Practical Aspects of Modern Cryptography

Winter 2011

Josh Benaloh
Brian LaMacchia

Agenda

- Integrity Checking (HMAC redux)
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

Message Authentication Codes

MAC key K, plaintext P, ciphertext C=E(P).

$$\text{MAC}=\text{H}(K,P)? \quad \text{MAC}=\text{H}(P,K)?$$

$$\text{MAC}=\text{H}(K,C)? \quad \text{MAC}=\text{H}(C,K)?$$

There are weaknesses with *all* of the above.

$$\text{HMAC} = \text{H}(K,\text{H}(K,P))$$

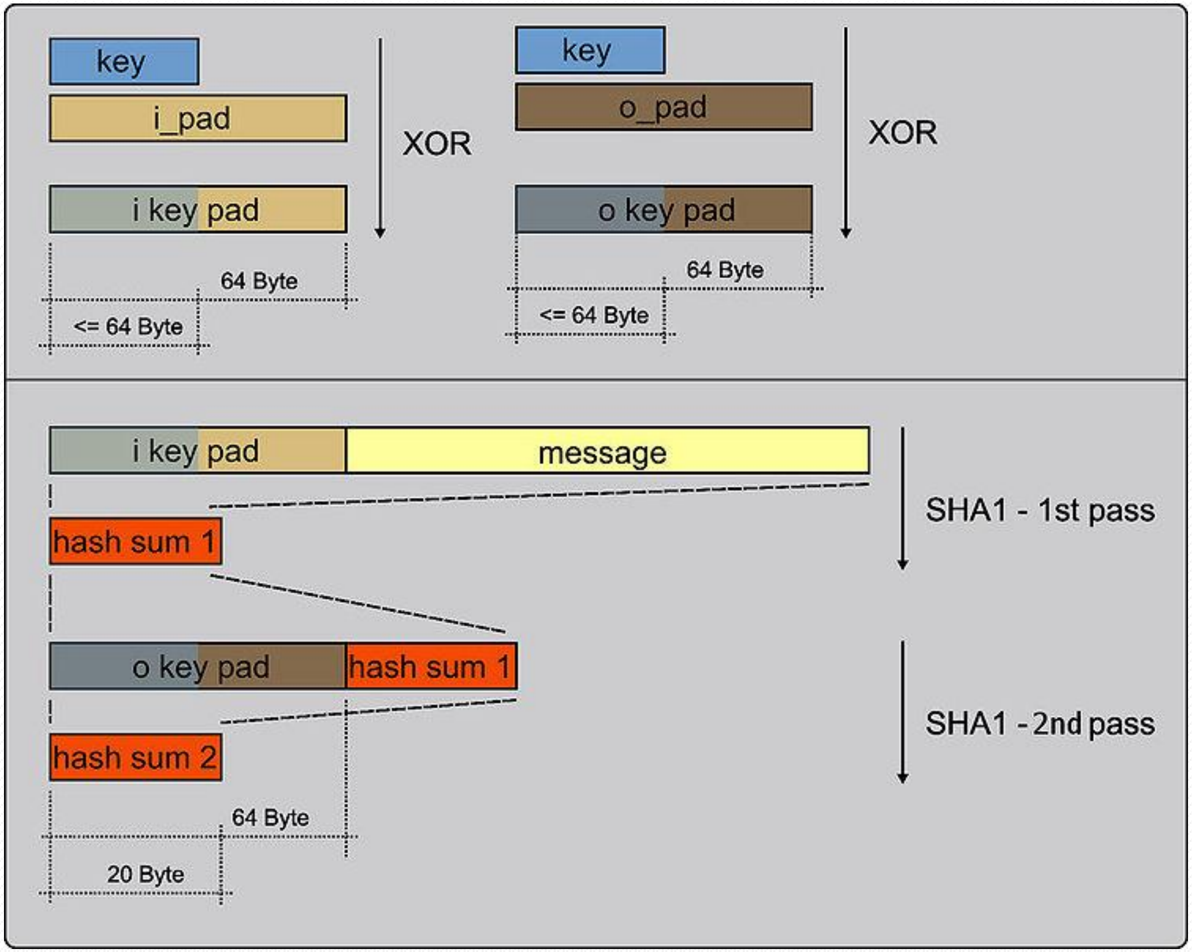
HMAC

- HMAC is a generic construction that build a MAC out of hash function (any hash function) and a secret key
- If $H(x)$ is a cryptographic hash function, then the HMAC function using $H(x)$ is:

$$HMAC(K, m) = H((K \oplus \text{opad}) \parallel H((K \oplus \text{ipad}) \parallel \text{m}))$$

- ipad = 0x36363636…36 (64 byte constant)
- opad = 0x5c5c5c5c…5c (64 byte constant)

Example: HMAC-SHA1



Crypto Hygiene

Do I really need to use different keys for encryption and integrity?

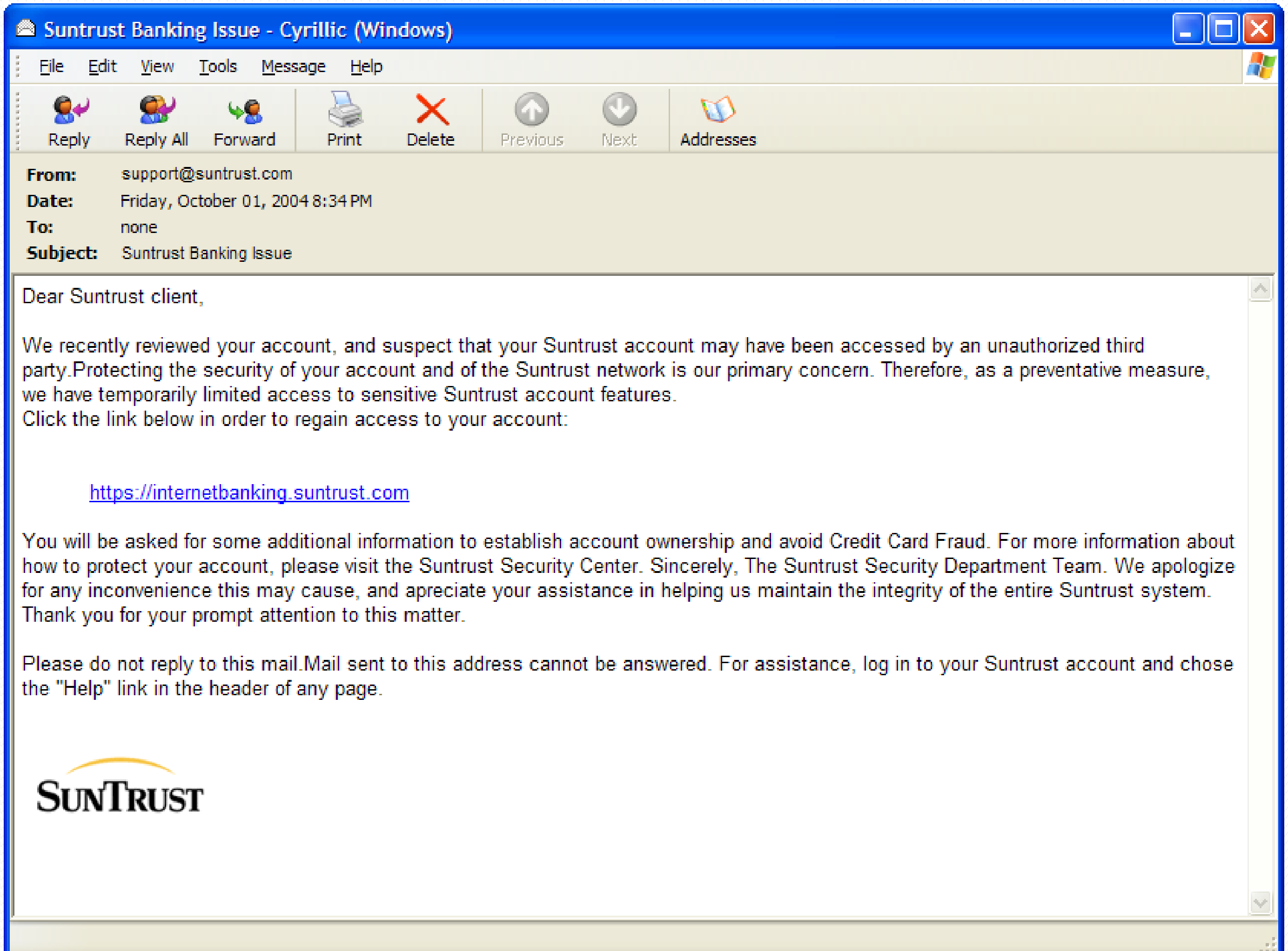
It's always a good idea to use separate keys for separate functions, **but the keys can be derived from the same master**.

K_1=H("Key1",K) K_2=H("Key2",K)

Agenda

- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

Motivation


Suntrust Banking Issue - Cyrillic (Windows)

File Edit View Tools Message Help

Reply Reply All Forward Print Delete Previous Next Addresses

From: support@suntrust.com
Date: Friday, October 01, 2004 8:34 PM
To: none
Subject: Suntrust Banking Issue

Dear Suntrust client,

We recently reviewed your account, and suspect that your Suntrust account may have been accessed by an unauthorized third party.Protecting the security of your account and of the Suntrust network is our primary concern. Therefore, as a preventative measure, we have temporarily limited access to sensitive Suntrust account features.
Click the link below in order to regain access to your account:

https://internetbanking.suntrust.com

You will be asked for some additional information to establish account ownership and avoid Credit Card Fraud. For more information about how to protect your account, please visit the Suntrust Security Center. Sincerely, The Suntrust Security Department Team. We apologize for any inconvenience this may cause, and apreciate your assistance in helping us maintain the integrity of the entire Suntrust system. Thank you for your prompt attention to this matter.

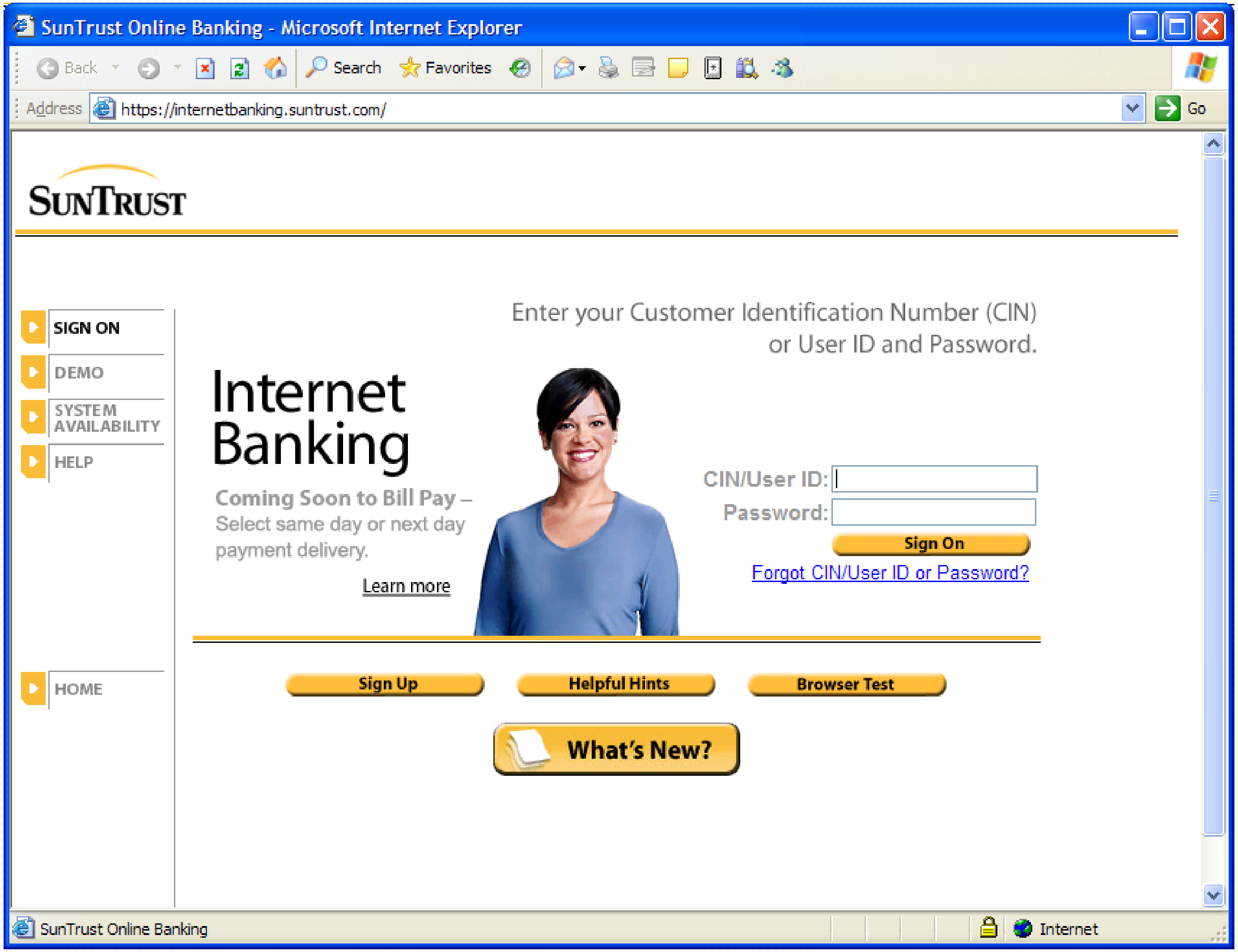
Please do not reply to this mail.Mail sent to this address cannot be answered. For assistance, log in to your Suntrust account and chose the "Help" link in the header of any page.

SunTrust

Motivation

WWW.SUNTRUSTBANKINGONLINE.COM - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://www.suntrustbankingonline.com/
Go
Links
SUNTRUST
SIGN ON
DEMO
SYSTEM AVAILABILITY
HELP
HOME
Enter your Customer Identification Number (CIN) and Password.
Internet Banking
Now you can update your Password and User ID online.
Learn more
CIN:
Password:
Sign On
Sign Up
Helpful Hints
Browser Test
What's New?
Internet

Motivation


SunTrust Online Banking - Microsoft Internet Explorer
Back
Search
Favorites
Address
https://internetbanking.suntrust.com/
Go
SunTrust
SIGN ON
DEMO
SYSTEM AVAILABILITY
HELP
HOME
Enter your Customer Identification Number (CIN) or User ID and Password.
Internet Banking
Coming Soon to Bill Pay –
Select same day or next day payment delivery.
Learn more
CIN/User ID:
Password:
Sign On
Forgot CIN/User ID or Password?
Sign Up
Helpful Hints
Browser Test
What's New?
SunTrust Online Banking
Internet

Motivation

- How do I know the web site I'm talking to is really who I think it is?
- Is it safe to view to give sensitive information over the Web?
  - What keeps my CC#, SSN, financial information or medical records out of the hands of the bad guys?
- How do I know that the information I'm looking at hasn't been malicious modified?
  - Has someone tampered with it?

Security Protocol Properties

- Confidentiality
  - Keeping message content secret, even if the information passes over a public channel
- Integrity
  - Keeping messages tamper-free from origin to destination
- Authentication
  - Determining the origin of messages (author and/or sender)

Agenda

- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

Kerberos History

- Based on symmetric Needham-Schroeder (1978)
- Designed as part of MIT's Project Athena in the 1980's
  - Kerberos v4 published in 1987
- Migration to the IETF
  - RFC 1510 (Kerberos v5, 1993)
- Used in a number of products
  - Example: Windows domains (since Windows 2000)
  - Many web-based authentication protocols (e.g. Windows Live ID) are essentially Kerberos (or Kerberos-inspired) using HTTP and client-side cookies.

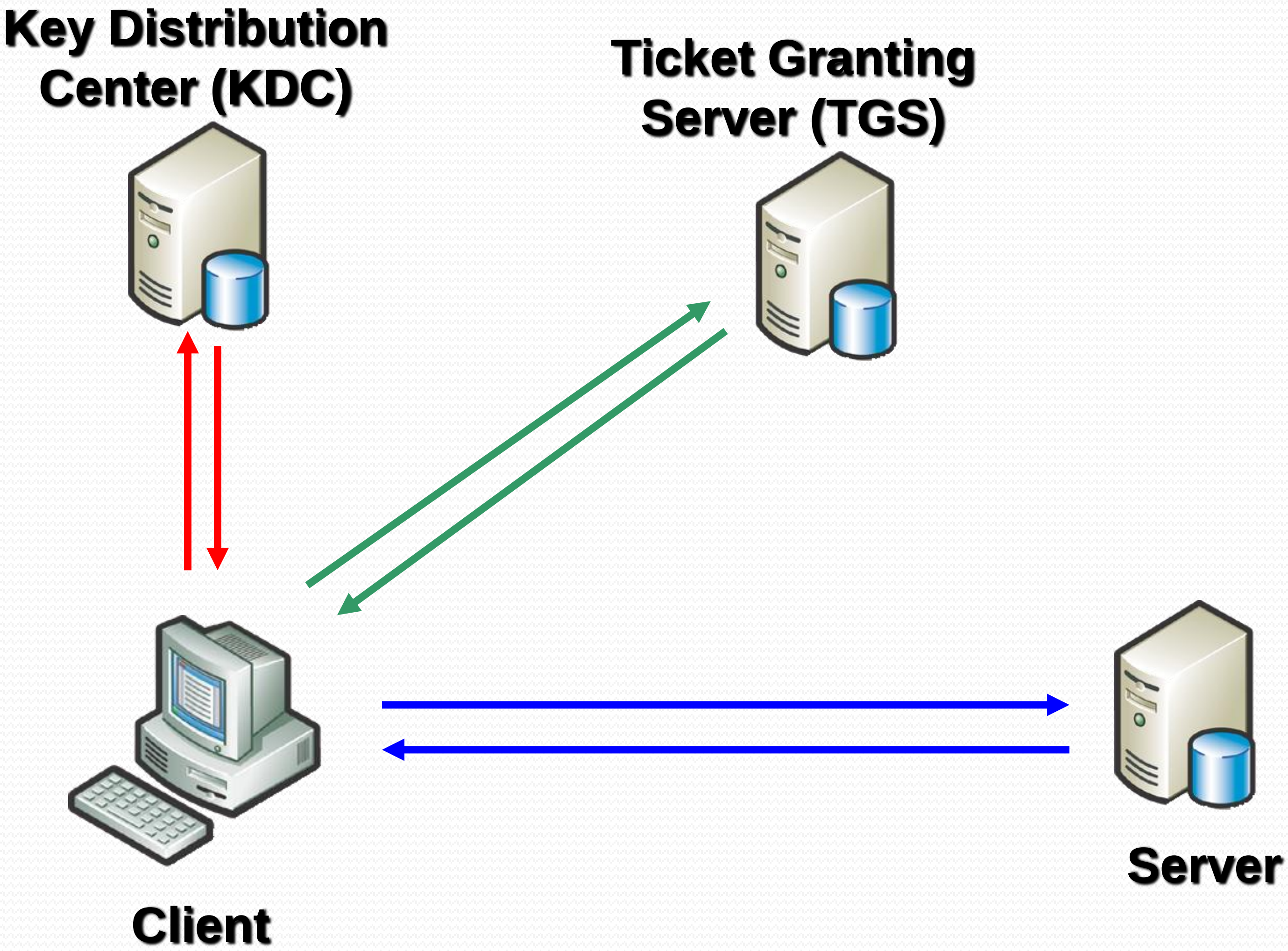
Kerberos

- Designed for a single "administration domain" of machines & users
- No public key crypto
- Provides authentication & encryption services
- "Kerberized" servers provide authorization on top of the authenticated identities

The Kerberos Model

- Clients
- Servers
- The Key Distribution Center (KDC)
- Centralized trust model
  - KDC is trusted by all clients & servers
  - KDC shares a secret, symmetric key with each client and server
- A "realm" is single trust domain consisting of one or more clients, servers, KDCs

Picture of a Kerberos Realm



Joining a Kerberos Realm

- One-time setup
  - Each client, server that wishes to participate in the realm exchanges a secret key with the KDC
  - If the KDC is compromised, the entire system is cracked
- Because the KDC knows everyone's individual secret key, the KDC can issue credentials to each realm identity

Kerberos Credentials

- Two types of credentials in Kerberos
  - Tickets
  - Authenticators
- Tickets are credentials issued to a client for communication with a specific server
- Authenticators are additional credentials that prove a client knows a key at a point in time
  - Basic idea: encrypt a “nonce”

The Basic Kerberos Protocol

Assume client C wishes to authenticate to and communicate with server S

Phase 1: C gets a Ticket-Granting Ticket (TGT) from the KDC

Phase 2: C uses the TGT to get a Ticket for S

Phase 3: C communicates with S

Protocol Definitions

- C = client, S = server
- TGS = ticket-granting service
- K_x = x's secret key
- $K_{x,y}$ = session key for x and y
- $\{m\}K_x$ = m encrypted in x's secret key
- $T_{x,y}$ = x's ticket to use y
- $A_{x,y}$ = authenticator from x to y
- N_x = a nonce generated by x

The Basic Kerberos Protocol (1)

Phase 1: C gets a Ticket-Granting Ticket

1. C sends a request to the KDC for a “ticket-granting ticket” (TGT)
   - A TGT is a ticket used to talk to the special ticket-granting service
   - A TGT is relatively long-lived (~8-24 hours typically)

$$C \rightarrow KDC: C, TGS, N_C$$

Sent in the clear!

The Basic Kerberos Protocol (2)

Phase 1: C gets a Ticket-Granting Ticket

2. KDC responds with two items
   - The ticket-granting ticket
     - A ticket for C to talk to TGS
   - A copy of the session key to use to talk to TGS, encrypted in C's shared key

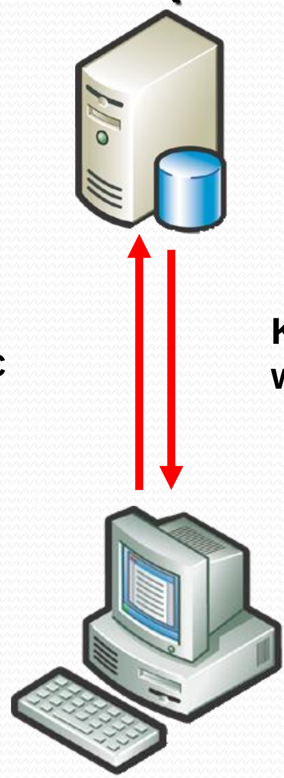
$$KDC \rightarrow C: T_{C,TGS}, \{K_{C,TGS}\}K_C$$

$$\text{where } T_{C,TGS} = TGS, \{C, C\text{-addr}, \text{lifetime}, K_{C,TGS}\}K_{TGS}$$

   - Only the TGS can decrypt the ticket
   - C can unlock the second part to retrieve $K_{C,TGS}$

Picture of a Kerberos Realm

Key Distribution Center (KDC)



Client

C ➔ KDC: C, TGS, N_C

KDC ➔ C: $T_{C,TGS}$, $\{K_{C,TGS}\}K_C$
where $T_{C,TGS}$ = TGS, $\{C, \text{C-addr}, \text{lifetime}, K_{C,TGS}\}K_{TGS}$

The Basic Kerberos Protocol (3)

Phase 2: C gets a Ticket for S

3. C requests a ticket to communicate with S from the ticket-granting service (TGS)
   - C sends TGT to S along with an authenticator requesting a ticket from C to S

$$C \rightarrow TGS: \{A_{C,S}\}K_{C,TGS}\ ,\ T_{C,TGS}$$

where $A_{c,s}$ = {c, timestamp, opt. subkey}

   - First part proves to TGS that C knows the session key
   - Second part is the TGT C got from the KDC

The Basic Kerberos Protocol (4)

Phase 2: C gets a Ticket for S

4. TGS returns a ticket for C to talk to S

   (Just like step 2 above...)

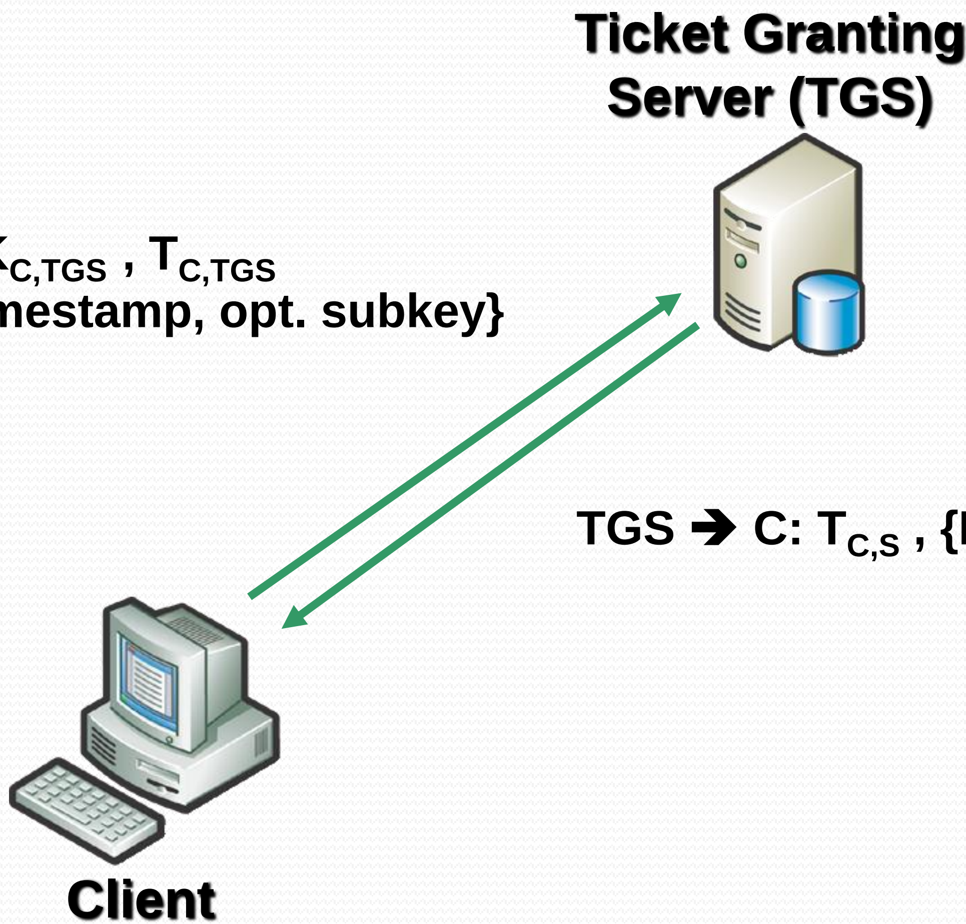
$$TGS \rightarrow C: T_{C,S}, \{K_{C,S}\}K_{C,TGS}$$

$$\text{Where } T_{C,S} = S, \{C, \text{C-addr}, \text{lifetime}, K_{C,S}\}K_S$$

- Only S can decrypt the ticket $T_{C,S}$
- C can unlock the second part to retrieve $K_{C,S}$

Picture of a Kerberos Realm

C ➔ TGS: $\{A_{C,S}\}K_{C,TGS}$, $T_{C,TGS}$
where $A_{c,s}$ = {c, timestamp, opt. subkey}



TGS ➔ C: $T_{C,S}$, $\{K_{C,S}\}K_{C,TGS}$

The Basic Kerberos Protocol (5)

Phase 3: C communicates with S

5. C sends the ticket to S along with an authenticator to establish a shared secret

$$C \rightarrow S: \{A_{C,S}\}K_{C,S}\ ,\ T_{C,S}$$

$$\text{where } A_{c,s} = \{c, \text{timestamp, opt. subkey}\}$$

$$T_{C,S} = S, \{C, \text{C-addr, lifetime}, K_{C,S}\}K_S$$

- S decrypts the ticket $T_{C,S}$ to get the shared secret $K_{C,S}$ needed to communicate securely with C

The Basic Kerberos Protocol (6)

Phase 3: C communicates with S

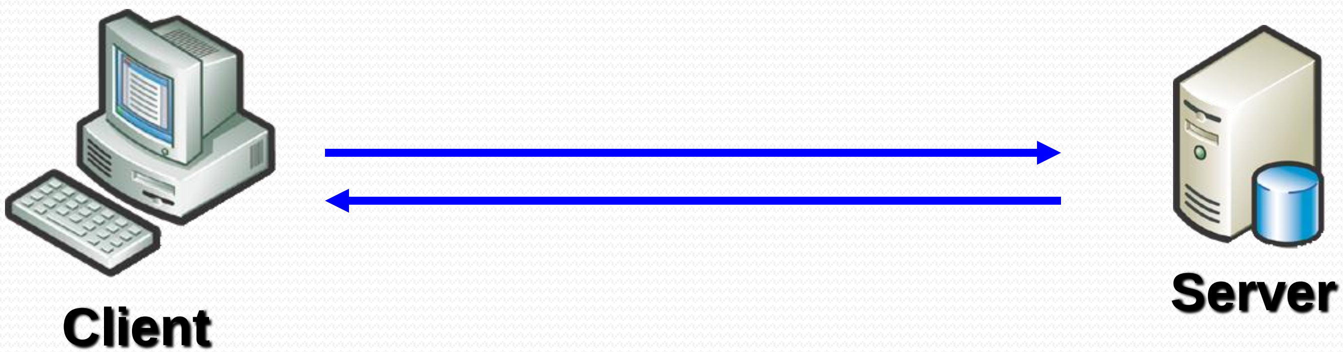
6. S decrypts the ticket to obtain the $K_{C,S}$ and replies to C with proof of possession of the shared secret (optional step)

   S ➔ C: {timestamp, opt. subkey}$K_{c,s}$

   Notice that S had to decrypt the authenticator, extract the timestamp & opt. subkey, and re-encrypt those two components with $K_{c,s}$

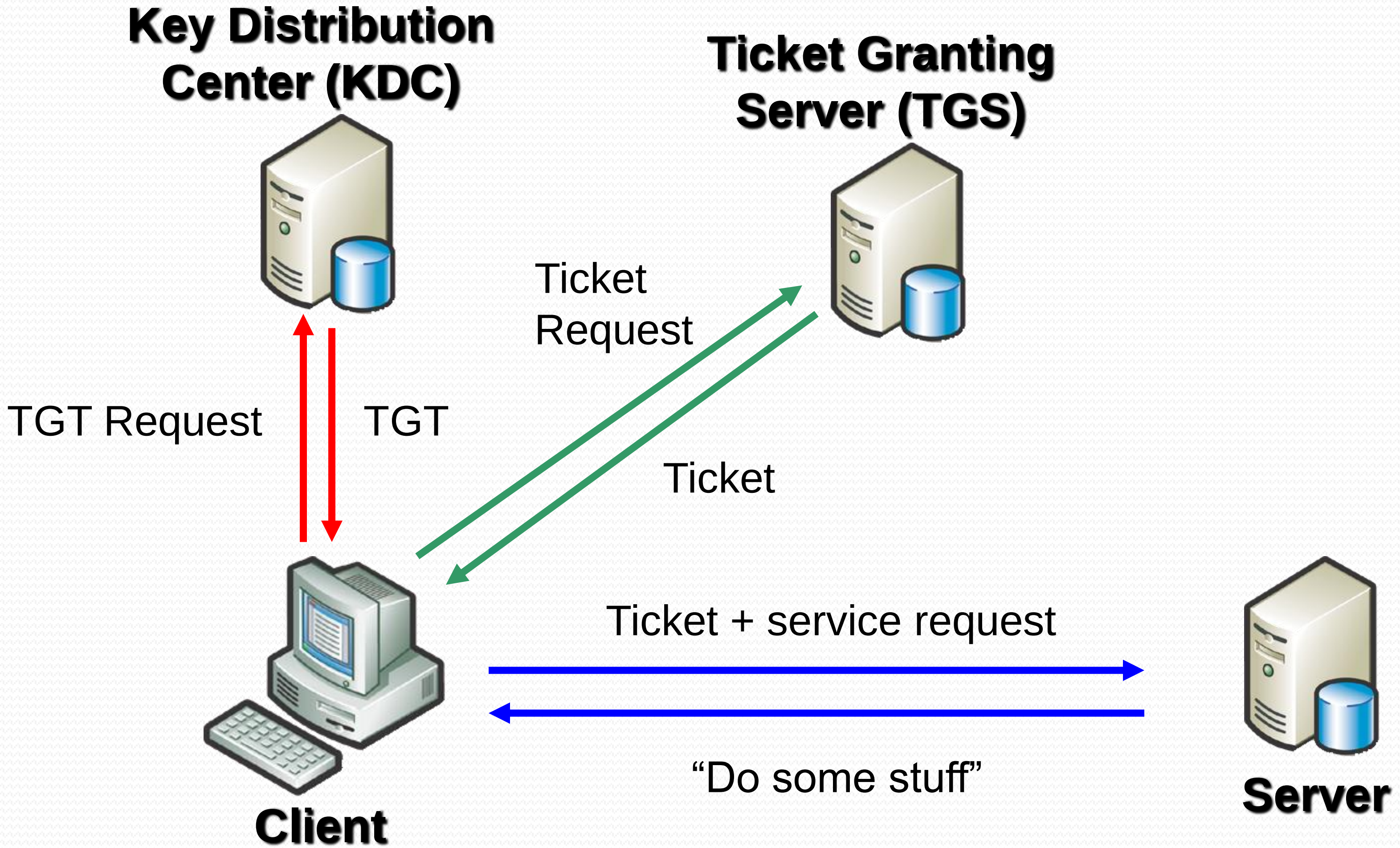
Picture of a Kerberos Realm

C ➔ S: $\{A_{C,S}\}\ K_{c,s},\ T_{C,S}$
where $A_{c,s}$ = {c, timestamp, opt. subkey}



S ➔ C: {timestamp, opt. subkey}$K_{c,s}$

Picture of a Kerberos Realm



Thoughts on Kerberos...

- Only the KDC needs to know the user's password (used to generate the shared secret)
  - You can have multiple KDCs for redundancy, but they all need to have a copy of the username/password database
- Only the TGS needs to know the secret keys for the servers
  - You can split KDC from TGS, but it is common for those two services to reside on the same physical machine

Thoughts on Kerberos…(2)

- “Time” is very important in Kerberos
  - All participants in the realm need accurate clocks
  - Timestamps are used in authenticators to detect replay; if a host can be fooled about the current time, old authenticators could be replayed
  - Tickets tend to have lifetimes on the order of hours, and replays are possible during the lifetime of the ticket

Thoughts on Kerberos…(3)

- Password-guessing attacks are possible
  - Capture enough encrypted tickets and you can brute-force decrypt them to discover shared keys
- It's possible to screw up the implementation
  - In fact, Kerberos v4 had a colossal security breach due to bad implementations

RNGs in Kerberos v4

- Session keys were generated from a PRNG seeded with the XOR of the following:
  - Time-of-day in seconds since 1/1/1970
  - Process ID of the Kerberos server process
  - Cumulative count of session keys generated
  - Fractional part of time-of-day seconds
  - Hostid of the machine running the server

RNGs in Kerberos v4 (continued)

- The seed is a 32-bit value, so while the session key is used for DES (64 bits long, normally 56 bits of entropy), it has only 32 bits of entropy
- What’s worse, the five values have predictable portions
  - Time is completely predictable
  - ProcessID is mostly predictable
  - Even hostID has 12 predictable bits (of 32 total)

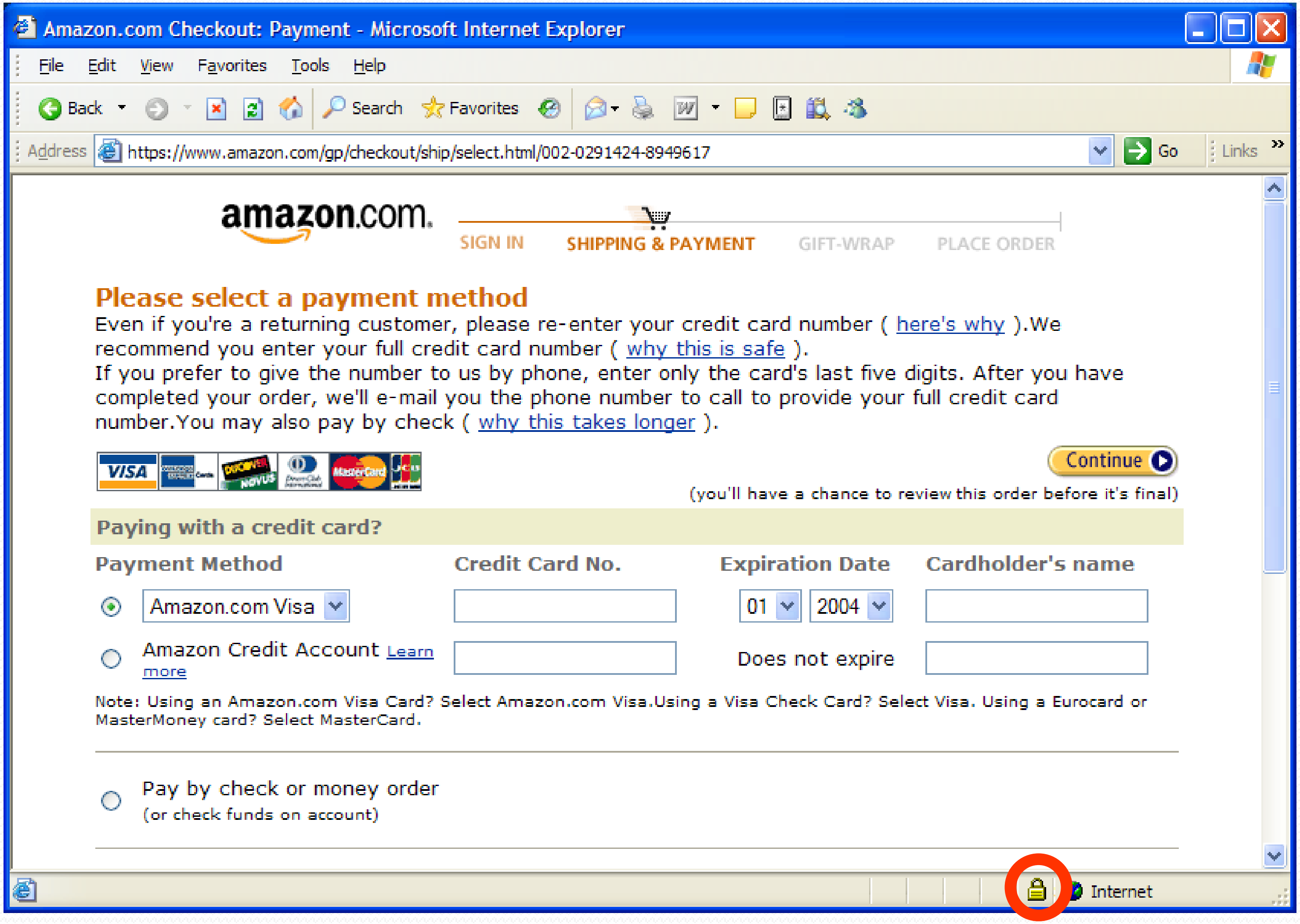
RNGs in Kerberos v4 (continued)

- Of the 32 seed bits, only 20 bits really change with any frequency, so Kerberos v4 keys (in the MIT implementation) only have 20 bits of randomness
  - They could be brute-force discovered in seconds
- The hole was in the MIT Kerberos sources for seven years!

Agenda

- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Public Key Infrastructure (PKI)

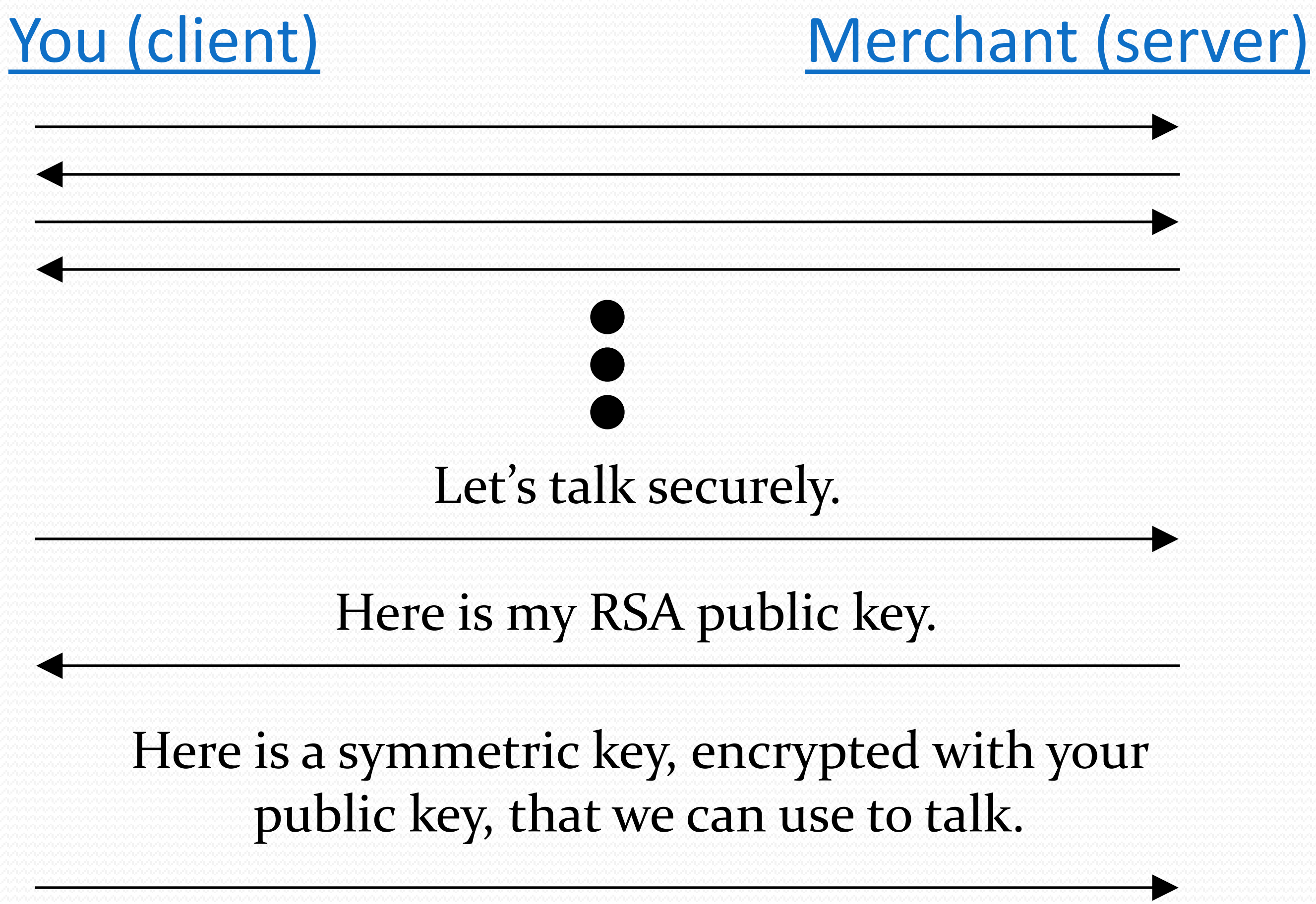
App-Level Security: SSL/TLS



SSL/PCT/TLS History

- 1994: Secure Sockets Layer (SSL) V2.0
- 1995: Private Communication Technology (PCT) V1.0
- 1996: Secure Sockets Layer (SSL) V3.0
- 1997: Private Communication Technology (PCT) V4.0
- 1999: Transport Layer Security (TLS) V1.0
- 2006: TLS V1.1 (RFC 4346)
- 2008: TLS V1.2 (RFC 5246)

Typical Scenario



SSL/TLS

You (client) Merchant (server)

Let's talk securely.

Here is my RSA public key.

Here is a symmetric key, encrypted with your public key, that we can use to talk.

SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.

→

Here is my RSA public key.

←

Here is a symmetric key, encrypted with your public key, that we can use to talk.

→

SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.

→

I choose this protocol and ciphers.
Here is my public key and
some other stuff.

←

Here is a symmetric key, encrypted with your public key, that we can use to talk.

→

SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.

→

I choose this protocol and ciphers.
Here is my public key and
some other stuff.

←

Using your public key, I've encrypted a
random symmetric key to you.

→

SSL/TLS

- All subsequent secure messages are sent using the symmetric key and a keyed hash for message authentication.

The five phases of SSL/TLS

1. Negotiate the ciphersuite to be used
2. Establish the shared session key
3. Client authenticates the server ("server auth")
   - Optional, but almost always done
4. Server authenticates the client ("client auth")
   - Optional, and almost never done
5. Authenticate previously exchanged data

Phase 1: Ciphersuite Negotiation

- Client hello (client→server)
  - “Hi! I speak these n ciphersuites, and here’s a 28-byte random number (nonce) I just picked”
- Server hello (client←server)
  - “Hello. We’re going to use this particular ciphersuite, and here’s a 28-byte nonce I just picked.”
- Other info can be passed along (we’ll see why a little later...)

TLS V1.0 ciphersuites

TLS_NULL_WITH_NULL_NULL
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_DSS_WITH_DES_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
TLS_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_WITH_DES_CBC_SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA

TLS-With-AES ciphersuites (RFC 3268)

```
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA

TLS_RSA_WITH_AES_256_CBC_SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA
```

ECC-based ciphersuites (RFC 4492)

```
TLS_ECDH_ECDSA_WITH_NULL_SHA
TLS_ECDH_ECDSA_WITH_RC4_128_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA

TLS_ECDHE_ECDSA_WITH_NULL_SHA
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
```

```
TLS_ECDH_RSA_WITH_NULL_SHA
TLS_ECDH_RSA_WITH_RC4_128_SHA
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_NULL_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

TLS_ECDH_anon_WITH_NULL_SHA
TLS_ECDH_anon_WITH_RC4_128_SHA
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_anon_WITH_AES_128_CBC_SHA
TLS_ECDH_anon_WITH_AES_256_CBC_SHA
```

Phase 2: Establish the shared session key

- Client key exchange
  - Client chooses a 48-byte "pre-master secret"
  - Client encrypts the pre-master secret with the server's RSA public key
  - Client➔server encrypted pre-master secret
- Client and server both compute
  - PRF (pre-master secret, "master secret", client nonce + server nonce)
  - PRF is a pseudo-random function
  - First 48 bytes output from PRF form master secret

TLS's PRF (V1.0 & V1.1)

- PRF(secret, label, seed) =
  P_MD5(S1, label + seed) XOR
  P_SHA-1(S2, label + seed);
  where S1, S2 are the two halves of the secret
- P_hash(secret, seed) =
  HMAC_hash(secret, A(1) + seed) + HMAC_hash(secret, A(2) + seed) + HMAC_hash(secret, A(3) + seed) + ...
- A(0) = seed
  A(i) = HMAC_hash(secret, A(i-1))

Phases 3 & 4: Authentication

- More on this in a moment...

Phase 5: Authenticate previously exchanged data

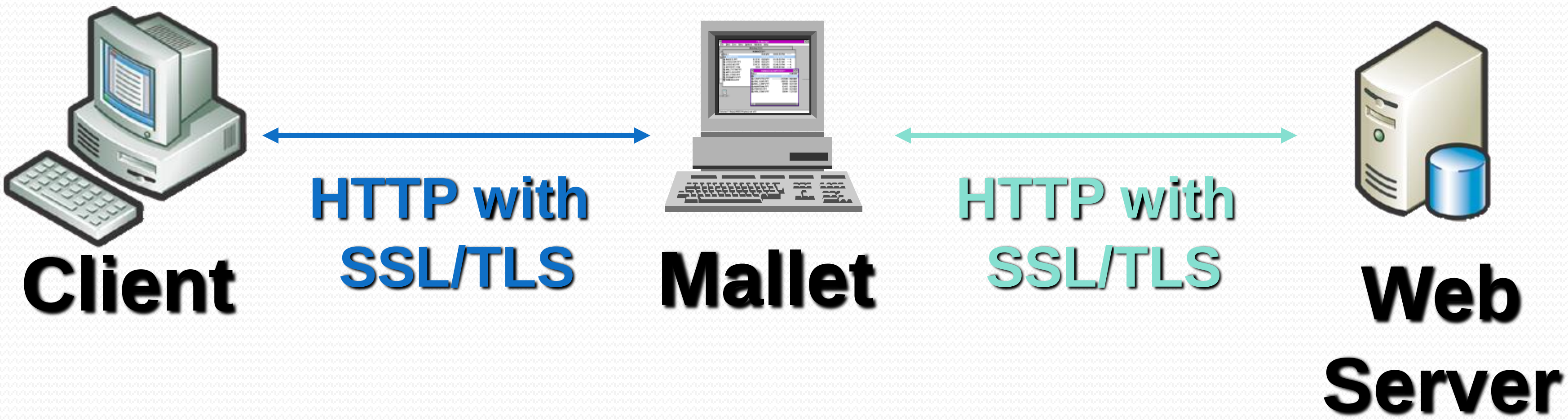
- “Change ciphersuites” message
  - Time to start sending data for real...
- “Finished” handshake message
  - First protected message, verifies algorithm parameters for the encrypted channel
  - 12 bytes from:
    PRF(master_secret, “client finished”,
    MD5(handshake_messages) +
    SHA-1(handshake_messages))

Why do I trust the server key?

- How do I know I'm really talking to Amazon.com?
- What defeats a man-in-the-middle attack?

Why do I trust the server key?

- How do I know I'm really talking to Amazon.com?
- What defeats a man-in-the-middle attack?



SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.

→

I choose this protocol and ciphers.
Here is my public key and
some other stuff that will make you
trust this key is mine.

←

Here is a fresh key encrypted with your key.

→

What's the "some other stuff"

- How can we convince Alice that some key belongs to Bob?
- Alice and Bob could have met previously & exchanged keys directly.
  - Jeff Bezos isn't going to shake hands with everyone he'd like to sell to...
- Someone Alice trusts could vouch to her for Bob and Bob's key
  - A third party can certify Bob's key in a way that convinces Alice.

Agenda

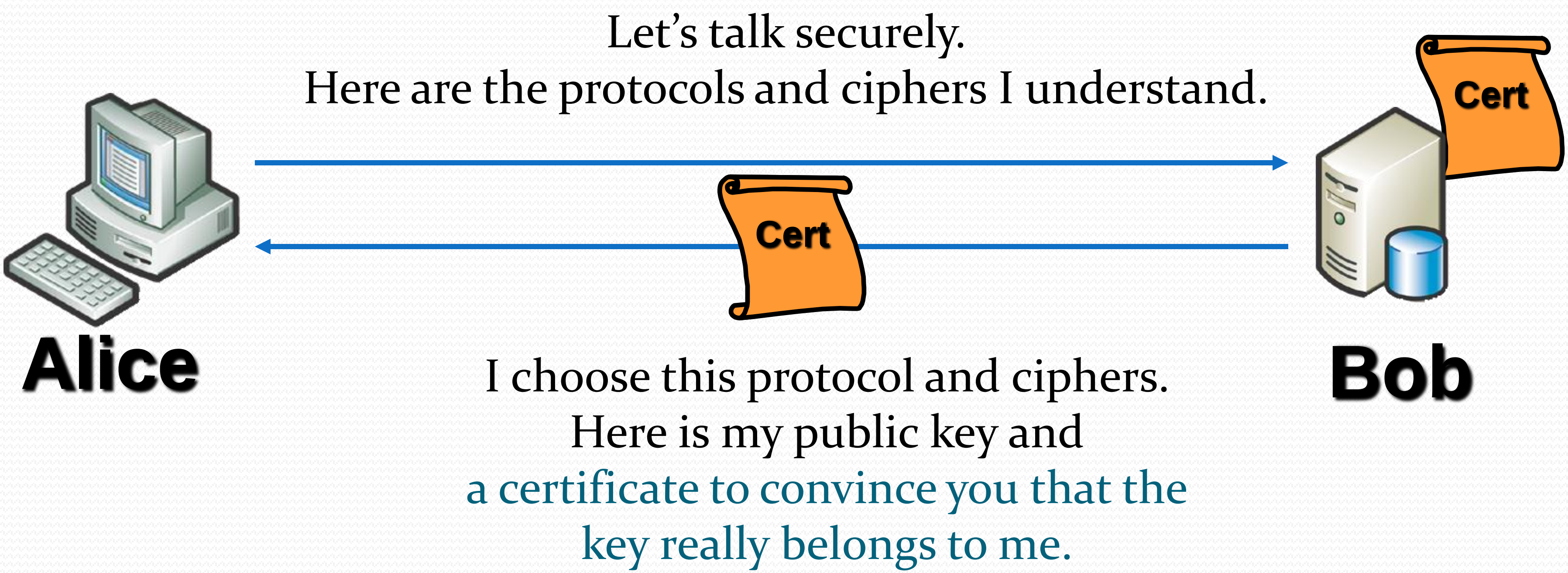
- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

What is a certificate?

- A certificate is a digitally-signed statement that binds a public key to some identifying information.
  - The signer of the certificate is called its issuer.
  - The entity talked about in the certificate is the subject of the certificate.
- That's all a certificate is, at the 30,000' level.

Defeating Mallet

- Bob can convince Alice that his key really does belong to him if he can also send along a digital certificate Alice will believe & trust



Certificates are Like Marriage

By the power vested in me I now declare this text and this bit string "name" and "key." What RSA has joined, let no man put asunder.

--Bob Blakley

Certs in the "real world"

- A driver's license is *like* a certificate
  - It is a "signed" document (sealed, tamper-resistant)
  - It is created and signed by an "issuing authority" (the WA Dept. of Licensing)
  - It binds together various pieces of identifying information
    - Name
    - License number
    - Driving restrictions (must wear glasses, etc.)

More certs in the real world

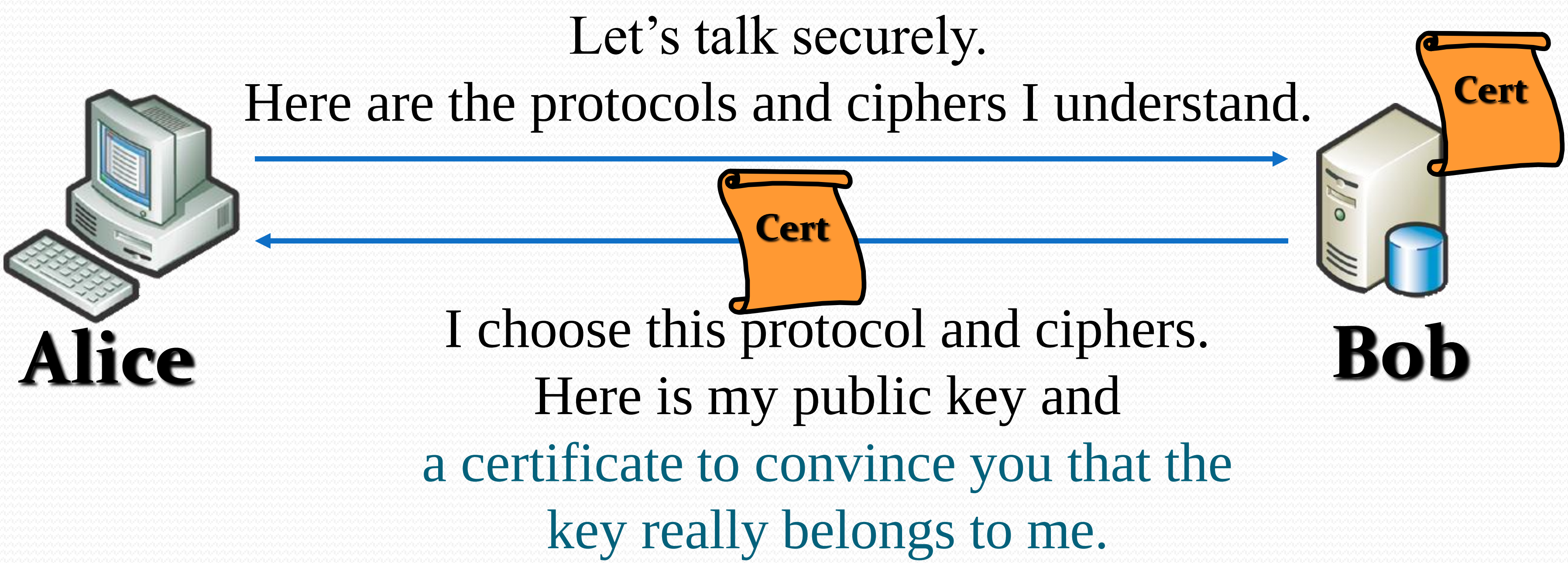
- Many physical objects are like certificates:
  - Any type of license – vehicle tabs, restaurant liquor license, amateur radio license, etc.
  - Government-issued IDs (passports, green cards)
  - Membership cards (e.g. Costco, discount cards)
- All of these examples bind an identity and certain rights, privileges or other identifiers
  - “BAL ==N1TJT” signed FCC

Why do we believe what certs say?

- In the physical world, why do we trust the statements contained on a physical cert?
  - We believe it’s hard to forge the cert
  - We trust the entity that “signed” the cert
- In the digital world we need those same two properties
  - We need to believe it’s hard to forge the digital signature on a signed document
  - We need to trust the issuer/signer not to lie to us

Defeating Mallet

- Bob can convince Alice that his key really does belong to him if he can also send along a digital certificate Alice will believe & trust



Getting a certificate

- How does Bob get a certificate for his key?
- He goes to a Certificate Authority (CA) that issues certificates and asks for one...
- The CA *issues* Bob a certificate for his public key.
  - CA is the issuer
  - Bob is the subject

Using Certificates

- Now that Bob has a certificate, is it useful?
- Alice will believe Bob's key belongs to Bob if Alice believes the certificate Bob gives her for his key.
- Alice will believe Bob's key belongs to Bob if Alice trusts the issuer of Bob's certificate to make key-name binding statements
- Have we made the situation any better?

Does Alice Trust Bob's CA?

- How can we convince Alice to trust Bob's CA?
- Alice and Bob's CA could have met previously & exchanged keys directly.
  - Bob's CA isn't going to shake hands with everyone he's certified, let alone everyone whom Bob wants to talk to.

Does Alice Trust Bob's CA?

- How can we convince Alice to trust Bob's CA?
- Alice and Bob's CA could have met previously & exchanged keys directly.
  - *Bob's CA isn't going to shake hands with everyone he's certified, let alone everyone whom Bob wants to talk to.*
- Someone Alice trusts could vouch to her for Bob's CA and Bob's CA's key
  - *Infinite Loop: See Loop, Infinite.*
  - Actually, it's just a bounded recursion...

What's Alice's Trust Model

- Alice has to implicitly trust some set of keys
  - Once she does that, those keys can introduce others to her.
- In the model used by SSL/TLS, CAs are arranged in a hierarchy
  - Alice, and everyone else, trusts one or more "root CA" that live at the top of the tree
- Other models work differently

Agenda

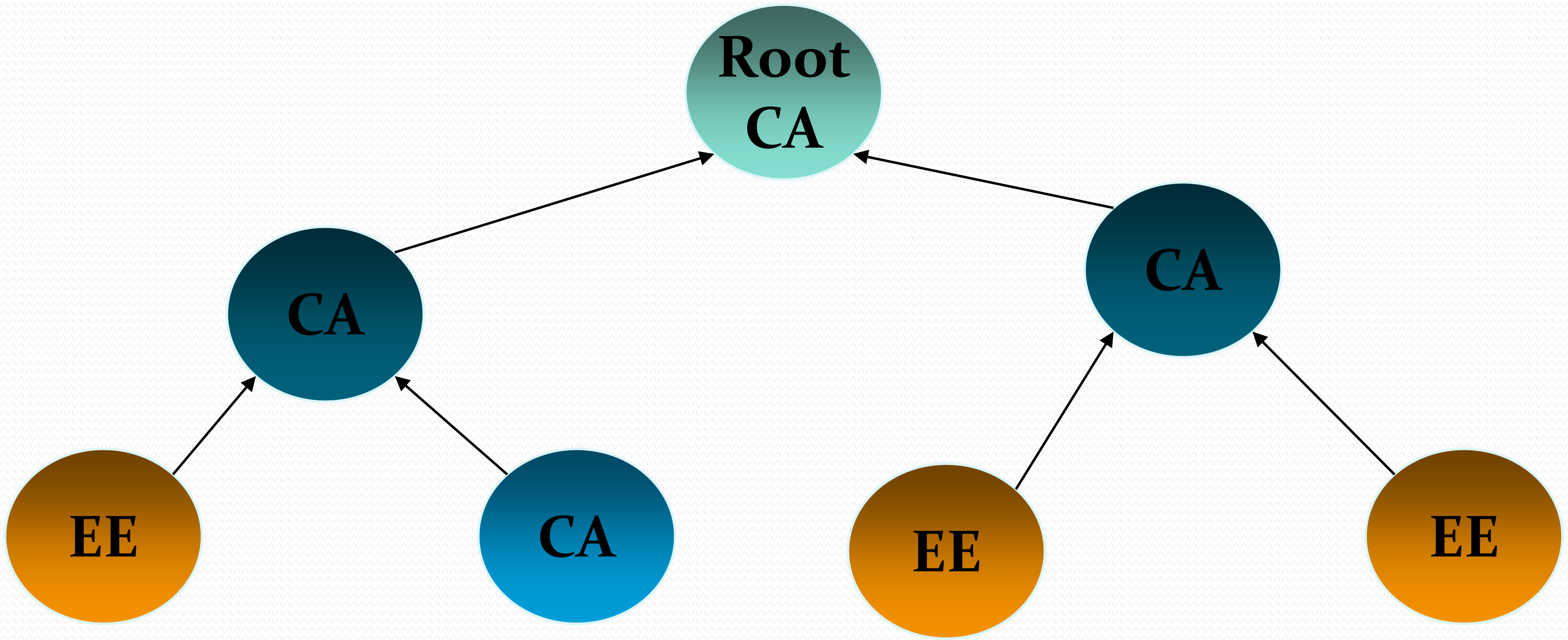
- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

Certificate Authorities

- A certificate authority (CA) guarantees the connection between a key and another CA or an "end entity."
- An end entity is:
  - A person
  - A role ("VP of sales")
  - An organization
  - A pseudonym
  - A piece of hardware or software
  - An account
- Some CA's only allow a subset of these types.

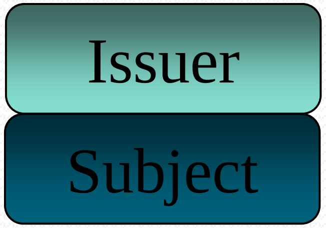
CA Hierarchies

- CAs can certify other CAs or “end entities” (EEs)
- Certificates are links in a tree of EEs & CAs



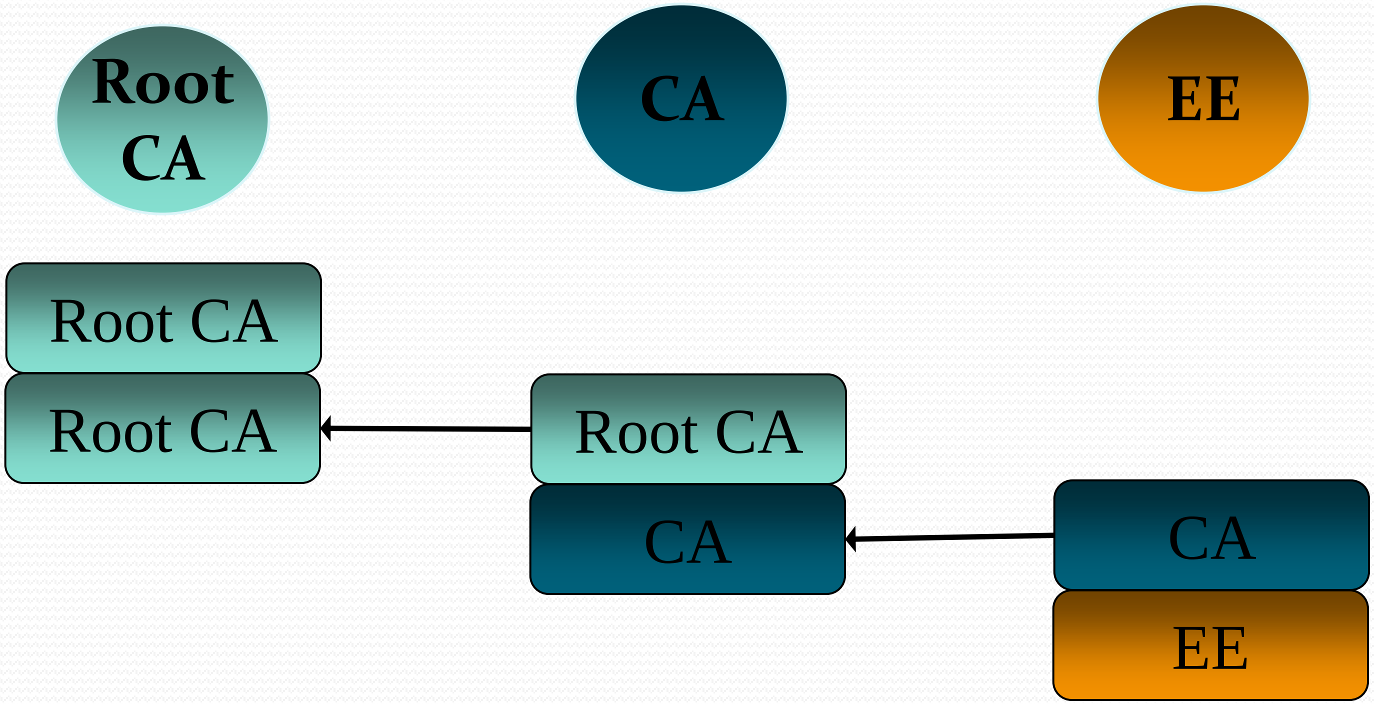
BAL's No-Frills Certs

- Certificates can contain all sorts of information inside them
  - We'll talk about the details in a little bit
- In the abstract, though, they're just statements by an issuer about a subject:



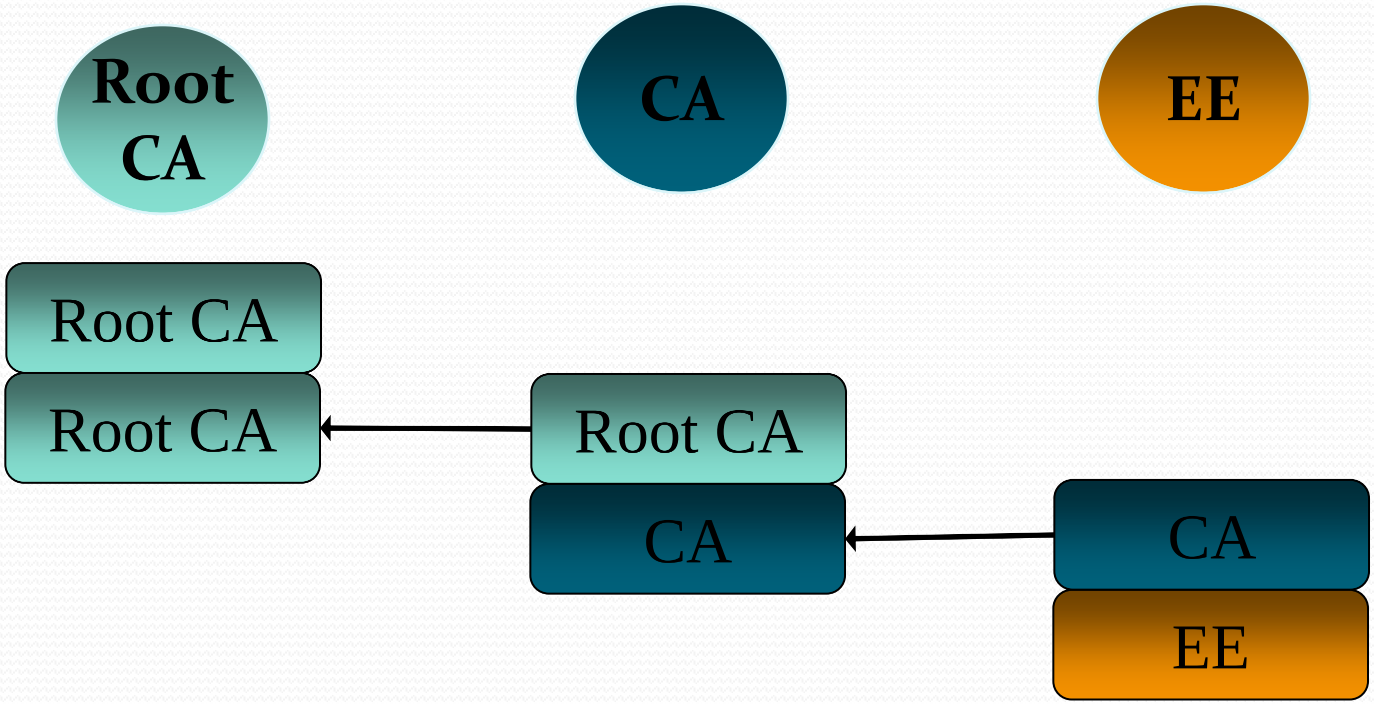
Does Alice trust Bob's Key?

- Alice trusts Bob's key if there is a chain of certificates from Bob's key to a root CA that Alice implicitly trusts



Chain Building & Validation

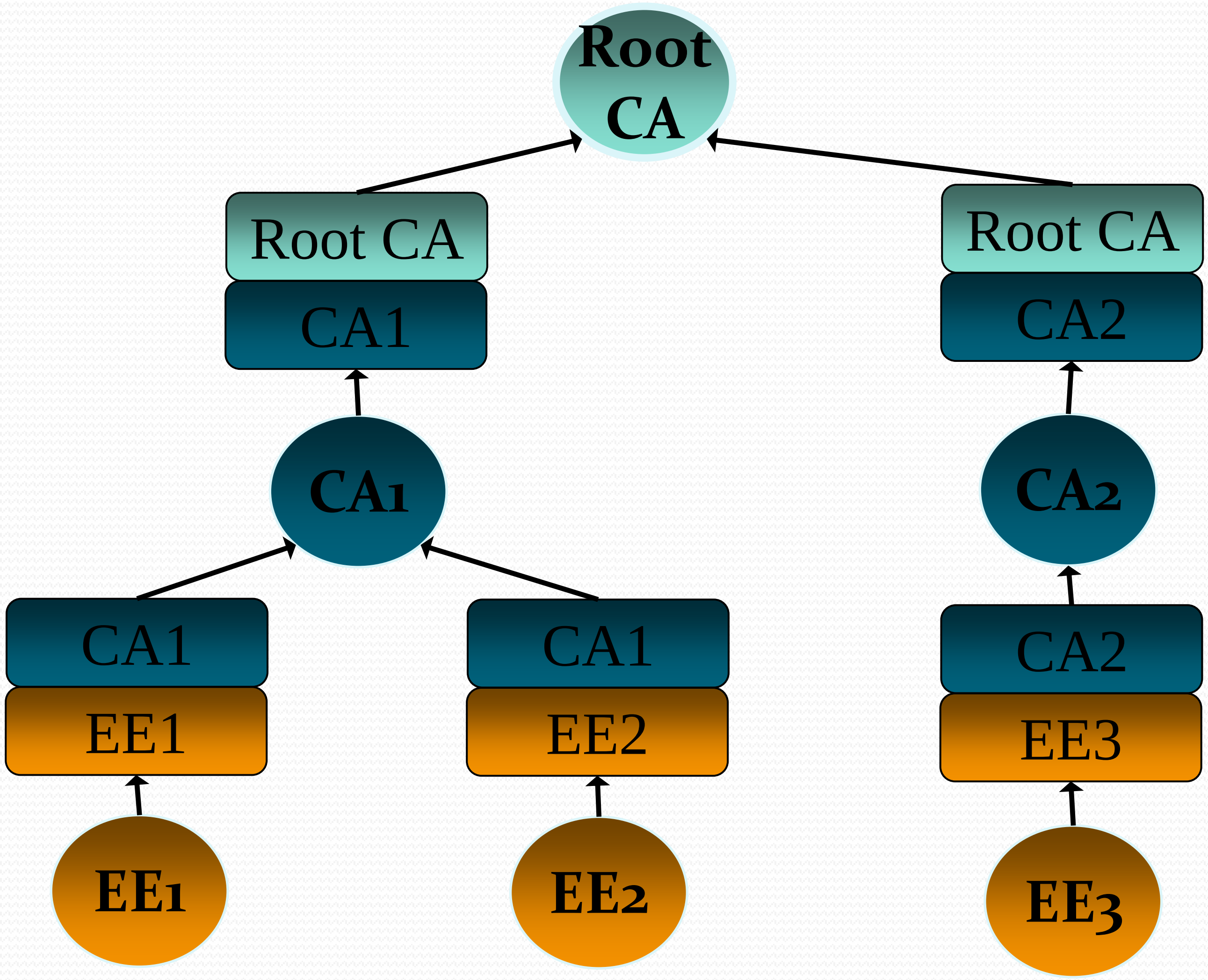
- "Given an end-entity certificate, does there exist a cryptographically valid chain of certificates linking it to a trusted root certificate?"



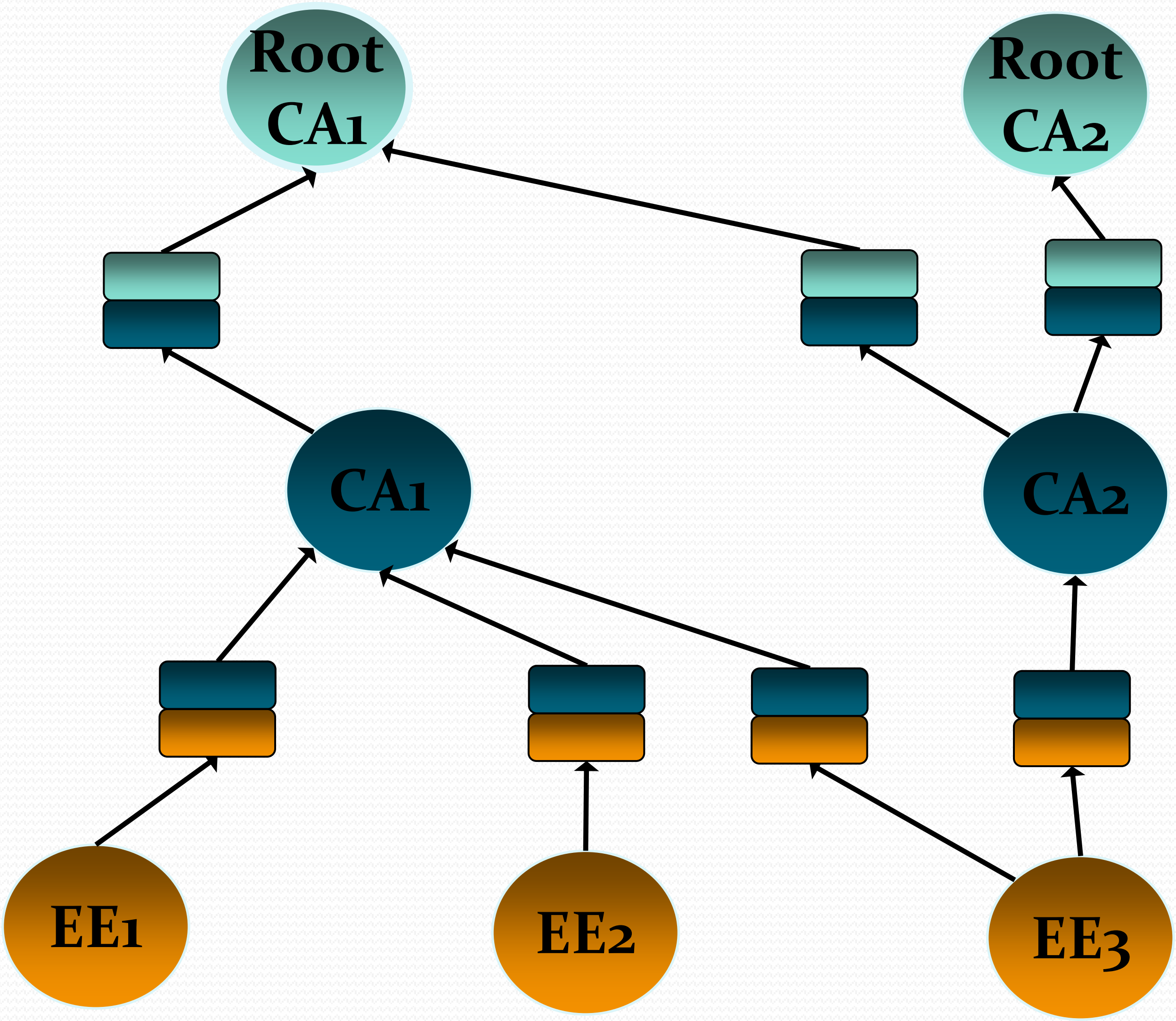
Chaining Certificates

- In theory, building chains of certificates should be easy
  - "Just link them together like dominos"
- In practice, it's a lot more complicated...

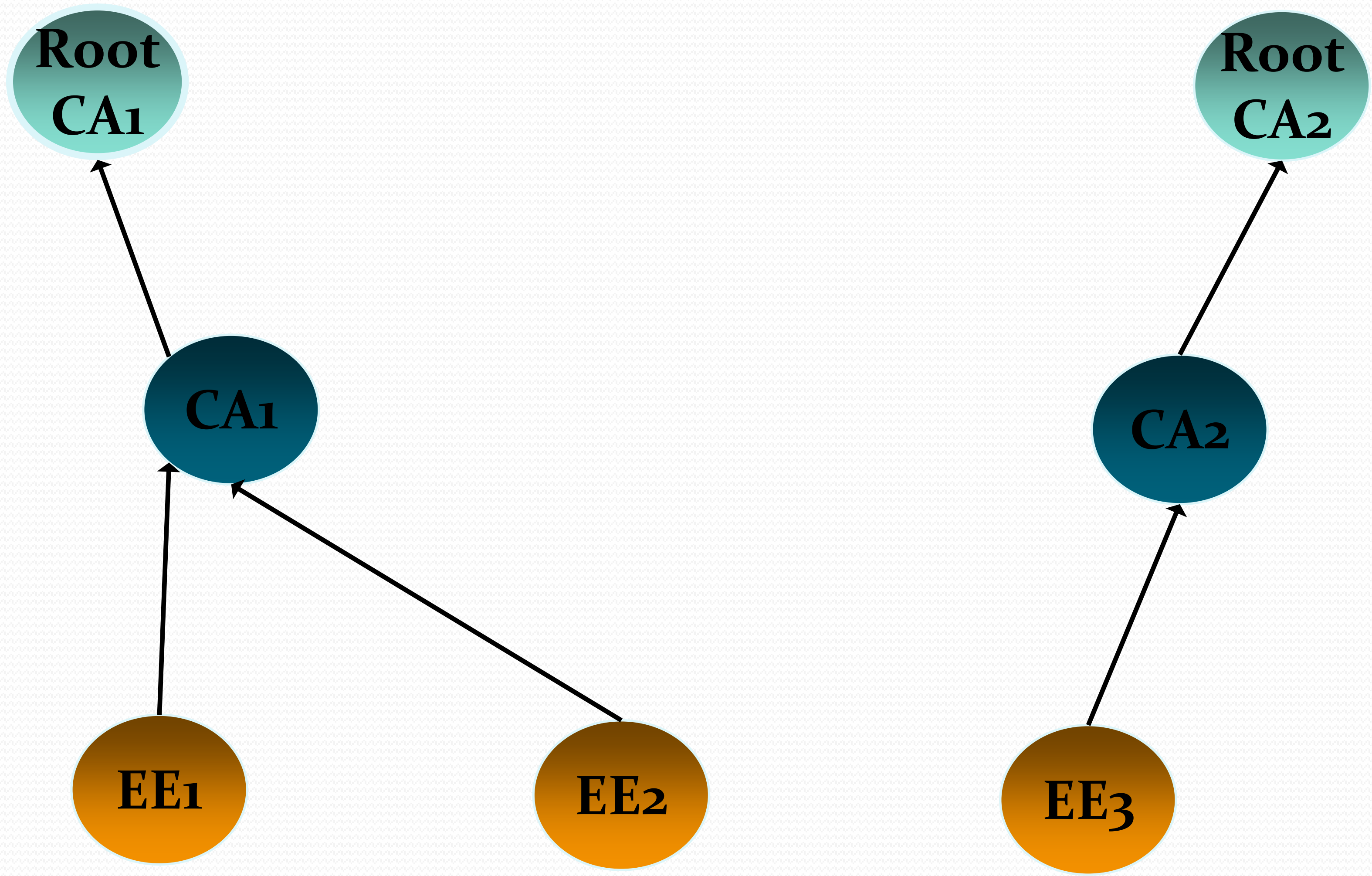
Chain Building Details (1)



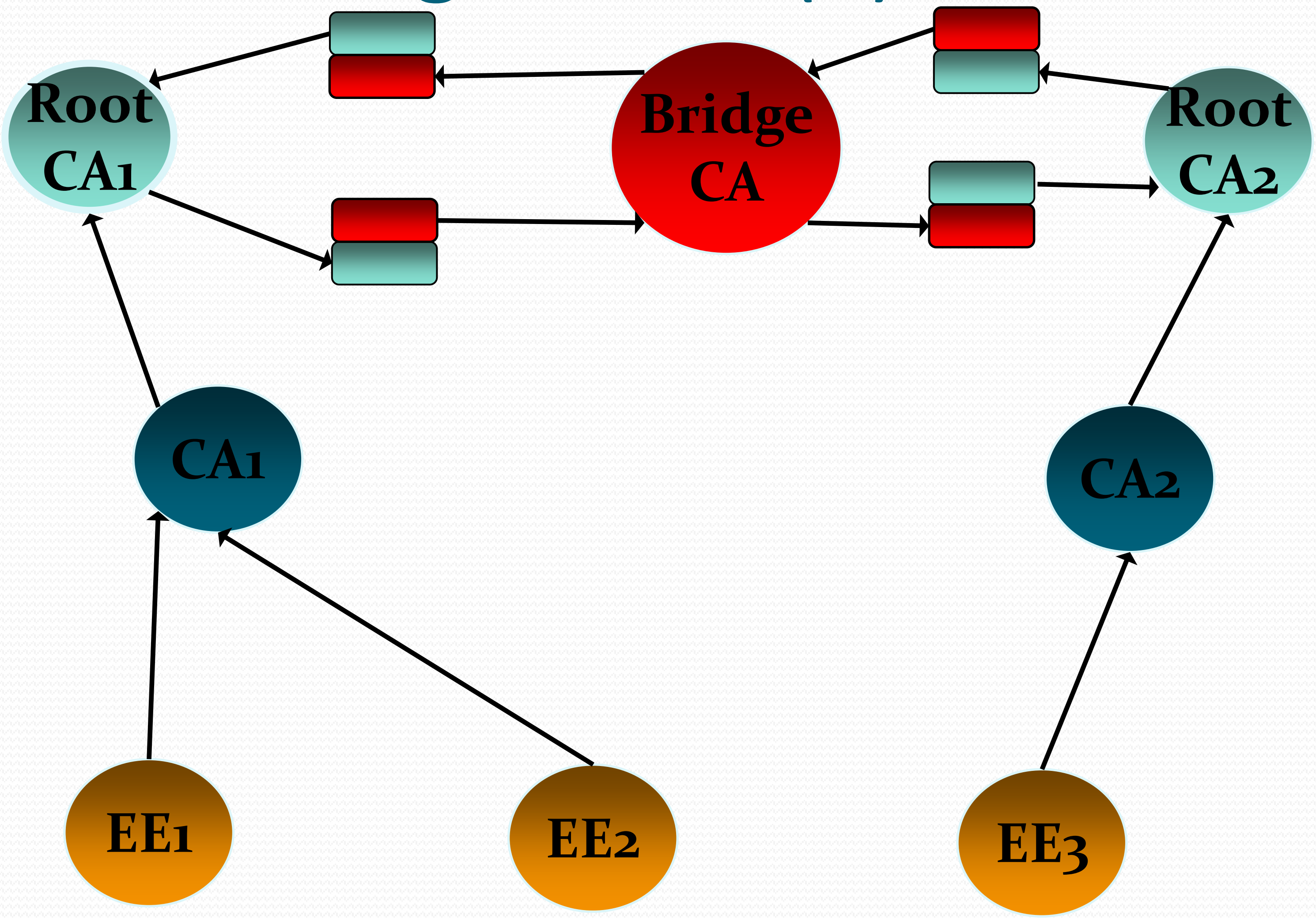
Chain Building Details (2)



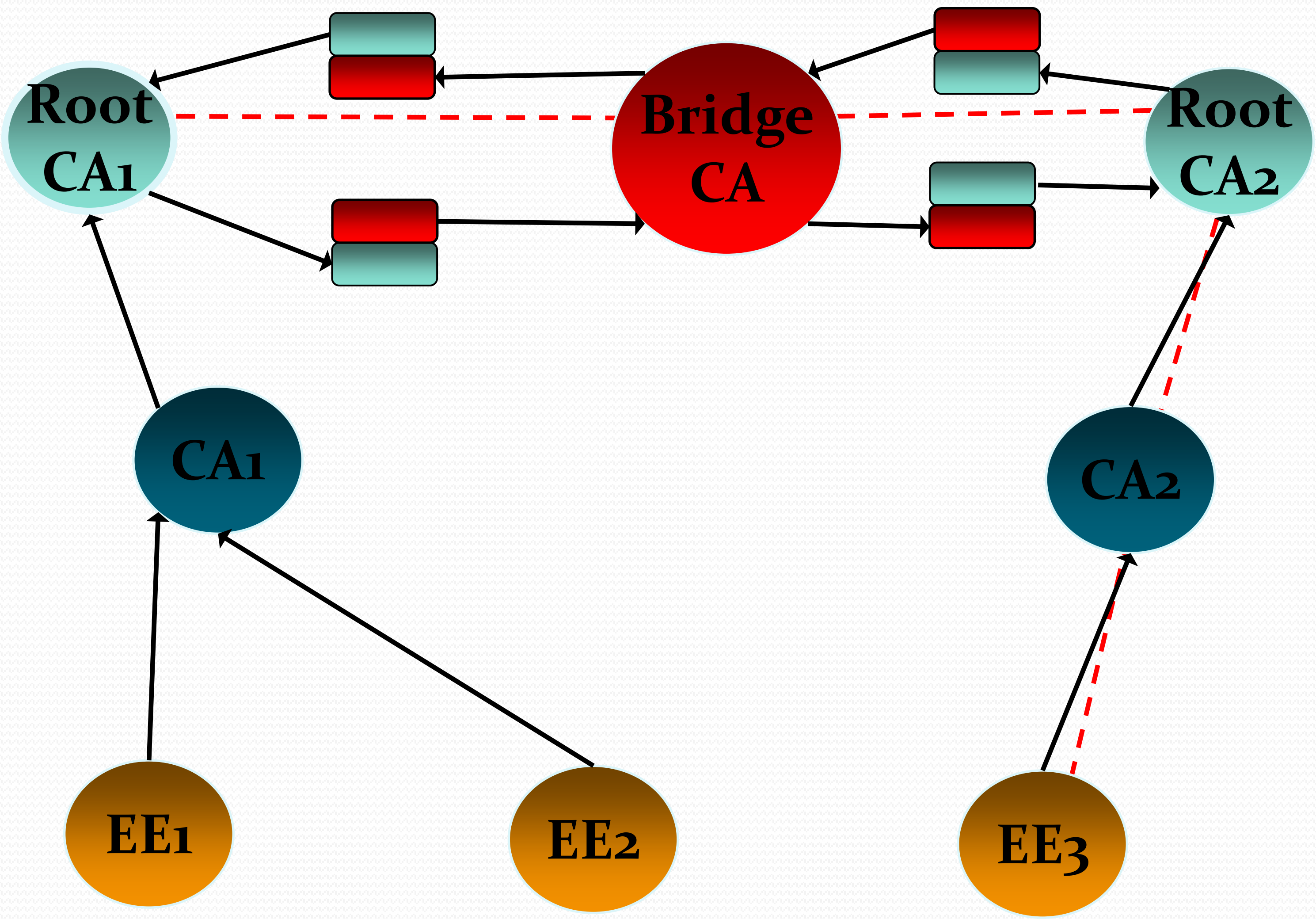
Chain Building Details (3)

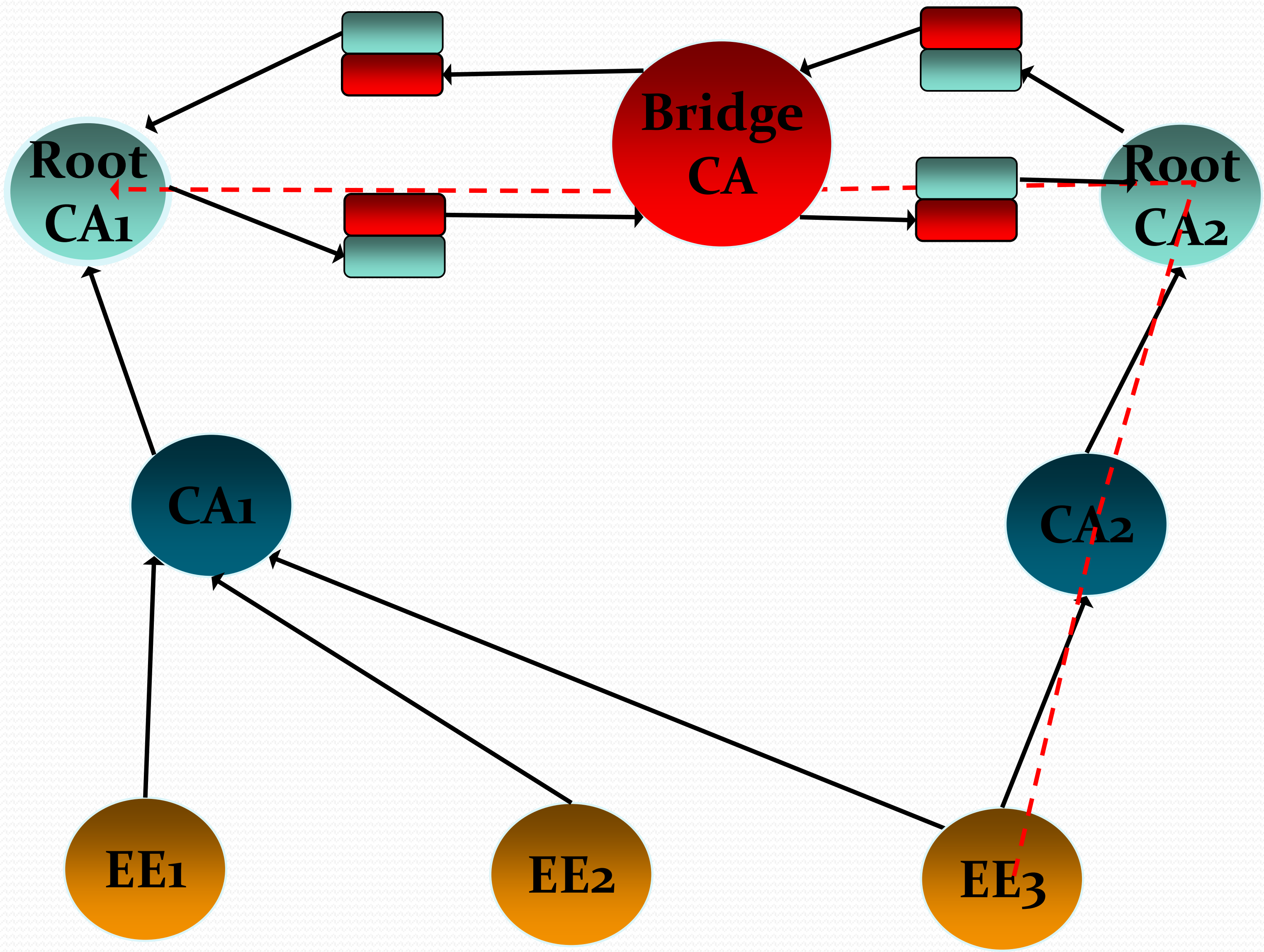


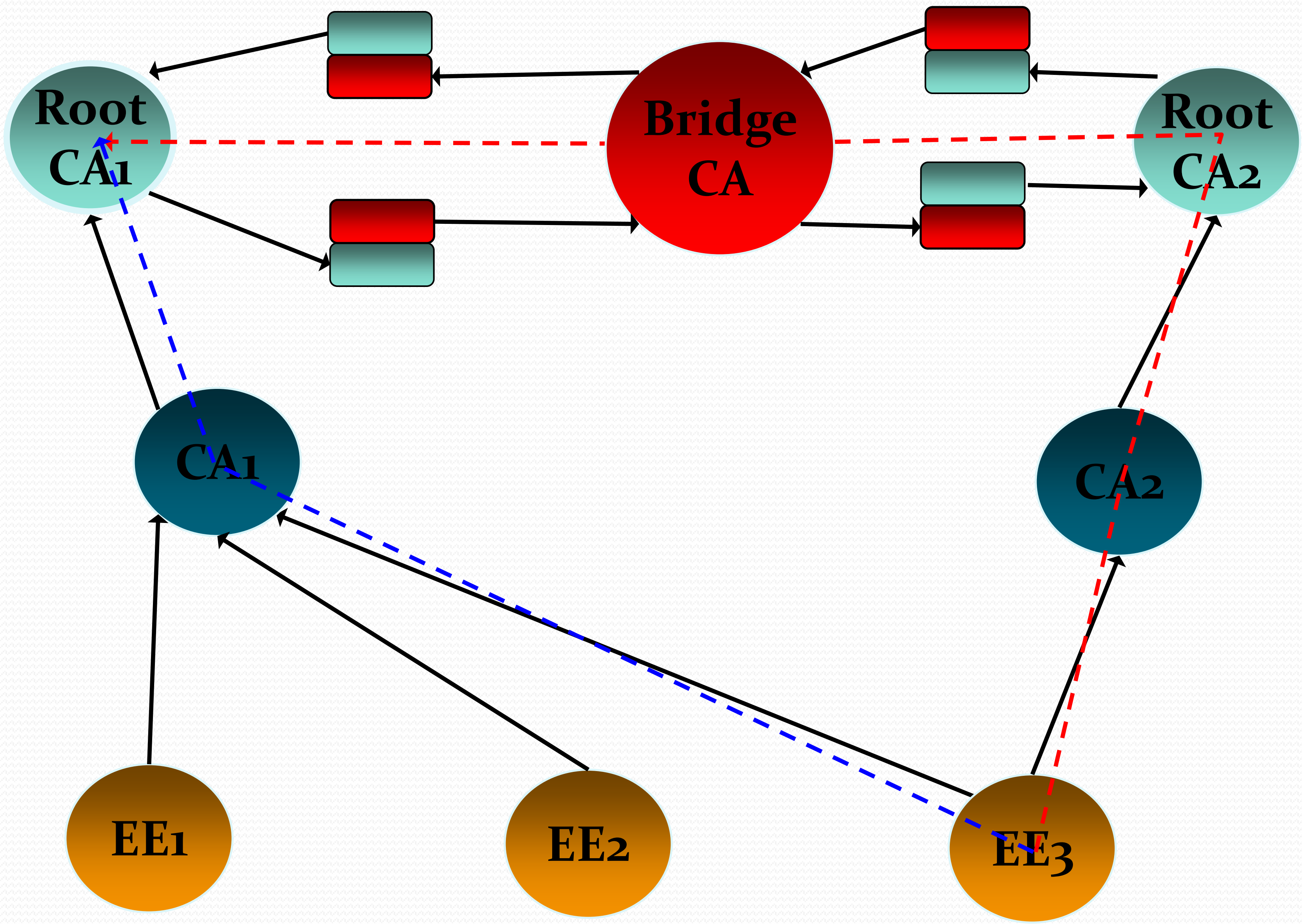
Chain Building Details (3)



Chain Building Details (3)




Bridge CA
Root CA1
Root CA2
CA1
CA2
EE1
EE2
EE3


Root
CA1
Bridge
CA
Root
CA2
CA1
CA2
EE1
EE2
EE3

Chaining Certificates

- How do we determine whether two certificates chain together?
  - You'd think this was an easy problem...
  - But it's actually a question with religious significance in the security community
  - "Are you a believer in names, or in keys?"
- The model SSL/TLS uses, the X.509 certificate model, is based on names
  - *"Names as principles"*

PKI Alphabet Soup

- X.509v3 - standard content of a certificate
- PKIX – IETF Working Group on PKI interoperability
  - PKIX == Public Key Infrastructure using X.509v3 certificates
- ASN.1 - Abstract Syntax Notation, exact description of a certificate format
- DER - Distinguished Encoding Rules, how to physically package a certificate

Key fields in a certificate

- The core fields of an X.509 certificate are
  - The subject public key
  - The subject Distinguished Name
  - The issuer Distinguished Name
- What's missing here?

Key fields in a certificate

- The core fields of an X.509 certificate are
  - The subject public key
  - The subject Distinguished Name
  - The issuer Distinguished Name
- What's missing here?
  - The issuer's public key is not present in the certificate.
  - You can't verify the signature on the cert without finding a parent cert!

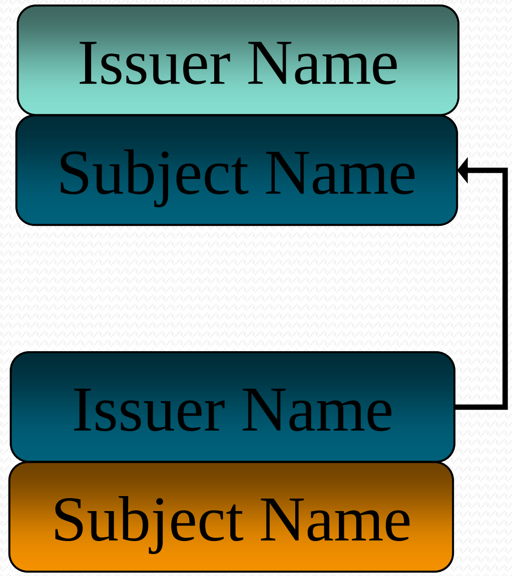
Back to Chain Building

- OK, assume we're a "relying party application" -- something that received an end-entity certificate and wants to verify it.
  - Our task is to build a cert chain from that end-entity cert to one of our trusted roots
- How do we do that?
  - We start with our EE cert, and using the information contained within we look for possible parent certificates.

Parent certs

- What’s a valid parent certificate?
  - In the raw X.509 model, parent-child relationships are determined solely by matching Issuer DN in the child to Subject DN in the parent
  - Recall that there’s an assumption that you have a big directory handy to find certs.
- If you don’t have a directory handy, you need to do the matching yourself
  - This is not as easy as you might think…

Name matching



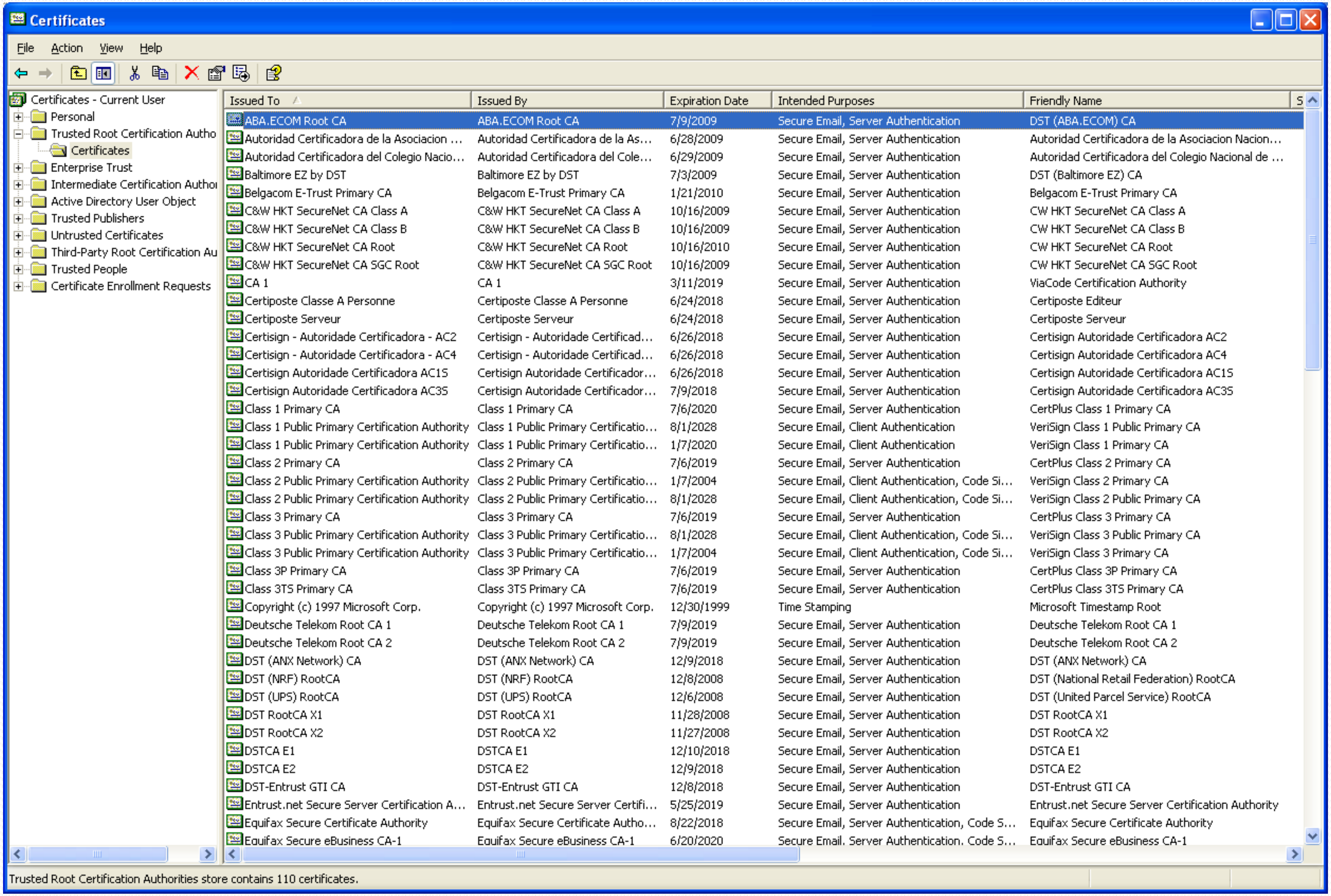
Even More Chain Building

- Name matching is just the beginning of the chain-building process
  - It is necessary that subject and issuer DNs exactly match for two certs to chain, but not always sufficient
- The chain building process is also influenced dynamically by other information contained within the certs themselves
  - Certificate Extensions

Trusted Root Certificates

- Who do I trust to be roots at the top of the cert chain?
- In theory, “anyone you want”
- In practice, trusted roots come from two sources
  - They’re baked into your web browser or operating system
  - They’re pushed onto your “enterprise managed desktop”

Trusted Root Certificates



Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Name
ABA.ECOM Root CA	ABA.ECOM Root CA	7/9/2009	Secure Email, Server Authentication	DST (ABA.ECOM) CA
Autoridad Certificadora de la Asociacion ...	Autoridad Certificadora de la As...	6/28/2009	Secure Email, Server Authentication	Autoridad Certificadora de la Asociacion Nacion...
Autoridad Certificadora del Colegio Nacio...	Autoridad Certificadora del Cole...	6/29/2009	Secure Email, Server Authentication	Autoridad Certificadora del Colegio Nacional de ...
Baltimore EZ by DST	Baltimore EZ by DST	7/3/2009	Secure Email, Server Authentication	DST (Baltimore EZ) CA
Belgacom E-Trust Primary CA	Belgacom E-Trust Primary CA	1/21/2010	Secure Email, Server Authentication	Belgacom E-Trust Primary CA
C&W HKT SecureNet CA Class A	C&W HKT SecureNet CA Class A	10/16/2009	Secure Email, Server Authentication	CW HKT SecureNet CA Class A
C&W HKT SecureNet CA Class B	C&W HKT SecureNet CA Class B	10/16/2009	Secure Email, Server Authentication	CW HKT SecureNet CA Class B
C&W HKT SecureNet CA Root	C&W HKT SecureNet CA Root	10/16/2010	Secure Email, Server Authentication	CW HKT SecureNet CA Root
C&W HKT SecureNet CA SGC Root	C&W HKT SecureNet CA SGC Root	10/16/2009	Secure Email, Server Authentication	CW HKT SecureNet CA SGC Root
CA 1	CA 1	3/11/2019	Secure Email, Server Authentication	ViaCode Certification Authority
Certiposte Classe A Personne	Certiposte Classe A Personne	6/24/2018	Secure Email, Server Authentication	Certiposte Editeur
Certiposte Serveur	Certiposte Serveur	6/24/2018	Secure Email, Server Authentication	Certiposte Serveur
Certisign - Autoridade Certificadora - AC2	Certisign - Autoridade Certificad...	6/26/2018	Secure Email, Server Authentication	Certisign Autoridade Certificadora AC2
Certisign - Autoridade Certificadora - AC4	Certisign - Autoridade Certificad...	6/26/2018	Secure Email, Server Authentication	Certisign Autoridade Certificadora AC4
Certisign Autoridade Certificadora AC1S	Certisign Autoridade Certificador...	6/26/2018	Secure Email, Server Authentication	Certisign Autoridade Certificadora AC1S
Certisign Autoridade Certificadora AC3S	Certisign Autoridade Certificador...	7/9/2018	Secure Email, Server Authentication	Certisign Autoridade Certificadora AC3S
Class 1 Primary CA	Class 1 Primary CA	7/6/2020	Secure Email, Server Authentication	CertPlus Class 1 Primary CA
Class 1 Public Primary Certification Authority	Class 1 Public Primary Certificatio...	8/1/2028	Secure Email, Client Authentication	VeriSign Class 1 Public Primary CA
Class 1 Public Primary Certification Authority	Class 1 Public Primary Certificatio...	1/7/2020	Secure Email, Client Authentication	VeriSign Class 1 Primary CA
Class 2 Primary CA	Class 2 Primary CA	7/6/2019	Secure Email, Server Authentication	CertPlus Class 2 Primary CA
Class 2 Public Primary Certification Authority	Class 2 Public Primary Certificatio...	1/7/2004	Secure Email, Client Authentication, Code Si...	VeriSign Class 2 Primary CA
Class 2 Public Primary Certification Authority	Class 2 Public Primary Certificatio...	8/1/2028	Secure Email, Client Authentication, Code Si...	VeriSign Class 2 Public Primary CA
Class 3 Primary CA	Class 3 Primary CA	7/6/2019	Secure Email, Server Authentication	CertPlus Class 3 Primary CA
Class 3 Public Primary Certification Authority	Class 3 Public Primary Certificatio...	8/1/2028	Secure Email, Client Authentication, Code Si...	VeriSign Class 3 Public Primary CA
Class 3 Public Primary Certification Authority	Class 3 Public Primary Certificatio...	1/7/2004	Secure Email, Client Authentication, Code Si...	VeriSign Class 3 Primary CA
Class 3P Primary CA	Class 3P Primary CA	7/6/2019	Secure Email, Server Authentication	CertPlus Class 3P Primary CA
Class 3TS Primary CA	Class 3TS Primary CA	7/6/2019	Secure Email, Server Authentication	CertPlus Class 3TS Primary CA
Copyright (c) 1997 Microsoft Corp.	Copyright (c) 1997 Microsoft Corp.	12/30/1999	Time Stamping	Microsoft Timestamp Root
Deutsche Telekom Root CA 1	Deutsche Telekom Root CA 1	7/9/2019	Secure Email, Server Authentication	Deutsche Telekom Root CA 1
Deutsche Telekom Root CA 2	Deutsche Telekom Root CA 2	7/9/2019	Secure Email, Server Authentication	Deutsche Telekom Root CA 2
DST (ANX Network) CA	DST (ANX Network) CA	12/9/2018	Secure Email, Server Authentication	DST (ANX Network) CA
DST (NRF) RootCA	DST (NRF) RootCA	12/8/2008	Secure Email, Server Authentication	DST (National Retail Federation) RootCA
DST (UPS) RootCA	DST (UPS) RootCA	12/6/2008	Secure Email, Server Authentication	DST (United Parcel Service) RootCA
DST RootCA X1	DST RootCA X1	11/28/2008	Secure Email, Server Authentication	DST RootCA X1
DST RootCA X2	DST RootCA X2	11/27/2008	Secure Email, Server Authentication	DST RootCA X2
DSTCA E1	DSTCA E1	12/10/2018	Secure Email, Server Authentication	DSTCA E1
DSTCA E2	DSTCA E2	12/9/2018	Secure Email, Server Authentication	DSTCA E2
DST-Entrust GTI CA	DST-Entrust GTI CA	12/8/2018	Secure Email, Server Authentication	DST-Entrust GTI CA
Entrust.net Secure Server Certification A...	Entrust.net Secure Server Certifi...	5/25/2019	Secure Email, Server Authentication	Entrust.net Secure Server Certification Authority
Equifax Secure Certificate Authority	Equifax Secure Certificate Autho...	8/22/2018	Secure Email, Server Authentication, Code S...	Equifax Secure Certificate Authority
Equifax Secure eBusiness CA-1	Equifax Secure eBusiness CA-1	6/20/2020	Secure Email, Server Authentication, Code S...	Equifax Secure eBusiness CA-1

Agenda

- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

Lifecycle Management

- Certificate Enrollment
  - Initial acquisition of a certificate based on other authentication information
- Renewal
  - Acquiring a new certificate for a key when the existing certificate expires
- Revocation
  - “Undoing” a certificate

Certificate Enrollment

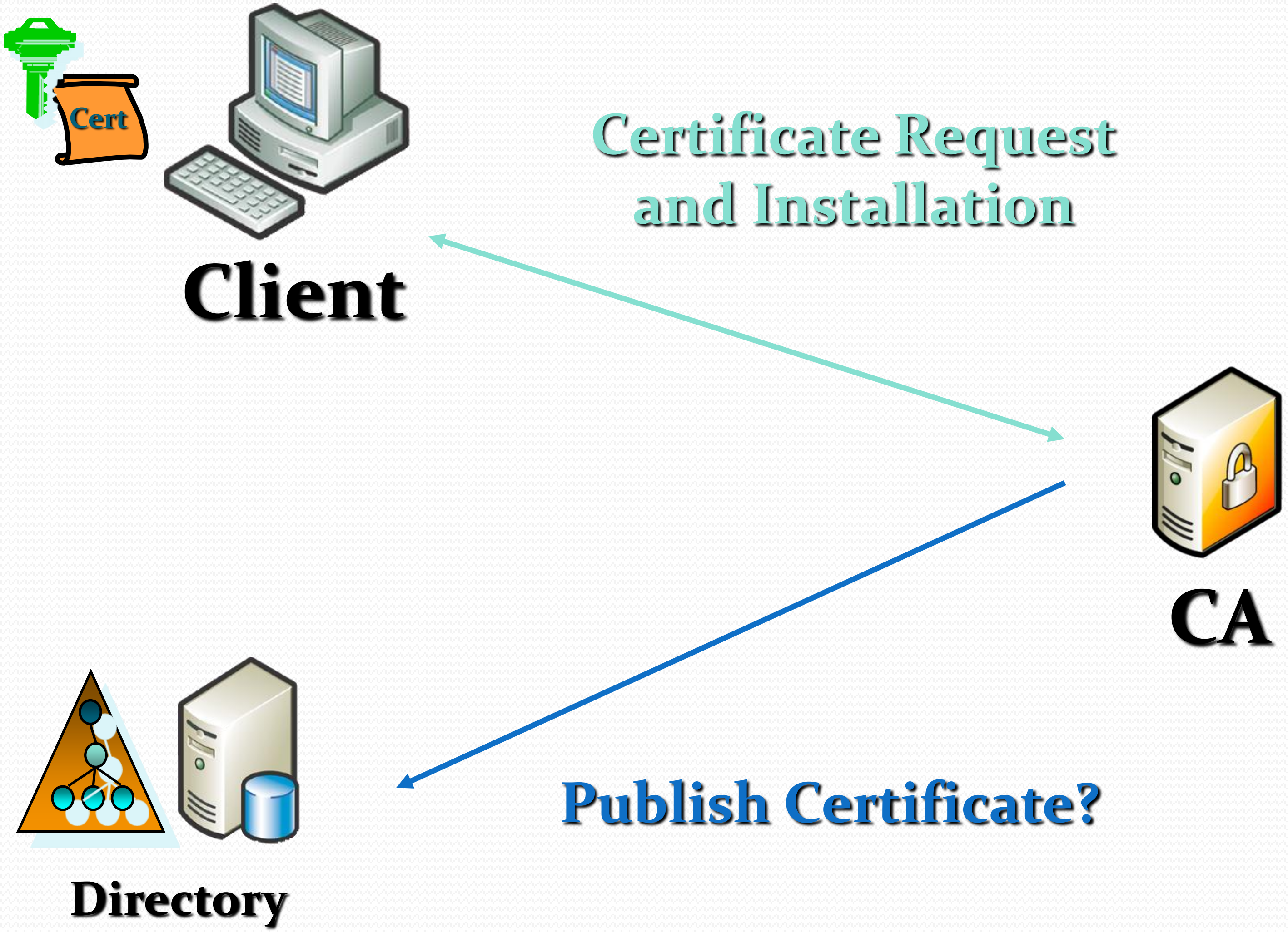
- *Enrollment* is the process of obtaining a certificate from a CA.

1. Alice generates a key pair, creates a message containing a copy of the public key and her identifying information, and signs the message with the private key (PKCS#10).
   - Signing the message provided "proof-of-possession" (POP) of the private key as well as message integrity
2. CA verifies Alice's signature on the message

Certificate Enrollment (2)

3. (Optional) CA verifies Alice's ID through out-of-band means.
4. CA creates a certificate containing the ID and public key, and signs it with the CA's own key
    - CA has certified the binding between key and ID
5. Alice verifies the key, ID & CA signature
6. Alice and/or the CA publish the certificate

Certificate Enrollment Flow



More PKI Alphabet Soup

- PKCS #10 – (old) standard message format for certificate requests
- PKCS #7 – (old) standard message format for encrypted/signed data
  - Also used for certificate request responses
  - Replaced by IETF CMS syntax
- CMC – “Certificate Management with CMS”
  - Replacement for PKCS #10/PKCS#7 in a certificate management context
- CMP – “Certificate Management Protocols”
  - Alternative to CMC

Agenda

- Integrity Checking
- Protocols (Part 1 – Session-based protocols)
  - Introduction
  - Kerberos
  - SSL/TLS
- Certificates and Public Key Infrastructure (PKI)
  - Certificates
  - Public Key Infrastructure
  - Certificate Lifecycle Management
  - Revocation

Expiration & Revocation

- Certificates (at least, all the ones we're concerned with) contain explicit validity periods – "valid from" & "expires on"
  - Expiration dates help bound the risk associated with issuing a certificate
- Sometimes, though, it becomes necessary to "undo" a certificate while it is still valid
  - Key compromise
  - Cert was issued under false pretenses
- This is called revoking a certificate

Status Info for Certificates

- Two standards within PKIX:
  - X.509v2/PKIX Part 1 Certificate Revocation Lists (CRLs)
  - Online Certificate Status Protocol (OCSP)
- Both methods state:
  - Whether a cert has been revoked
  - A “revocation code” indicating why the cert was revoked
  - The time at which the cert was revoked

Certificate Revocation

- A CA revokes a certificate by placing the cert on its Certificate Revocation List (CRL)
  - Every CA issues CRLs to cancel out issued certs
  - A CRL is like anti-matter – when it comes into contact with a certificate it lists it cancels out the certificate
  - Think “1970s-style credit-card blacklist”
- Relying parties are expected to check CRLs before they rely on a certificate
  - “The cert is valid unless you hear something telling you otherwise”

The Problem with CRLs

- Blacklists have numerous problems
  - Not issued frequently enough to be effective against a serious attack
  - Expensive to distribute (size & bandwidth)
  - Vulnerable to simple DOS attacks
    - If you block on lack of CRL access, why have off-line support in the first place?

The Problem with CRLs (2)

- CRL design made it worse
  - CRLs can contain retroactive invalidity dates
  - A CRL issued today can say a cert was invalid as of last week.
    - Checking that something was valid at time t wasn't sufficient!
    - Back-dated CRLs can appear at any time in the future
  - If you rely on certs & CRLs you're screwed because the CA can change the rules out from under you later.

The Problem with CRLs (3)

- Revoking a CA cert is more problematic than revoking an end-entity cert
  - When you revoke a CA cert, you potentially take out the entire subordinate structure, depending on what chaining logic you use
- How do you revoke a self-signed cert?
  - "The cert revokes itself."
    - Huh?
  - Do I accept the CRL as valid & bounce the cert?
  - Do I reject the CRL because the cert associated with the CRL signing key was revoked?

The Problem with CRLs (4)

- You can't revoke a CRL
  - Once you commit to a CRL, it's a valid state for the entirety of its validity period
- What happens if you have to update the CRL while the CRL you just issued is still valid?
  - You can update it, but clients aren't required to fetch it since the one they have is still valid!
- Bottom line: yikes!
  - We need something else

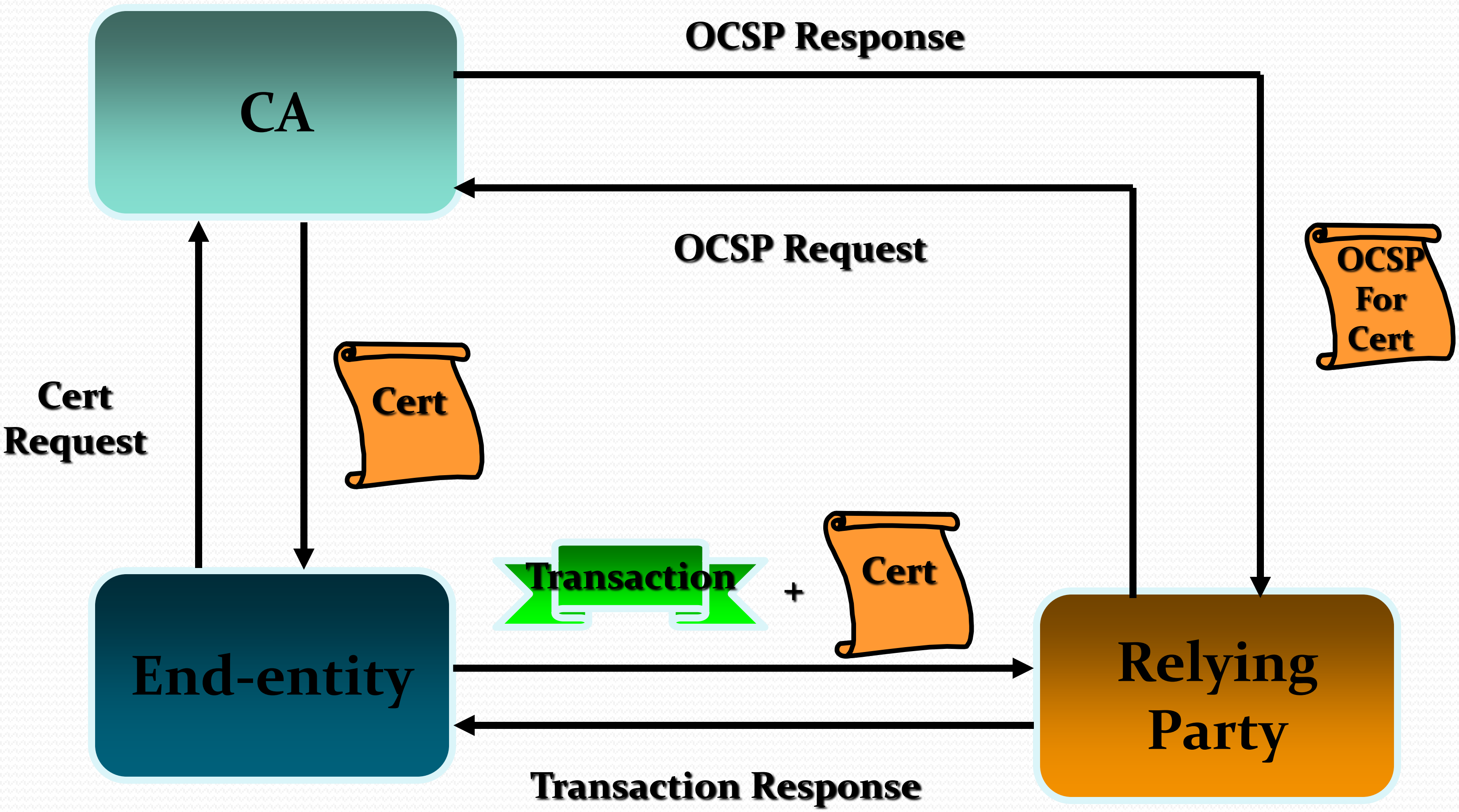
CRLs vs. OCSP Responses

- Aggregation vs. Freshness
  - CRLs combine revocation information for many certs into one long-lived object
  - OCSP Responses designed for real-time responses to queries about the status of a single certificate
- Both CRLs & OCSP Responses are generated by the issuing CA or its designate. (Generally this is not the relying party.)

Online Status Checking

- OCSP: Online Certificate Status Protocol
  - A way to ask "is this certificate good right now?
  - Get back a signed response from the OCSP server saying, "Yes, cert C is good at time t"
    - Response is like a "freshness certificate"
- OCSP response is like a selective CRL
  - Client indicates the certs for which he wants status information
  - OCSP responder dynamically creates a lightweight CRL-like response for those certs

OCSP in Action



Final thoughts on Revocation

- From a financial standpoint, it's the revocation data that is valuable, not the issued certificate itself
  - For high-valued financial transactions, seller wants to know your cert is good right now
  - Same situation as with credit cards, where the merchant wants the card authorized right now at the point-of-sale
- Card authorizations transfer risk from merchant to bank – thus they're worth $$$
  - Same with cert status checks

Using Certificates

- Most certificate uses do not require any sort of directory
  - Only needed to locate someone else's certificate for encryption
- Authentication protocols have the client present their certificate (or chain) to the server
  - Ex: SSL, TLS, Smart card logon
  - Rules for mapping a certificate to user account vary widely
    - Cert fields, name forms, binary compare
- Signing operations embed the certificates with the signature
  - How else would you know who signed it?

Using Certificates (2)

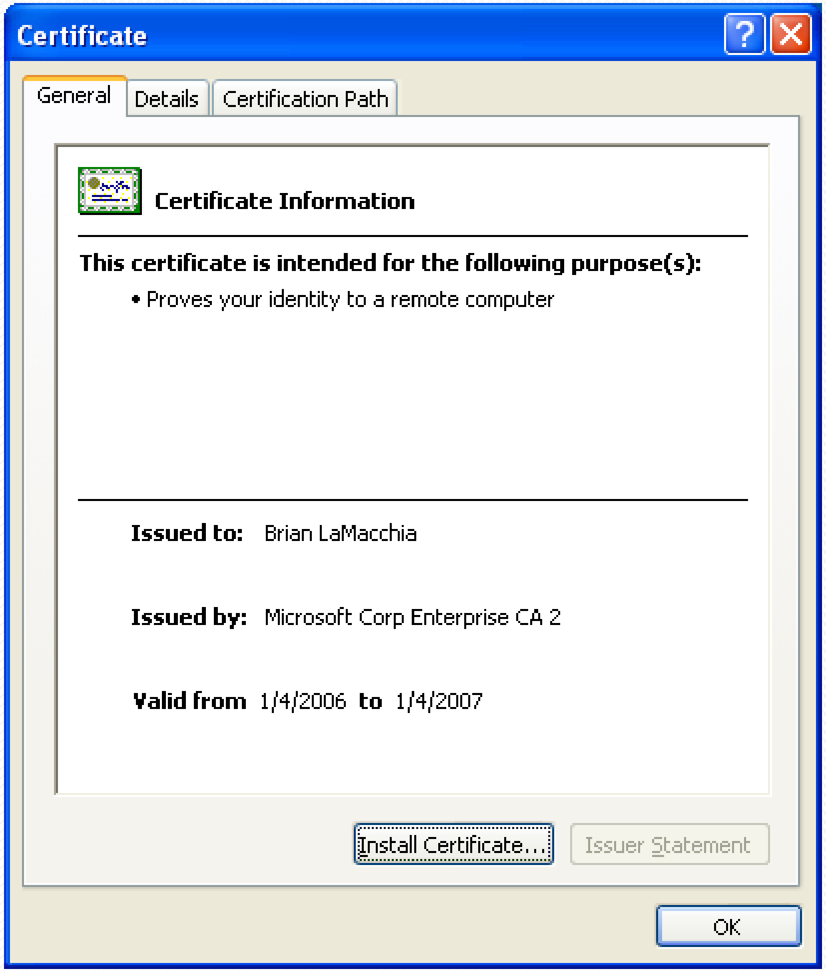
- X.509 and PKIX define the basic structure of certificates
  - If you understand X.509, you can parse any certificate you're presented
- However, every protocol defines a certificate profile for certificate use in that particular protocol
  - Ex: TLS, S/MIME, IPSEC, WPA/WPA2
- CAs/organizations define profiles too
  - Ex: US DoD Common Access Card certs

Additional Implementation Considerations

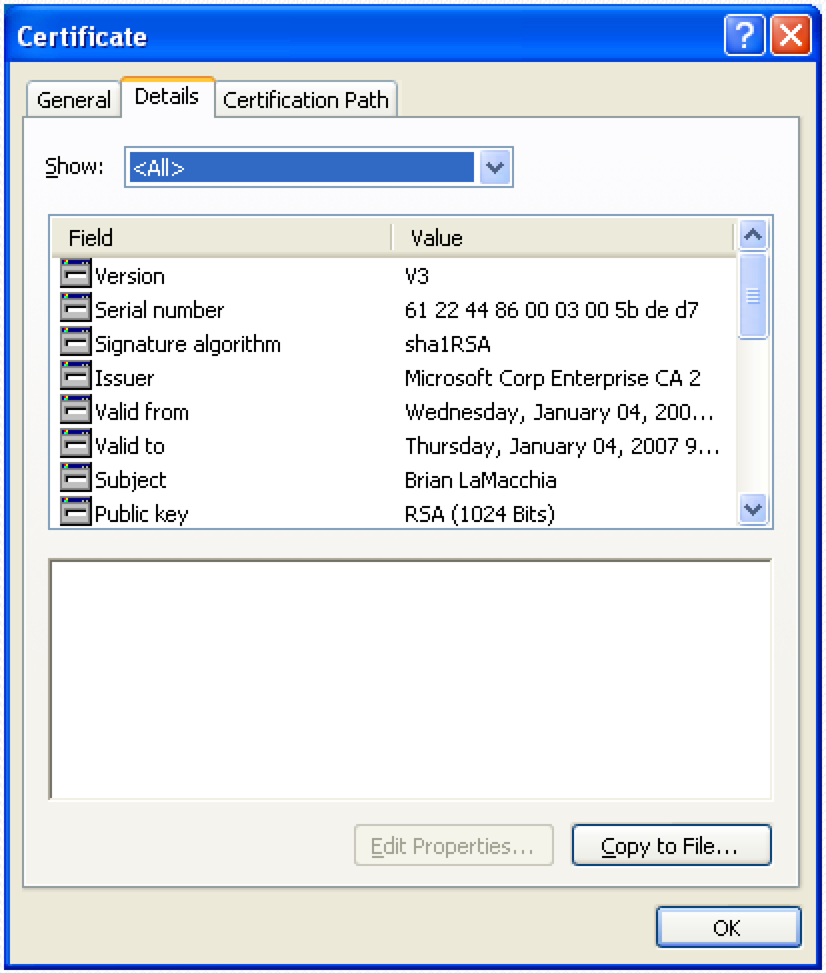
- Publishing certificates
  - How? Where? What format?
- Key escrow / data recovery for encryption keys/certs
- Auto-enrollment (users & machines)
- Establishing trusts / hierarchies
- Protecting private keys
- Disseminating root certificates

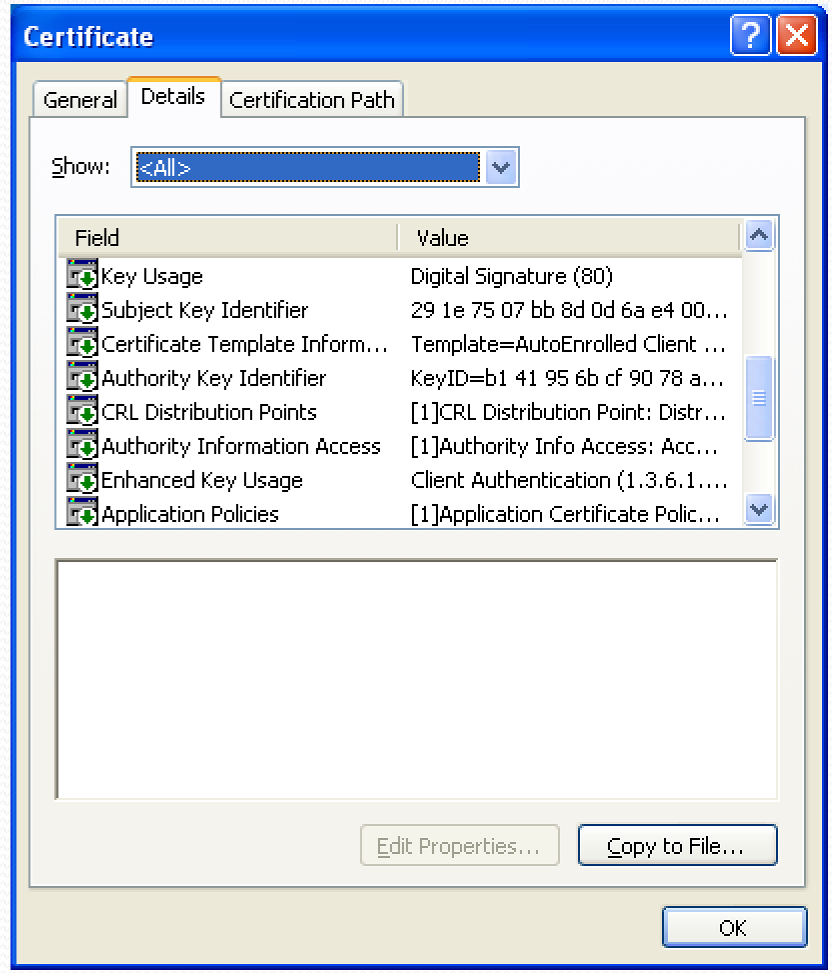
Supplemental Material on Certificate Extensions (only if time permits)

Exploring inside an X.509 Cert

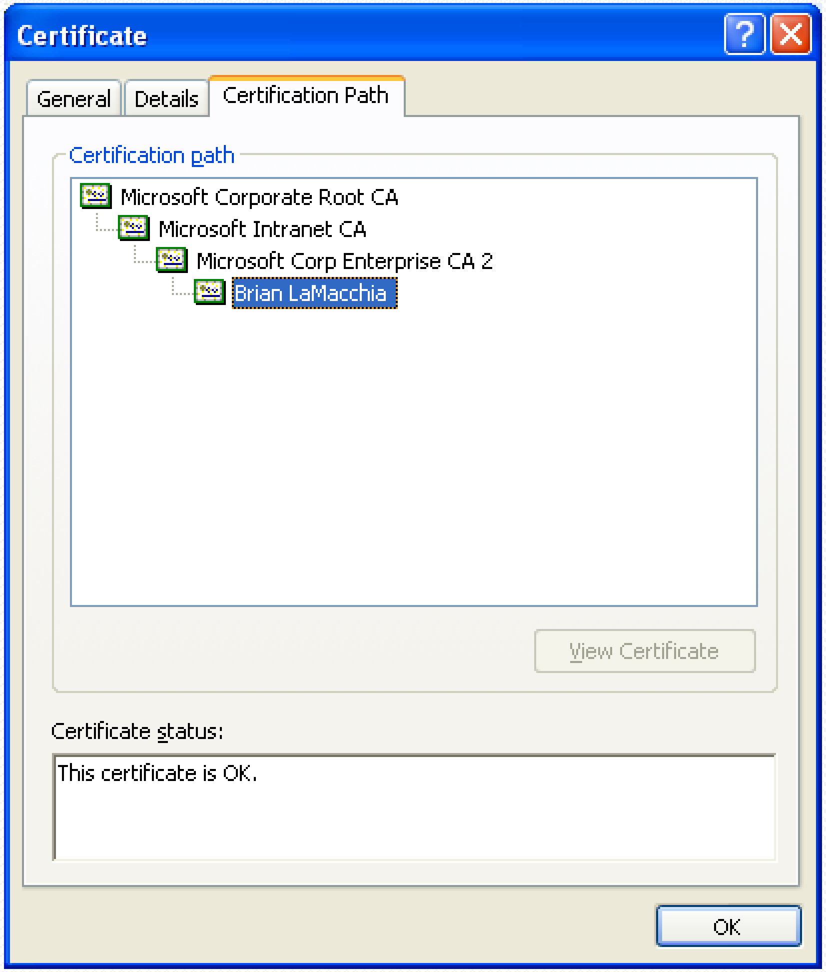


Exploring inside an X.509 Cert

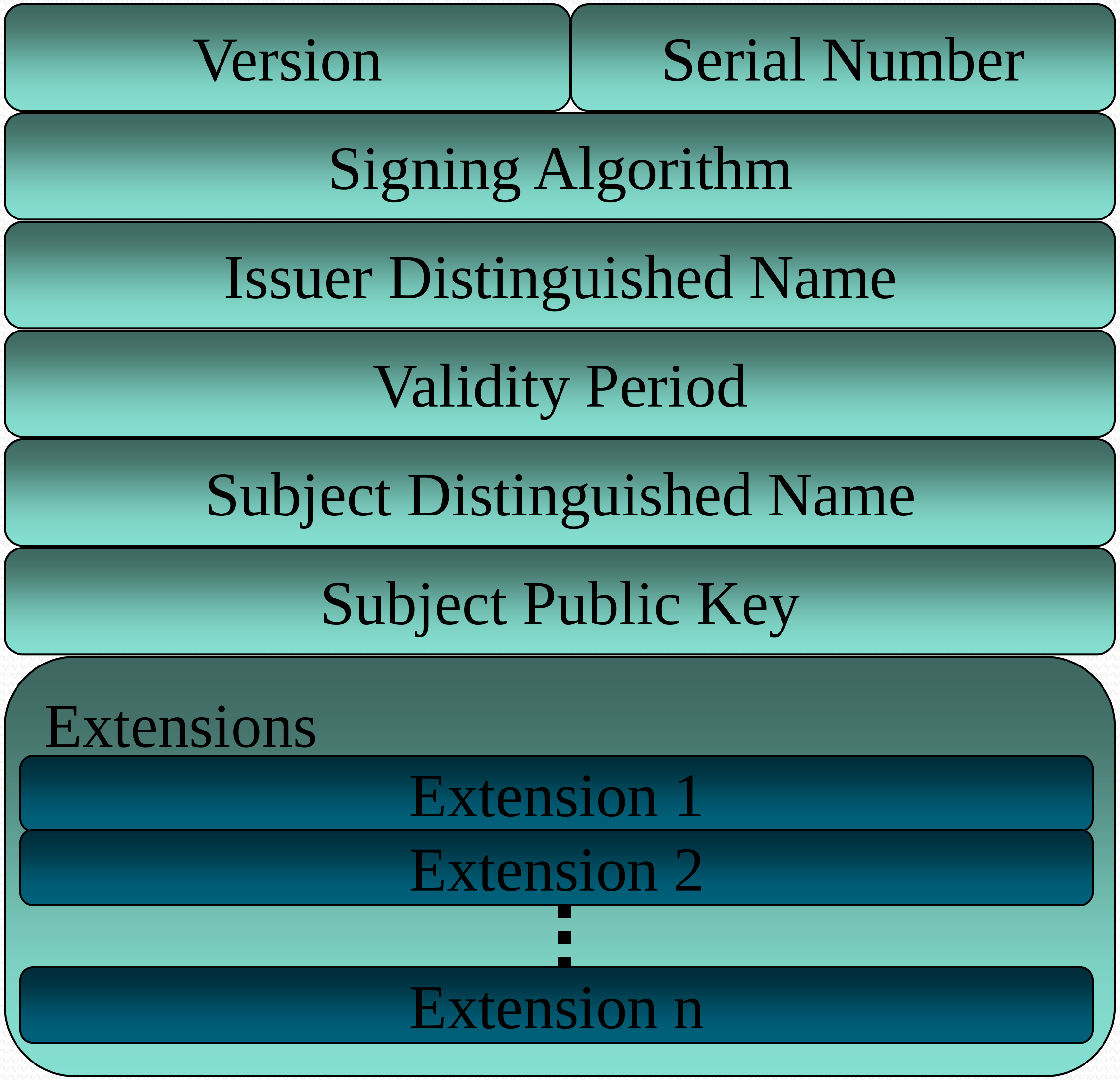




Exploring inside an X.509 Cert



Inside an X.509v3 Certificate



Certificate Extensions

- An extension consists of three things:
  - A "critical" flag (boolean)
  - A type identifier
  - A value
    - Format of the value depends on the type identifier

Certificate Extensions

Extensions

Critical?	Key Usage
Critical?	Subject Key ID
Critical?	Authority Key ID
Critical?	CRL Distribution Points
Critical?	Authority Info Access
Critical?	Extended Key Usage
Critical?	Subject Alt Name
Critical?	Certificate Policies
Critical?	Proprietary Extension 1
Critical?	Proprietary Extension n

Critical Flags

- The "critical flag" on an extension is used to protect the issuing CA from assumptions made by software that doesn't understand (implement support for) a particular extension
  - If the flag is set, relying parties must process the extension if they recognize it, or reject the certificate
  - If the flag is not set, the extension may be ignored

Critical Flags (2)

- Some questions you might be asking yourself right now...
- What does “must process the extension if they recognize it” mean?
  - What does “recognize” mean?
  - What does “process” mean?
  - You’ve got me....
  - The IETF standards folks didn’t know either...

Critical Flags (3)

- Actual definitions of flag usage are vague:
  - X.509: Non-critical extension “is an advisory field and does not imply that usage of the key is restricted to the purpose indicated”
  - PKIX: “CA’s are required to support constrain extensions” but “support” is never defined.
  - S/MIME: Implementations should “correctly handle” certain extensions
  - Verisign: “All persons shall process the extension...or else ignore the extension”

Types of Extensions

- There are two flavors of extensions
  - Usage/informational extensions, which provide additional info about the subject of the certificate
  - Constraint extensions, which place restrictions on one or more of:
    - Use of the certificate
    - The user of the certificate
    - The keys associated with the certificate

Some common extensions

- Key Usage
  - digitalSignature
    - "Sign things that don't look like certs"
  - keyEncipherment
    - Exchange encrypted session keys
  - keyAgreement
    - Diffie-Hellman
  - keyCertSign/keyCRLSign
    - "Sign things that look like certs"
  - nonRepidiation

NonRepudiation

- The nonRepudiation bit is the black hole of PKIX
  - It absorbs infinite amounts of argument time on the mailing list without making any progress toward understanding what it means
  - What does it mean? How do you enforce that?
  - No one knows...
- "Nonrepudiation is anything which fails to go away when you stop believing in it"

More Extensions

- Subject Key ID
  - Short identifier for the subject public key
- Authority Key ID
  - Short identifier for the issuer’s public key – useful for locating possible parent certs
- CRL Distribution Points
  - List of URLs pointing to revocation information servers
- Authority Info Access
  - Pointer to issuer cert publication location

Even More Extensions

- Basic constraints
  - Is the cert a CA cert?’
  - Limits on path length beneath this cert
- Name constraints
  - Limits on types of certs this key can issue
- Policy mappings
  - Convert one policy ID into another
- Policy constraints
  - Anti-matter for policy mappings

Still More Extensions

- Extended Key Usage
  - Because Key Usage wasn’t confusing enough!
- Private Key Usage Period
  - CA attempt to limit key validity period
- Subject Alternative names
  - Everything which doesn’t fit in a DN
  - RFC822 names, DNS names, URIs
  - IP addresses, X.400 names, EDI, etc.

Yet Still More Extensions

- Certificate policies
  - Information identifying the CA policy that was in effect when the cert was issued
  - Policy identifier
  - Policy qualifier
  - Explicit text
  - Hash reference (hash + URI) to a document
- X.509 defers cert semantics to the CA's issuing policy
- Most CA policies disclaim liability

Extensions and Chain Building

- When you build a cert chain, you start with the EE cert and discover possible parent certificates by matching DNs
  - “Build the chain from the bottom up.”
- However, to verify a cert chain, you have to start and the root and interpret all the extensions that may constrain subordinate CAs (and EEs)
  - “Build the chain from the top down.”