ECE-GY 9383
Special Topics in Network Security

3 - Key distribution

Slides credits: Shivendra S. Panwar, Fraida Fund

CSE/ECE zjzhao

NYU | TANDON SCHOOL OF ENGINEERING

In this lecture

- Using symmetric encryption to distribute symmetric keys
- Kerberos
- Using asymmetric encryption to distribute symmetric keys
- Certificates
- Public key infrastructure

(**Reference:** Chapter 4, Stallings)

Using symmetric encryption to distribute symmetric keys

For symmetric encryption:

- Both parties must possess the secret key
- Frequent key changes limit the amount of data compromised if key is leaked

→ We need a practical, secure method to distribute the key to the involved parties

Key distribution options

1. User A selects key and *physically* delivers it to User B
2. A third party C selects key and *physically* delivers it to User A and User B

   Options above are not practical in many circumstances.

3. User A and User B have an existing shared secret key. User A encrypt a new key using the old key, then transmits the encrypted new key to User B

   This option leaves new key vulnerable in case old key is compromised.

4. User A and User B each have an encrypted connection to a third party C. C chooses a key and delivers it on the encrypted links to A and B.

   Option 4 is most often used for end-to-end encryption.

Two kinds of keys used in Option 4

- Session key: two communication endpoints establish a logical connection. For the duration of connection, all data is encrypted with one-time session key, which is destroyed at the end of the session.

- Permanent key: used between two endpoints for use in distributing session keys.

Using a key distribution center (KDC)

KDC is a necessary element of option 4

1. User A wants to connect to User B. User A sends a connection request to KDC, encrypted with a permanent key shared with KDC.
2. If KDC approves request, it generates session key, then
   - encrypts session key with KDC-A’s permanent key, and sends to User A.
   - also encrypts session key with KDC-B’s permanent key and sends to User B.
3. User A and User B exchange messages encrypted with temporary session key.

Kerberos

Kerberos

- Key distribution and user authentication service developed at MIT
- 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

Two versions are in use

- Version 4 implementations still exist, although this version is being phased out
- Version 5 corrects some of the security deficiencies of the previous version and has been issued as IETF RFC 4120

Kerberos (cont'd)

Key distribution and user authentication service using symmetric encryption

Scenario: users at workstations access services on servers, via network. Services should only be available to authorized users.

Potential threats:

- User may gain access to workstation and pretend to be another user operating at that workstation
- User may spoof network address, making messages from their workstation appear to be coming from another workstation.
- User may eavesdrop, use replay attack to access unauthorized services.

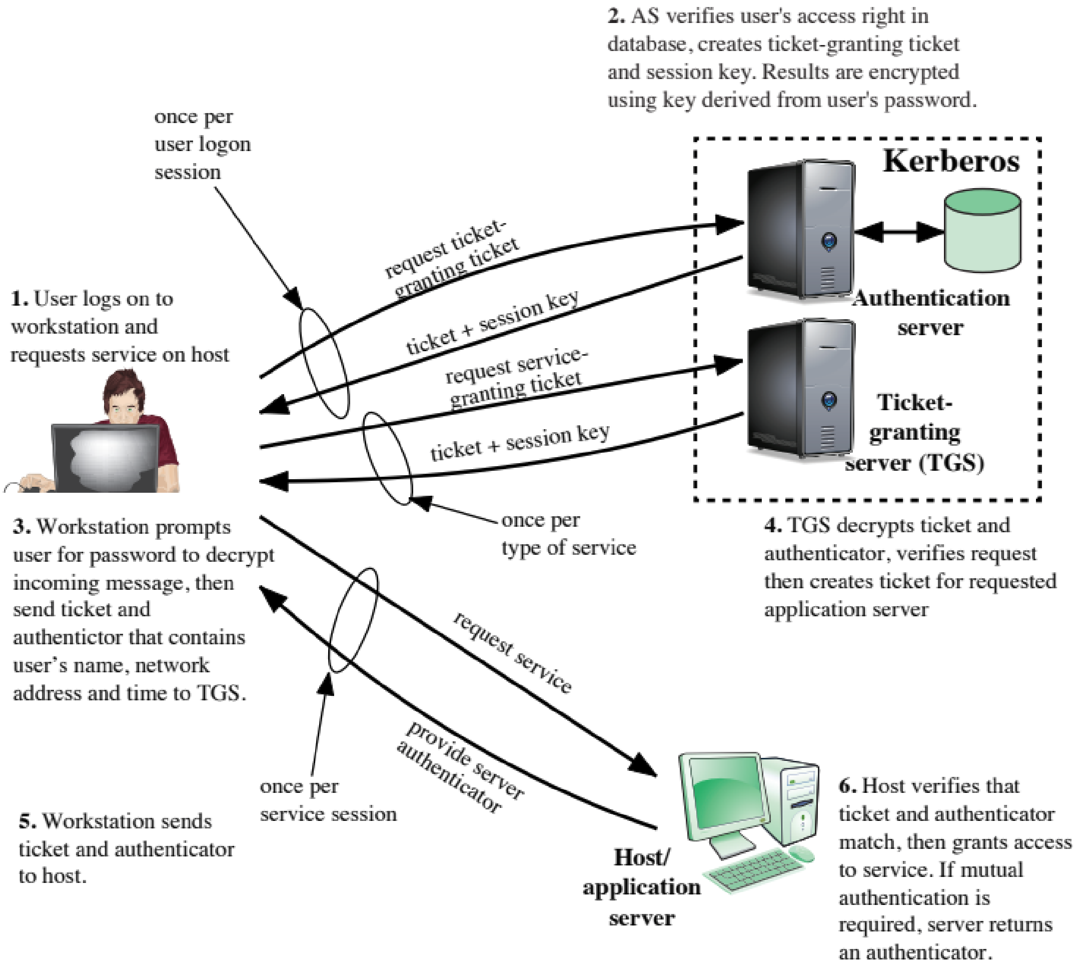


Figure 4.2 Overview of Kerberos

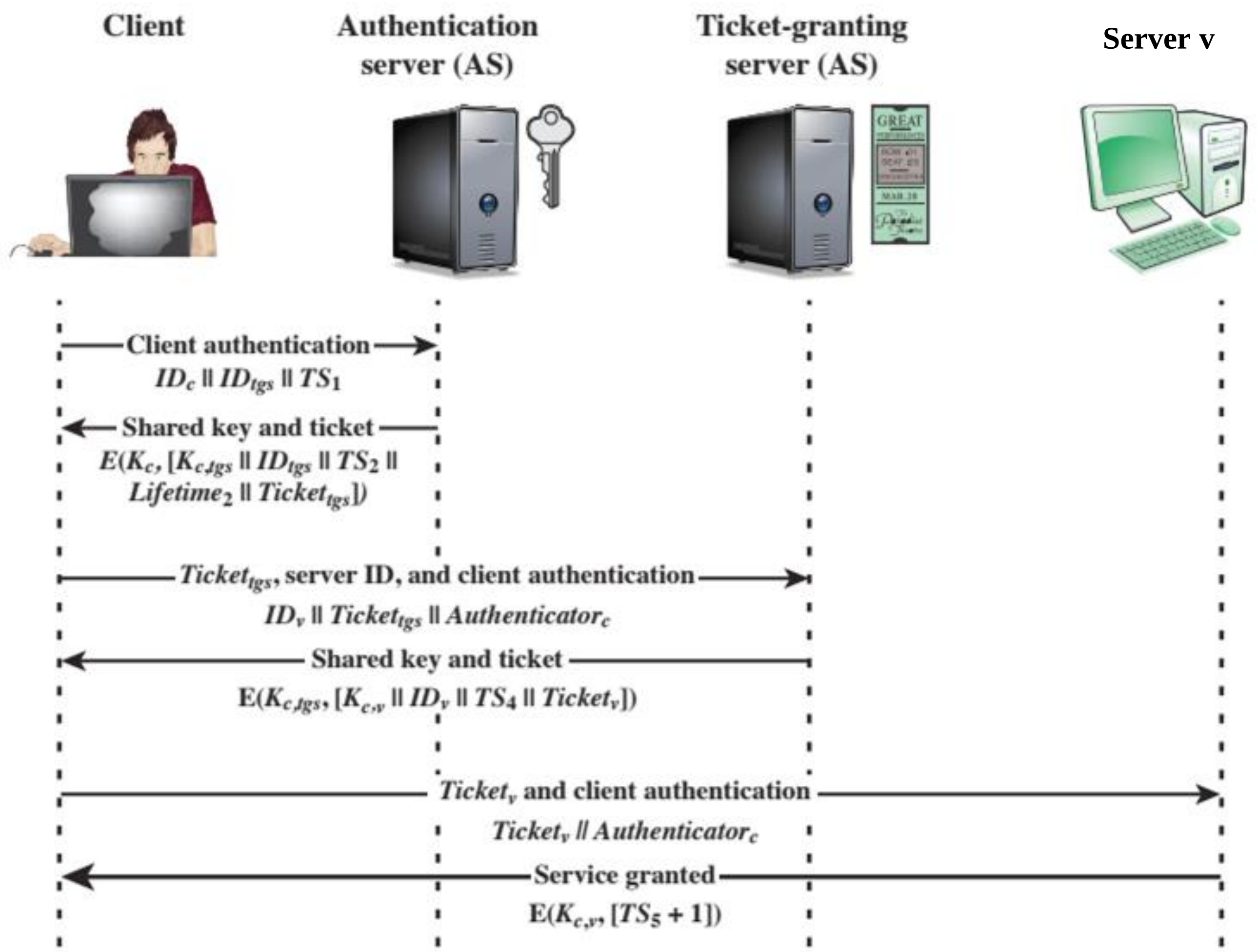
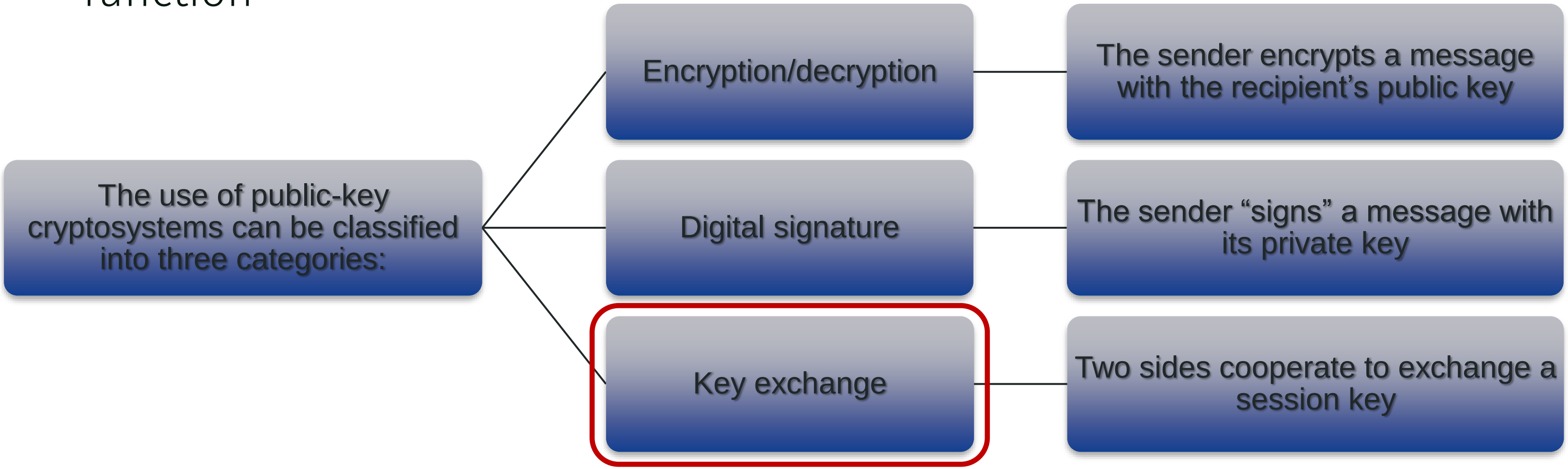


Figure 4.3 Kerberos Exchanges

Key distribution with public-key encryption

Applications for public-key cryptosystems

- Public-key systems are characterized by the use of a cryptographic type of algorithm with two keys, one held private and one available publicly
- Depending on the application, the sender uses either the sender’s private key, the receiver’s public key, or both to perform some type of cryptographic function



Key distribution using asymmetric encryption

- Two aspects to the use of public-key encryption in this regard:
  1. The distribution of public keys
  2. The use of public-key encryption to distribute secret keys

- Public-key certificate
  - Consists of a public key plus a user ID of the key owner, with the whole block signed by a trusted third party

    (Third party is a certificate authority (CA) that is trusted by the user community)
  - A user can present his/her public key to the CA in a secure manner to obtain a certificate. And then publish the certificate
  - Anyone needing this user’s public key can obtain the certificate and verify that it is valid by way of the attached trusted signature

Certificates

Why certificates?

- An attacker can forge a public key, e.g. announce “I am User X and here is my public key” when it is not actually User X.
- Unless this forgery is detected, other users will think they are securely communicating with User X.
- Solution: public key certificate signed by trusted third party

Public key certificates

- User certificates are created by a trusted **certification authority** (CA)
- User presents public key to CA
- CA creates and **signs** certificate with its private key.

  (See: digital signature process ➔)
- Any user can verify that it is valid using CA's public key.

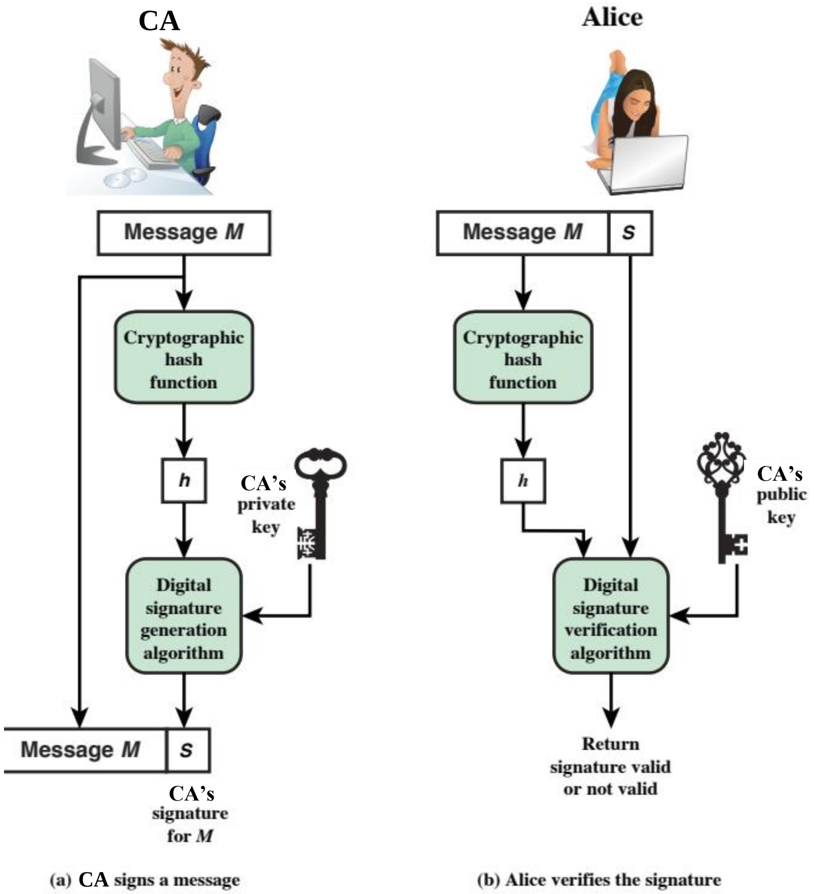
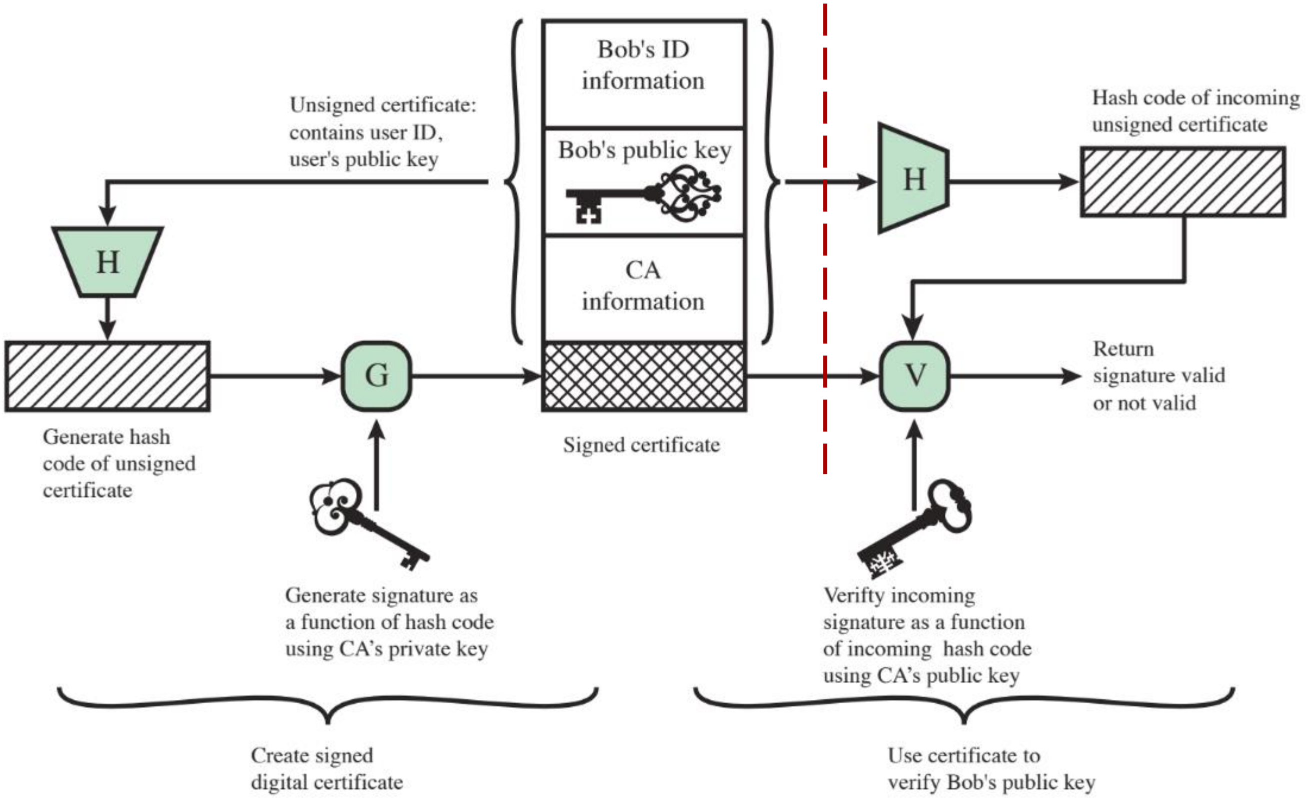


Figure 3.15 Simplified Depiction of Essential Elements of Digital Signature Process

Public-key certificate use (Figure 4.4)



Contents of a user certificate

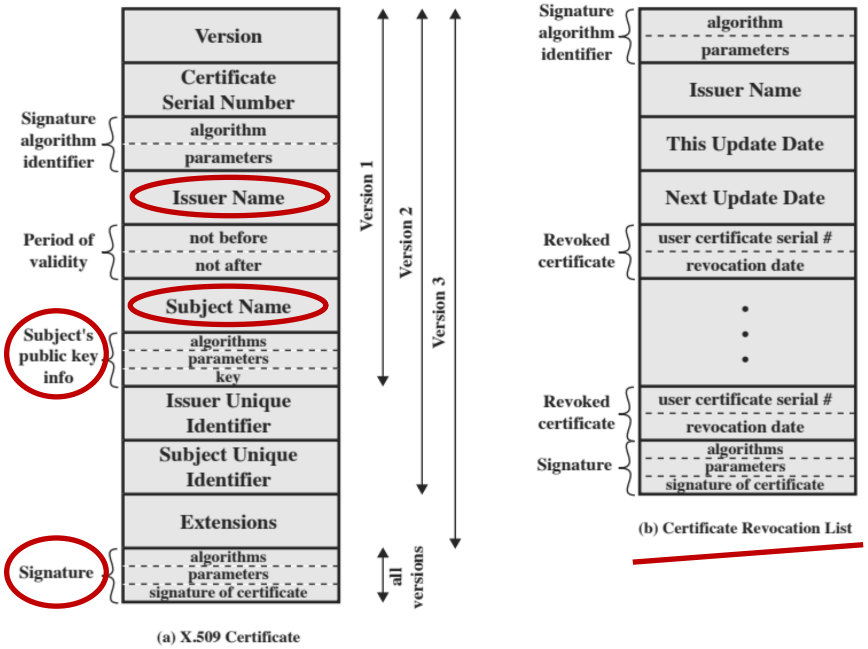


Figure 4.5 X.509 Formats

Using a chain of certificates - overview

What if:

- User A has certificate from CA1
- User B has certificate from CA2
- User A does not know the public key of CA2 → A can't get B's public key (signed by CA2)

If CA1 and CA2 securely exchange public keys, then A can get B's public key:

1. User A gets certificate of CA2 signed by CA1. Then A can get CA2's public key from its certificate and verify it.
2. A gets certificate of B signed by CA2. Now A can verify CA2's signature and get B's public key.

Using chain of certificates - expression

- This chain, as A getting B's public key, is expressed as:

  CA1 << CA2 >> CA2 << B >>

  And B can get A's public key with reverse chain:

  CA2 << CA1 >> CA1 << A >>

- An arbitrarily long path of CAs can produce a chain.

Using chain of certificates – two types of certificates

Directory for each CA includes two types of certificates -

- **Forward certificates**: Certificates of this CA generated by other CAs
  - CAx <<CA1>>
- **Reverse certificates**: Certificates of other CAs generates by this CA
  - CA1 <<CAx>>

Example:

- User A can establish certification path to User B:

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

- User B can establish certification path to User A:

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

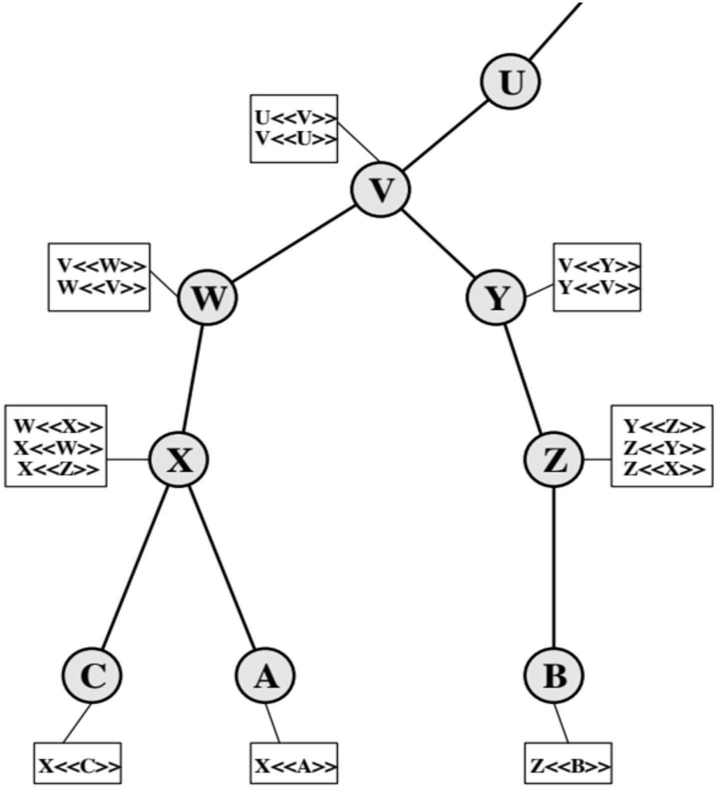
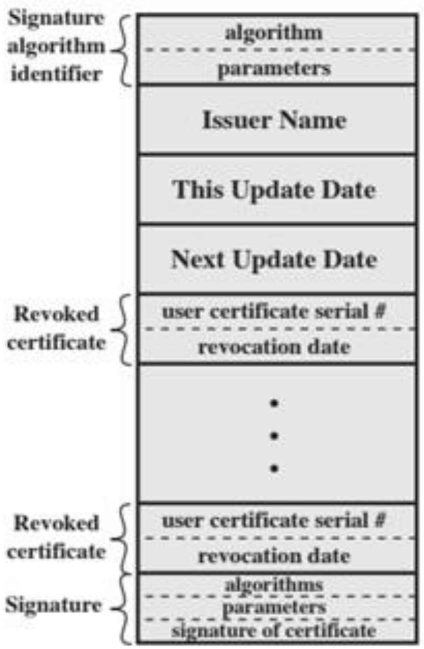


Figure 4.6 X.509 CA Hierarchy: a Hypothetical Example

Certificate revocation

- Need to revoke a certificate if:
  1. User's private key is compromised
  2. User is no longer certified by CA (e.g. user's name has changed)
  3. CA's certificate is compromised

- Each CA posts list of revoked certificates, that users can check when verifying a certificate.



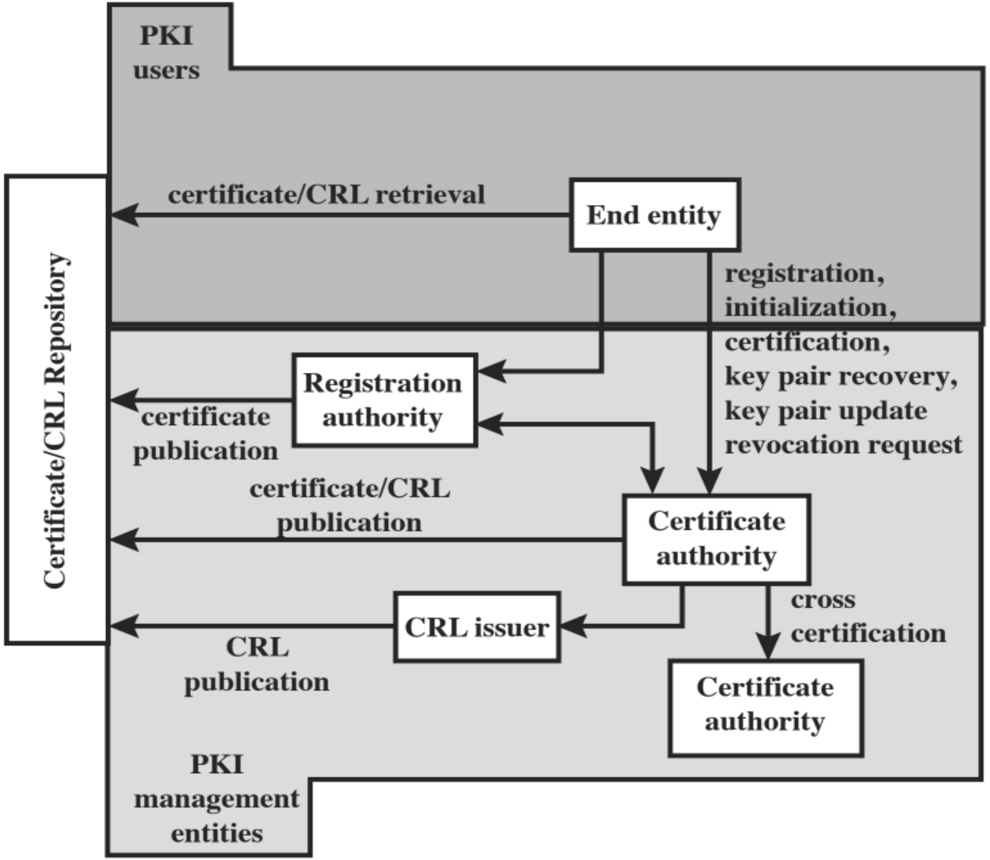
(b) Certificate Revocation List

Public key infrastructure (PKI)

Key entities

- **End entity**: end users, devices, or any other entity identified in "subject" field of PK certificate
- **CA**: issuer of certificates and certificate revocation lists (CRLs)
- **Registration authority (RA)**: optional administrative "helper" to CA
- **CRL issuer**: optional component that CA can delegate publication of CRLs to
- **Repository**: store of certificates and CRLs that can be retrieved by end entities

PKIX architectural model (Figure 4.7)



Summary: what we have discussed so far

- Encryption with public/private key pair
- Digital signatures with public key cryptography
- Using symmetric encryption to distribute symmetric keys
- Using asymmetric/public-kay encryption to distribute symmetric keys
- Certificates, public key infrastructure

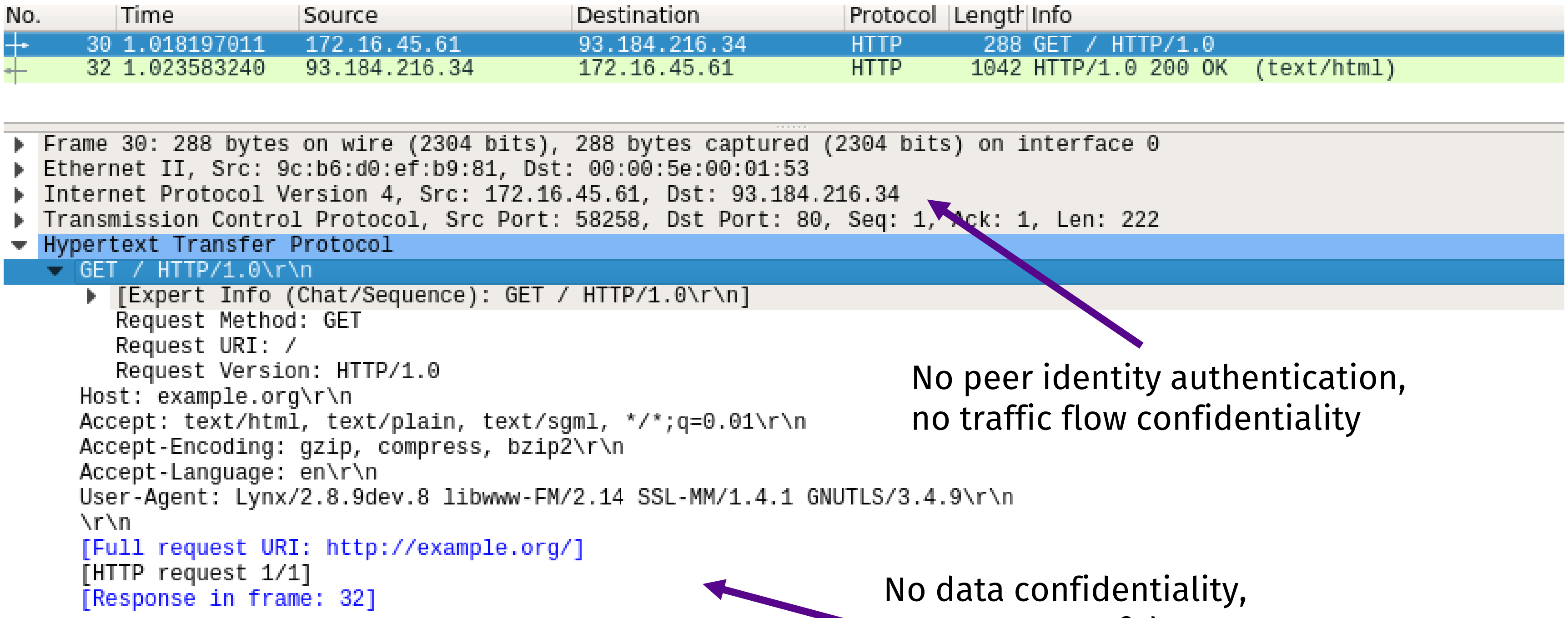
Next: what we can do with encryption basics?

- Internet is not secure
- Internet protocols are hierarchically structured
- Venerable points in protocol layers
- Venerable points in network infrastructure

“Basic” networking

No.	Time	Source	Destination	Protocol	Length	Info
30	1.018197011	172.16.45.61	93.184.216.34	HTTP	288	GET / HTTP/1.0
32	1.023583240	93.184.216.34	172.16.45.61	HTTP	1042	HTTP/1.0 200 OK (text/html)

```
Frame 30: 288 bytes on wire (2304 bits), 288 bytes captured (2304 bits) on interface 0
Ethernet II, Src: 9c:b6:d0:ef:b9:81, Dst: 00:00:5e:00:01:53
Internet Protocol Version 4, Src: 172.16.45.61, Dst: 93.184.216.34
Transmission Control Protocol, Src Port: 58258, Dst Port: 80, Seq: 1, Ack: 1, Len: 222
Hypertext Transfer Protocol
  GET / HTTP/1.0\r\n
    [Expert Info (Chat/Sequence): GET / HTTP/1.0\r\n]
    Request Method: GET
    Request URI: /
    Request Version: HTTP/1.0
  Host: example.org\r\n
  Accept: text/html, text/plain, text/sgml, */*;q=0.01\r\n
  Accept-Encoding: gzip, compress, bzip2\r\n
  Accept-Language: en\r\n
  User-Agent: Lynx/2.8.9dev.8 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.4.9\r\n
  \r\n
  [Full request URI: http://example.org/]
  [HTTP request 1/1]
  [Response in frame: 32]
```



No peer identity authentication,
no traffic flow confidentiality

No data confidentiality,
no assurance of data
integrity

What kind of security assurances do I have?

Can I **be confident** that I was indeed corresponding with example.org, and that the HTTP response came from them?

Can the remote site **be confident** about who they were corresponding with, and that the HTTP request came from me?

Is the data inside the HTTP request and response **confidential**?

Is the fact of my having visited example.org, and the basic parameters of my visit (e.g. when, for how long, how much data was transferred) **confidential**?

Can I be sure that the data I received in the HTTP response was not tampered with (**data integrity**)?

Can the remote site be confident that the HTTP request they received was **not tampered with**?

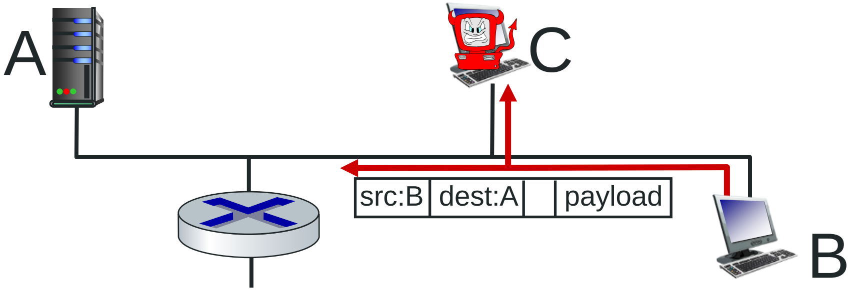
Could the remote site have **restricted my access** based on my network address?

Packet sniffing

Anybody with physical access to the local network can “listen” to all the traffic traversing a link

Ethernet MAC flooding attack can turn a switch into a hub -

- Normally, a switch learns which host is on which port and then sends unicast traffic only to the network segment where that host is connected.
- Attacker can generate fake traffic to fill MAC address table on switch with fake entries, so there is no memory for real entries.
- Traffic will be flooded out of all ports.

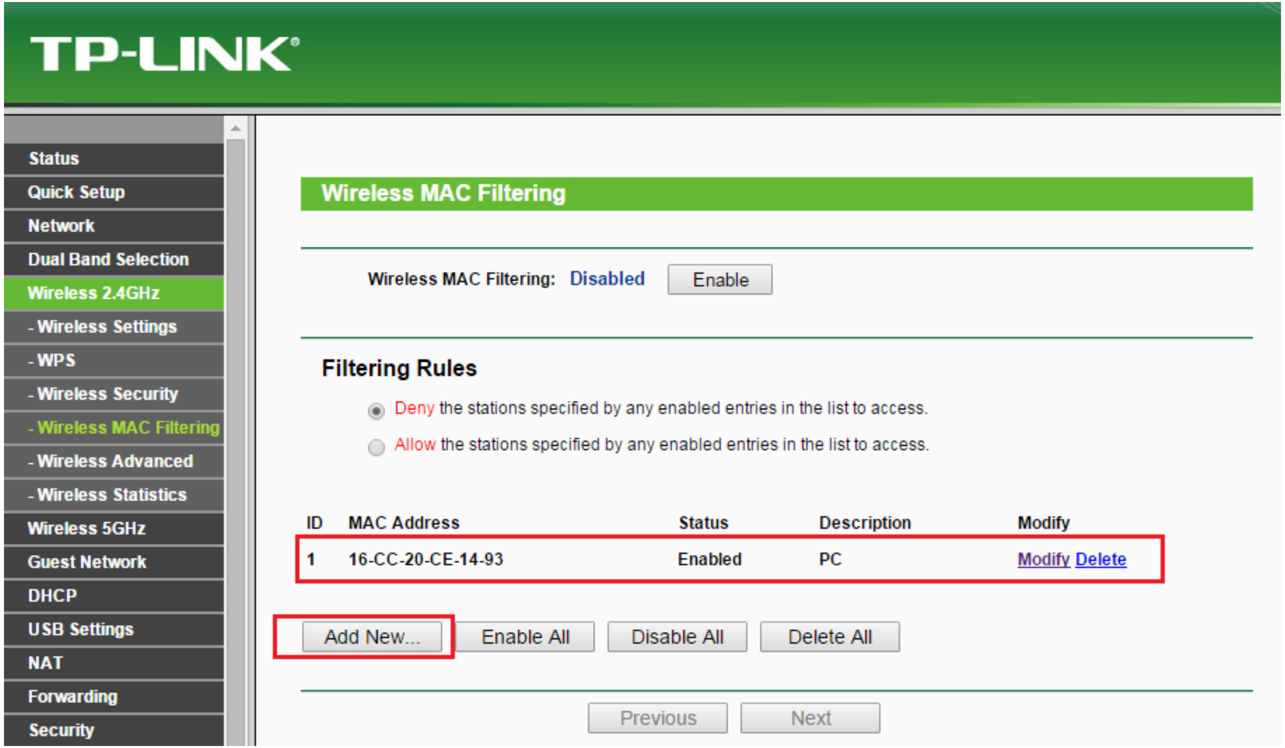


MAC address spoofing

Trivial to change the MAC address of a network interface card in software

Implications -

- Bypass any MAC address filtering.
- Impersonate another host.



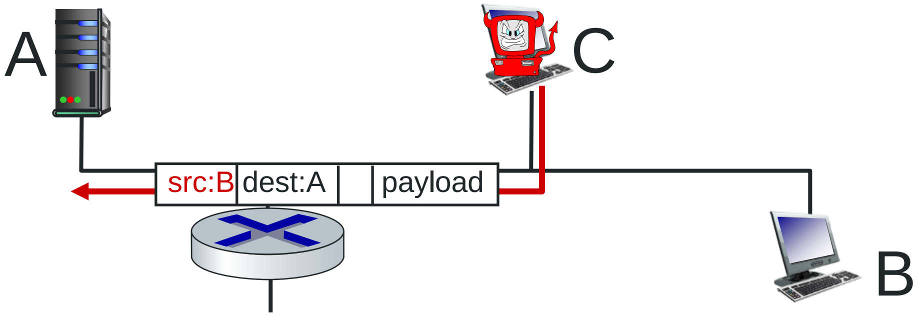
IP address spoofing

There is no source IP authentication

"Anyone" can put an arbitrary source IP into a packet and send it out

Implications -

- Anonymous attacks - no accountability
- Can't get non-repudiation from addresses alone

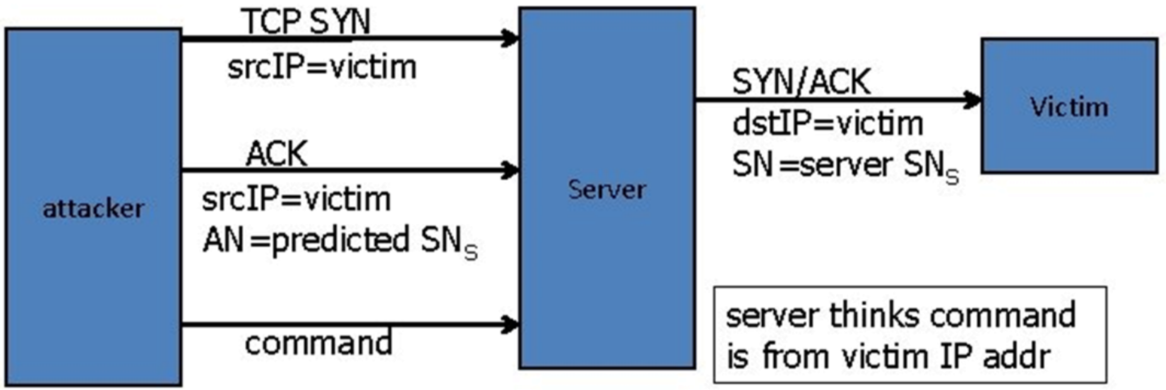


TCP connection spoofing

- Injecting IP packets with spoofed IP address insufficient - segments with wrong sequence numbers will be discarded.
- To impersonate: need to guess a reasonable sequence number
  Easy if you can sniff; in some situations, it can be done even without access to network – not an easy one with ISN randomization

Ex. based on a slide

from CS155 at Stanford



Where to put security facilities?

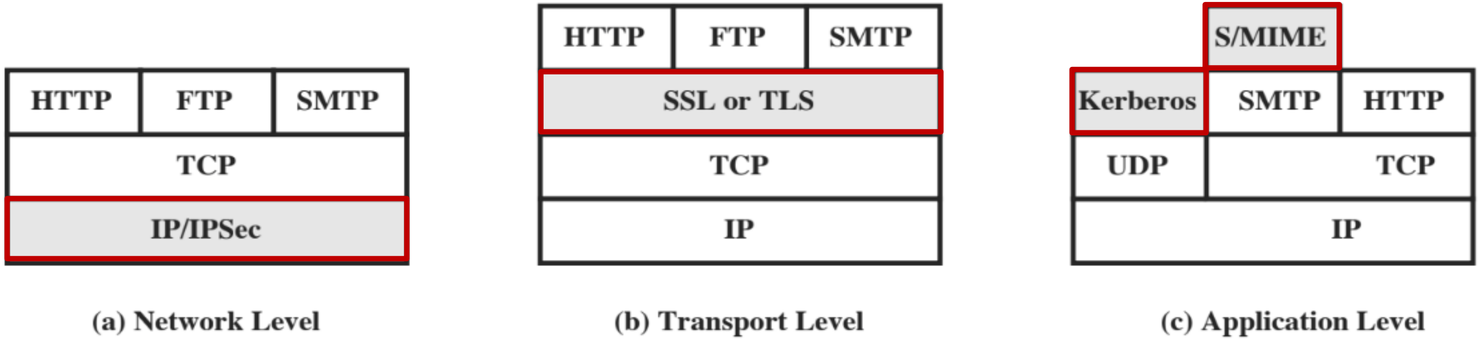


Figure 6.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack