CYB 241 Digital Cryptography Techniques

Key Distribution for Symmetric Key Cryptography

Key Distribution Technique

- Used to deliver a key to two parties who wish to exchange data without allowing others to see the key
- For symmetric encryption, the two parties must share the same key, and that key must be protected from access by others
- Frequent key changes are desirable to limit the amount of data compromised if an attacker learns the key

The need for key distribution

- Large group of people, processes, or systems want to communicate with one another in a secure fashion.
  - Entities also want to join or leave the group at any time.
- Simple solution is each party physically exchanging an encryption key with everyone.
  - Enable establishing a secure communication link using the encryption key.
  - Not feasible. Why?
- Alternative using key distribution center (KDC)
  - Provide every group member with a single key for securely communicate with KDC
  - This key is called a **master key**.
  - When A wants to establish a secure communication link with B, A requests a **session key** from KDC and shared with B.

- **Session Key**
  - A temporary key used for the duration of a logical connection then discarded
  - Obtained from key distribution center (KDC)
  - transmitted in encrypted form, using a master key
- **Master Key**
  - Each user has unique master key shared with KDC
  - *N* master keys are required for *N* entities communicating in pair
  - Master keys can be distributed in non-cryptographic way, such as physical delivery.

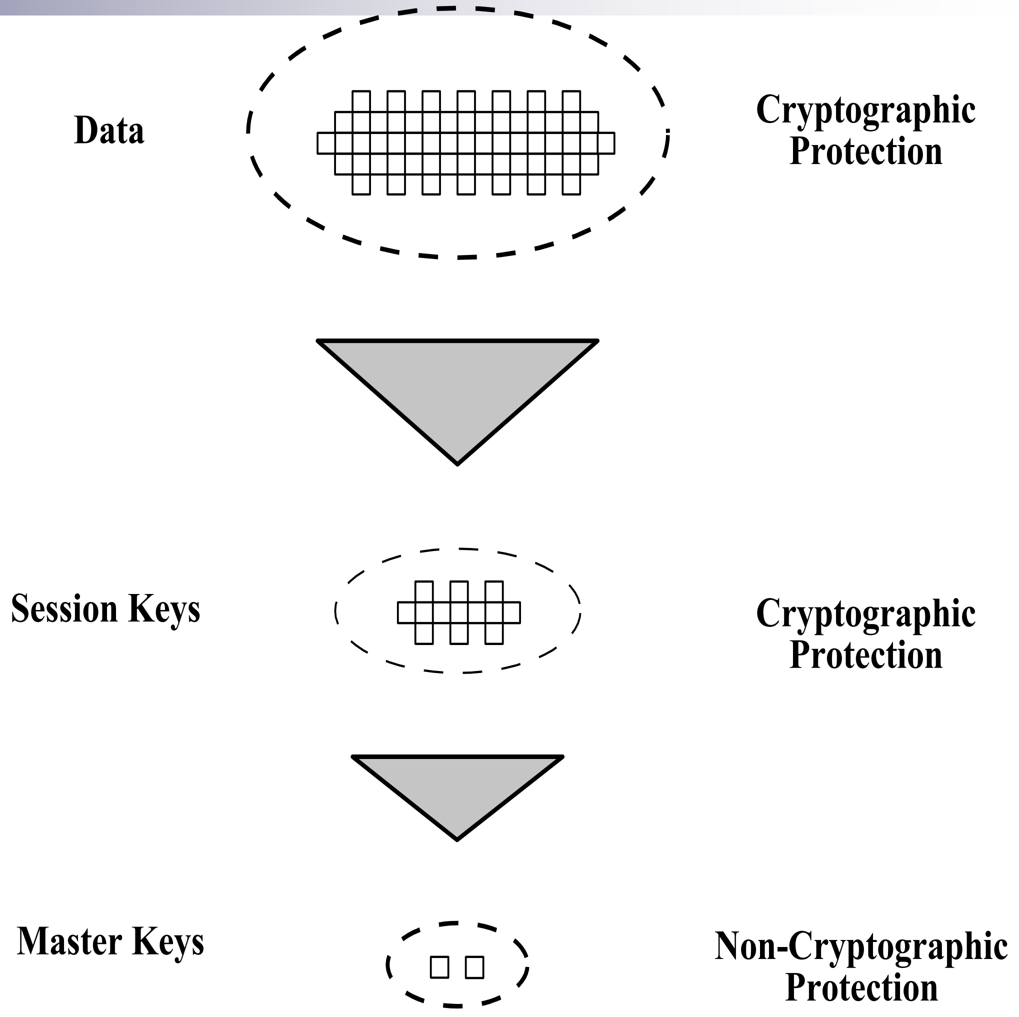


Figure 14.2 The Use of a Key Hierarchy

Number of Required Session Keys

- If there are *N* hosts
- Number of key required is

  [N(N-1)]/2

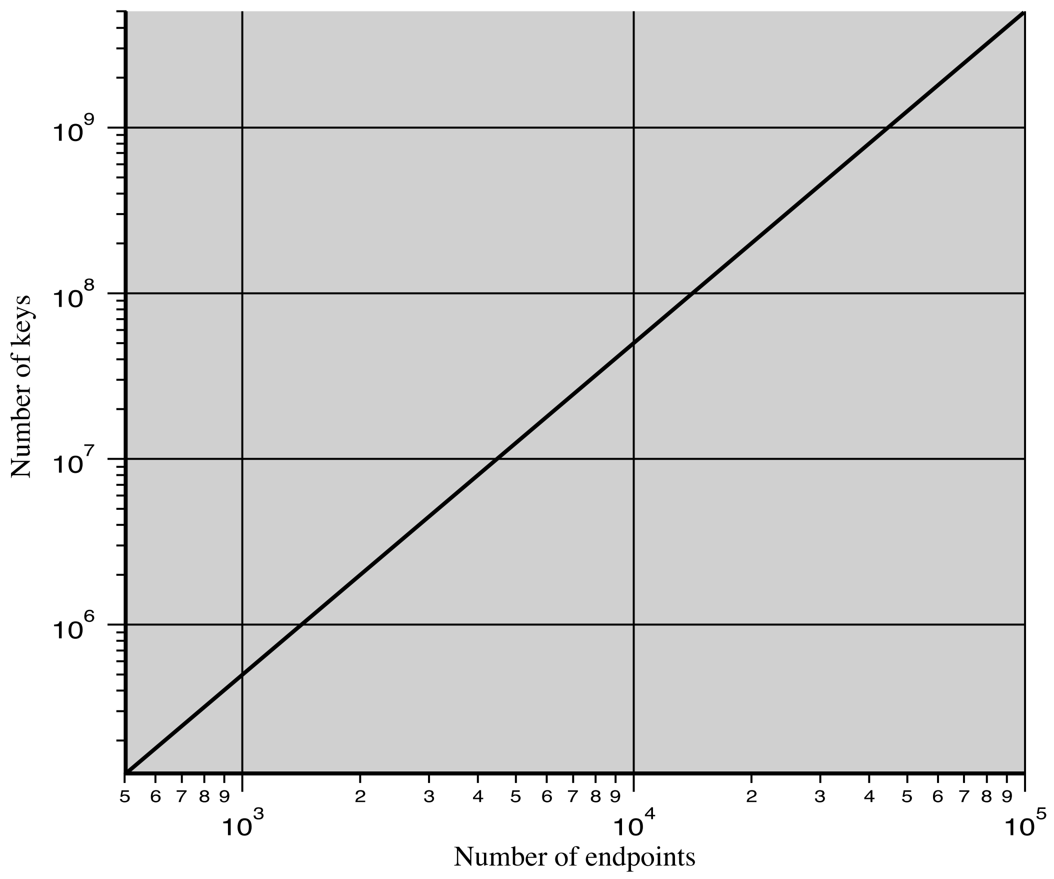
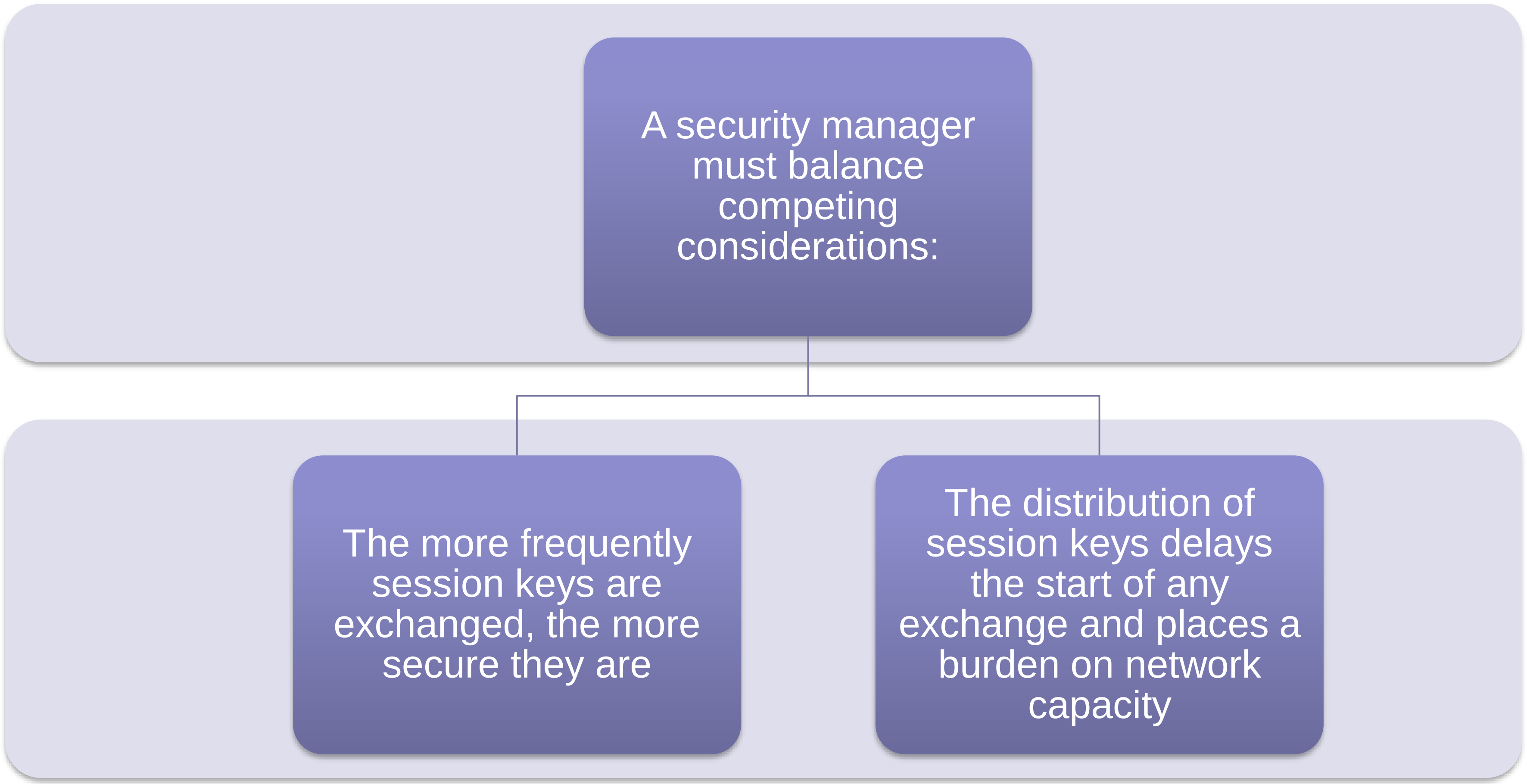


Figure 14.1 Number of Keys Required to Support Arbitrary Connections Between Endpoints

Session Key Lifetime



A security manager must balance competing considerations:

The more frequently session keys are exchanged, the more secure they are

The distribution of session keys delays the start of any exchange and places a burden on network capacity

Needham-Schroeder protocol

- A party named A wants to establish a secure communication link with another party B.
- Both the parties A and B possess master keys $\mathbf{K_a}$ and $\mathbf{K_b}$, respectively used to communicate with KDC
- User A sends a request to KDC for a session key for communicating with user B.
  - The message sent by A to KDC includes A's network address ($\mathbf{ID_A}$), B's network address ($\mathbf{ID_B}$), and a unique session identifier (nonce) $\mathbf{N_1}$.
- KDC responds to A with a message encrypted using the key $\mathbf{K_a}$ contains:
  - The session-key $\mathbf{K_s}$
  - The original message received from A
  - A ticket that encrypted with $\mathbf{K_b}$ to be sent to B
    - The ticket has the session key $\mathbf{K_s}$, and A's identifier $\mathbf{ID_A}$

Needham-Schroeder protocol

- A decrypts the message received from KDC and keeps the session key $\mathbf{K_s}$ then sends the ticket to B.
- B decrypts the message received from A using the master key $\mathbf{K_b}$ and compares the $\mathbf{ID_A}$ with the sender identifier to make certain that no one is masquerading A.
- Using the session key $\mathbf{K_s}$, B sends back to A a nonce N2.
- A responds with N2 + 1 encrypted also with $\mathbf{K_s}$
- The last two steps to confirm that:
  - The session keys are working properly
  - No replay attack. Why?
- Timestamp can be added to step 3. Why?

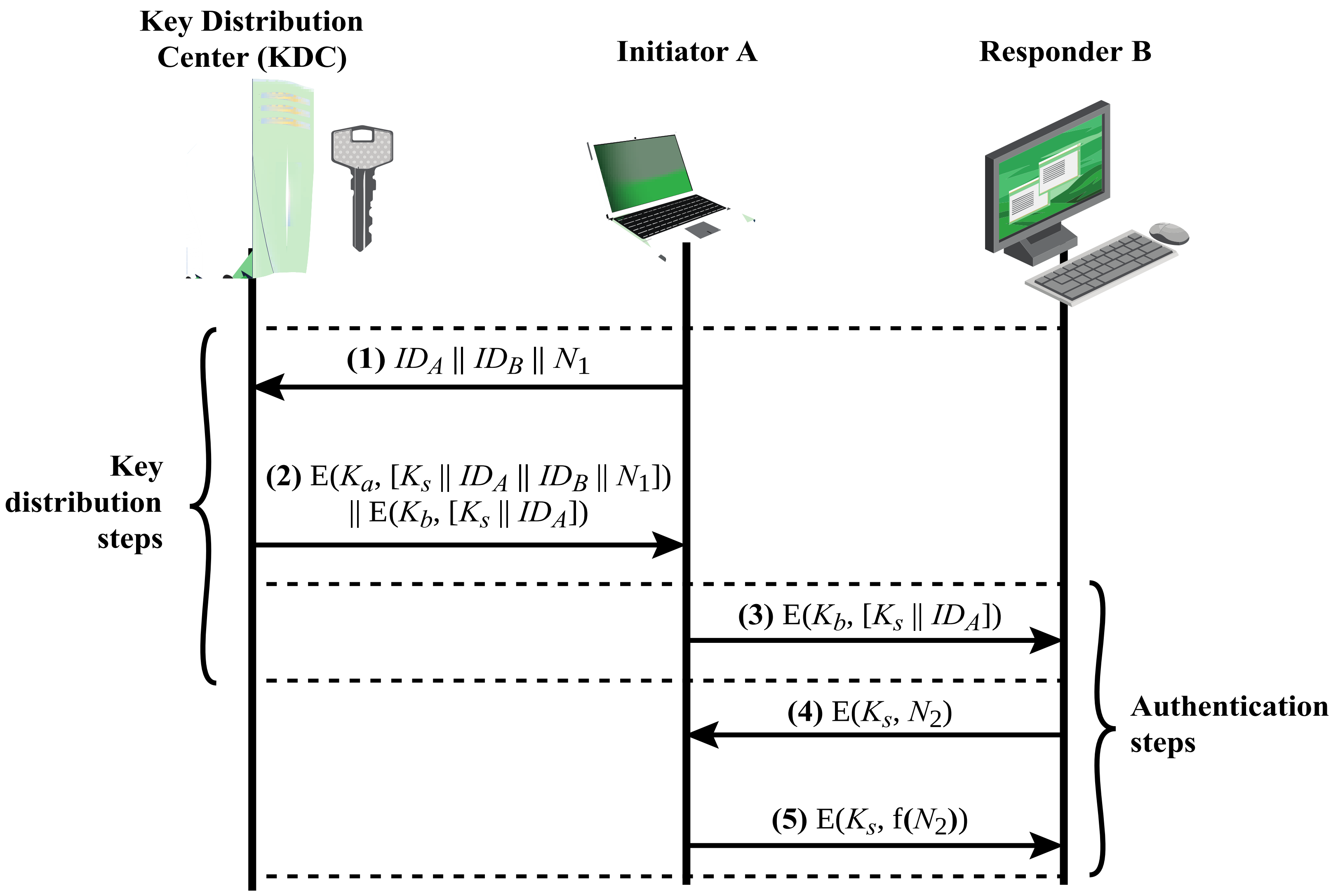

Key Distribution Center (KDC)
Initiator A
Responder B
(1) $ID_A \| ID_B \| N_1$
(2) $E(K_a, [K_s \| ID_A \| ID_B \| N_1]) \| E(K_b, [K_s \| ID_A])$
(3) $E(K_b, [K_s \| ID_A])$
(4) $E(K_s, N_2)$
(5) $E(K_s, f(N_2))$
Key distribution steps
Authentication steps

Figure 14.3 Key Distribution Scenario

Hierarchical KDC

- A hierarchy of KDCs can be used for very large network.
- Local KDC is responsible for key distribution within the same local domain
- If two entities in different domains desire a shared key, then the corresponding local KDC’s can communicate through a global KDC
- The hierarchical concept can be extended to three or more layers
- It minimizes the effort involved in master key distribution
    - Limits the range of a faulty or subverted KDC to its local area only

Kerberos

- Developed by MIT
- Assume open distributed environment
- User at workstations want to access services on distributed servers over network
- Servers must be able to authenticate requests to services
- Workstation cannot be trusted to identify its users to network services.

Possible Threats

- User gains access to workstation
  - Pretend to be another user
- User alters network address of workstation
  - Requests appear from impersonated workstation
- User eavesdrops on exchanges
  - Use replay attack to gain access to server or to disrupt operations

Security Approaches

- Small network by single organization
  - can rely on client workstation to assure user identity
- More open environment
  - require user to prove identity for each service
  - require severs to prove identity to client workstation
  - this approach is used in Kerberos
- Kerberos provides a centralized authentication server to authenticate users to servers and servers to users
  - Relies exclusively on symmetric encryption

Terminology

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

Simple Authentication Dialogue

(1) C → AS: $ID_C || P_C || ID_v$

(2) AS → C: Ticket

(3) C → V: $ID_C || Ticket$

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

Simple Authentication Dialogue

- (1), AS must verify
  - user password correct
  - user permitted access to V
- (3), C requests service from V
  - V decrypts Ticket, verify ID_C = encrypted ID_C
- ID_V included so V verify decrypted properly
- ID_C verify ticket issued on behalf of C
- AD_C prevent capture, send from another workstation

Problems

- Need to enter password for each service
  - file server, print server, mail server
- Password transmitted in plaintext
  - eavesdropper can capture and use it

Kerberos 4 Overview

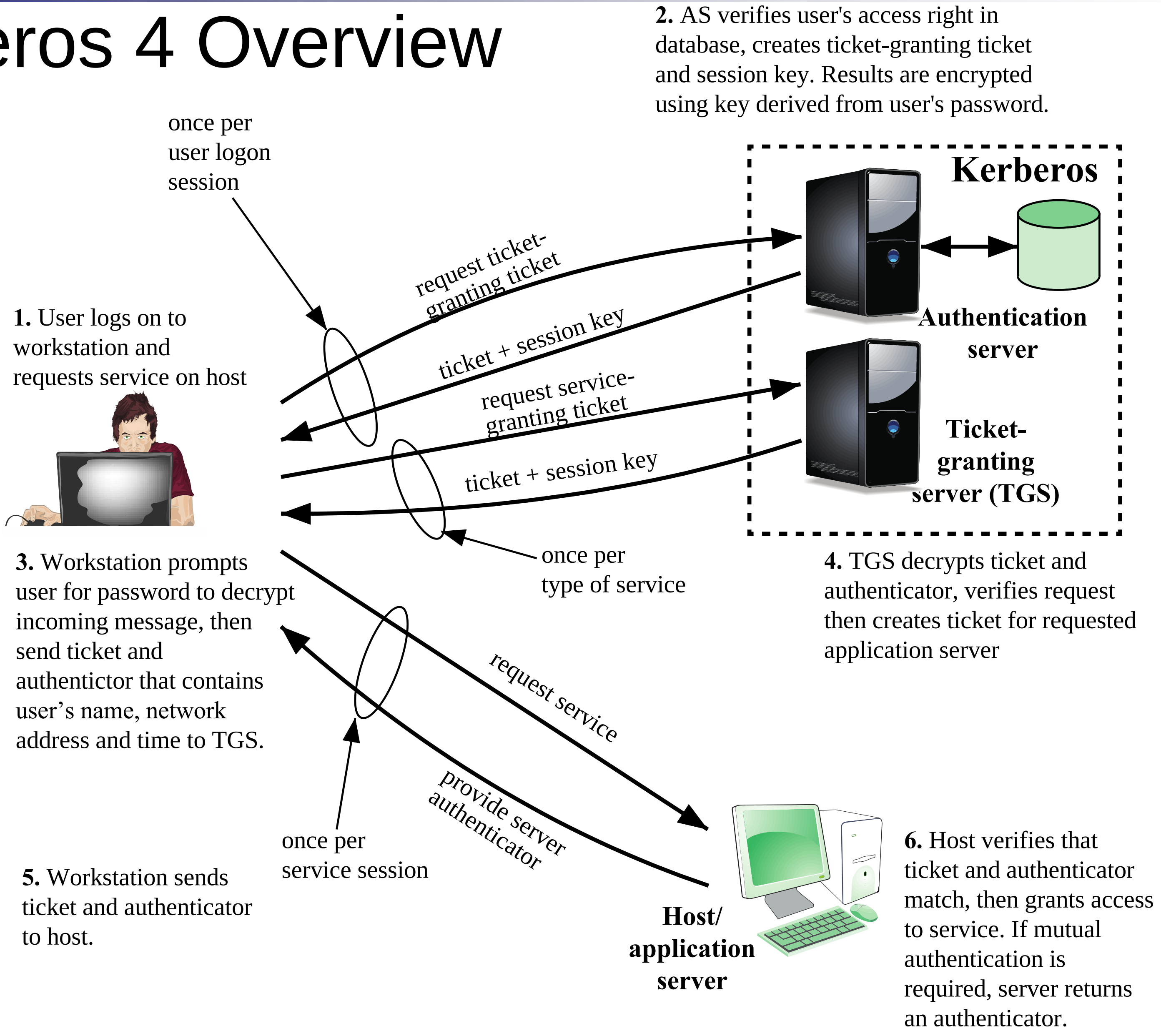

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
once per user logon session
Kerberos
request ticket-granting ticket
ticket + session key
Authentication server
1. User logs on to workstation and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

Figure 15.1 Overview of Kerberos

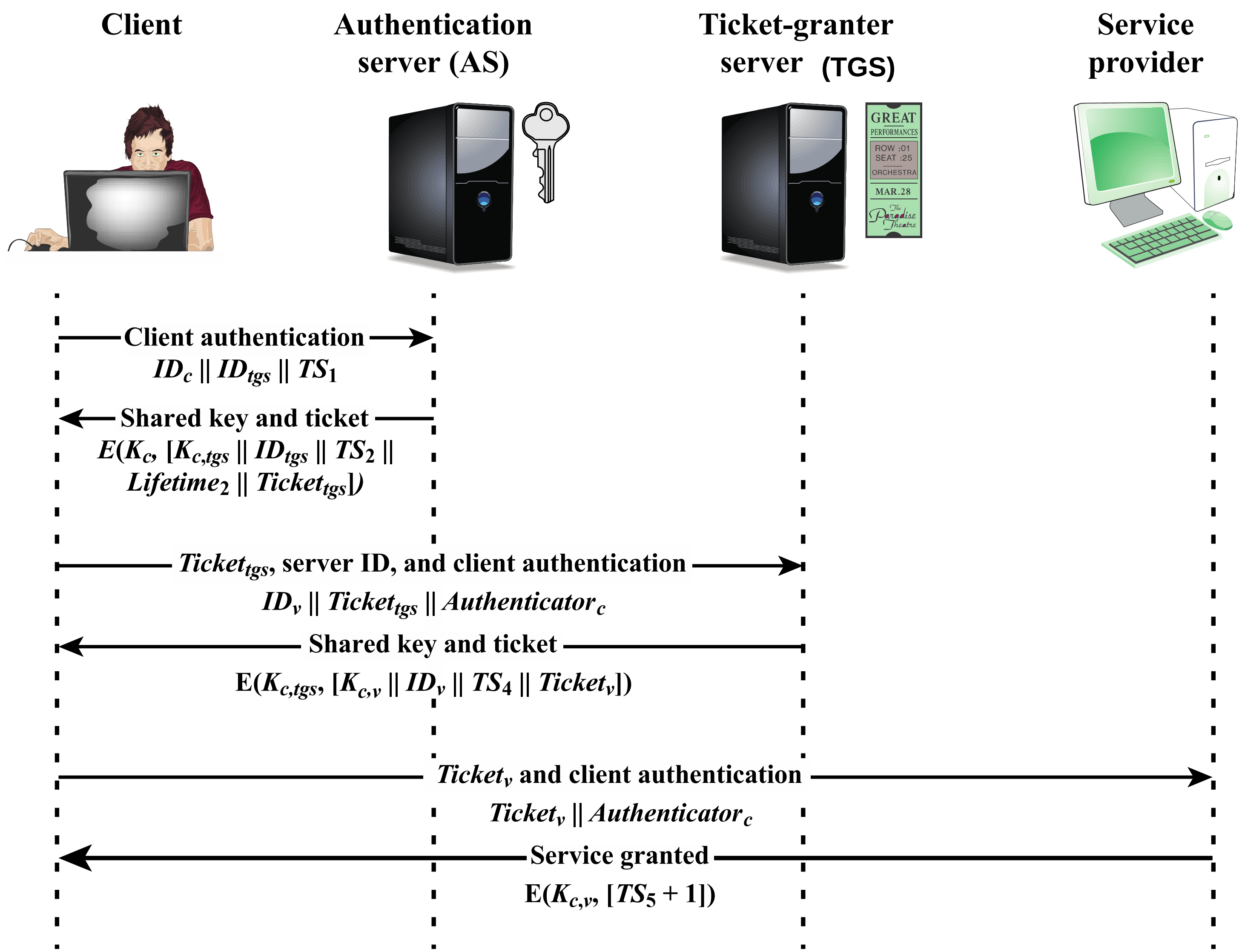

Client
Authentication server (AS)
Ticket-granter server (TGS)
Service provider
GREAT PERFORMANCES
ROW :01
SEAT :25
ORCHESTRA
MAR.28
The Paradise Theatre
Client authentication
$ID_c \| ID_{tgs} \| TS_1$
Shared key and ticket
$E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$
$Ticket_{tgs}$, server ID, and client authentication
$ID_v \| Ticket_{tgs} \| Authenticator_c$
Shared key and ticket
$E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$
$Ticket_v$ and client authentication
$Ticket_v \| Authenticator_c$
Service granted
$E(K_{c,v}, [TS_5 + 1])$

Figure 15.2 Kerberos Exchanges

Kerberos 4 Authentication

(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

CYB 241 Digital Cryptography Techniques

Key Management and Public Key Cryptography

Key Distribution Using Asymmetric Encryption

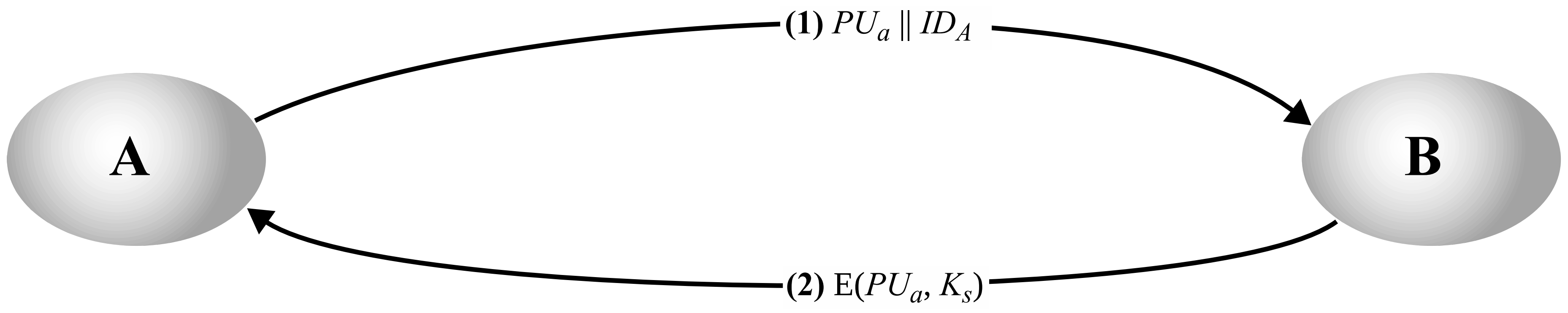
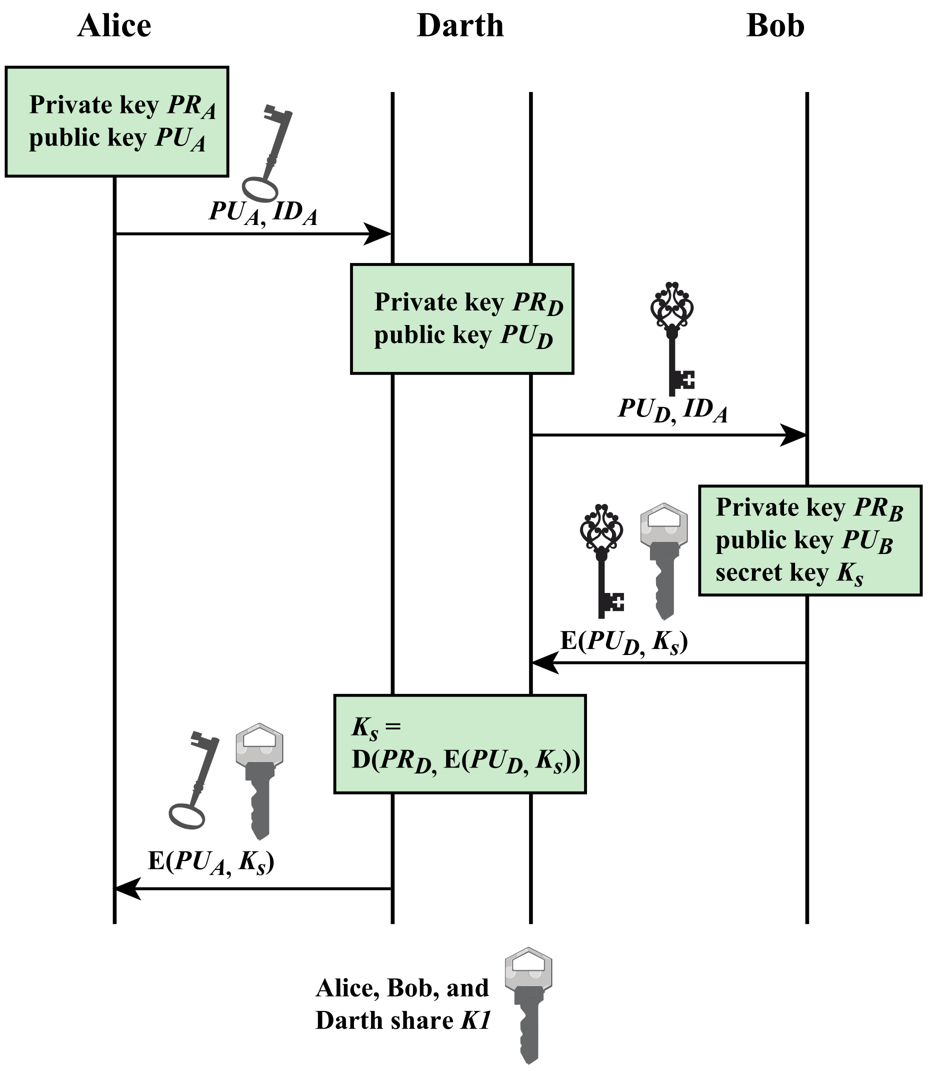


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

Man-in-the-Middle Attack

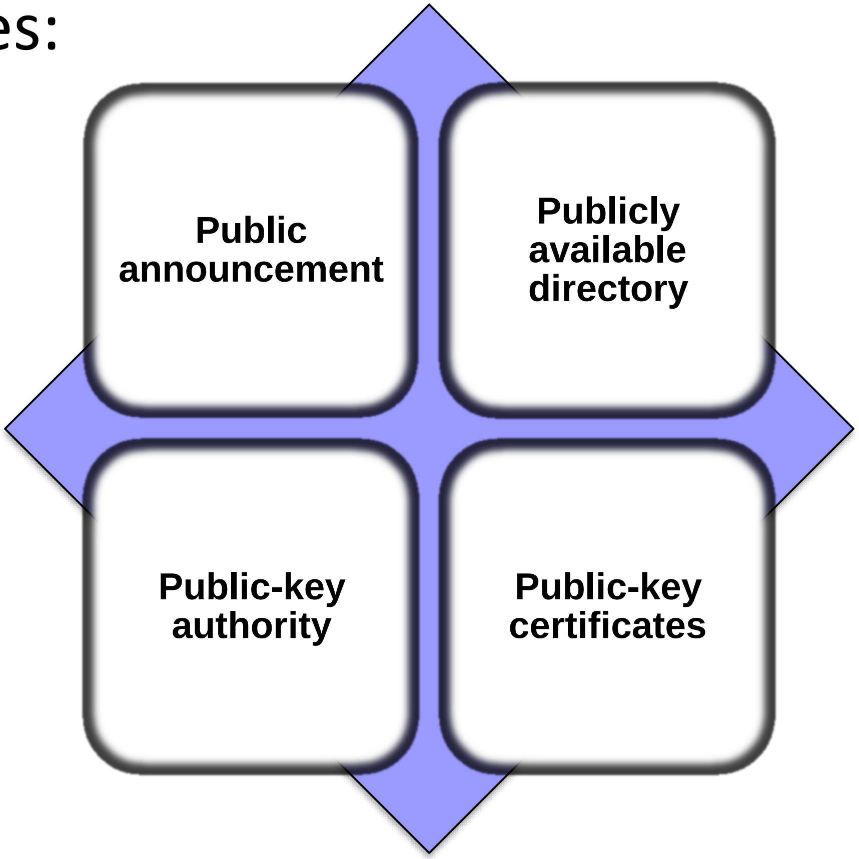


Alice
Darth
Bob
Private key PR_A public key PU_A
PU_A, ID_A
Private key PR_D public key PU_D
PU_D, ID_A
Private key PR_B public key PU_B secret key K_s
E(PU_D, K_s)
K_s = D(PR_D, E(PU_D, K_s))
E(PU_A, K_s)
Alice, Bob, and Darth share $K1$

Figure 14.8 Another Man-in-the-Middle Attack

Distribution of Public Keys

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



Public Announcement

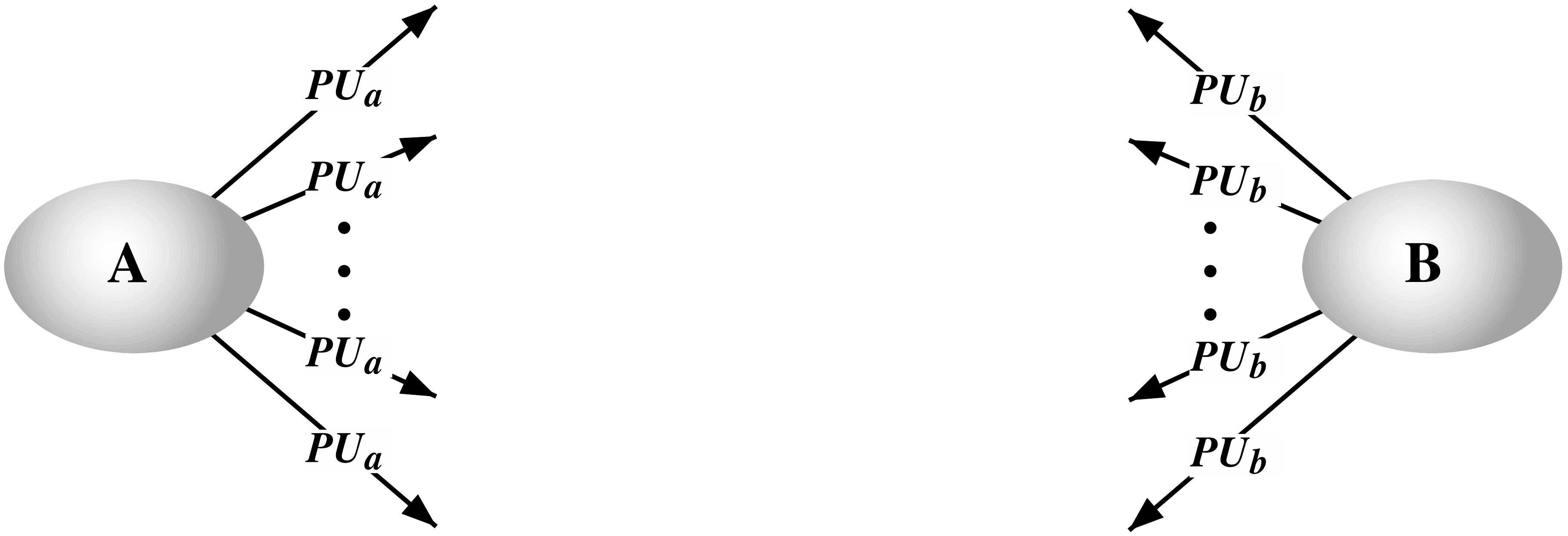


Figure 14.10 Uncontrolled Public Key Distribution

Publicly Available Directory

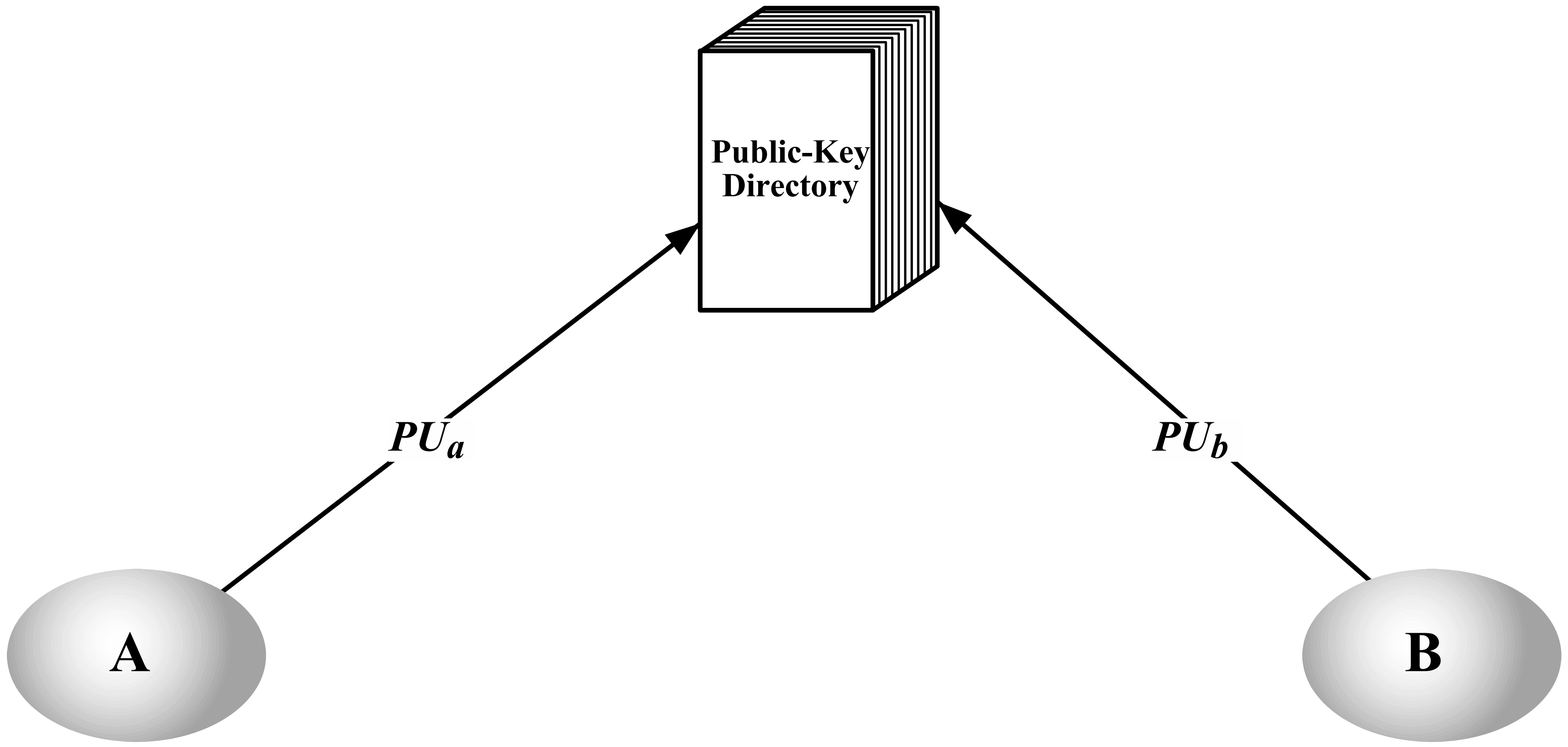


Figure 14.11 Public Key Publication

Public Key Authority

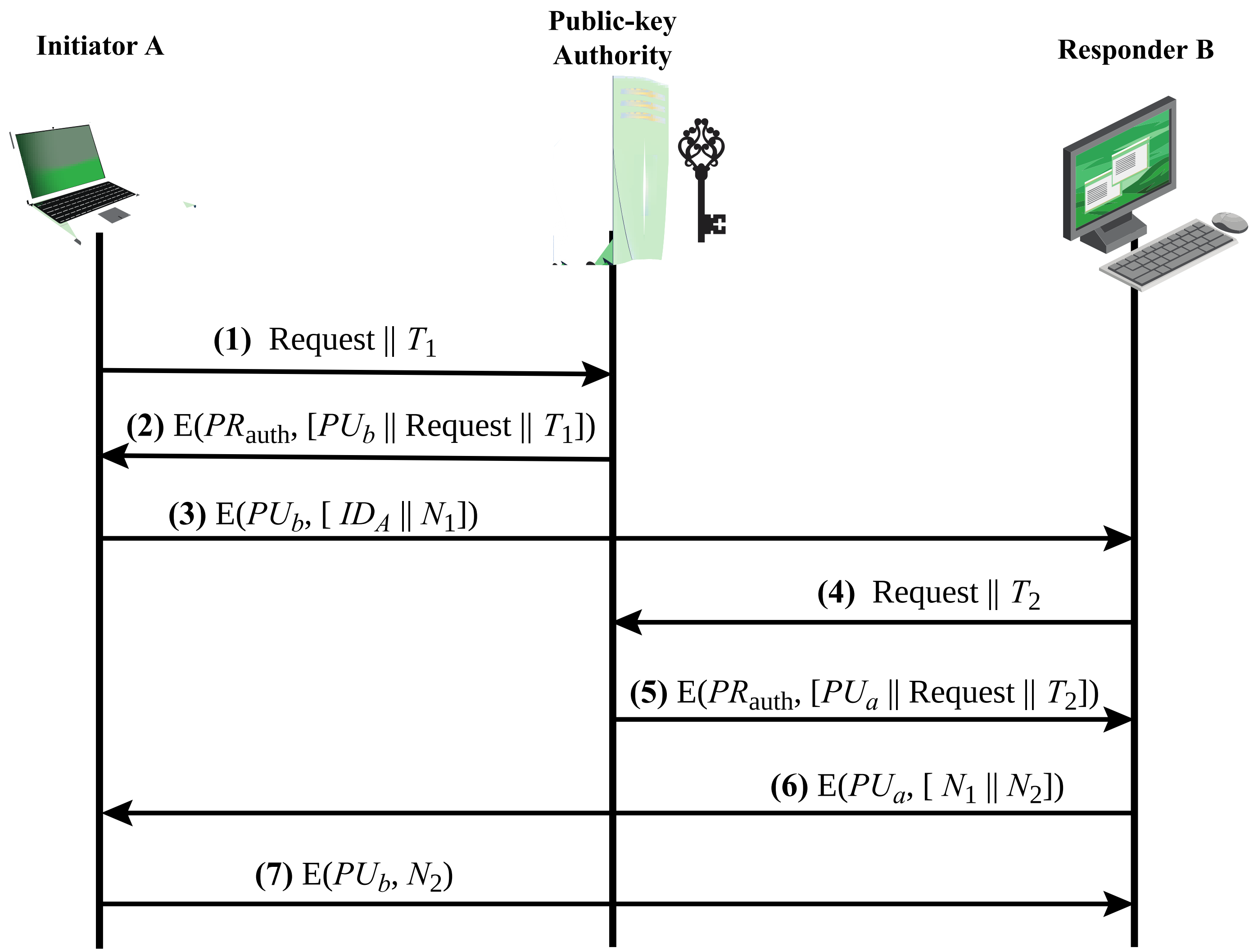
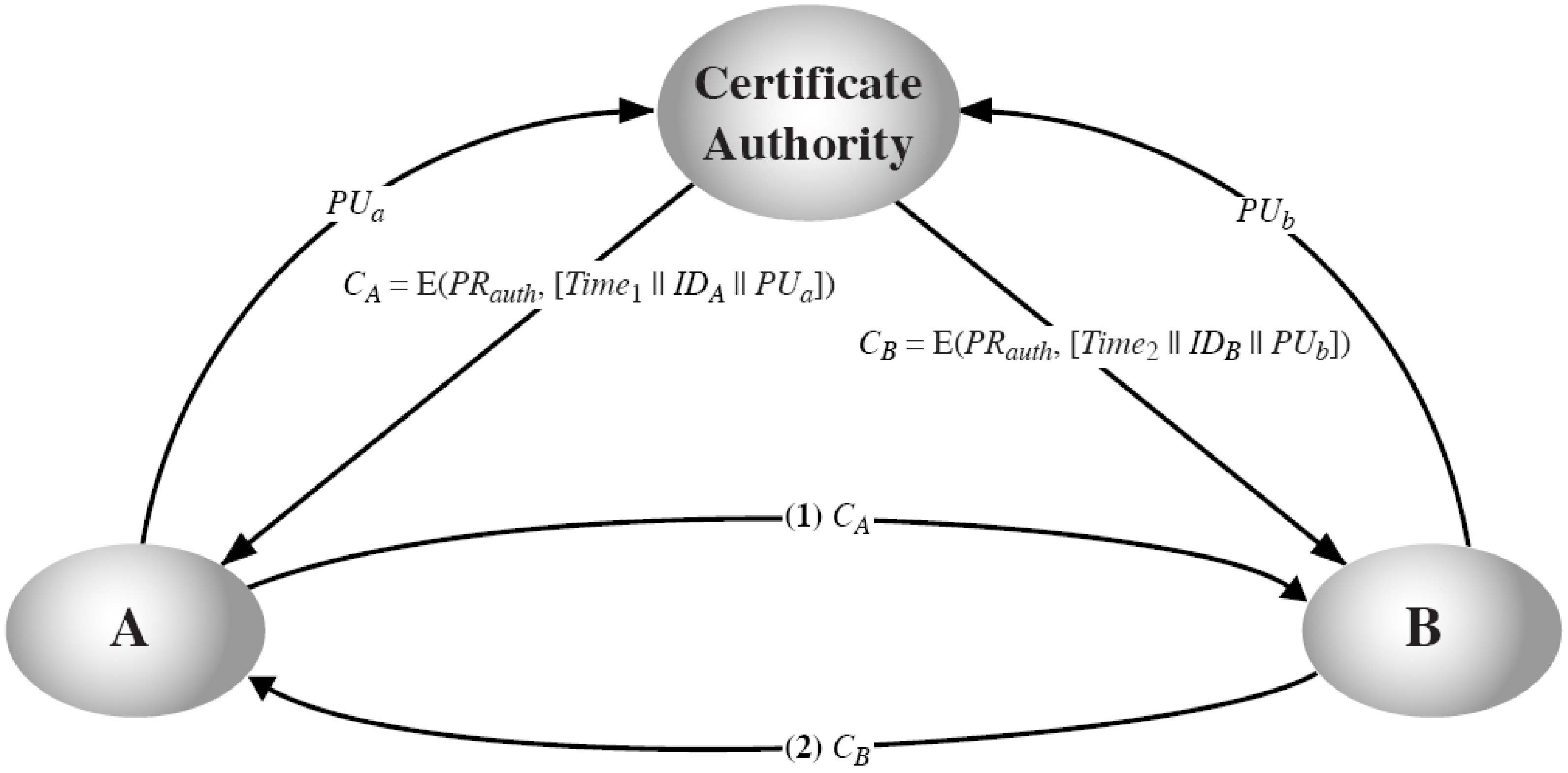

Initiator A
Public-key Authority
Responder B
(1) Request || T_1
(2) E(PR_{auth}, [PU_b || Request || T_1])
(3) E(PU_b, [ID_A || N_1])
(4) Request || T_2
(5) E(PR_{auth}, [PU_a || Request || T_2])
(6) E(PU_a, [N_1 || N_2])
(7) E(PU_b, N_2)

Figure 14.12 Public-Key Distribution Scenario

Public-Key Certificates

- Certificate contains
  - public key of certificate holder
  - ID of certificate holder
- Certificate encrypted using PR_{auth}
  - serves as a trusted signature
- Users can verify certificate using PU_{auth}
- Certificate authority
  - government agency or financial institution

Public-Key Certificates



Certificate Authority
PU_a
PU_b
$C_A = E(PR_{auth}, [Time_1 \| ID_A \| PU_a])$
$C_B = E(PR_{auth}, [Time_2 \| ID_B \| PU_b])$
(1) C_A
(2) C_B
A
B

Public-Key Certificates

- Participant A applies to auth for certificate
  - supplies PU_A and request certificate
  - in person or by secure communication
- Authority provides certificate
  - $C_A = E(PR_{auth}, [T||ID_A||PU_A])$
- A may pass this certificate to others (B)
- B can verify certificate
  - $D(PU_{auth}, C_A) = (T||ID_A||PU_A)$

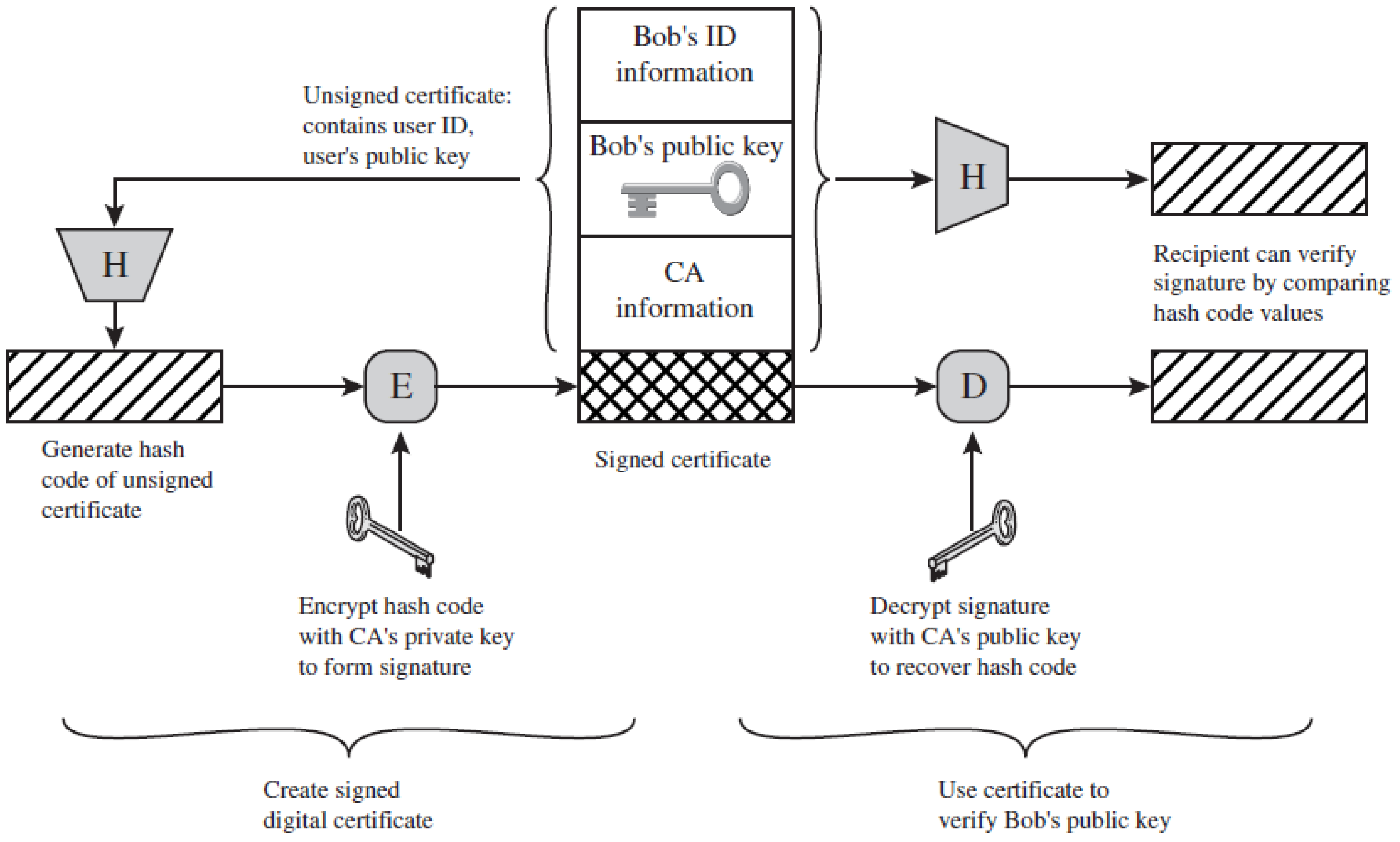
Public-Key Certificates

- Timestamp serves as expiration date
- Sufficiently old certificate assumed expired
- If PR is compromised
  - A applies for new certificate
  - still at risk until other communicants are aware
- X.509 is universally accepted standard for certificate format

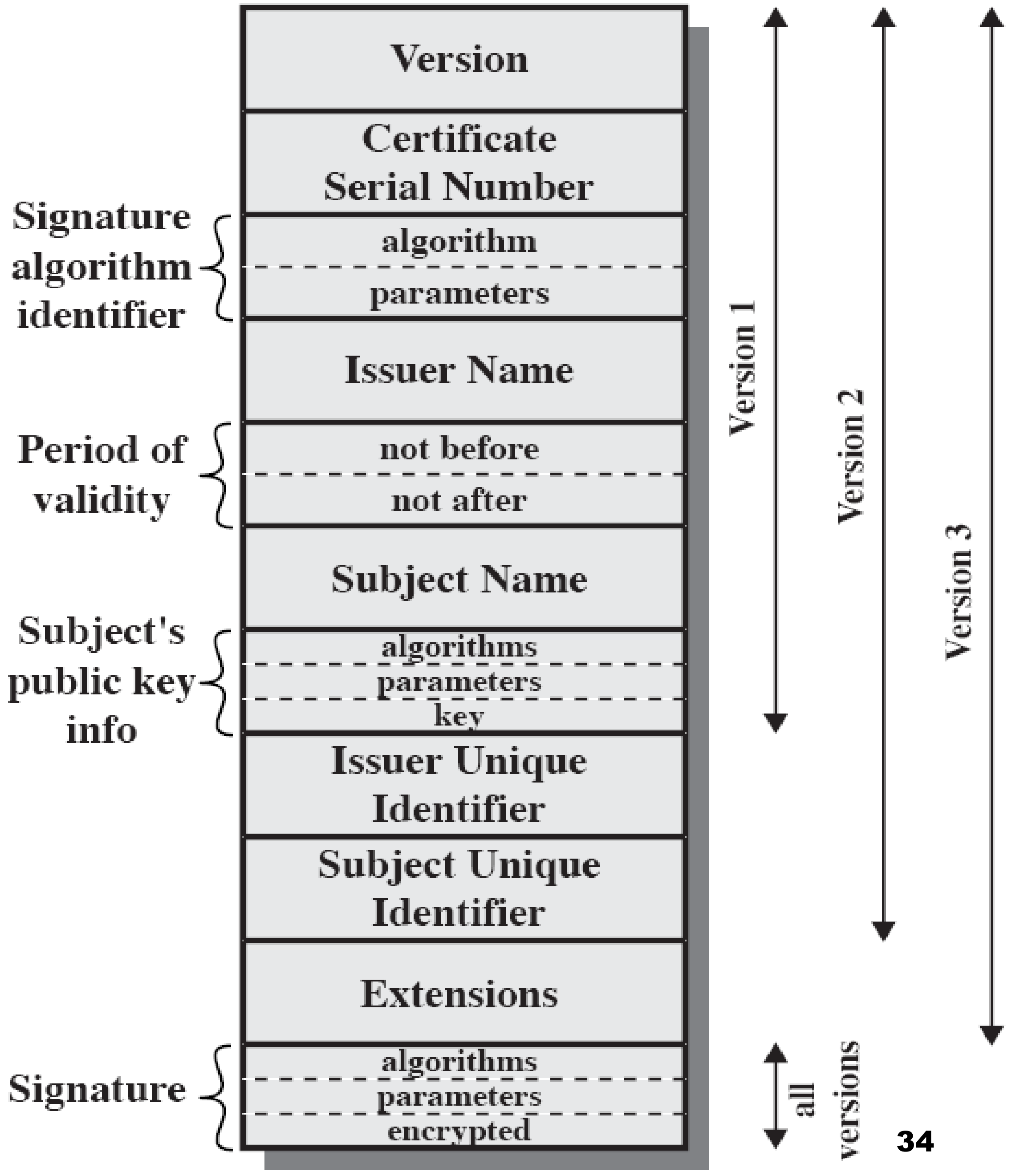
X.509 Certificates

- Public-key certificate associated with user
- Assumed created by trusted CA
- Placed in directory server
  - provides easily accessible location
  - not responsible for creating certification
- Based on Public-key and digital signatures
  - Public-key: no specific algorithm, recommends RSA
  - Digital signature
    - requires using hash function
    - no specific hash algorithm
- Used in S/MIME, IPSec and SSL/TLS

X.509 Certificates


Unsigned certificate: contains user ID, user's public key
Bob's ID information
Bob's public key
CA information
H
H
Recipient can verify signature by comparing hash code values
E
D
Signed certificate
Generate hash code of unsigned certificate
Encrypt hash code with CA's private key to form signature
Decrypt signature with CA's public key to recover hash code
Create signed digital certificate
Use certificate to verify Bob's public key

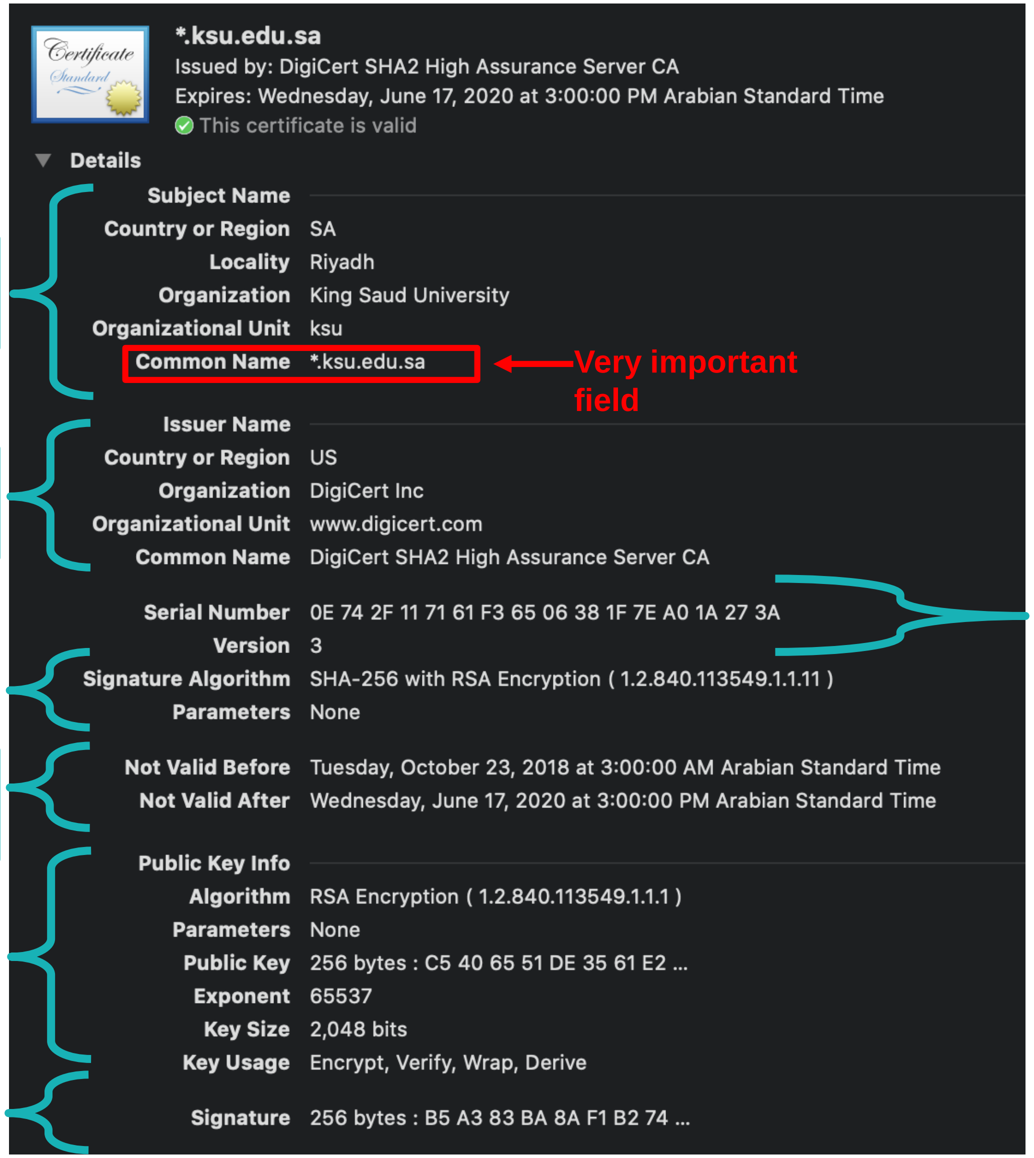
X.509 Certificates



Certificate Format

- Issuer name
  - Name of CA created and signed this certificate
- Subject name
  - Name of user to whom certificates refers
- Subject’s public-key information
  - Public-key of subject + algorithm id + parameters
- Signature
  - Hash code of other fields, encrypted with CA Private-key
- Y <<X>> = the certificate of user X issued by certification authority Y

Example: X.509 Digital Certificate



*.ksu.edu.sa
Issued by: DigiCert SHA2 High Assurance Server CA
Expires: Wednesday, June 17, 2020 at 3:00:00 PM Arabian Standard Time
This certificate is valid

Details

Subject Name	
Country or Region	SA
Locality	Riyadh
Organization	King Saud University
Organizational Unit	ksu
Common Name	*.ksu.edu.sa
Issuer Name	
Country or Region	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert SHA2 High Assurance Server CA
Serial Number	0E 74 2F 11 71 61 F3 65 06 38 1F 7E A0 1A 27 3A
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None
Not Valid Before	Tuesday, October 23, 2018 at 3:00:00 AM Arabian Standard Time
Not Valid After	Wednesday, June 17, 2020 at 3:00:00 PM Arabian Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : C5 40 65 51 DE 35 61 E2 ...
Exponent	65537
Key Size	2,048 bits
Key Usage	Encrypt, Verify, Wrap, Derive
Signature	256 bytes : B5 A3 83 BA 8A F1 B2 74 ...

Owner's information

Very important field

Issuer's information

Certificate's serial number

Issuer's signature algorithm

Validity period

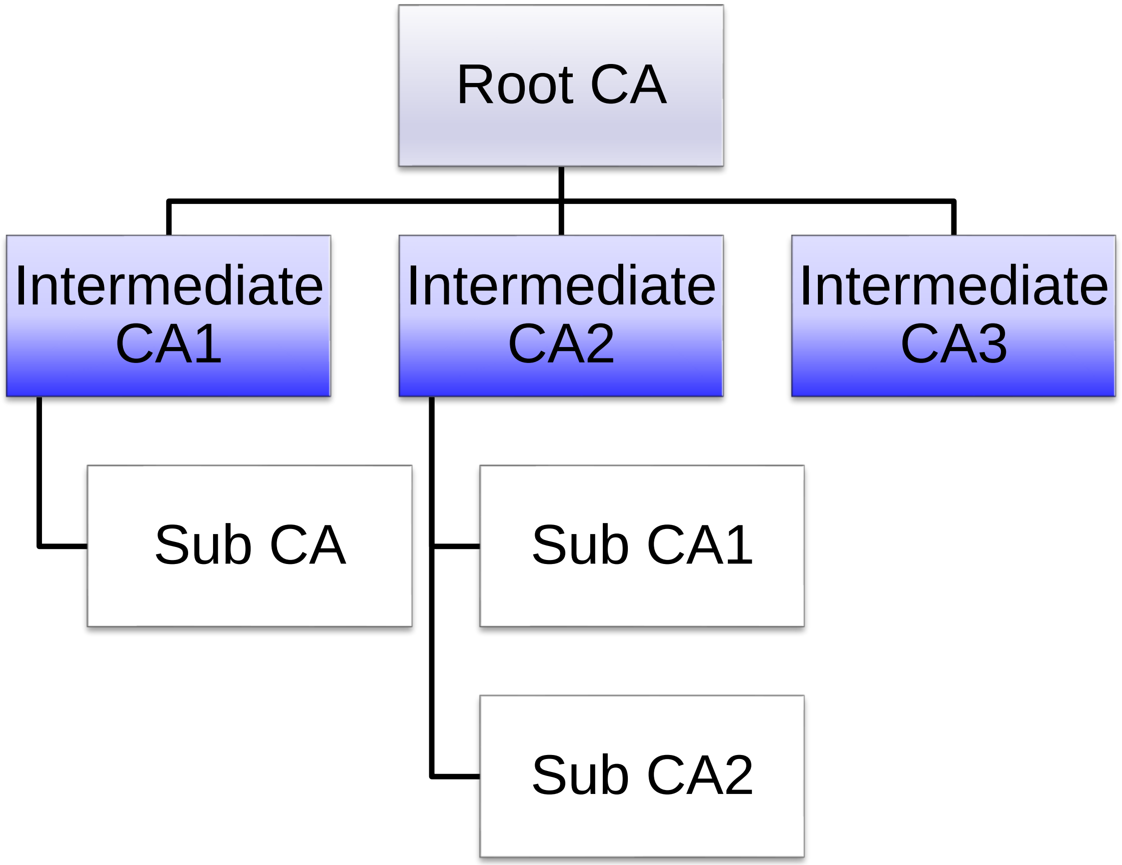
Owner's public key

Issuer's digital signature

Optional extensions (not shown)

Hierarchical Approach

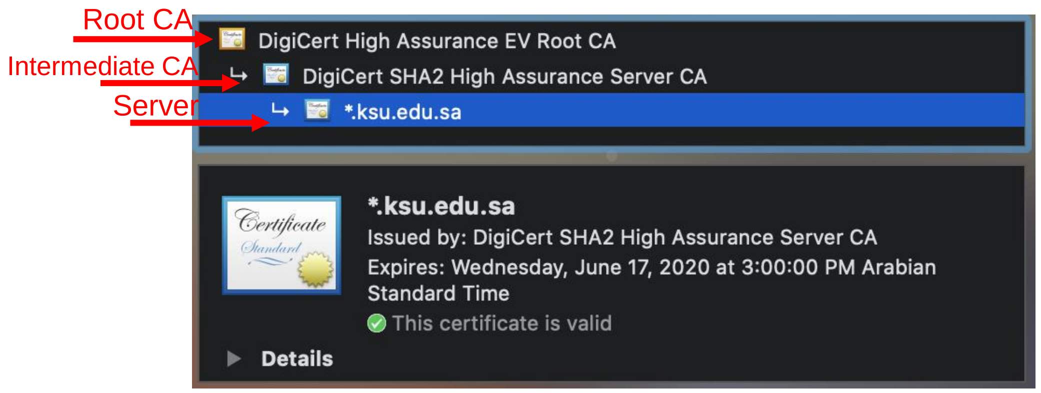
Hierarchy of Certificate Authorities (CAs):



Hierarchical Approach

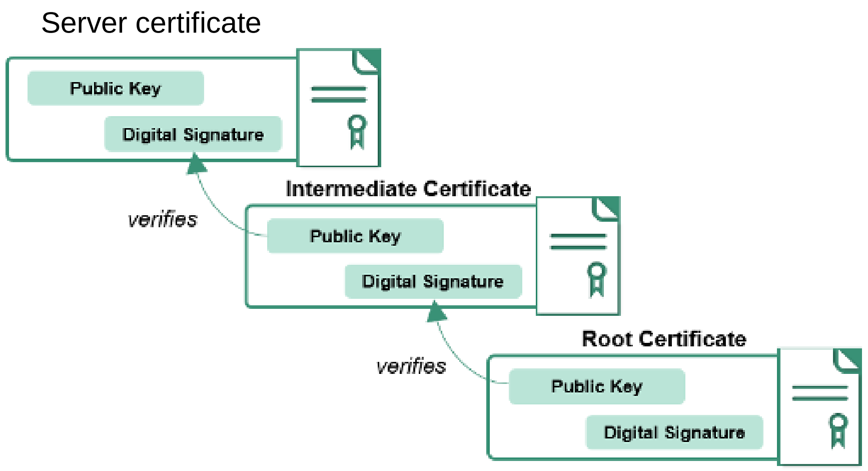
- Single CA certifying every public key is impractical
- The CAs operate through a strict hierarchical organization
  - The CAs at the top of the hierarchy are known as Root CAs
  - The CAs below the root are generally referred to as Intermediate-Level CAs.
- Your computer comes pre-loaded with the public keys for the root CAs.
  - For example, Verisign
- Root CA signs certificates for Intermediate-Level CAs, Intermediate-Level CAs sign certificates for individual networks, and so on

Hierarchical Approach



DigiCert High Assurance EV Root CA
↳ DigiCert SHA2 High Assurance Server CA
↳ *.ksu.edu.sa

*.ksu.edu.sa
Issued by: DigiCert SHA2 High Assurance Server CA
Expires: Wednesday, June 17, 2020 at 3:00:00 PM Arabian Standard Time
This certificate is valid
Details



Example:

To verify the certificate for https://ksu.edu.sa, the browser follows these steps:

1. Checks whether the **root CA** is on the **browser's trusted CAs list**. If so, the browser already has the root CA's public key, if not, a warning is displayed.
2. Verify the **intermediate CA's certificate** using the root CA's public key.
3. **Verify the server's certificate** using the intermediate's CA's public key.

Obtaining a Certificate

User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified
- No party other than the certification authority can modify the certificate without this being detected

- If both users (A, B) are with same CA
  - A can directly use CA Public-key to verify cert.
- If A with CA X_1, B with CA X_2
  - A obtains cert. of X_2 signed by X_1, get Public-key of X_2
  - A obtains cert. of B signed by X_2, get Public-key of B

Obtaining User's Certificate

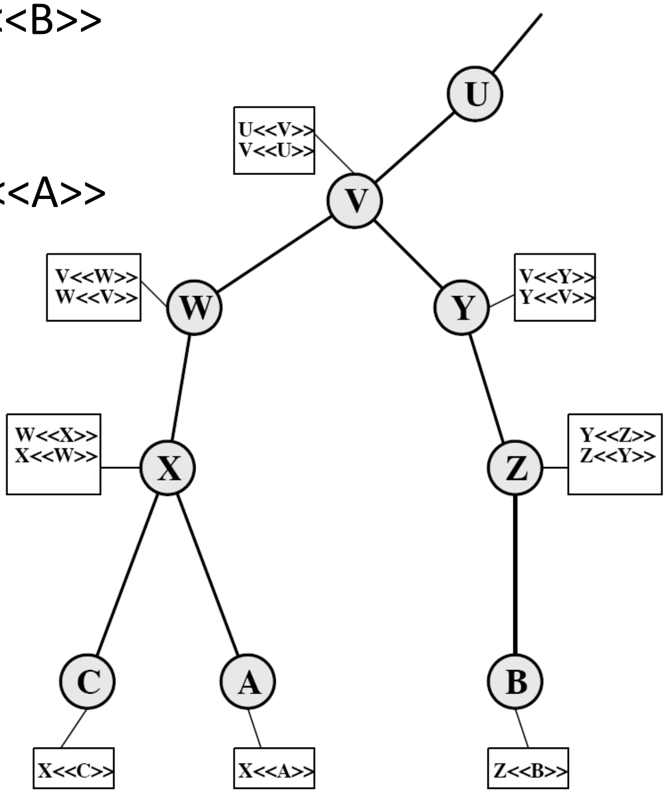
- A obtain B's PU
  - X_1 <<X_2>> X_2 <<B>>
- B obtain A's PU
  - X_2 <<X_1>> X_1 <<A>>
- If N CA's in the path
  - X_1 <<X_2>> X_2 <<X_3>> … X_N <<B>>

Obtaining User's Certificate

A get PU of B
X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>

B get PU of A
Z<<Y>>Y<<V>>V<<W>>W<<X>>X<<A>>

Forward certificate ⟶
Reverse certificate ⟶



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

Reading Assignment

- Textbook
  - Chapter 14
    - 14.1, 14.2, 14.3, 14.4
  - Chapter 15
    - 15.1, 15.2, 15.3