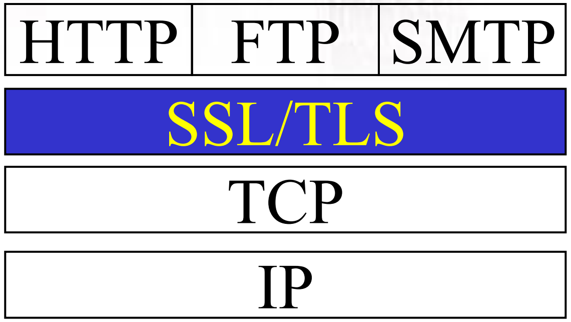
Web Security
SSL/TLS and Certificates

Chun-Jen (James) Chung

Arizona State University

What is SSL/TLS?

- SSL – Secure Socket Layer
- TLS – Transport Layer Security

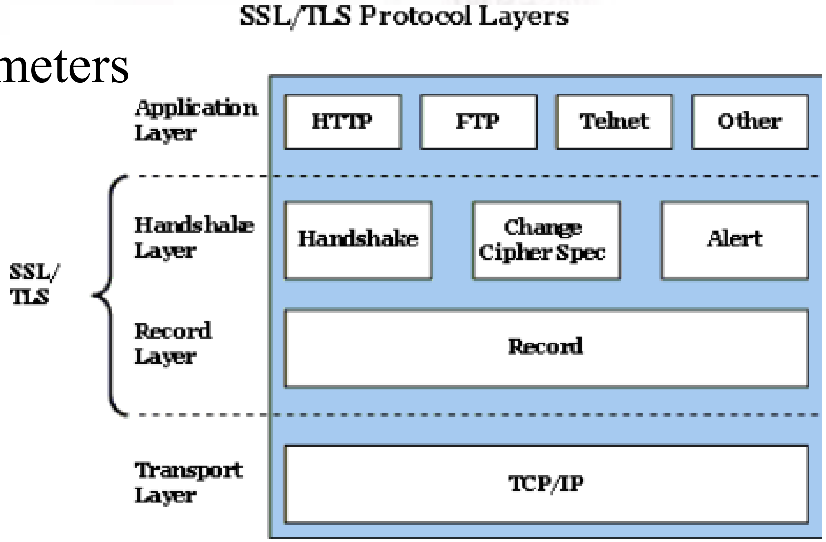


At session (or above) layer

- Both provide a secure transport connection (data encryption and authentication) between applications and servers.
- SSL version 3.0 has been implemented in many web browsers and widely used on the Internet.
- TLS can be viewed as SSL v3.1
    - Same protocol design, different algorithm

SSL/TLS Components

- Handshake Protocol
  - negotiation of security algorithms and parameters
  - Use public-key cryptography to *establish a shared secret key* between client and server
  - server authentication and optionally client authentication


SSL/TLS Protocol Layers

- Record Protocol
  - Fragmentation/compression/encryption
  - Using secret key to provide message authentication and integrity protection
- Alert Protocol
  - error messages (fatal alerts and warnings)
- Change Cipher Spec Protocol
  - a single message that indicates the end of the SSL handshake

Sessions and Connections

- Connection:
  - A peer-to-peer relationships in the transport layer. Every connection is associated with one session.
- Session:
  - An association between a client and a server created by the handshake protocol.
  - Define a set of cryptographic security parameters, which can be shared among multiple connections.
  - Avoid the expensive negotiation of new security parameters for each connection.

SSL Statefullness

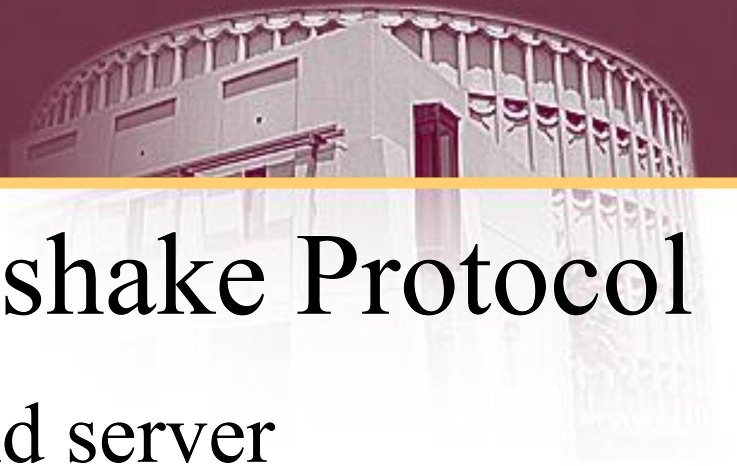
- Multiple secure connections in a session
- Connections of the same session share the session state
- Current operating state for read and write (receive and send)
- Pending read and write states created during Handshake Protocol

Session State

- session identifier
  - arbitrary byte sequence chosen by the server to identify the session
- peer certificate
  - X509 certificate of the peer; may be null
- compression method
- cipher spec
  - encryption (null, DES, 3DES) and MAC (MD5, SHA-1) algorithm used, and cryptographic attributes (e.g., hash size, IV size, …)
- master secret
  - 48-byte secret shared between the client and the server
- is resumable
  - a flag indicating whether the session can be used to initiate new connections
- connection states

Connection State

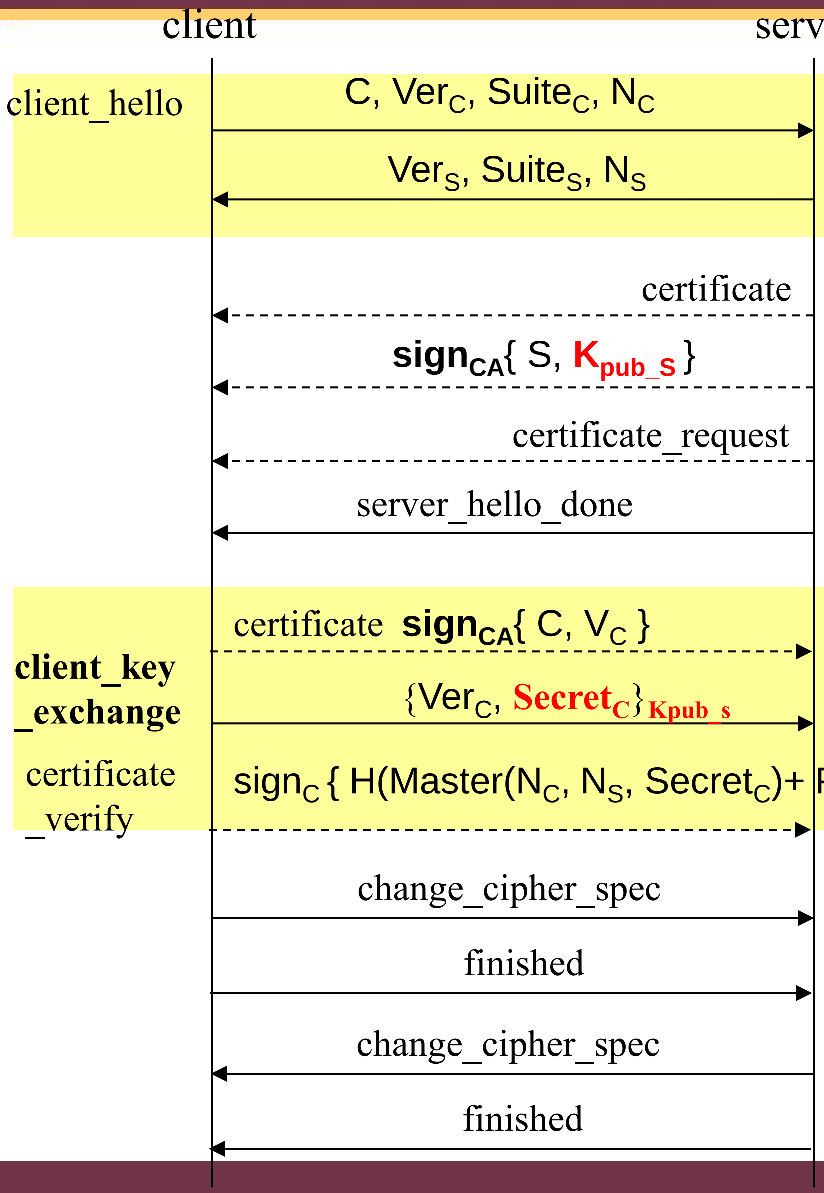
- server and client random
  - random byte sequences chosen by the server and the client for every connection
- server/client write MAC secret
  - secret key used in MAC operations on data sent by the server/client
- server/client write key
  - secret encryption key for data encrypted by the server/client
- initialization vectors
  - an IV is maintained for each encryption key if DES CBC mode is used
- sending and receiving sequence numbers
  - sequence numbers are 64 bits long
  - reset to zero after each Change Cipher Spec message



SSL/TLS Handshake Protocol

- Two parties: client and server
- Negotiate version of the protocol and the set of cryptographic algorithms to be used
  - Interoperability between different implementations of the protocol
- Authenticate client and server (optional)
  - Use *digital certificates* to learn each other’s *public keys* and verify each other’s identity
- Use **public keys** to establish a shared secret

SSL Handshake Protocol



client

server

client_hello

C, Ver_C, $Suite_C$, N_C

Ver_S, $Suite_S$, N_S

server_hello

Phase 1: Negotiation of the session ID, protocol version, cipher suites (RSA or DH), and exchange of initial random numbers

certificate

server_key_exchange

$sign_{CA}${ S, K_{pub_S} }

certificate_request

server_hello_done

Phase 2: Server may send its certificate and key exchange message, and it may request the client to send a certificate. Server signals end of hello phase.

certificate **$sign_{CA}$**{ C, V_C }

client_key_exchange

{Ver_C, $Secret_C$}$_{Kpub\_s}$

certificate_verify

$sign_C$ { H(Master(N_C, N_S, $Secret_C$)+ Pad_2 +H(Msgs +C+ Master(N_C, N_S, $Secret_C$) + Pad_1)) }

Phase 3: Client sends certificate if requested and may send an explicit certificate verification message. Client always sends its key exchange message.

Phase 4: Change cipher spec and finish handshake

change_cipher_spec

finished

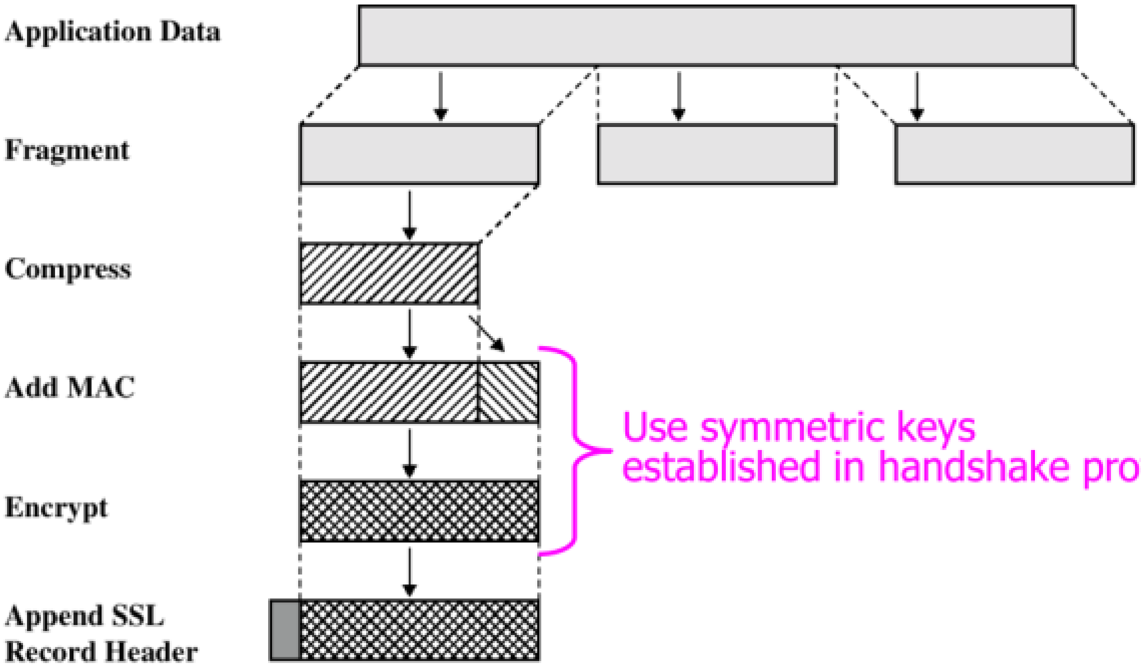
{H(Master(N_C, N_S, $Secret_C$) + Pad_2 + H(Msgs + **C** + Master(N_C, N_S, $Secret_C$) + Pad_1)) }Master(N_C,N_S, $Secret_C$)

change_cipher_spec

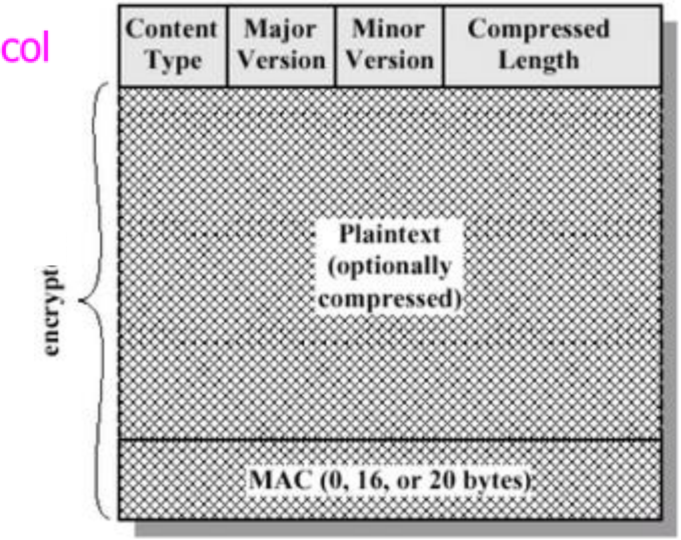
finished

{H(Master(N_C, N_S, $Secret_C$) + Pad_2 + H(Msgs + **S** + Master(N_C, N_S, $Secret_C$) + Pad_1)) }Master(N_C,N_S, $Secret_C$)

SSL/TLS Record Protocol



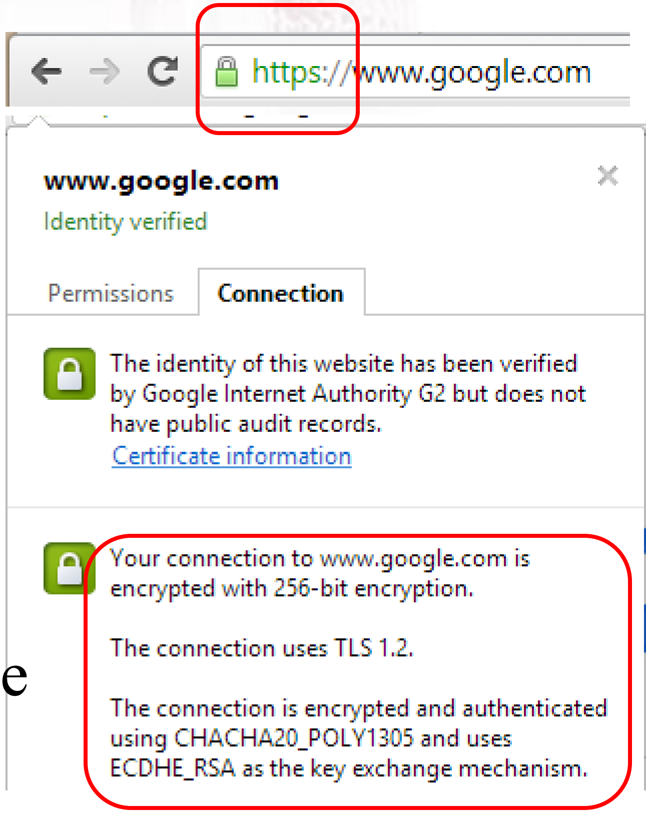
Use symmetric keys established in handshake protocol



TLS uses HMAC

HTTPS

- HTTP Secure, HTTP over SSL, HTTP over TLS
- HTTPS connections need their own port - port 443
- Require X.509 certificates to check the identity of the peer
- Require Certificate Authority (CA) and Public-key Infrastructure (PKI) to verify the relation between owner of a certificate and the certificate, as well as to generate, sign, and administer the validity of certificates



www.google.com

Identity verified

The identity of this website has been verified by Google Internet Authority G2 but does not have public audit records.

Certificate information

Your connection to www.google.com is encrypted with 256-bit encryption.

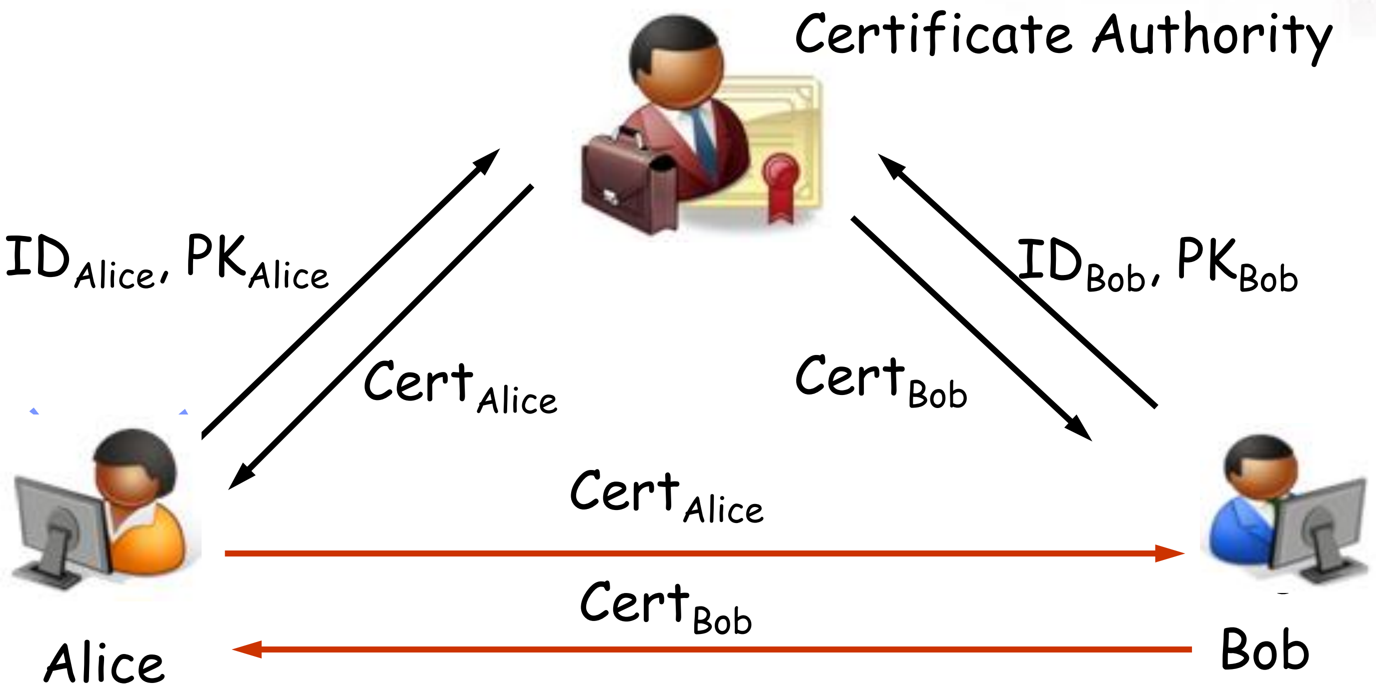
The connection uses TLS 1.2.

The connection is encrypted and authenticated using CHACHA20_POLY1305 and uses ECDHE_RSA as the key exchange mechanism.

Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - $\mathrm{SIG}_{\mathrm{Alice}}$(“Bob”, PK_{B})
- Common approach: certificate authority (CA)
  - An agency responsible for certifying public keys
  - Browsers are pre-configured with 100+ of trusted CAs
  - A public key for any website in the world will be accepted by the browser if certified by one of these CAs

Public-Key Certificates



Certificate Authority

ID_{Alice}, PK_{Alice}

$Cert_{Alice}$

ID_{Bob}, PK_{Bob}

$Cert_{Bob}$

$Cert_{Alice}$

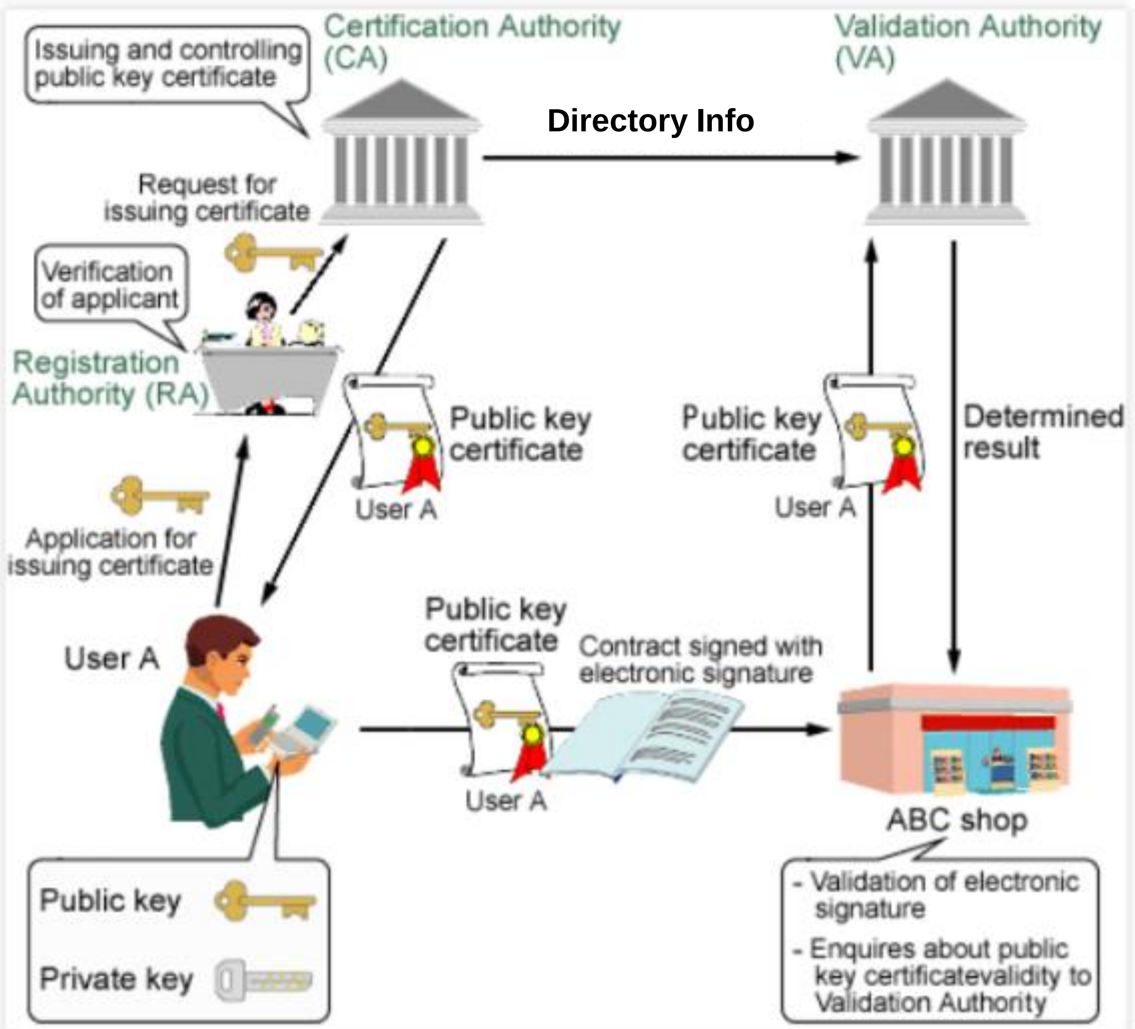
$Cert_{Bob}$

Alice

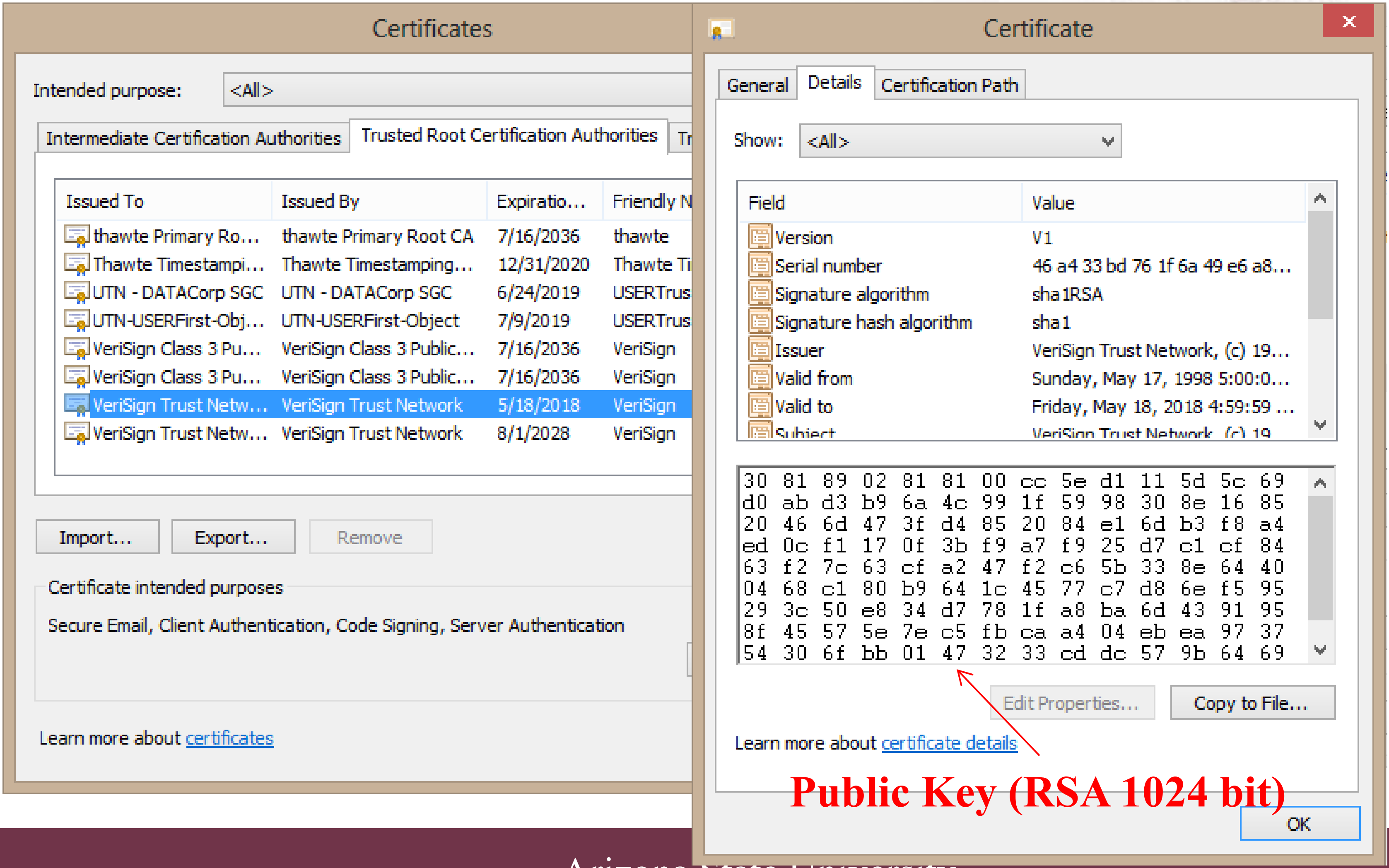
Bob

$Cert_{Alice}$ = < ID_{Alice}, SN, Expiry, PK_{Alice}, $Sig_{CA}(ID_{Alice}$, SN, Expiry, $PK_{Alice})$ >

Public Key Infrastructure


Issuing and controlling public key certificate
Certification Authority (CA)
Validation Authority (VA)
Directory Info
Request for issuing certificate
Verification of applicant
Registration Authority (RA)
Public key certificate
User A
Public key certificate
User A
Determined result
Application for issuing certificate
User A
Public key certificate
Contract signed with electronic signature
User A
ABC shop
- Validation of electronic signature
- Enquires about public key certificatevalidity to Validation Authority
Public key
Private key

Pre-installed Trusted CAs



Certificates

Intended purpose: <All>

Intermediate Certification Authorities | Trusted Root Certification Authorities | Tr

Issued To	Issued By	Expiratio...	Friendly N
thawte Primary Ro...	thawte Primary Root CA	7/16/2036	thawte
Thawte Timestampi...	Thawte Timestamping...	12/31/2020	Thawte Ti
UTN - DATACorp SGC	UTN - DATACorp SGC	6/24/2019	USERTrus
UTN-USERFirst-Obj...	UTN-USERFirst-Object	7/9/2019	USERTrus
VeriSign Class 3 Pu...	VeriSign Class 3 Public...	7/16/2036	VeriSign
VeriSign Class 3 Pu...	VeriSign Class 3 Public...	7/16/2036	VeriSign
VeriSign Trust Netw...	VeriSign Trust Network	5/18/2018	VeriSign
VeriSign Trust Netw...	VeriSign Trust Network	8/1/2028	VeriSign

Certificate intended purposes

Secure Email, Client Authentication, Code Signing, Server Authentication

Learn more about certificates

Certificate

General | Details | Certification Path

Show: <All>

Field	Value
Version	V1
Serial number	46 a4 33 bd 76 1f 6a 49 e6 a8...
Signature algorithm	sha1RSA
Signature hash algorithm	sha1
Issuer	VeriSign Trust Network, (c) 19...
Valid from	Sunday, May 17, 1998 5:00:0...
Valid to	Friday, May 18, 2018 4:59:59 ...
Subject	VeriSign Trust Network, (c) 19

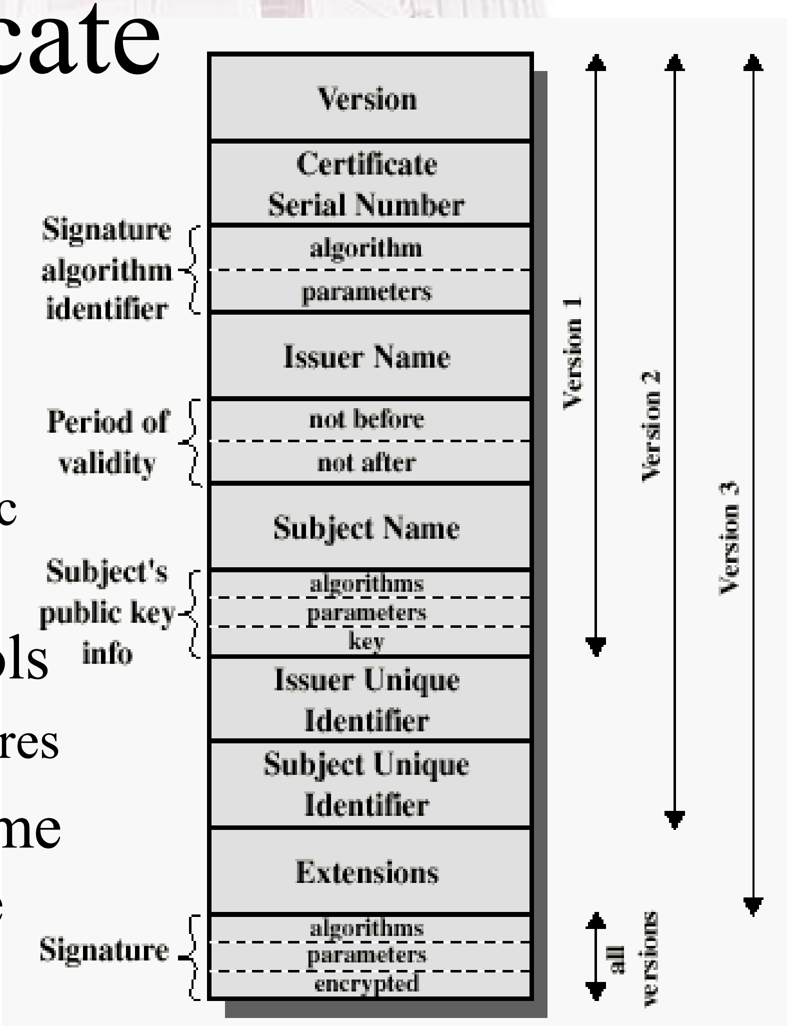
```
30 81 89 02 81 81 00 cc 5e d1 11 5d 5c 69
d0 ab d3 b9 6a 4c 99 1f 59 98 30 8e 16 85
20 46 6d 47 3f d4 85 20 84 e1 6d b3 f8 a4
ed 0c f1 17 0f 3b f9 a7 f9 25 d7 c1 cf 84
63 f2 7c 63 cf a2 47 f2 c6 5b 33 8e 64 40
04 68 c1 80 b9 64 1c 45 77 c7 d8 6e f5 95
29 3c 50 e8 34 d7 78 1f a8 ba 6d 43 91 95
8f 45 57 5e 7e c5 fb ca a4 04 eb ea 97 37
54 30 6f bb 01 47 32 33 cd dc 57 9b 64 69
```

Learn more about certificate details

Public Key (RSA 1024 bit)

X.509 Certificate

- Internet standard (1988-2000)
- Specifies certificate format
  - used in IPsec and SSL/TLS
- Specifies certificate directory service
  - For retrieving other users' CA-certified public keys
- Specifies a set of authentication protocols
  - For proving identity using public-key signatures
- Can use with any digital signature scheme and hash function, but must hash before signing

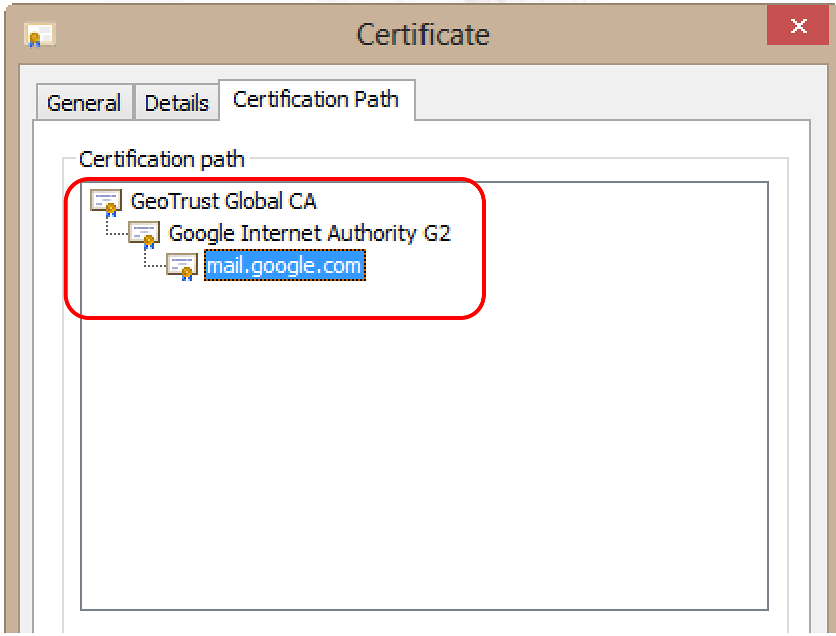


Certificate Example

User Name	User Name: www.google.com
Certificate Version	Certificate Version: V3
Validity Period	Validity Period: Feb 12, 14 – June 11, 14
Serial No	Serial No: 4d cc 87 66 51 3f 02 14
User's Public Key	User's Public Key: RSA (2048 bits)
Other user attributes	Other attributes: e.g. signing algorithm: sha1RSA
CA's name	CA's name: Google Internet Authority G2
CA's signature (of all the above)	CA's signature: 1024-bit data

$\text{Cert}_A = < \text{ID}_A, \text{PK}_A, \text{Validity Period},\ldots \text{Sign}_{CA}(\text{ID}_A, \text{PK}_A, \text{Validity Period}, \ldots) >$

CA Hierarchy



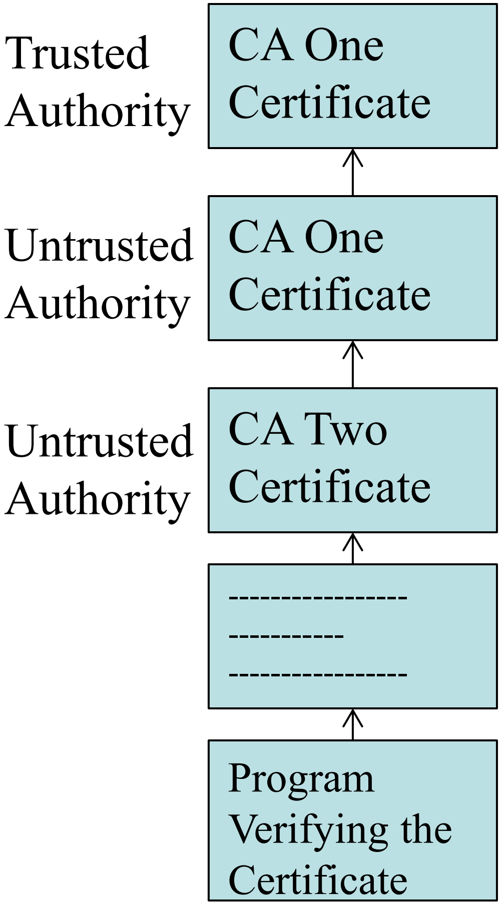
Certificate

Certification path

- GeoTrust Global CA
  - Google Internet Authority G2
    - mail.google.com

- Browsers have several trusted root certificate authorities
- A Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - Certificate "chain of trust"
  - GeoTrust (root) → Google Internet Authority → mail.google.com
- Client (browser) verifies this chain of certificates beginning from the leaf to the root CA.

Verifying a Certificate Chain



Trusted Authority — CA One Certificate

Untrusted Authority — CA One Certificate

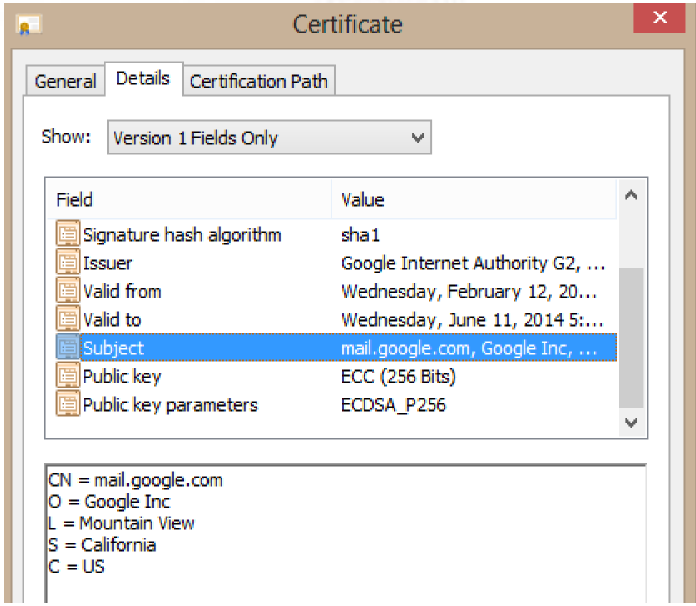
Untrusted Authority — CA Two Certificate

Program Verifying the Certificate

Verify validity period and verify that it is signed by the root CA. Root CA is trusted, verification stop here.

Verify validity period and verify that it is signed by CA One. CA One is not trusted, check the next certificate.

Verify validity period and verify that it is signed by CA Two. CA two is not trusted, check the next certificate.



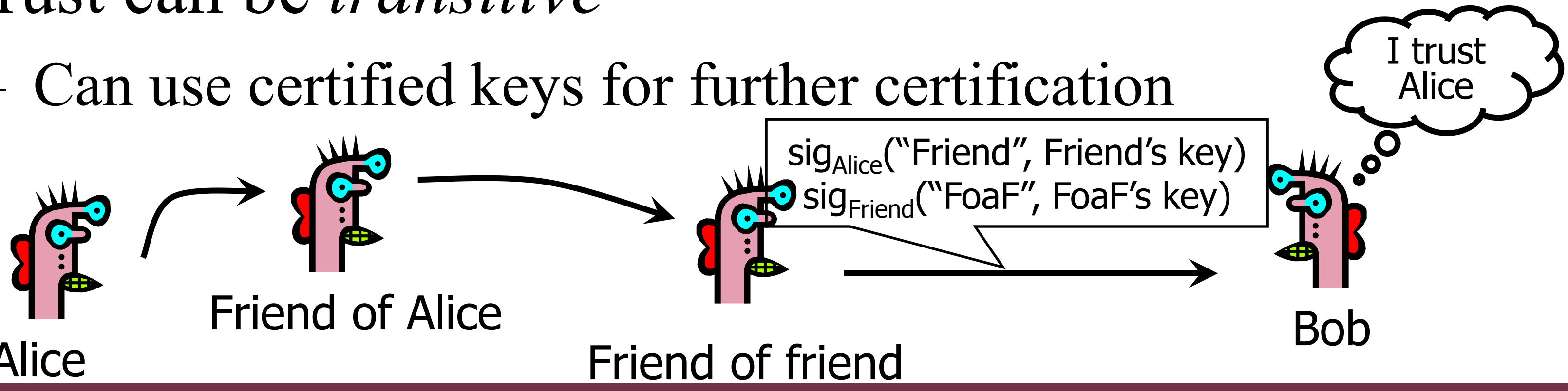
Show: Version 1 Fields Only

Field	Value
Signature hash algorithm	sha1
Issuer	Google Internet Authority G2, ...
Valid from	Wednesday, February 12, 20...
Valid to	Wednesday, June 11, 2014 5:...
Subject	mail.google.com, Google Inc, ...
Public key	ECC (256 Bits)
Public key parameters	ECDSA_P256

CN = mail.google.com
O = Google Inc
L = Mountain View
S = California
C = US

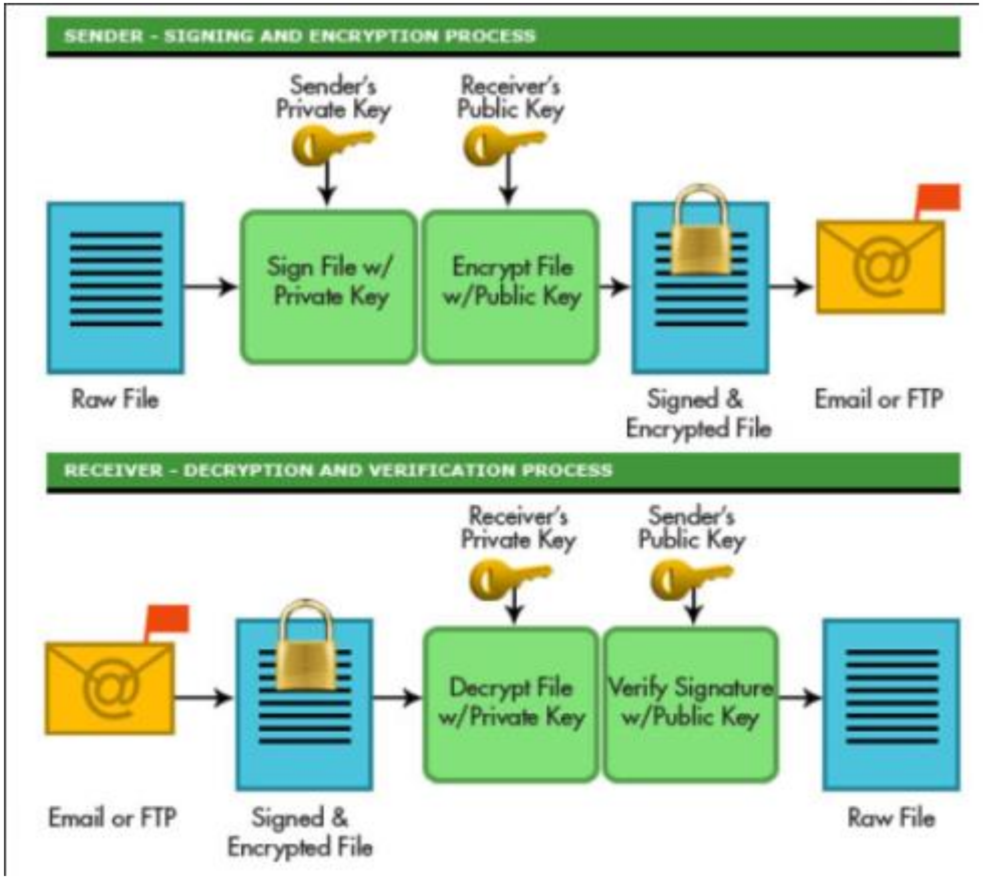
Alternative: Web of Trust

- A decentralized trust model used in PGP (Pretty Good Privacy)
- Instead of a single root certificate authority (centralized), each person has *a set of keys they "trust"*
  - If public-key certificate is signed by *one of the "trusted" keys*, the public key contained in it will be deemed valid
- Trust can be *transitive*
  - Can use certified keys for further certification

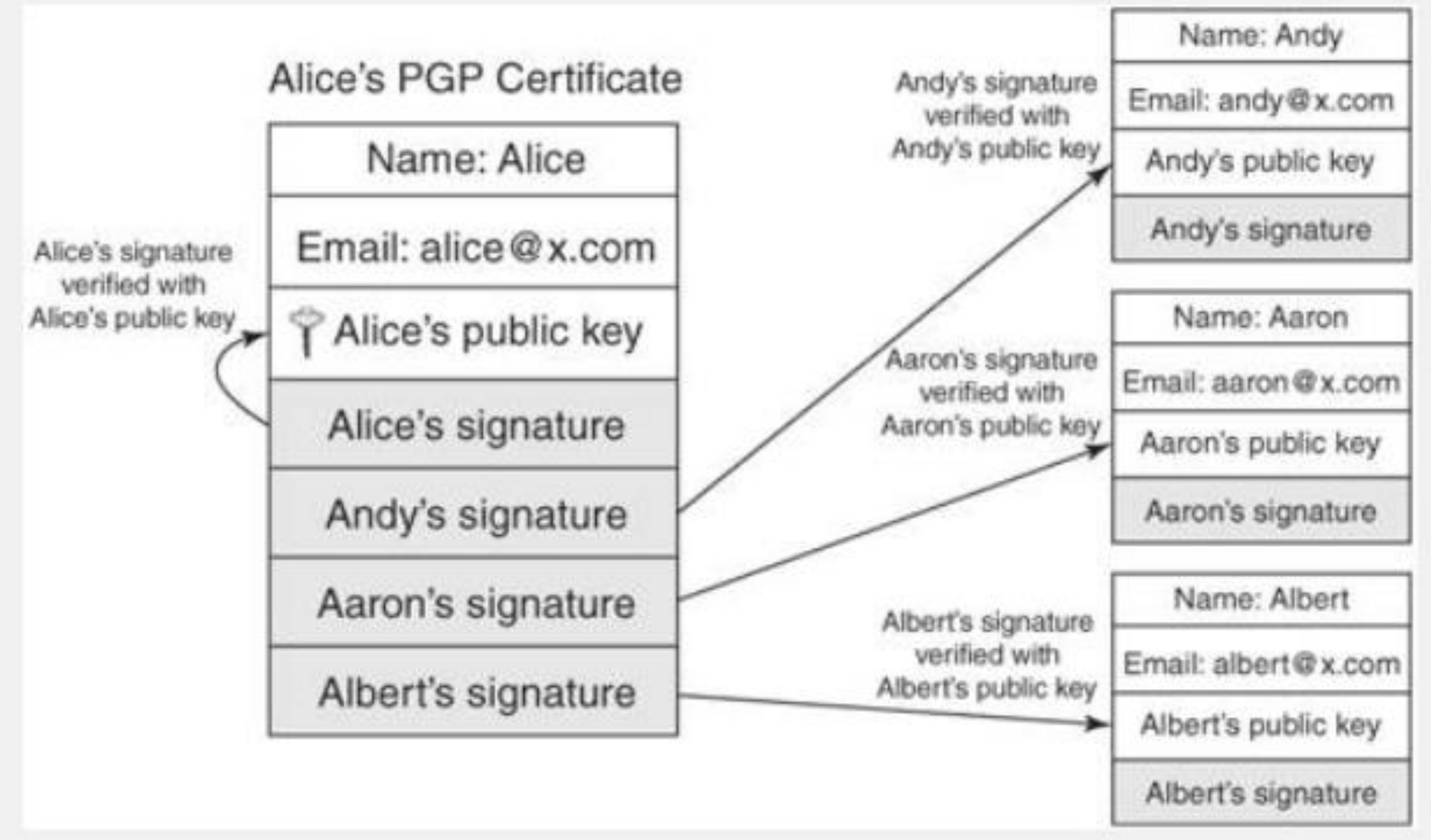


$\text{sig}_{\text{Alice}}$("Friend", Friend's key)
$\text{sig}_{\text{Friend}}$("FoaF", FoaF's key)

PGP Encryption/Decryption

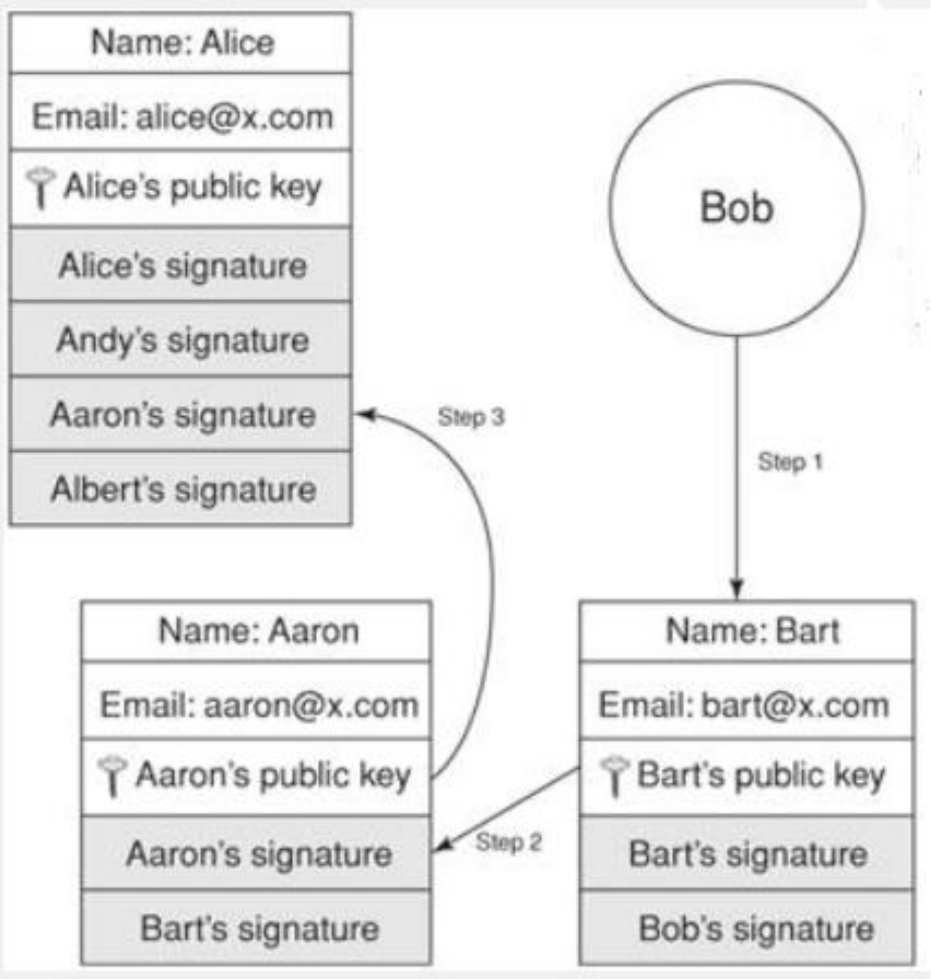


PGP certificate with multiple signatures



If Bob wants to send a secure e-mail to Alice, he looks up Alice's public key certificate and then checks the signatures to see if any of them are from entities that he trusts.

Indirect verification (FoaF) of a PGP certificate

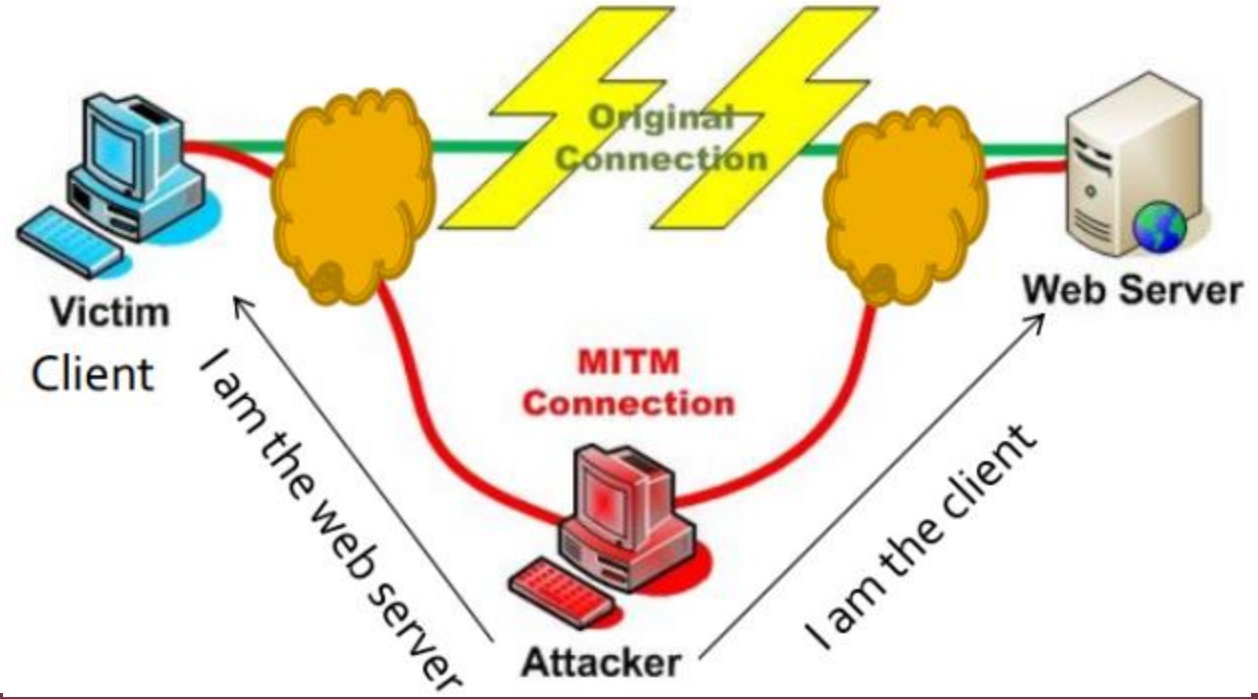


To verify Alice's information:

1. Bob trusts Bart and has personally verified Bart's info and public key.
2. Bob verifies Bart's signature on Aaron's certificate; if signature is good. Bob knows that Bart has verified Aaron's info and public key.
3. Bob verifies Aaron's signature on Alice's certificate; if signature is good. Bob knows that Aaron has verified Alice's info and public key.

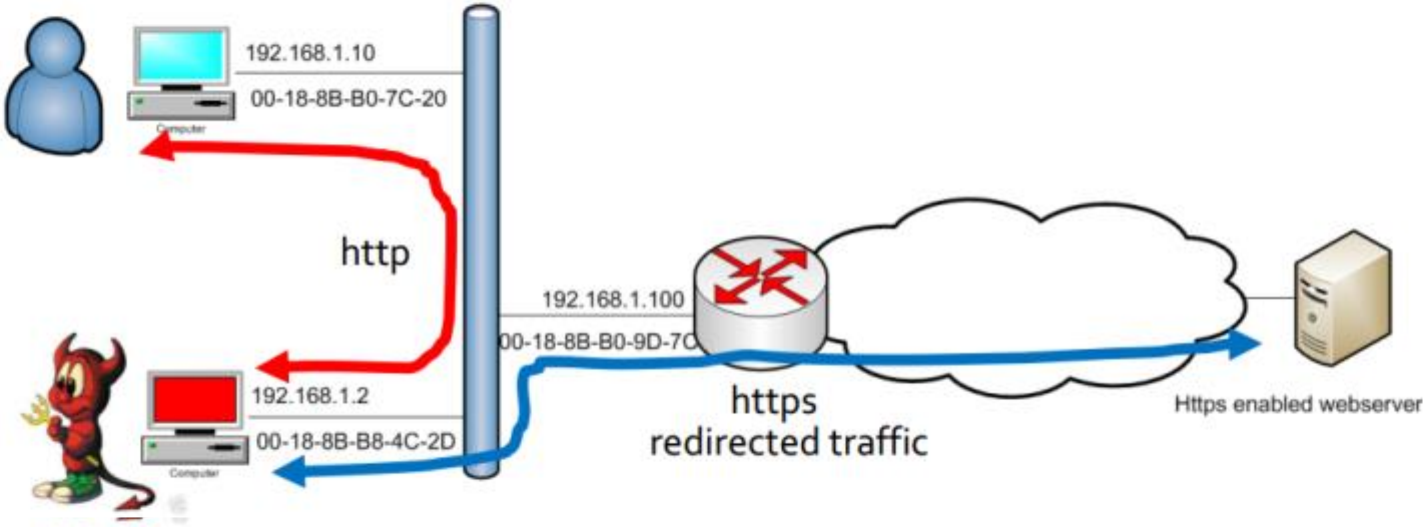
MITM attacks in Web-based Apps

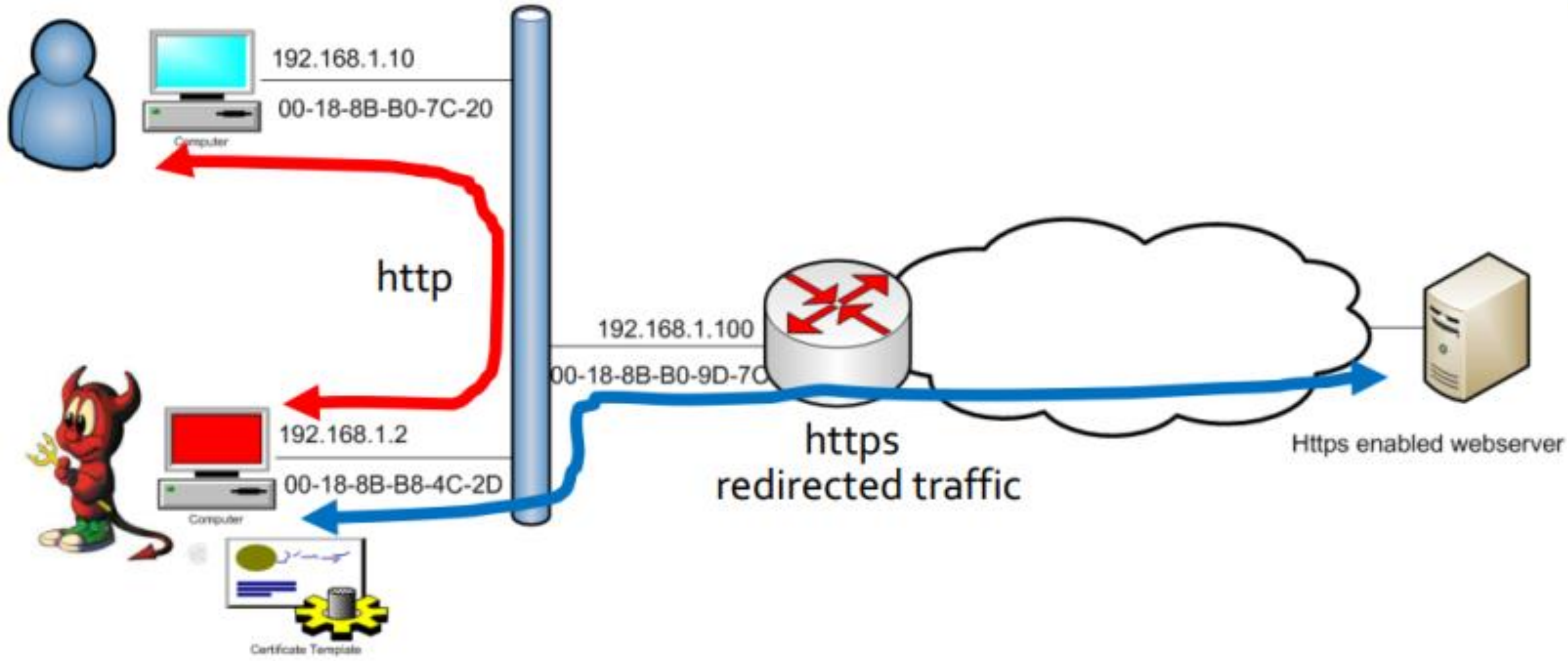
- A man-in-the-middle attack can succeed only when the attacker can impersonate each endpoint to the satisfaction of the other — it is an attack on mutual authentication.



SSL Strip MITM Attacks

- SSL Strip is a tool written by Moxie Marlinspike and released at Black Hat DC 2009. (http://www.thoughtcrime.org/software/sslstrip/)
- It basically reroutes encrypted HTTPS requests to plaintext HTTP requests, effectively sniffing all credentials passed along the network via SSL.
- It lets users connect via HTTP, logs their information, then redirects their connection to the originally-intended HTTPS server on the internet.





```
1) echo '1' > /proc/sys/net/ipv4/ip_forward
2) iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT
   --to-port 10000
3) arpspoof -i eth0 -t 192.168.1.100 192.168.1.2
4) sslstrip -k -l 10000
```

Strip actively intercepting packets from user to remote server.

How SSLStrip attack works?

- It does an MITM on the HTTP connection
- It replaces all the HTTPS links with HTTP ones but remembers the links which were changed
- It communicates with the victim client on an HTTP connection for any secure link
- It communicates with the legitimate server over HTTPS for the same secure link
- Communication is transparently proxied between the victim client and the legitimate server
- Images such as the favicon are replaced by images of the familiar "secure lock" icon, to build trust
- As the MITM is taking places all passwords, credentials etc are stolen without the Client knowing

How to counter SSL Strip attack?

- Force using HTTPS
  - HTTP Strict Transport Security (HSTS)
- Check certificate validity
- Detect ARP spoofing
  - Set fixed MAC address of the default gateway
- Make sure https and the lock icon appear in the address bar.