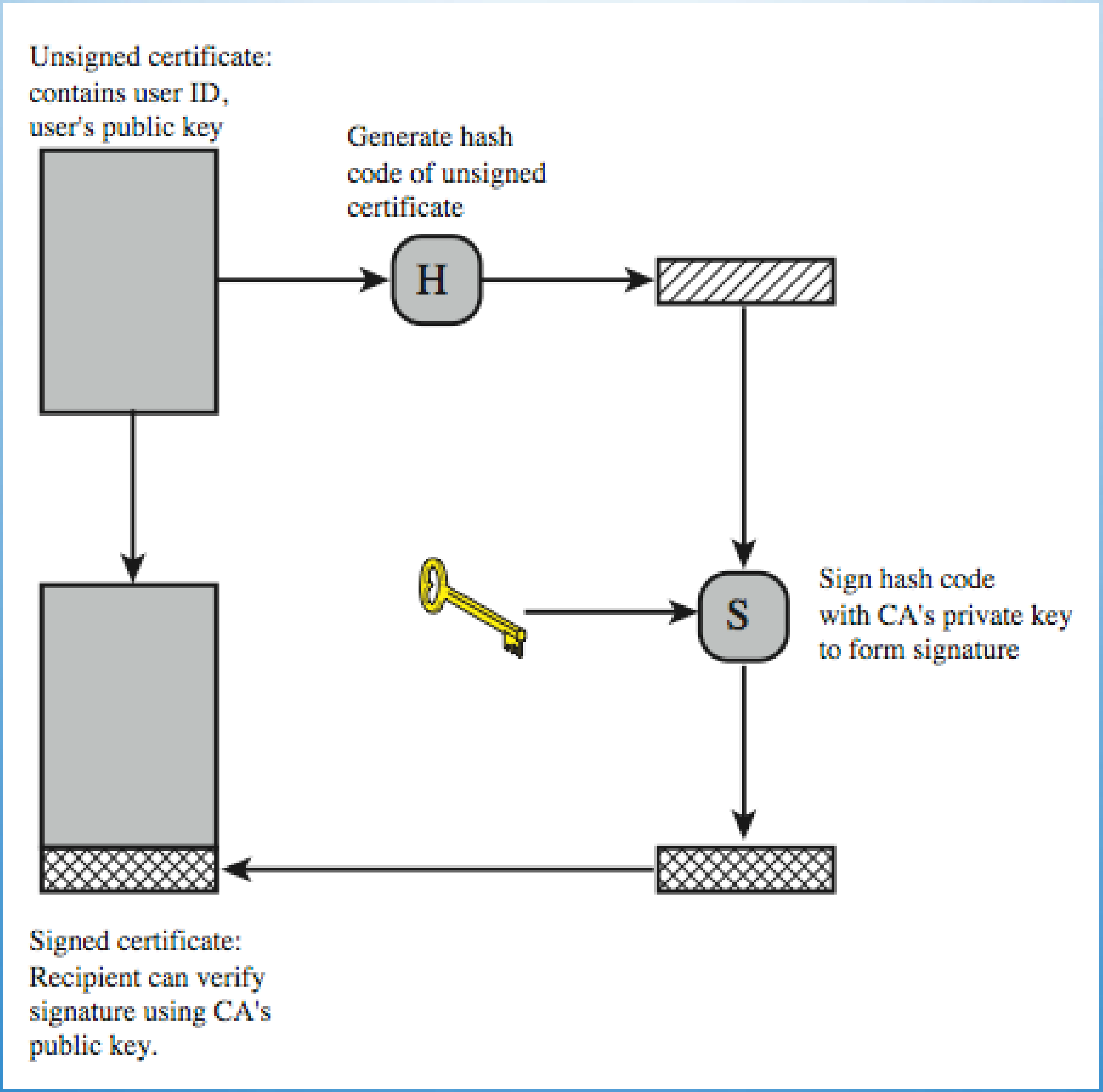
KEY MANAGEMENT

There are two aspects to the use of public key encryption in this target:

- The distribution of Public Keys
- The use of Public key encryption to distribute secret keys.

The distribution of Public Keys (Digital Certificates)

- The weakness in Public key sharing- Any one can forge such a public announcement. Some user could pretend to be user A and send a public key to another participant or broadcast such a public key.
- Until such time as user A discovers the forgery and alerts other participants the forger is able to read all encrypted messages intented for A.
- Solution- PUBLIC KEY CERTIFICATES
- A certificate consists of a Publickey plus a user Id of the Key owner with the whole block signed by a trusted third party called Certificate Authority.
- It is trusted by user community such as Government Agency.
- A user present his or her public key to the authority in a secure manner and obtain a Certificate.


Unsigned certificate:
contains user ID,
user's public key
Generate hash
code of unsigned
certificate
H
S
Sign hash code
with CA's private key
to form signature
Signed certificate:
Recipient can verify
signature using CA's
public key.

KERBEROS

- Kerberos is a key distribution and user authentication service developed at MIT.

- The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network.

- We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service.

- In this environment, a workstation cannot be trusted to identify its users correctly to network services.

- In particular, the following three threats exist:

KERBEROS

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

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

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

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access.

Kerberos

- Rather than building elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users.
- relies exclusively on symmetric encryption, making no use of public-key encryption.
  - ➢ allows users access to services distributed through network
  - ➢ without needing to trust all workstations
  - ➢ 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

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.
- An opponent can pretend to be another client and obtain unauthorized privileges on server machines.
- To counter this threat, servers must be able to confirm the identities of clients who request service.
- Each server can be required to undertake this task for each client/server interaction, 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.

- These keys have been distributed physically or in some other secure manner.

- Consider the following hypothetical dialogue:1

(1) $C \rightarrow AS$: $ID_C \| P_C \| ID_V$

(2) $AS \rightarrow C$: $Ticket$

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

$Ticket = E(K_v, [ID_C \| AD_C \| ID_V])$

where

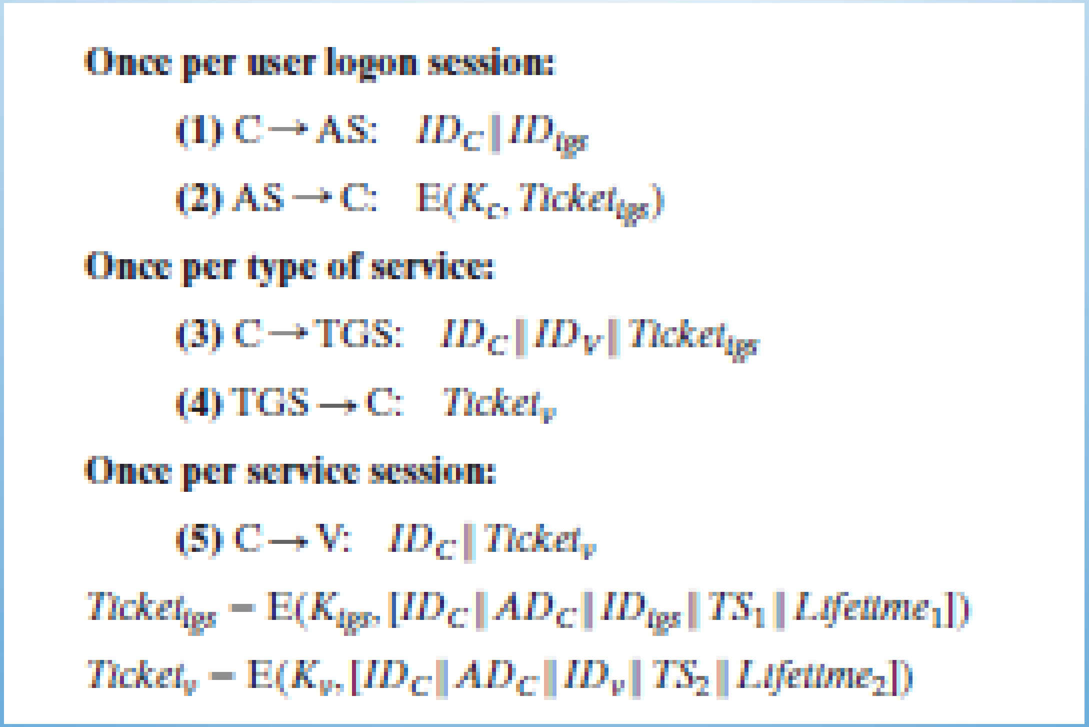
C	= client
AS	= authentication server
V	= server
ID_C	= identifier of user on C
ID_V	= identifier of V
P_C	= password of user on C
AD_C	= network address of C
K_v	= secret encryption key shared by AS and V

A MORE SECURE AUTHENTICATION DIALOGUE

- However, under this scheme, it remains the case that a user would need a new ticket for every different service.

- If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

- The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.
- To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the **ticket-granting server (TGS).**



Once per user logon session:

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

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

Once per type of service:

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

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

Once per service session:

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

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

$Ticket_v = E(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$

➢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 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 (*KC), which is already stored at the AS.*
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.

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 using a key shared only by the AS and the TGS (*Ktgs) 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.

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.

Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain.

The heart of the first problem is the lifetime associated with the ticket-granting ticket.

If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users.

Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location.

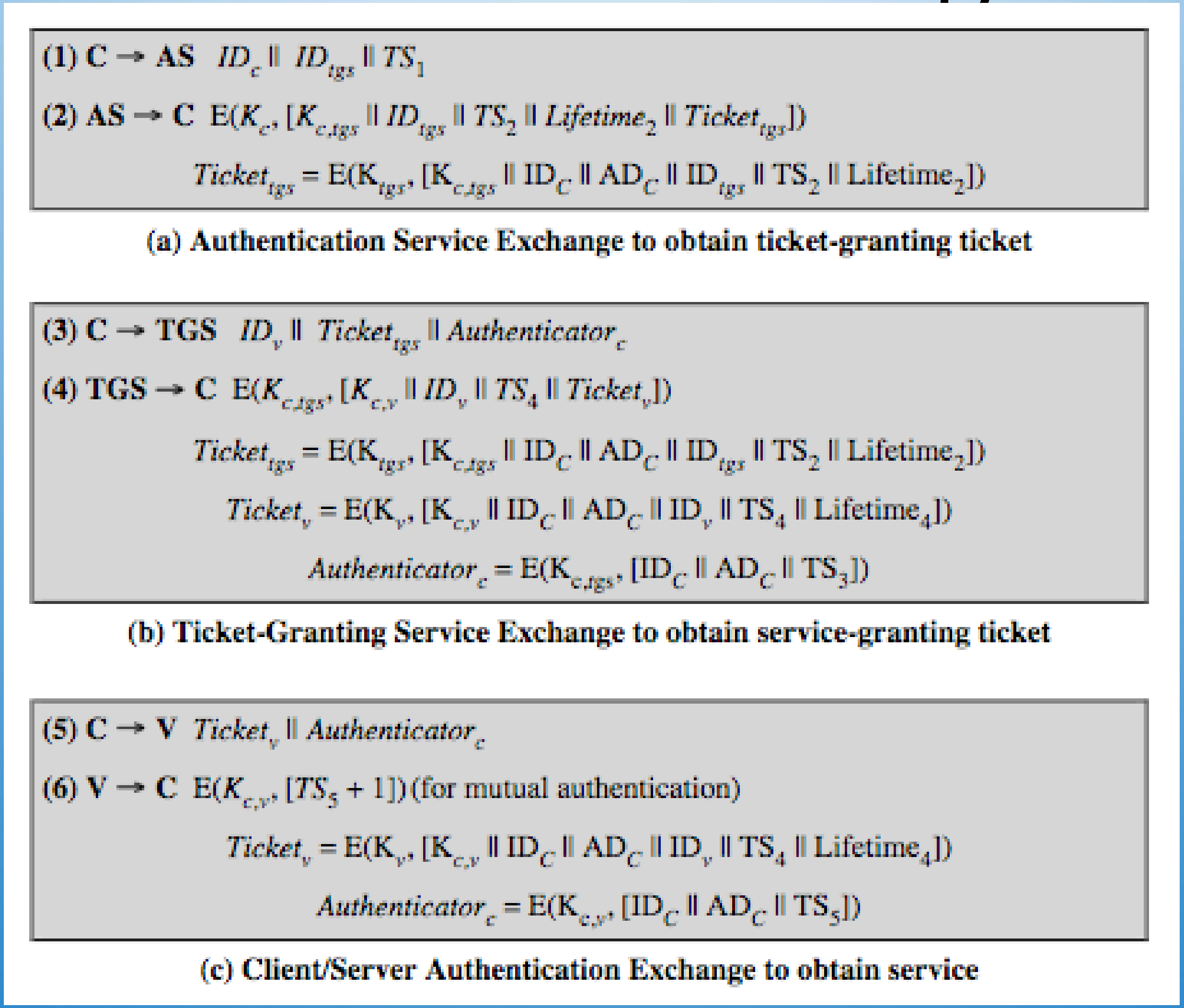
The false server then would be in a position to act as a real server, capture any information from the user, and deny the true service to the user.

We examine these problems in turn and refer to Table 4.1, which shows the actual Kerberos protocol.

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
- using a complex protocol using DES

Kerberos v4 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

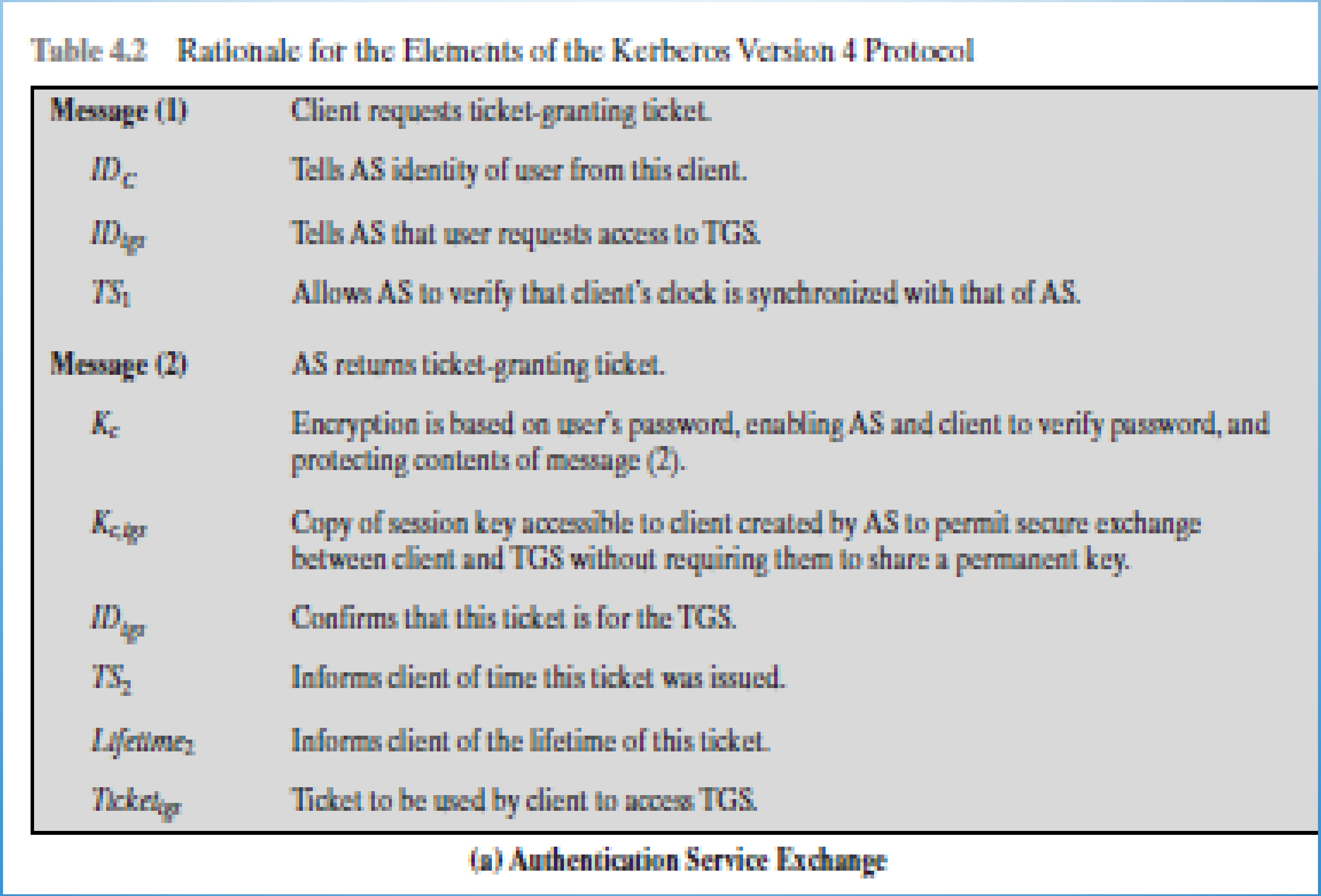
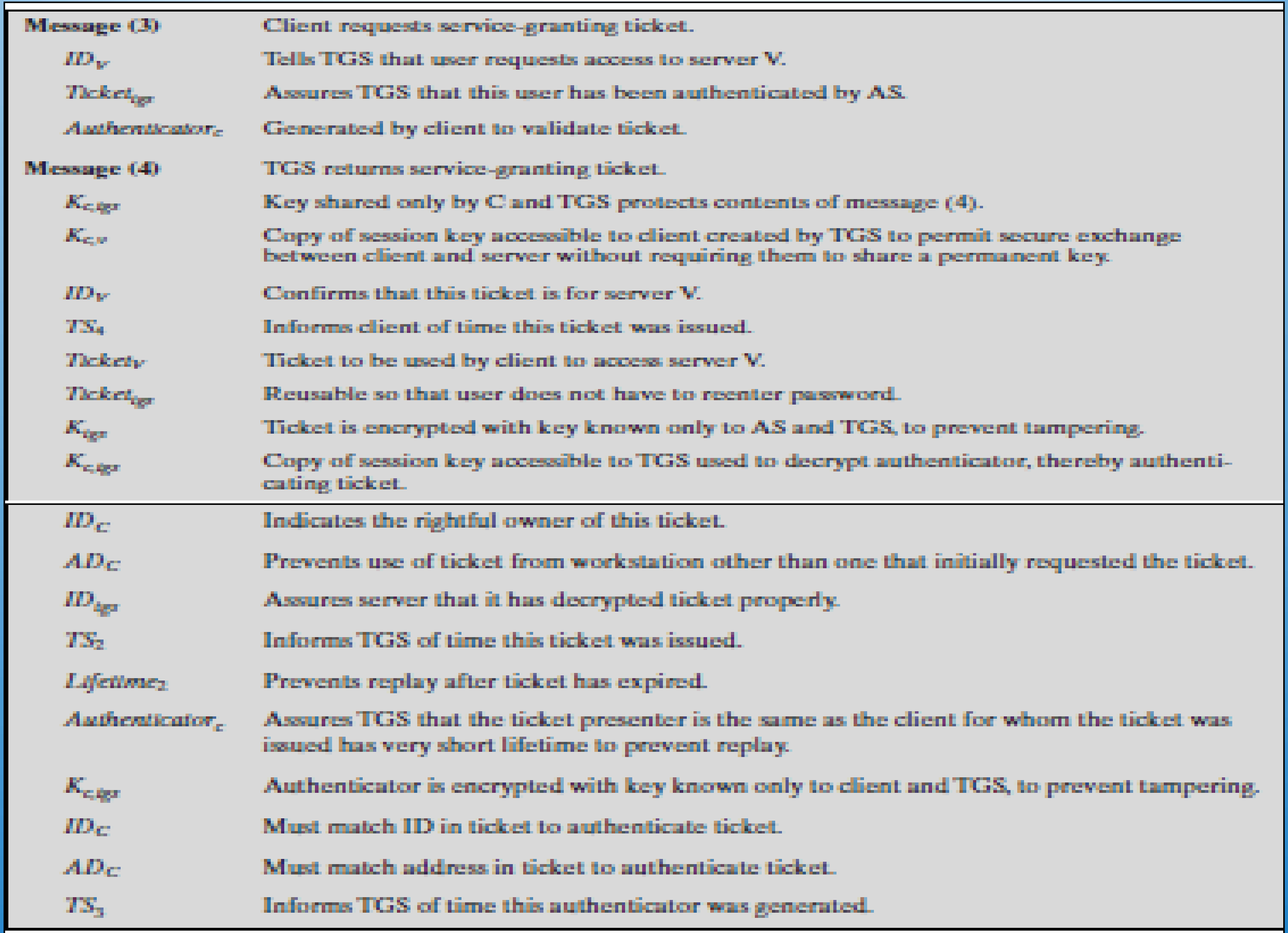


Table 4.2 Rationale for the Elements of the Kerberos Version 4 Protocol

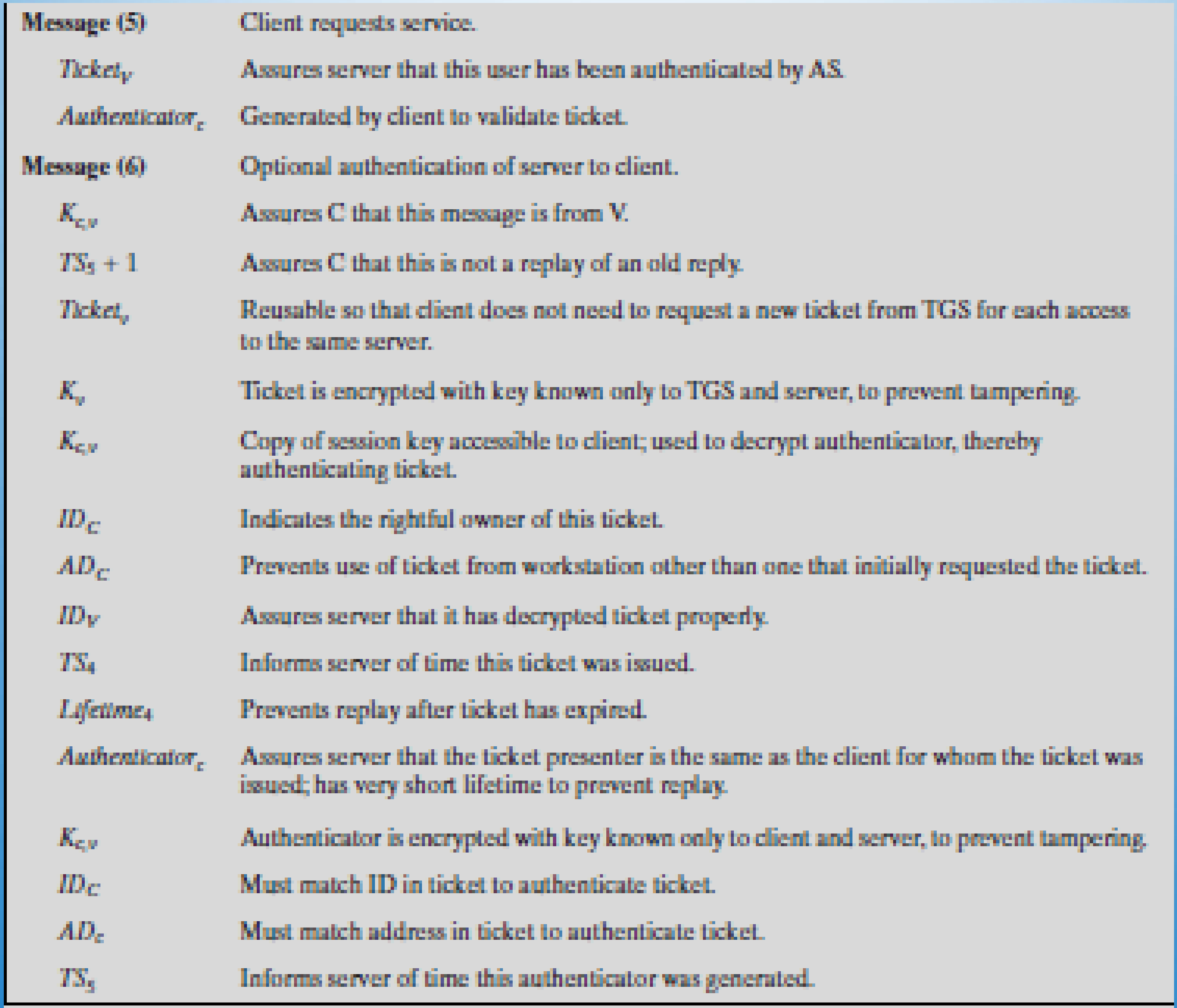
Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user requests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2).
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(a) Authentication Service Exchange



Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

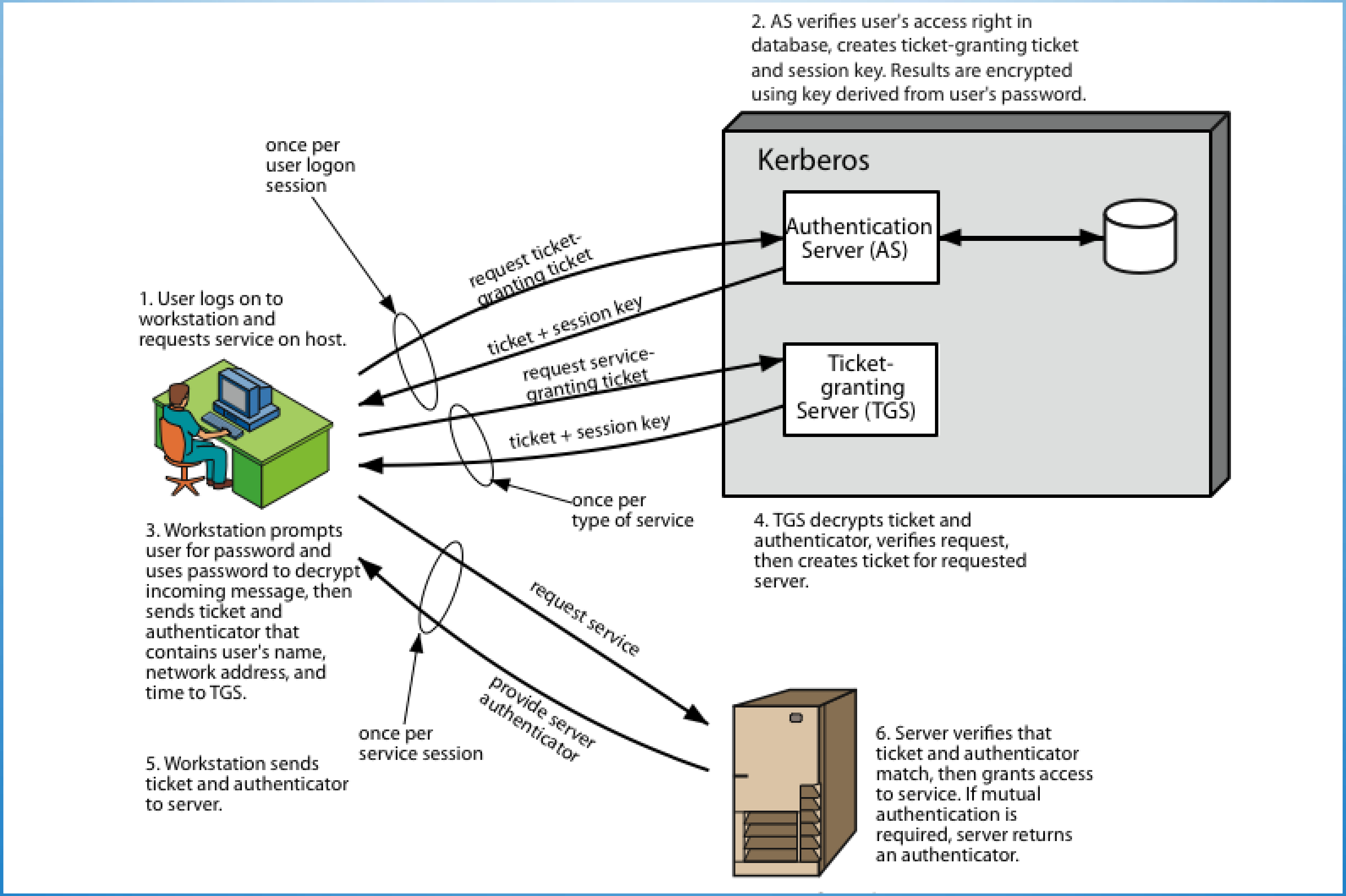
(b) Ticket-Granting Service Exchange



Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.
TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_c	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

(c) Client/Server Authentication Exchange

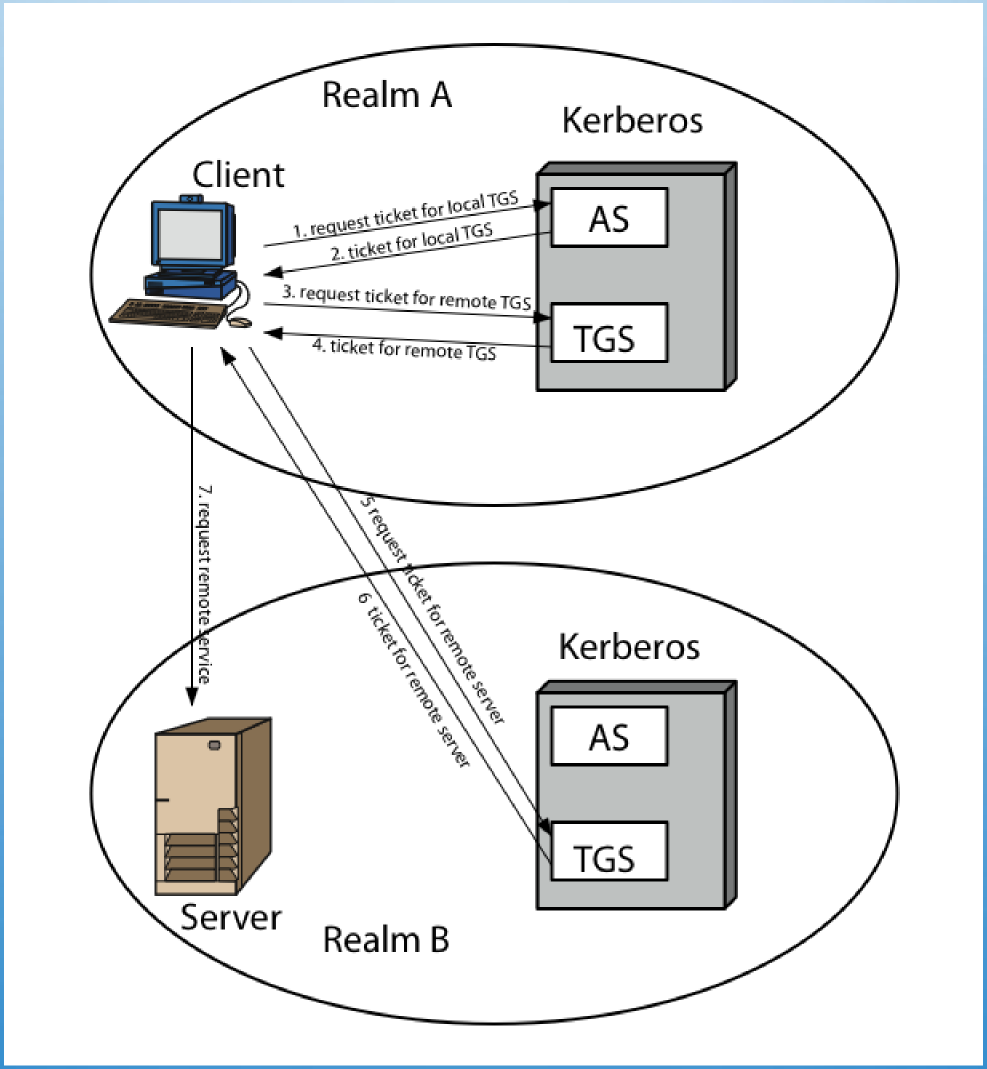
Kerberos 4 Overview


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

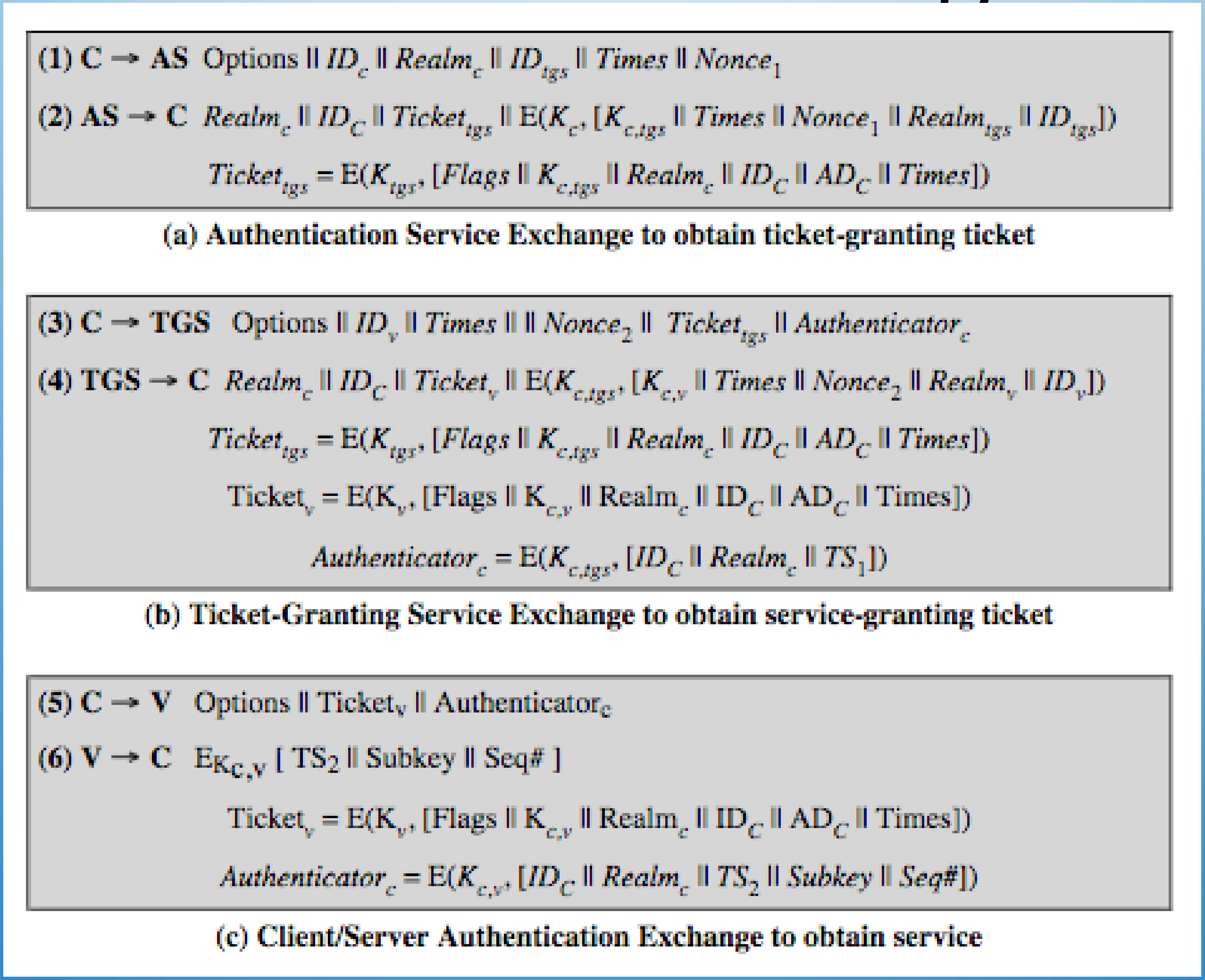
Kerberos Realms


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

Kerberos v5 Dialogue



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

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

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

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

(3) **C → TGS** Options $\| ID_v \| Times \| \| Nonce_2 \| \; Ticket_{tgs} \| Authenticator_c$

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

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

$\mathrm{Ticket}_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{Flags} \| \mathrm{K}_{c,v} \| \mathrm{Realm}_c \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{Times}])$

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

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

(5) **C → V** Options $\|$ Ticket_v $\|$ $\mathrm{Authenticator}_c$

(6) **V → C** $\mathrm{E}_{K_{c,v}}$ [TS_2 $\|$ Subkey $\|$ Seq#]

$\mathrm{Ticket}_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{Flags} \| \mathrm{K}_{c,v} \| \mathrm{Realm}_c \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{Times}])$

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

(c) Client/Server Authentication Exchange to obtain service

X.509 Certificates

- ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service.

- The directory is, in effect, a server or distributed set of servers that maintains a database of information about users.

- The information includes a mapping from user name to network address, as well as other attributes and information about the users.

- X.509 defines a framework for the provision of authentication services by the X.500 directory to its users.

- The directory may serve as a repository 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.

- In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 Certificates

- ➢X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90]

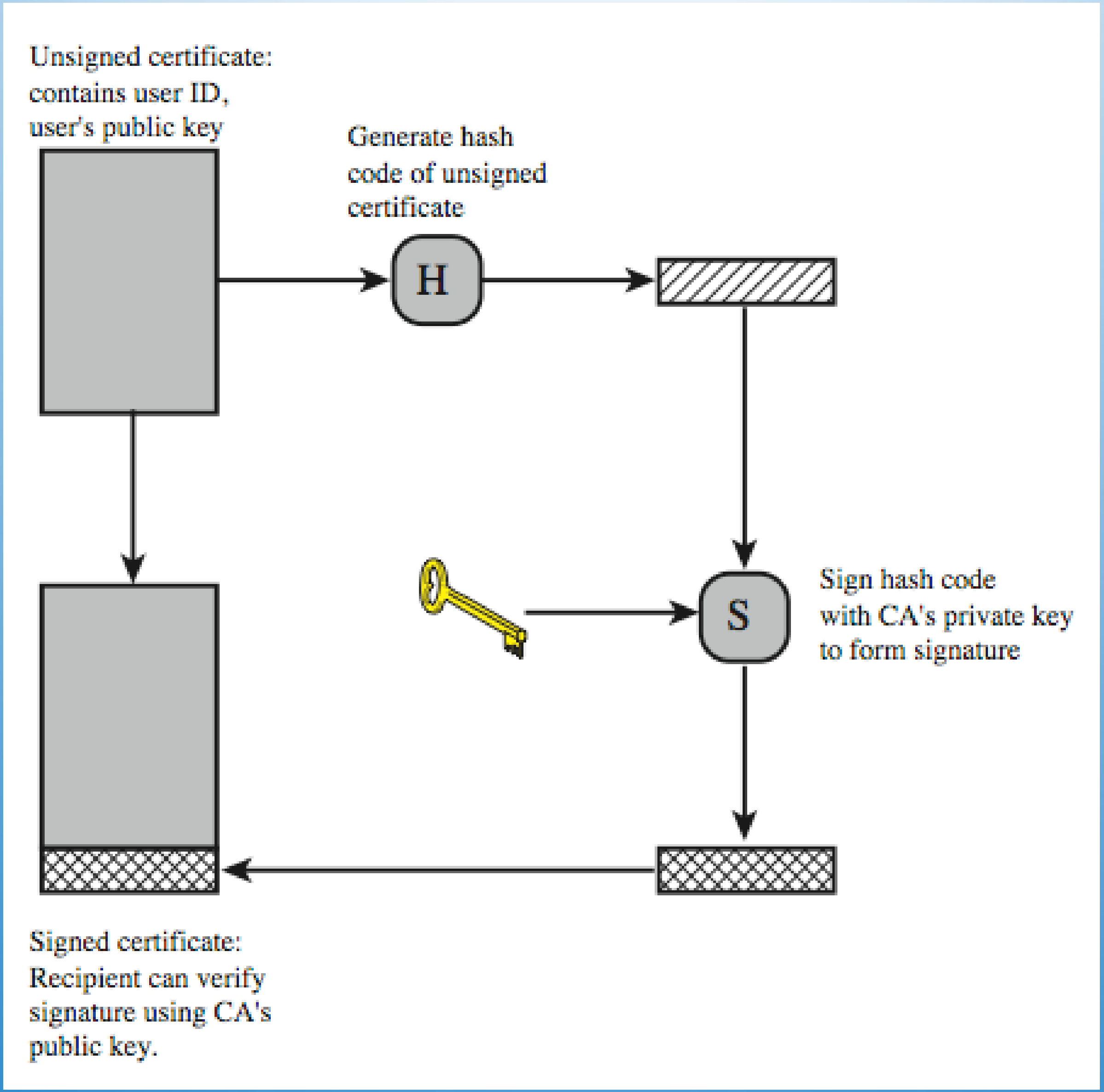
- ➢A revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

- ➢X.509 is based on the use of public-key cryptography and digital signatures.

- ➢The standard does not dictate the use of a specific algorithm but recommends RSA.

- ➢The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm.

- ➢The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation.
- ➢Figure illustrates the generation of a public-key certificate.


Unsigned certificate:
contains user ID,
user's public key
Generate hash
code of unsigned
certificate
H
S
Sign hash code
with CA's private key
to form signature
Signed certificate:
Recipient can verify
signature using CA's
public key.

- The heart of the X.509 scheme is the public-key certificate associated with each user.

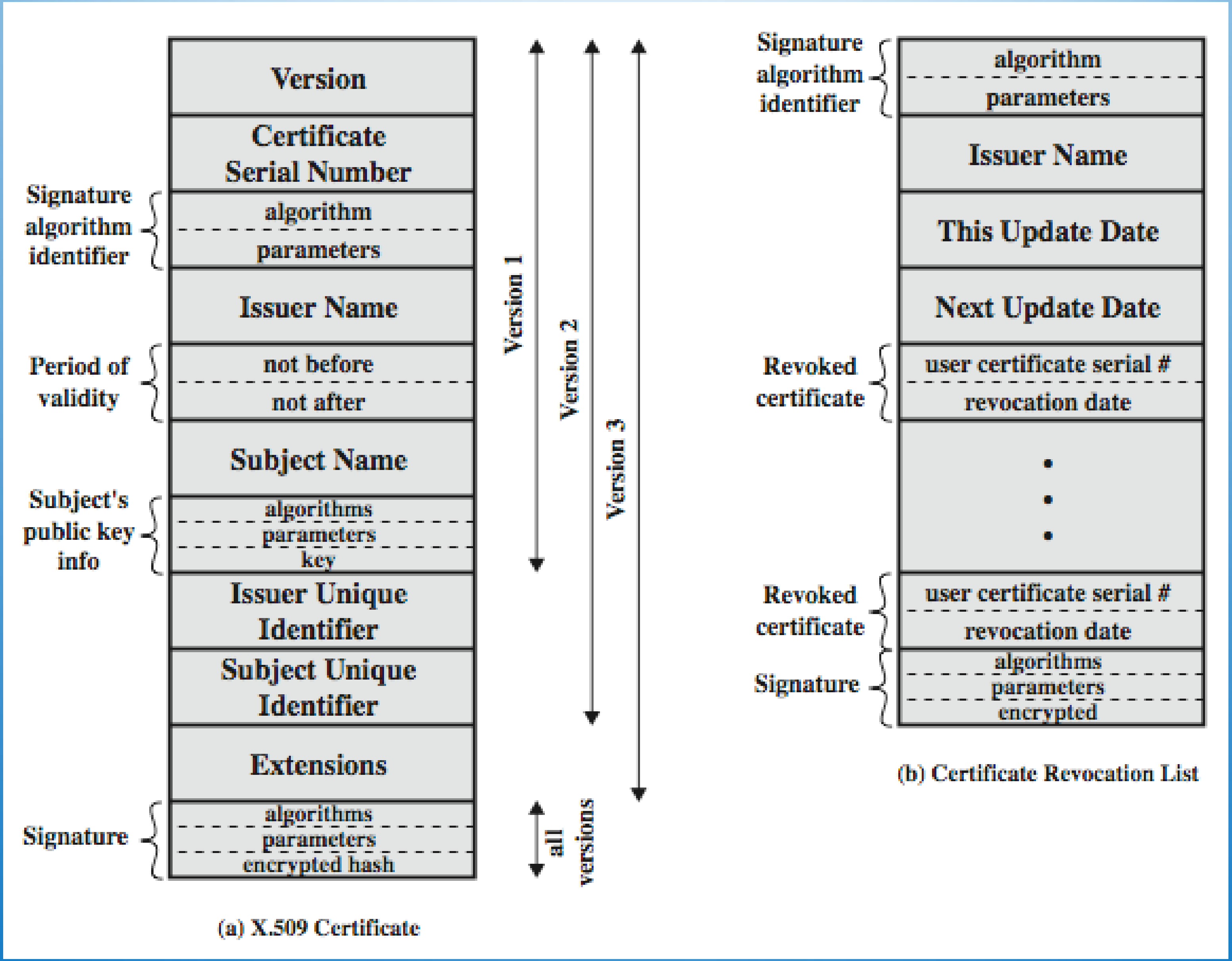
- These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user.

- The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version V (1, 2, or 3)
  - serial number SN (unique within CA) identifying certificate
  - signature algorithm identifier AI
  - issuer X.500 name CA)
  - period of validity TA (from - to dates)
  - subject X.500 name A (name of owner)
  - subject public-key info Ap (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



Version: Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

Serial number: An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.

Signature algorithm identifier: The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.

Issuer name: X.500 name of the CA that created and signed this certificate.

Period of validity: Consists of two dates: the first and last on which the certificate is valid.

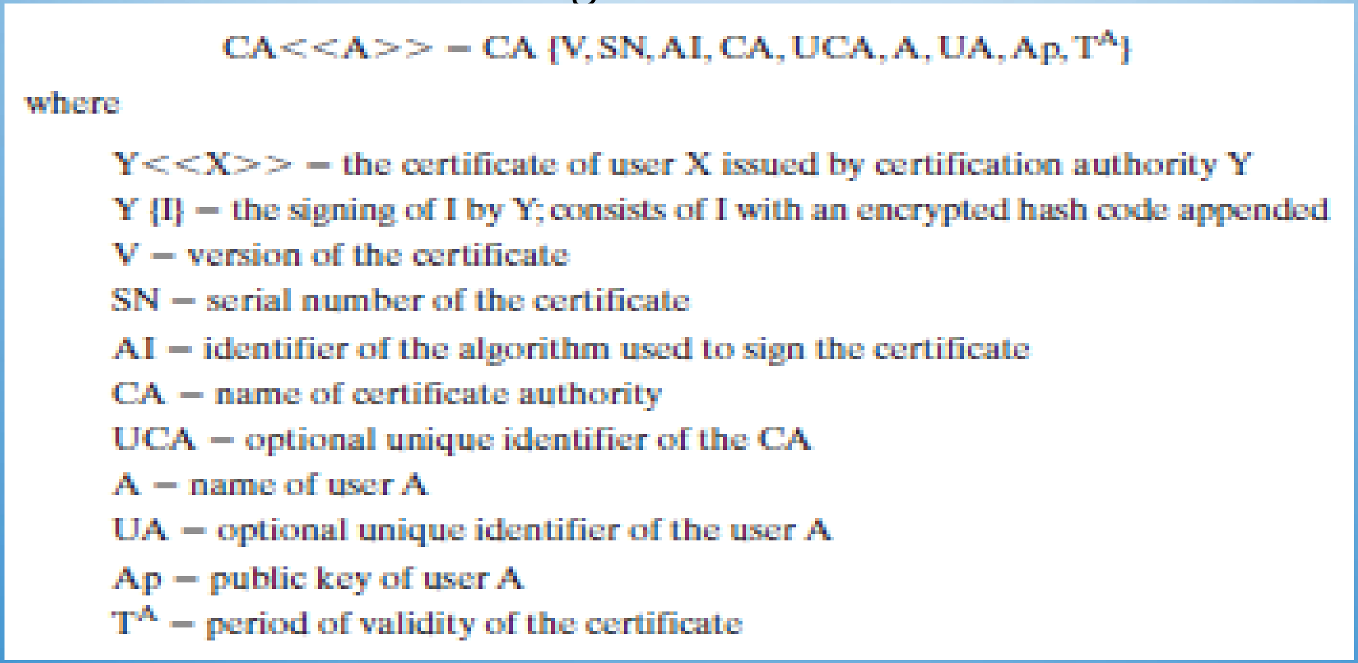
Subject name: The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

Subject’s public-key information: The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
Issuer unique identifier: An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
Subject unique identifier: An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
Extensions: A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
Signature: Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA’s private key.
This field includes the signature algorithm identifier.

- ➢The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time.

- ➢These fields are rarely used.

- ➢The standard uses the following notation to define a certificate:



$$CA<<A>> = CA\ \{V, SN, AI, CA, UCA, A, UA, Ap, T^A\}$$

where

$Y<<X>>$ = the certificate of user X issued by certification authority Y
$Y\ \{I\}$ = the signing of I by Y; consists of I with an encrypted hash code appended
V = version of the certificate
SN = serial number of the certificate
AI = identifier of the algorithm used to sign the certificate
CA = name of certificate authority
UCA = optional unique identifier of the CA
A = name of user A
UA = optional unique identifier of the user A
Ap = public key of user A
T^A = period of validity of the certificate

- ➢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.
- ➢This is the typical digital signature approach

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
- If all users subscribe to the same CA, then there is a common trust of that CA.
- All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users
- If there is a large community of users, it may not be practical for all users to subscribe to the same CA.
- Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures.
- This public key must be provided to each user in an absolutely secure way
- so that the user has confidence in the associated certificates.

➢Example: A has obtained a certificate from certification authority X1 and B has obtained a certificate from CA X2.

➢If A does not securely know the public key of X2, then B’s certificate, issued by X2, is useless to A.

➢A can read B’s certificate, but A cannot verify the signature.

➢However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B’s public key.

1. A obtains (from the directory) the certificate of X2 signed by X1. Because A securely knows X1’s public key, A can obtain X2’s public key from its certificate and verify it by means of X1’s signature on the certificate.

2. A then goes back to the directory and obtains the certificate of B signed by X2. Because A now has a trusted copy of X2’s public key, A can verify the signature and securely obtain B’s public key. A has used a chain of certificates to obtain B’s public key.

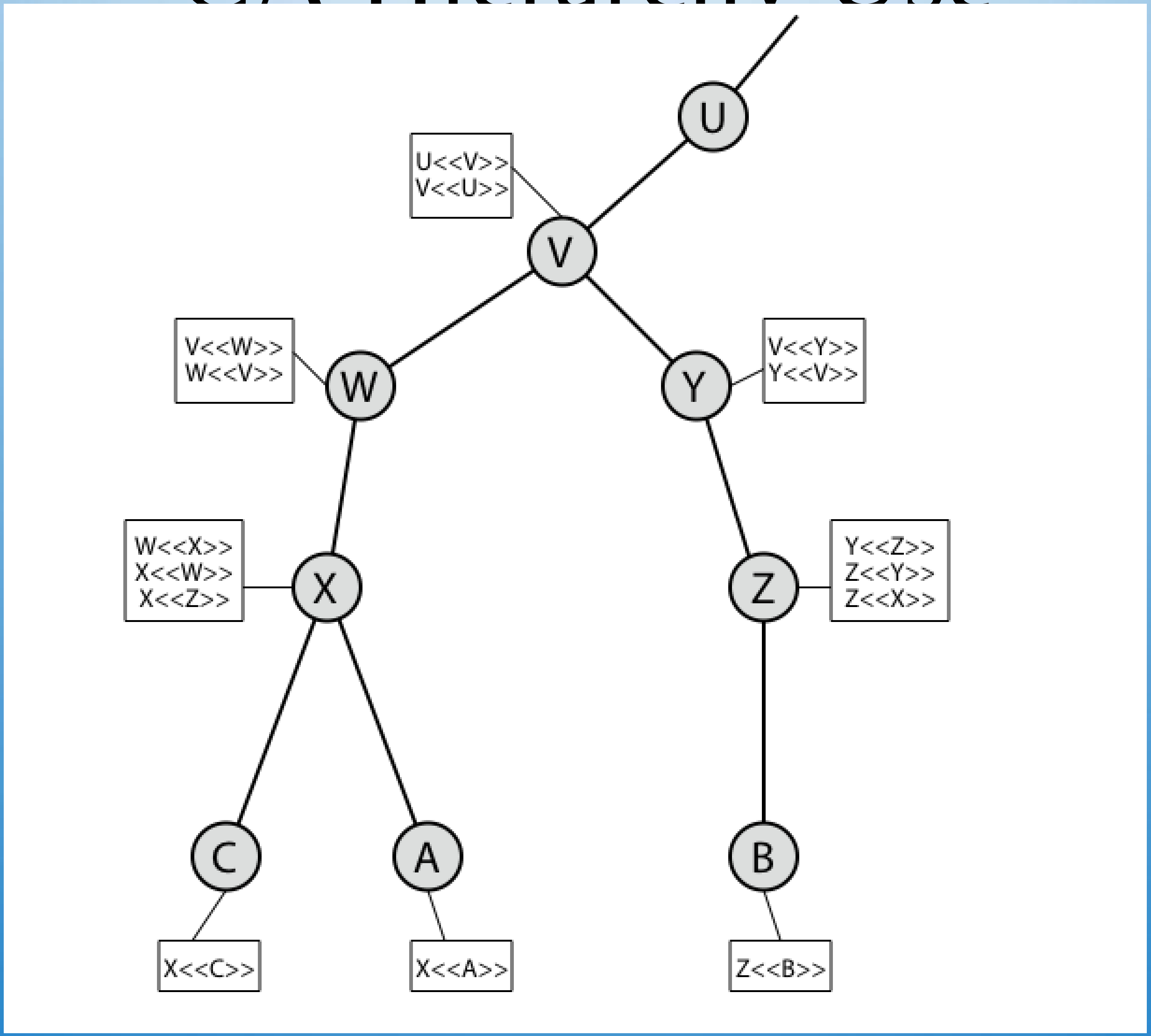
CA Hierarchy

- 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

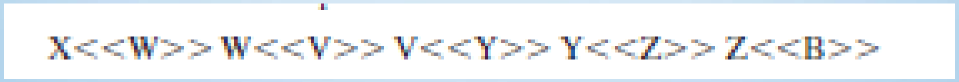
Forward certificates: Certificates of X generated by other CAs

Reverse certificates: Certificates generated by X that are the certificates of other CAs

CA Hierarchy Use

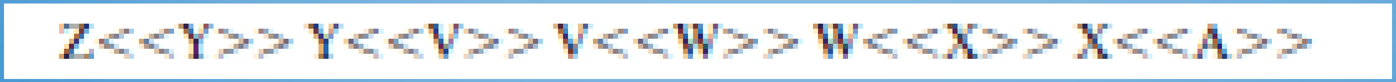


➢In this example, user A can acquire the following certificates from the directory to establish a certification path to B:


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

➢When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B’s public key.

Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A’s public key, which can be obtained from the certification path:


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

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

X-509 supports three types of authenticating using public key signatures.
The types of authentication are

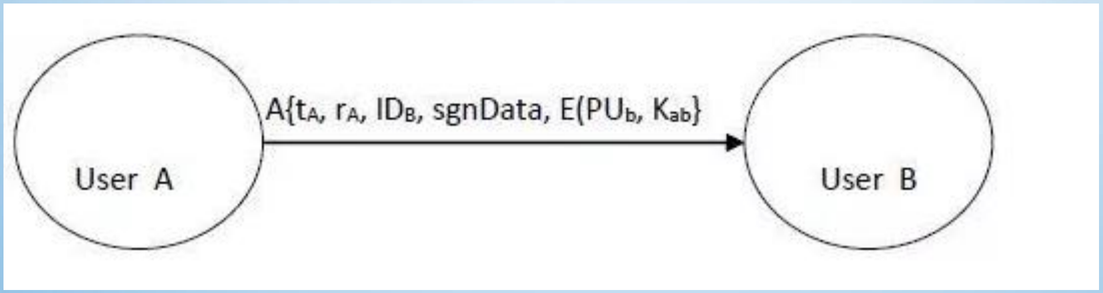
One-way authentication

Two- way authentication

Three-way authentication

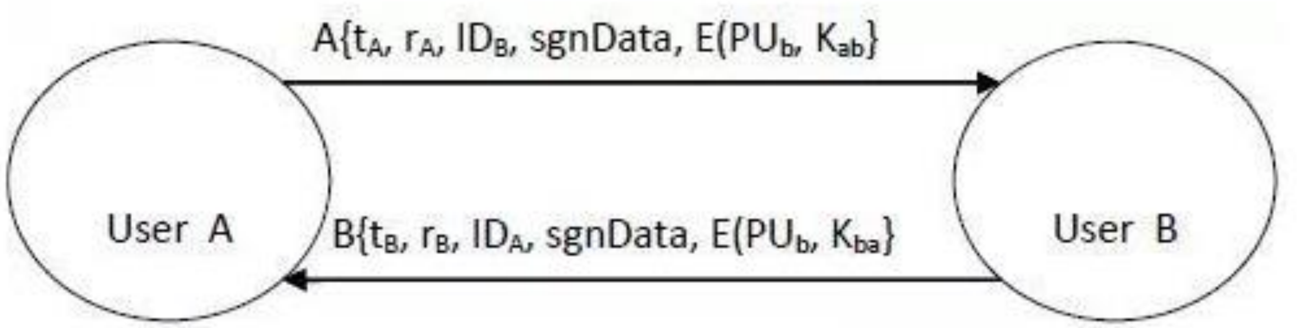
One-way authentication: Authentication Procedures X-509
It involves the single transfer of information from one user (say A) to other (B).
This method authenticates the identity of A to B and the integrity of the message.

Here, a message in the {} is signed by A.
Moreover, sign data is the information that needs to be conveyed. $\mathbf{t_A}$ is timestamp and rA is the nonce.



Two- way authentication: Authentication Procedures X-509

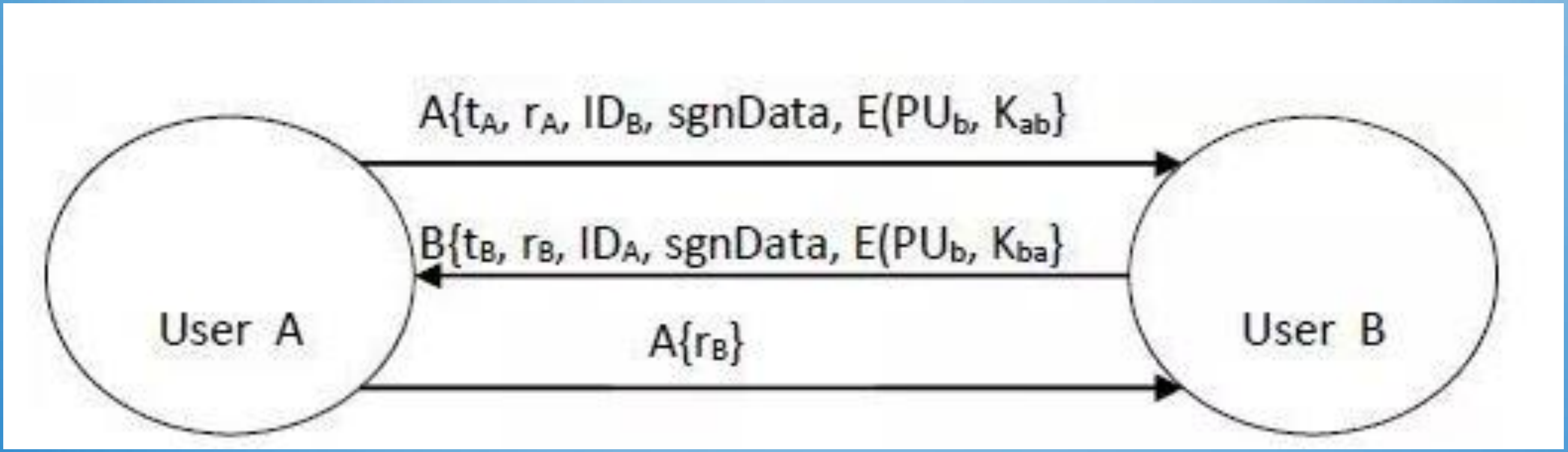
Two- way authentication allows both parties to communicate and verify the identity of each other.


A{t_A, r_A, ID_B, sgnData, E(PU$_b$, K$_{ab}$}
User A
B{t_B, r_B, ID_A, sgnData, E(PU$_b$, K$_{ba}$}
User B

Three-way authentication: Authentication Procedures X-509

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

This method includes an additional message from A.


$A\{t_A, r_A, ID_B, sgnData, E(PU_b, K_{ab}\}$
$B\{t_B, r_B, ID_A, sgnData, E(PU_b, K_{ba}\}$
$A\{r_B\}$
User A
User B

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

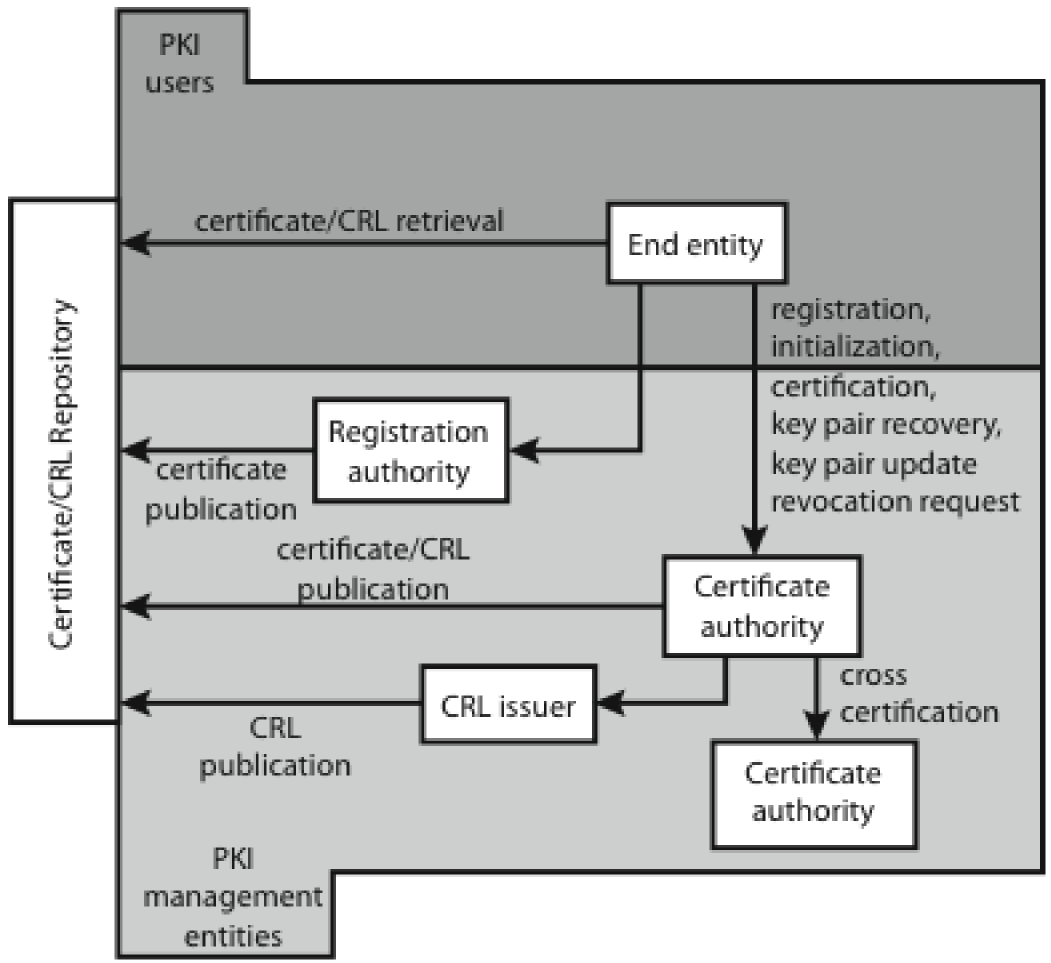
- RFC 2822 (Internet Security Glossary) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

- 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.

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
cross certification
CRL issuer
CRL publication
Certificate authority
PKI management entities

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.

PKIX Management

- functions:
  - registration
  - initialization
  - certification
  - key pair recovery
  - key pair update
  - revocation request
  - cross certification
- protocols: CMP, CMC

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 off-line 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.

THANK YOU