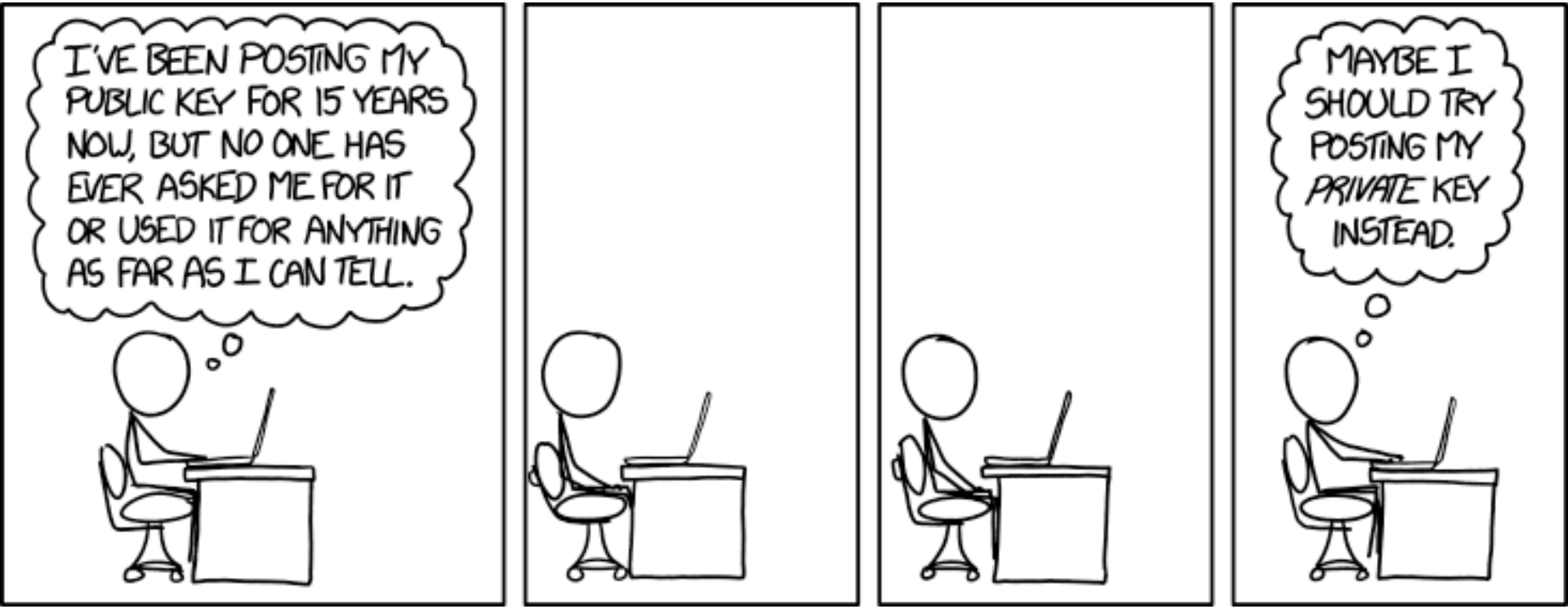
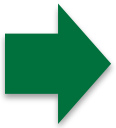
CS 55:
Security and Privacy

Public Key Infrastructure (PKI)



https://xkcd.com/1553/

Agenda



1. Man-in-The-Middle attacks
2. Public Key Certificates
3. Certificate authorities
4. Root and intermediate authorities

Asymmetric encryption can defeat passive eavesdroppers

Alice

Alice
public key

Bob gets Alice's public key, uses it to encrypt message

Bob

Ciphertext

Send ciphertext to Alice

Alice
private key

Data in ciphertext can be a symmetric key for use with AES in following messages (session key)

Data can only be decrypted using Alice's private key

Alice decrypts message with her private key, recovers Bob's message

Asymmetric encryption still has a Man-in-The-Middle (MiTM) problem

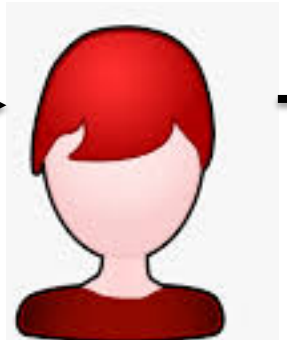
Mallory intercepts Alice's public key and replaces it with her own public key

Authentication problem

Bob has no way to verify it really is Alice's key, assumes Mallory's key is Alice's

Alice

Alice public key



Mallory

Mallory public key

Bob

Alice private key

Secret data, encrypted with Alice's public key

Secret data, encrypted with Mallory's public key

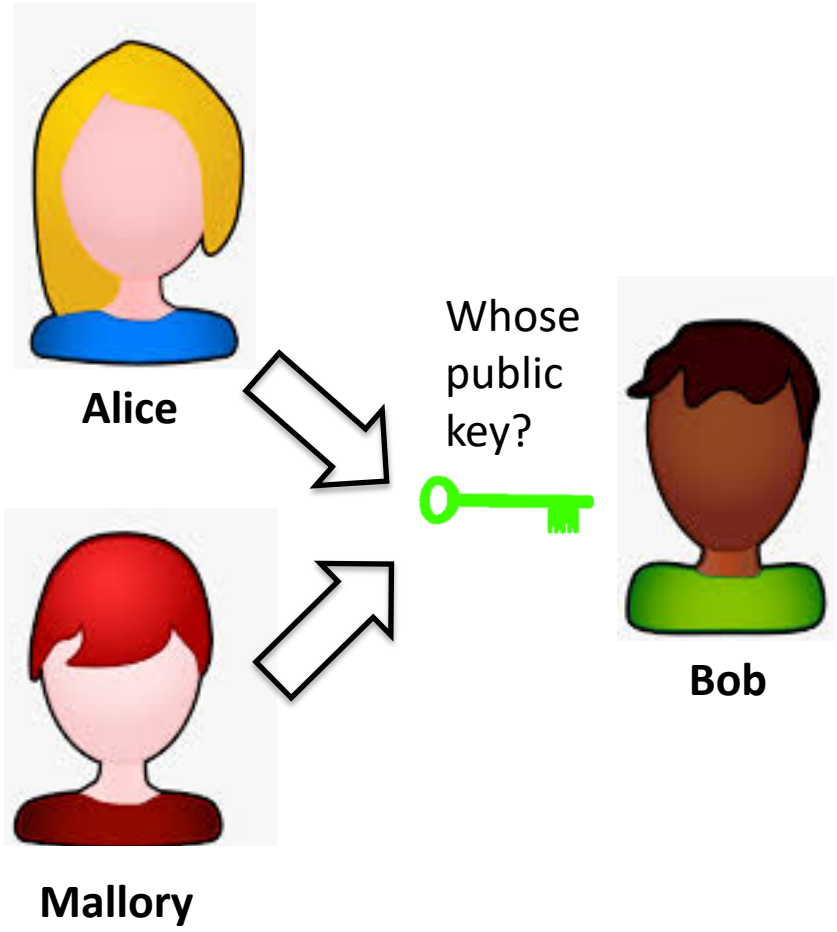
Previously we've said Bob gets Alice's public key but didn't say how!

Alice believes message was from Bob

Mallory decrypts message with her private key, reads message, possibly changes it, re-encrypts message with Alice's public key

Bob encrypts message with public key he received (Mallory's)

The fundamental problem: there is no authentication with asymmetric encryption



Bob receives a public key

Bob has no way to tell whether the public key he received belongs to Alice or Mallory masquerading as Alice

Solution:

- Find a third party that Bob trusts to verify Alice's identity
- Third party creates a certificate with Alice's public key
- Ensure the certificate cannot be forged or altered

Agenda

1. Man-in-The-Middle attacks
2. Public Key Certificates
3. Certificate authorities
4. Root and intermediate authorities

We need a trusted third party to verify public keys belong to a particular principle

Simplified

Alice generates public and private keys

Alice

Alice public key

Alice private key

Certificate Authorities match public keys with principles by issuing a certificate

Alice sends her public key to trusted third party

Simplified

Alice public key

Alice

CERTIFICATE AUTHORITY

Bob

Trusted third party verifies Alice's identify and public key

- **Creates a digital certificate with Alice's name, public key, serial number and expiration date**
- **CA signs certificate with its private key**
- **Certificate has standard format X.509**

Note: you can create your own certificates

Alice public key

Alice private key

Certificate Authorities match public keys with principles by issuing a certificate

Alice sends her public key to trusted third party

Simplified

Alice public key

Alice private key

Certificates can be used to verify a message's integrity and sender

Simplified

CERTIFICATE AUTHORITY

Alice

Bob

Message

Signature

Alice sends to Bob:

- **Plaintext**
- **Signature on hash of plaintext (encrypted with Alice's private key)**
- **Certificate which contains Alice's public key**

Alice public key

Alice private key

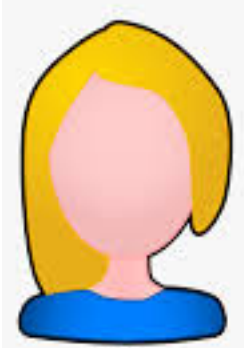
Certificates can be used to verify a message's integrity and sender

But how did Bob get the CA's certificate?
A list of valid root CA's built into browsers and other software
What about revocation?

CERTIFICATE AUTHORITY

Simplified

Bob knows message was from Alice because Bob trusts the CA to verify Alice before signing her certificate



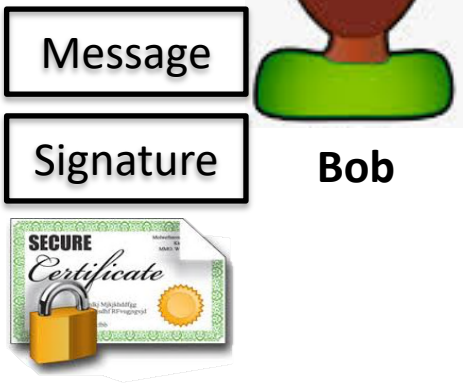
Alice

Alice public key

Alice private key

Bob

- **Verifies certificate is valid**
  - **Not expired or revoked**
  - **Verifies CA's signature on certificate using CA's public key (know certificate is from CA and is unaltered)**
- **Verifies message signature using Alice's public key on certificate**



Bob

Sometimes certificates are revoked, clients should confirm certificates are still valid

Common reasons why a certificate would be revoked

- Certificate is no longer used
- Details of certificate changed
- Certificate owner's private key compromised
- CA compromised

Methods to check if certificate valid

1. Certificate Revocation List (CRL) – client downloads CRL from CA, searches long list for this certificate
2. Online Certificate Status Protocol (OCSP) – client asks CA to confirm still valid, responses: good, revoked, or unknown
3. OCSP stapling – shift burden from client to web server, at regular intervals (hours to days), web server contacts CA to check revocation status, CA sends back timestamped response, web server sends OSCP response "stapled" with certificate
4. Certificate pinning – certificate compiled into app, get certificate from server, see if it matches hardcoded certificate

Adapted from https://www.youtube.com/watch?v=WXNKQ_otO_g

PKI is often used with web sites

CERTIFICATE AUTHORITY

Company purchases certificate from CA

Web server

PKI is often used with web sites

CERTIFICATE AUTHORITY

Company purchases certificate from CA

Certificate installed on server

Web server

CA verifies company's identity

- CA asks company to post file on web server as proof they control the web server
- CA confirms file appears on web service
- CA knows the company controls the server

Certificate contains

- Server public key
- Name of server, serial number and expiration date

CA signs certificate with its private key

PKI is often used with web sites

CERTIFICATE AUTHORITY

Company purchases certificate from CA

Certificate installed on server

Web server

Web browser

Client types web address into URL
Browser initiates secure TLS connection with server
Server sends browser its certificate
Browser has list of root CAs installed, verifies CA's signature on certificate, gets server's public key from certificate
Browser knows it is talking to the right server
We are trusting the browser has only valid root CAs

An adversary could try to compromise the process, but it would be difficult to succeed

Could try to compromise the CA

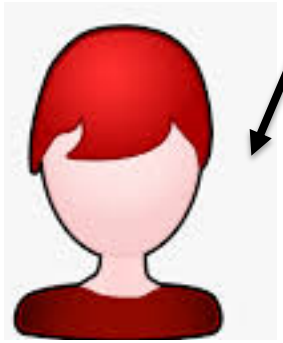
If get CA's private key, can issue fraudulent certificates

Could bribe an employee to issue a fake certificate

CERTIFICATE AUTHORITY

Web server

Web browser



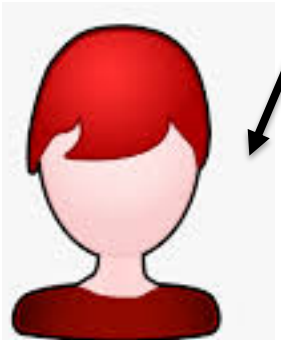
Mallory

An adversary could try to compromise the process, but it would be difficult to succeed

CERTIFICATE AUTHORITY

Web server

Web browser



Mallory

Could try to compromise browser and get adversary's certificate installed in list of trusted root CAs

An adversary could try to compromise the process, but it would be difficult to succeed

CERTIFICATE AUTHORITY

Mallory could forward legitimate certificate

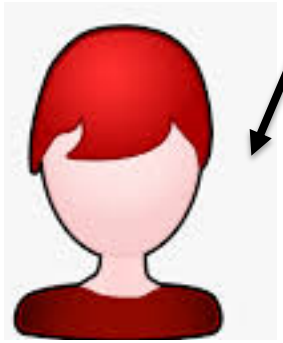
Browser uses web site's public key to encrypt data

Mallory does not have private key and cannot see data

Web server

Web browser

Web site certificate



Mallory

An adversary could try to compromise the process, but it would be difficult to succeed

CERTIFICATE AUTHORITY

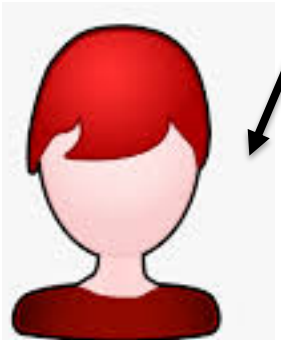
Could forward M's own legitimate certificate (signed by CA)

Names would not match (Mallory vs. web site URL)

Web server

Web browser

Mallory's certificate



Mallory

An adversary could try to compromise the process, but it would be difficult to succeed

CERTIFICATE AUTHORITY

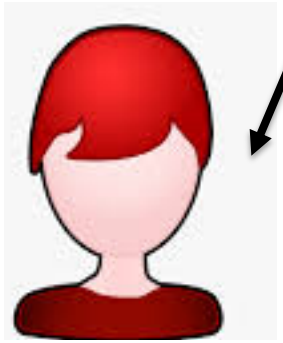
Could forward fake certificate to browser (certificate with M's public key)

Will not be signed by a known CA, browser issues warning

Web server

Web browser

Fake certificate



Mallory

We can get a certificate from a real web site using OpenSSL

```
$ openssl s_client -showcerts -connect www.dartmouth.edu:443 </dev/null
# copy from ----- BEGIN CERTIFICATE ----- to ----- END CERTIFICATE ---- to dartmouth.pem
# convert pem format to something more readable
$ openssl x509 -in dartmouth.pem -text –noout
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number:
         0d:d5:d1:e4:e3:90:0c:42:39:07:43:52:d0:7d:98:95
   Signature Algorithm: sha256WithRSAEncryption
      Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
      Validity
         Not Before: May 26 00:00:00 2020 GMT
         Not After : Jun 16 12:00:00 2022 GMT
      Subject: C=US, ST=New Hampshire, L=Hanover, O=Trustees of Dartmouth College, CN=*.dartmouth.edu
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
               00:aa:44:f7:b4:e8:7c:3a:18:d8:00:c7:58:fb:57:
               <snip>
            Exponent: 65537 (0x10001)
      <snip>
```

Each certificate has unique serial number

Certificate issued by digicert.com

Issued to Trustees of Dartmouth College

Common name (CN) Good for all .dartmouth.edu sites

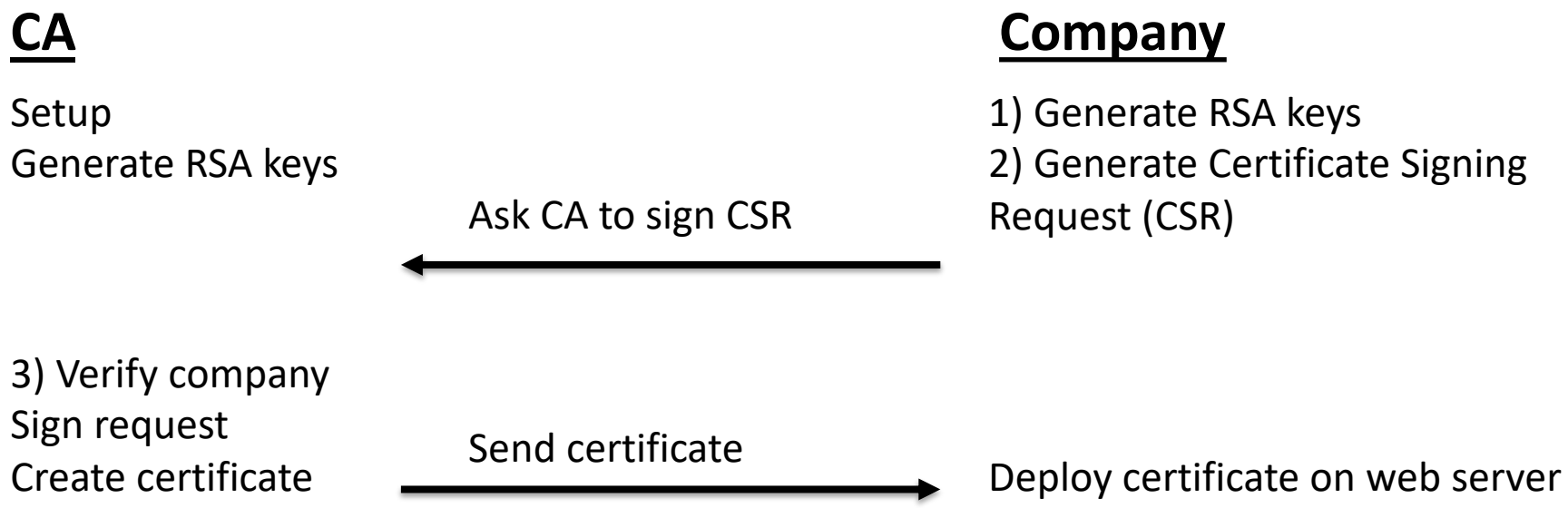
Public key: n e

CA's signature not shown

Agenda

1. Man-in-The-Middle attacks
2. Public Key Certificates
3. Certificate authorities
4. Root and intermediate authorities

Getting a certificate starts with a Certificate Signing Request made by company



We can set up our own CA with a "self-signed" certificate

#set up directories on CA
$ mkdir demoCA
$ cd demoCA/
$ mkdir certs crl newcerts
$ touch index.txt serial
$ echo 1000 > serial
$ cd ..

Companies sometimes do this for their internal networks (do not need to pay CA for each certificate)
Devices must be configured to trust the internal CA (we will do this in a few slides)

Generate public/private key pair using 4096-bit RSA

Store public and private keys in modelCA_key.pem

#create public/private keys and certificate for CA
$ openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -keyout modelCA_key.pem -out modelCA_cert.pem
Generating a 4096 bit RSA private key
<snip>

Valid for 10 years!

Use SHA-256 for hasing

Certificate stored in modelCA_cert.pem

If certificate is self-signed, why would anyone trust it?
They wouldn't!

1) Create key pair for company

```
# generate key pair for company
$ openssl genrsa -aes128 -out company_key.pem 2048

# look at results
$ openssl rsa -noout -text -in company_key.pem
Enter pass phrase for company_key.pem:
Private-Key: (2048 bit)
modulus:
    00:e2:b5:fc:36:9e:da:d7:5c:70:b3:df:92:a1:6a:
    <snip>
publicExponent: 65537 (0x10001)
privateExponent:
    00:82:60:77:f8:0d:68:fa:fb:0d:51:54:1c:a6:39:
    <snip>
prime1:
    00:f6:4d:4e:f4:e9:76:02:d5:3c:14:00:e0:21:e2:
    <snip>
prime2:
    00:eb:a3:33:75:94:27:cc:80:52:8e:b1:08:03:4a:
    <snip>
```

company_key.pem contains both public and private keys

2) Company generates Certificate Signing Request (CSR)

$ openssl req -new -key company_key.pem -out company.csr -sha256
Enter pass phrase for company_key.pem: **cs55**
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:**US**
State or Province Name (full name) [Some-State]:**NH**
Locality Name (eg, city) []:**Hanover**
Organization Name (eg, company) [Internet Widgits Pty Ltd]:**CS55 Ltd**
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:**cs55.dartmouth.edu**
Email Address []:**admin@cs55.dartmouth.edu**

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

Results of company generated certificate signing request

```
$ openssl req -in company.csr -text –noout
Certificate Request:
   Data:
      Version: 0 (0x0)
      Subject: C=US, ST=NH, L=Hanover, O=CS55 Ltd,
CN=cs55.dartmouth.edu/emailAddress=admin@cs55.dartmouth.edu
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
               00:e2:b5:fc:36:9e:da:d7:5c:70:b3:df:92:a1:6a:
               <snip>
            Exponent: 65537 (0x10001)
      Attributes:
         a0:00
   Signature Algorithm: sha256WithRSAEncryption
      e0:5f:b8:a0:2f:6d:61:8b:55:ad:b0:cb:9e:43:4e:52:c5:b2:
      <snip>
```

Info we entered about company

Public key

Signature using company's private key (allows the CA to verify public key belongs to company)

Company asks CA to sign request, our own CA signs request

```
# edit openssl.cnf to allow signing, set policy = policy_anything (from policy_match)
$ sudo nano /usr/lib/ssl/openssl.cnf

# sign company's request
$ openssl ca -in company.csr -out company_cert.pem -md sha256 -cert modelCA_cert.pem -keyfile
modelCA_key.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for modelCA_key.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Nov 23 22:38:01 2020 GMT
            Not After : Nov 23 22:38:01 2021 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = NH
            localityName              = Hanover
            organizationName          = CS55 Ltd
            commonName                = cs55.dartmouth.edu
            emailAddress              = admin@cs55.dartmouth.edu
<snip>
with 1 new entries
Data Base Updated
```

Company's signature request

Signed certificate

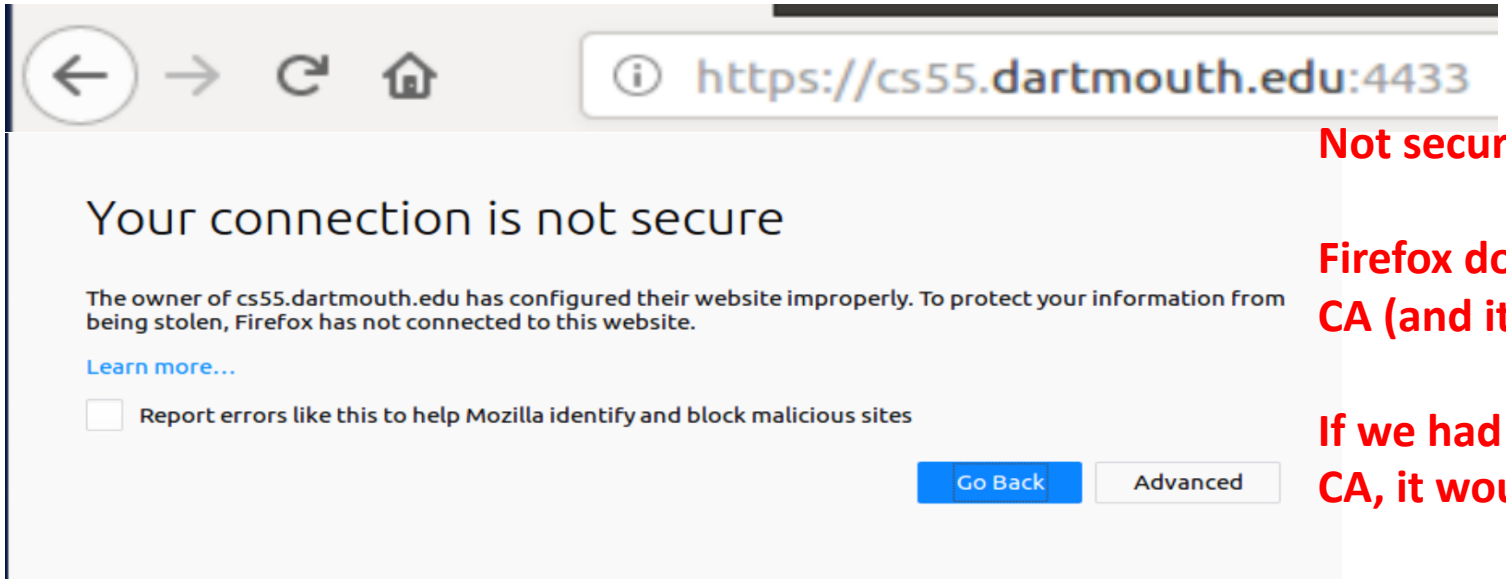
CSR from a real CA only signed after identify verified!

Company deploys certificate on web server

```
# edit /etc/hosts to add cs55.dartmouth.edu at 127.0.0.1
$ sudo nano /etc/hosts

#copy company_key.pem and company_cert.pem into one file
$ cp company_key.pem company_all.pem
$ cat company_cert.pem >> company_all.pem

#start a simple web server running using cert, listening on port 4433
$ openssl s_server -cert company_all.pem -accept 4433 -www
```



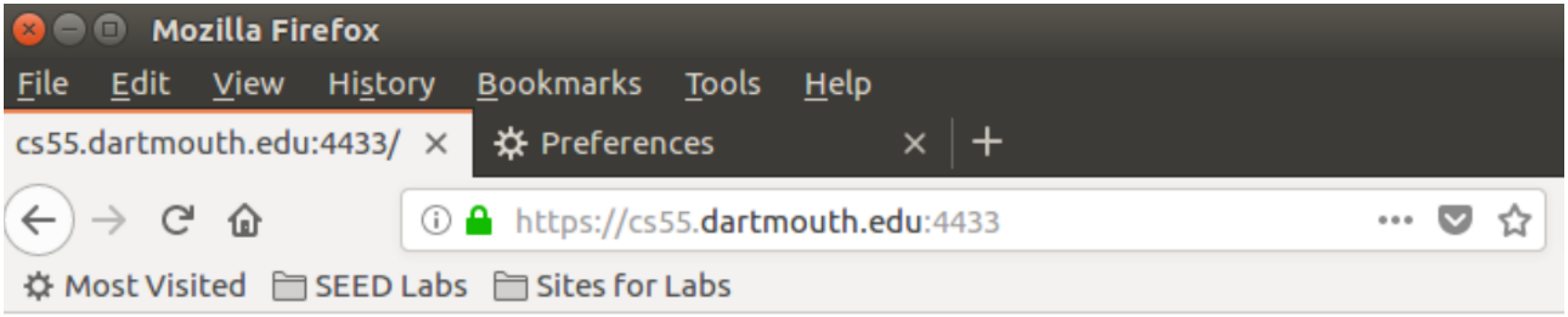
Not secure! Why not!!!

Firefox doesn't trust this CA (and it should not!)

If we had used a trusted CA, it would have worked

Tell Firefox to trust this CA and all works as expected!

From Firefox: Edit->Preferences->Privacy & Security->View Certificates
Import modelCA_cert.pem with option ”Trust this CA to identify web sites”
Try browsing to site again



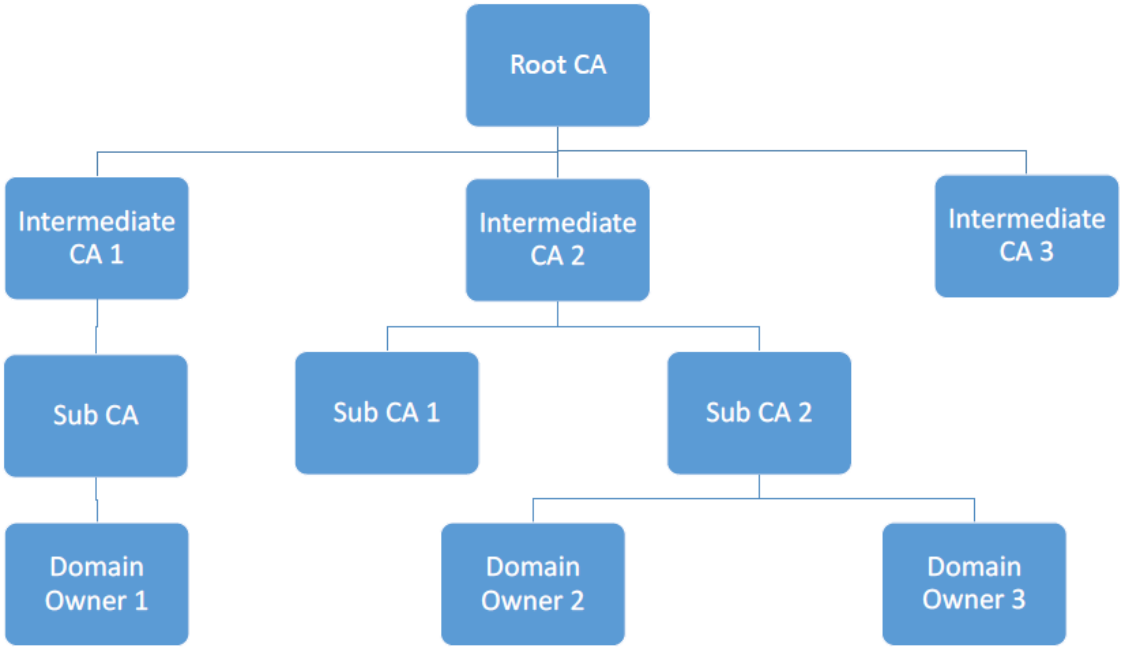

```
s_server -cert company_all.pem -accept 4433 -www
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1/SSLv3:ECDHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:ECDHE-ECDSA-AES256-GCM-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA384   TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA384
TLSv1/SSLv3:ECDHE-RSA-AES256-SHA      TLSv1/SSLv3:ECDHE-ECDSA-AES256-SHA
TLSv1/SSLv3:SRP-DSS-AES-256-CBC-SHA   TLSv1/SSLv3:SRP-RSA-AES-256-CBC-SHA
TLSv1/SSLv3:SRP-AES-256-CBC-SHA       TLSv1/SSLv3:DH-DSS-AES256-GCM-SHA384
TLSv1/SSLv3:DHE-DSS-AES256-GCM-SHA384TLSv1/SSLv3:DH-RSA-AES256-GCM-SHA384
TLSv1/SSLv3:DHE-RSA-AES256-GCM-SHA384TLSv1/SSLv3:DHE-RSA-AES256-SHA256
TLSv1/SSLv3:DHE-DSS-AES256-SHA256     TLSv1/SSLv3:DH-RSA-AES256-SHA256
TLSv1/SSLv3:DH-DSS-AES256-SHA256      TLSv1/SSLv3:DHE-RSA-AES256-SHA
TLSv1/SSLv3:DHE-DSS-AES256-SHA        TLSv1/SSLv3:DH-RSA-AES256-SHA
TLSv1/SSLv3:DH-DSS-AES256-SHA         TLSv1/SSLv3:DHE-RSA-CAMELLIA256-SHA
TLSv1/SSLv3:DHE-DSS-CAMELLIA256-SHA   TLSv1/SSLv3:DH-RSA-CAMELLIA256-SHA
```

Works!

Agenda

1. Man-in-The-Middle attacks
2. Public Key Certificates
3. Certificate authorities
4. Root and intermediate authorities

There are many CAs in the world, and they are organized in a hierarchical structure



Root CA's public keys are self-signed

```
Issuer: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network,
    OU=(c) 2006 VeriSign, Inc. - For authorized use only,
    CN=VeriSign Class 3 Public Primary Certification Authority - G5
Subject: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network,
    OU=(c) 2006 VeriSign, Inc. - For authorized use only,
    CN=VeriSign Class 3 Public Primary Certification Authority - G5
```

How can we trust it?

- **Known CAs are preloaded on OS, browsers, other software**
- **As long as we trust the software, we are trusting the public keys that came with it!**

Root CAs vouch for intermediate CAs (root CA can go offline once intermediate CAs created)

Why?

- **Reduces load on Roots (creating and validating)**
- **If there is a compromise of an intermediate CA, only its certificates are compromised, not all**

CA's sometimes meshed (but doesn't scale well)

Web of trust (I trust, you trust someone else, so I trust them)

Root CAs vouch for subordinates in a "chain of trust"

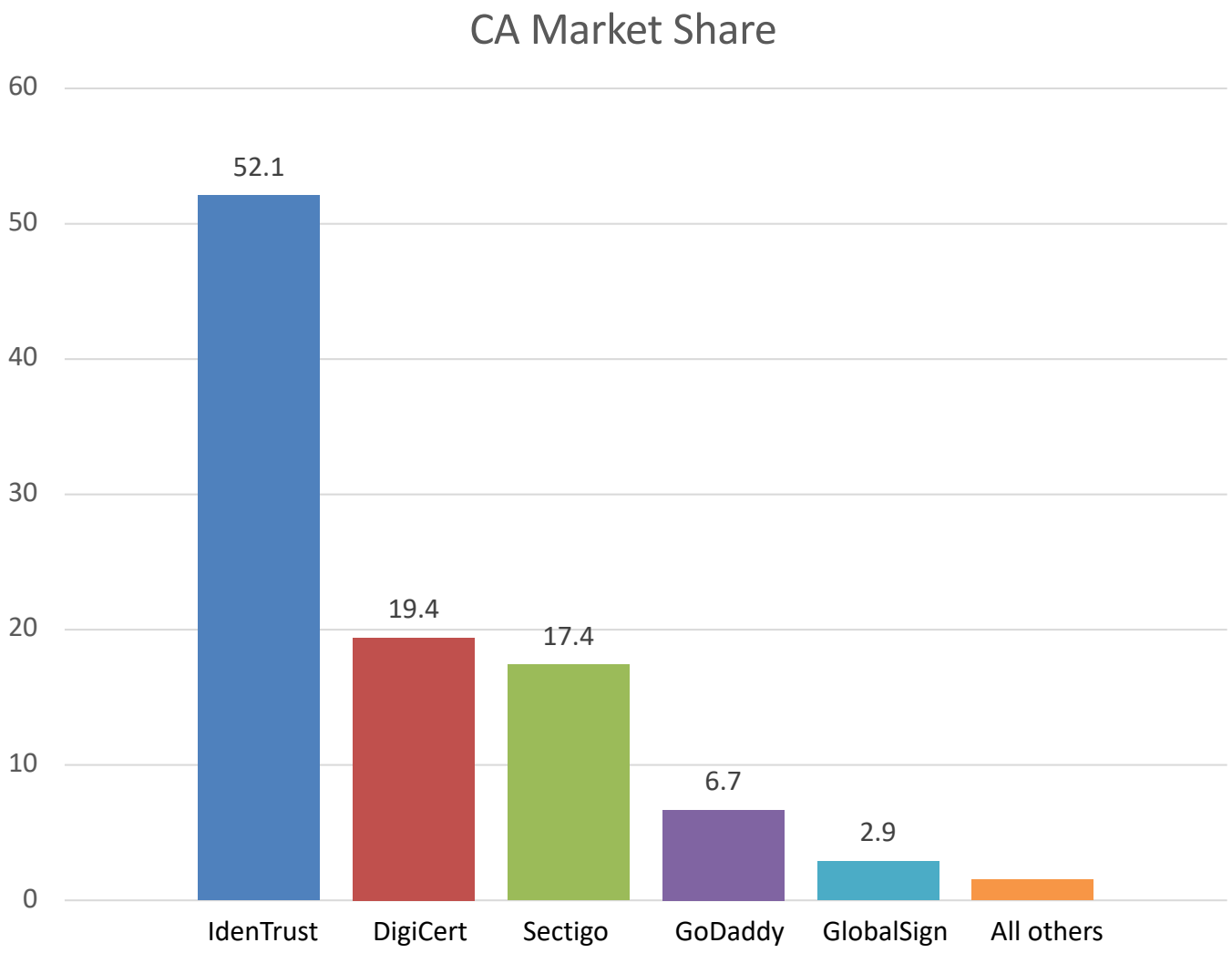
```
$ openssl s_client -showcerts -connect www.paypal.com:443
Certificate chain
 0 s:/businessCategory=Private
Organization/jurisdictionC=US/jurisdictionST=Delaware/serialNumber=3014267/C=US/ST=California/
L=San Jose/O=PayPal, Inc./OU=CDN Support/CN=www.paypal.com
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 Extended Validation Server CA
-----BEGIN CERTIFICATE-----
MIIHdDCCBlygAwIBAgIQB0Haxhm5e7comqWUzibAzTANBgkqhkiG9w0BAQsFADB1
<snip>
 1 s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 Extended Validation Server CA
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance EV Root CA
-----BEGIN CERTIFICATE-----
MIIEtjCCA56gAwIBAgIQDHmpRLCMEZUgkmFf4msdgzANBgkqhkiG9w0BAQsFADBs
<snip>
```

PayPal certificate signed by DigiCert subordinate

Subordinate signed by root

Browser checks whether server's certificate signed by root (browser knows root CAs ahead of time)
If signed by intermediate, check if intermediate signed by root
Check in chain until find a root

Global market for CAs is dominated by a few major players (but many small players)



https://w3techs.com/technologies/overview/ssl_certificate

There are three main types of certificates CAs issue: DV, OV, EV

Domain Validated (DV) Certificates

- Most popular type of certificate
- The CA verifies the domain records to check if the domain belongs to applicant
- Domain Control Validation (DCV) is performed on domain name in the certificate request
- DCV uses information in the WHOIS database
- DCV is conducted via
  - Email
  - HTTP
  - DNS

Note: there is also a Root certificate that Root CAs get, but companies normally don't!

Root CAs then issue intermediate CA certificates

There are three main types of certificates CAs issue: DV, OV, EV

Organizational Validated (OV) Certificates

- Not very popular type of certificate
- CAs verify the following before issuing OV certificates:
  - Domain control validation
  - Applicant's identity and address
  - Applicant's link to organization
  - Organization's address
  - Organization's WHOIS record
  - Callback on organization's verified telephone number

There are three main types of certificates CAs issue: DV, OV, EV

Extended Validated (EV) Certificates

- Not very popular type of certificate
- Name appears in green in browser
- CAs verify the following before issuing OV certificates:
  - Domain control validation
  - Applicant's identity and address
  - Applicant's link to organization
  - Organization's address
  - Organization's WHOIS record
  - Callback on organization's verified telephone number

Browsers display certificate types differently