Certificates, Trust & PKI

Brian A. LaMacchia
bal@cs.washington.edu
bal@microsoft.com

Certificates

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.*

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.

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

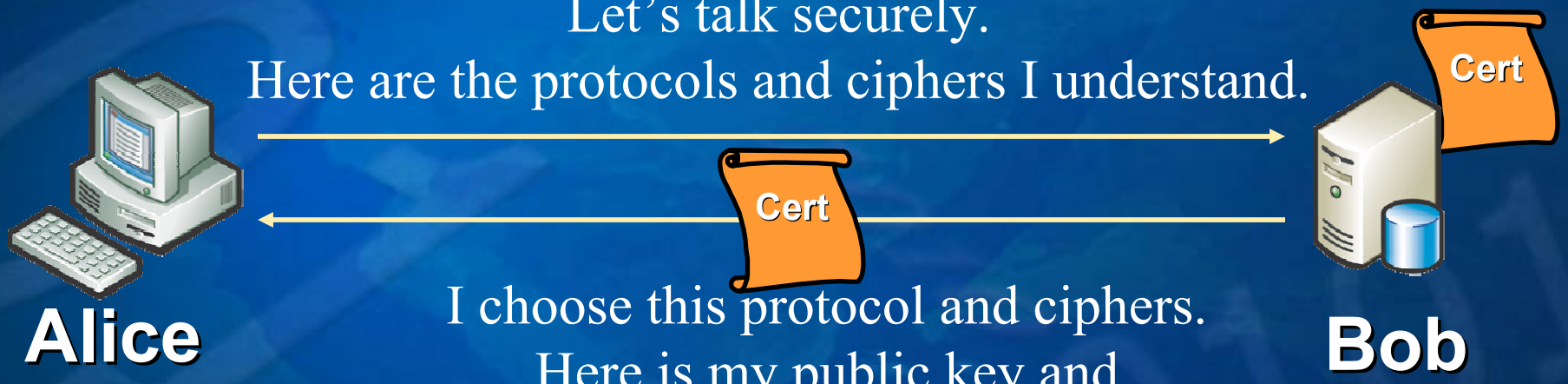
- Many physical objects are like certificates:
  - Any type of license – vehicle tabs, restaurant liquor license, amateur radio license, etc.
  - Government-issued IDs (passports, green cards)
  - Membership cards (e.g. Costco, discount cards)
- All of these examples bind an identity and certain rights,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

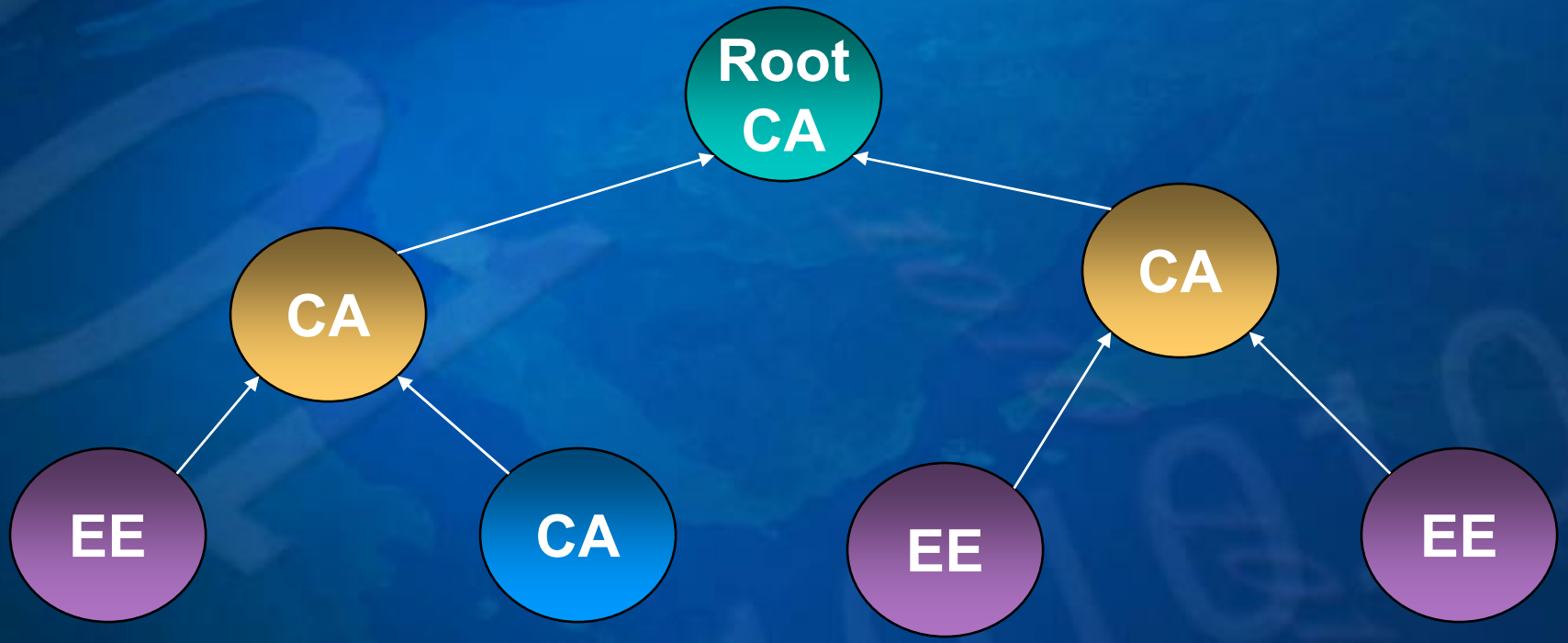
Public Key Infrastructure

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"
- Certificates are links in a tree of EEs & CAs

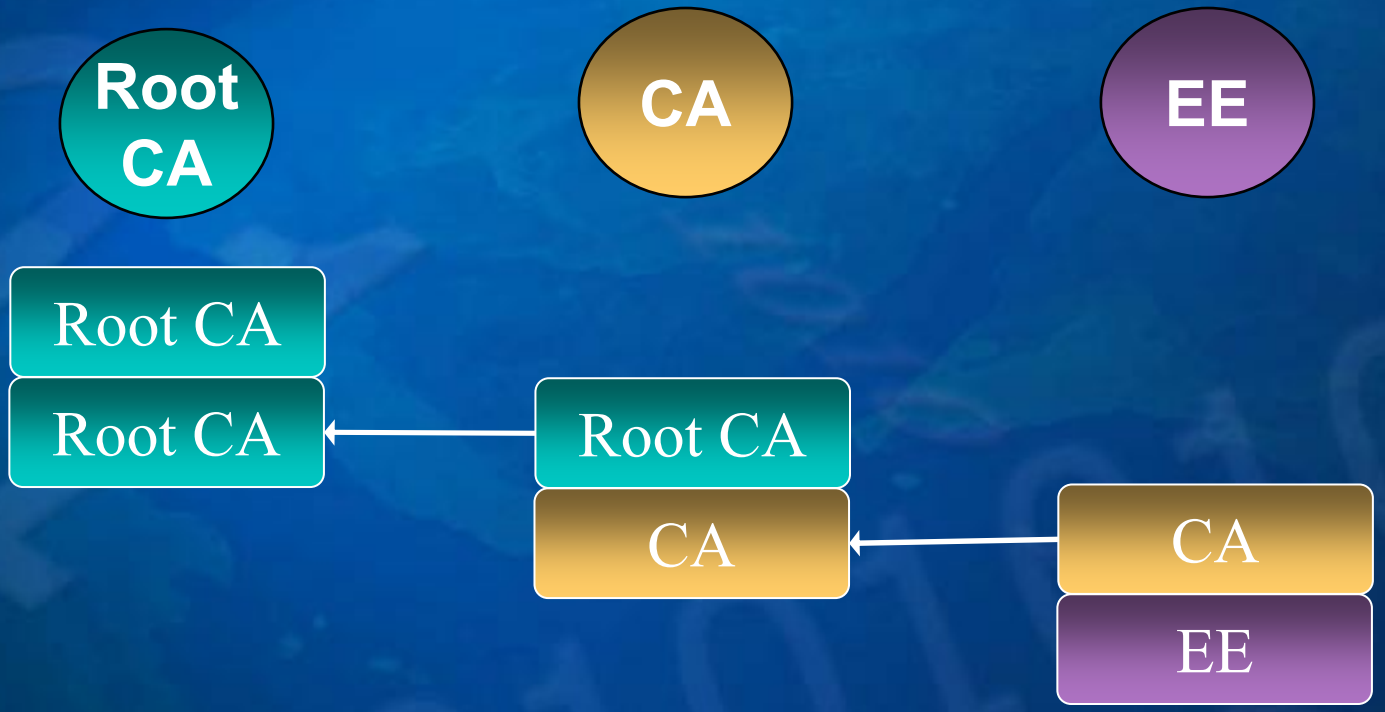


BAL's No-Frills Certs

- 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:

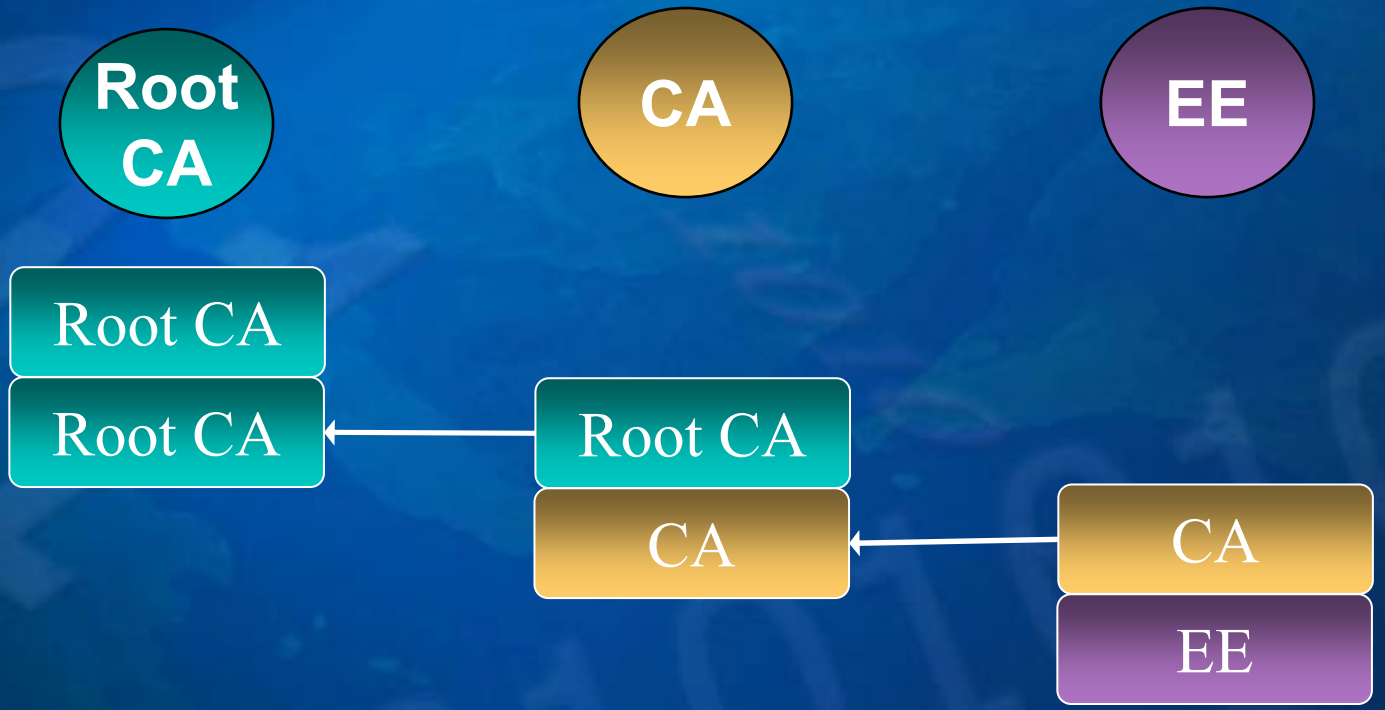
Does Alice trust Bob's Key?

- 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

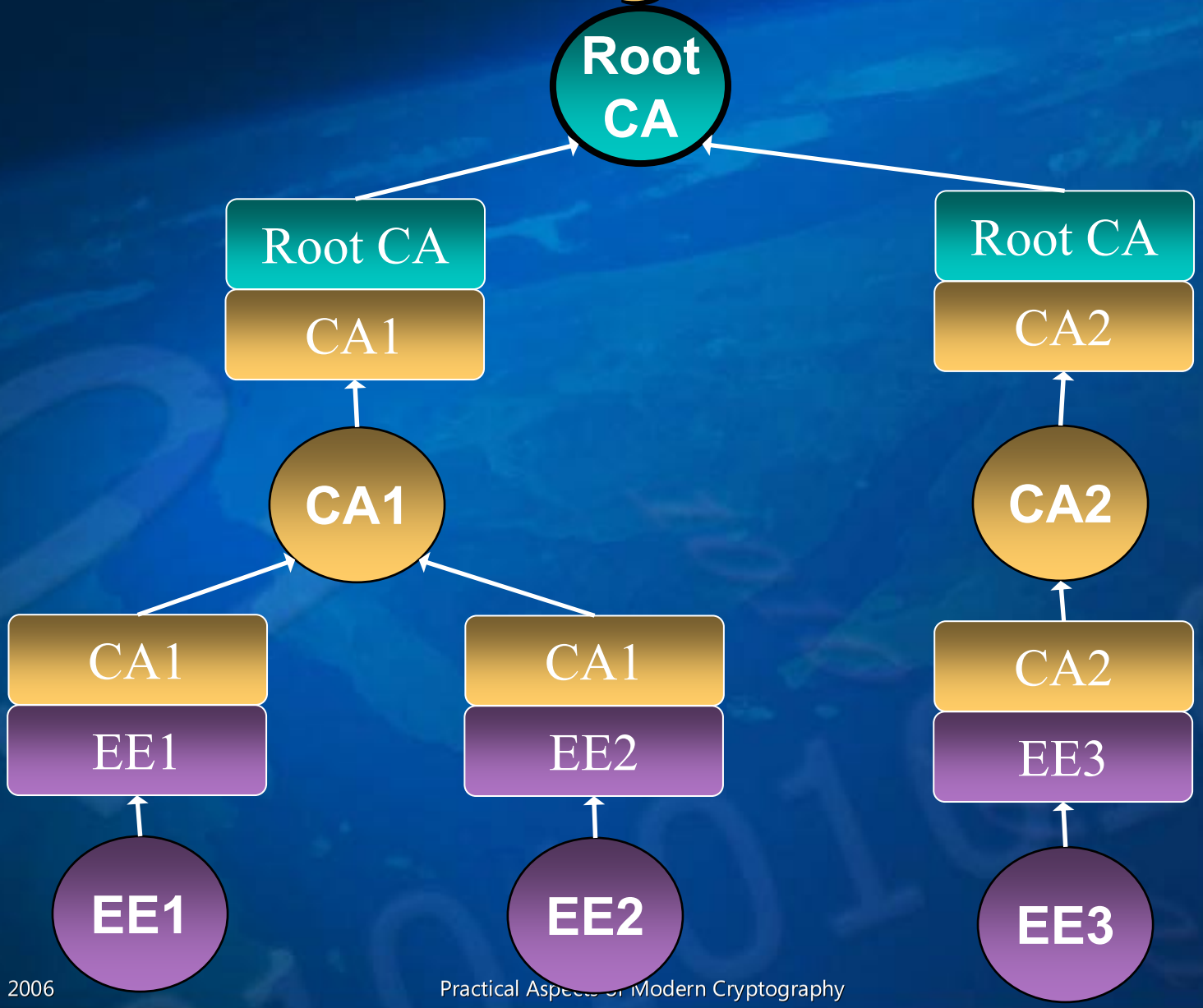
- "Given an end-entity certificate, does there exist a cryptographically valid chain of certificates linking it to a trusted root certificate?"



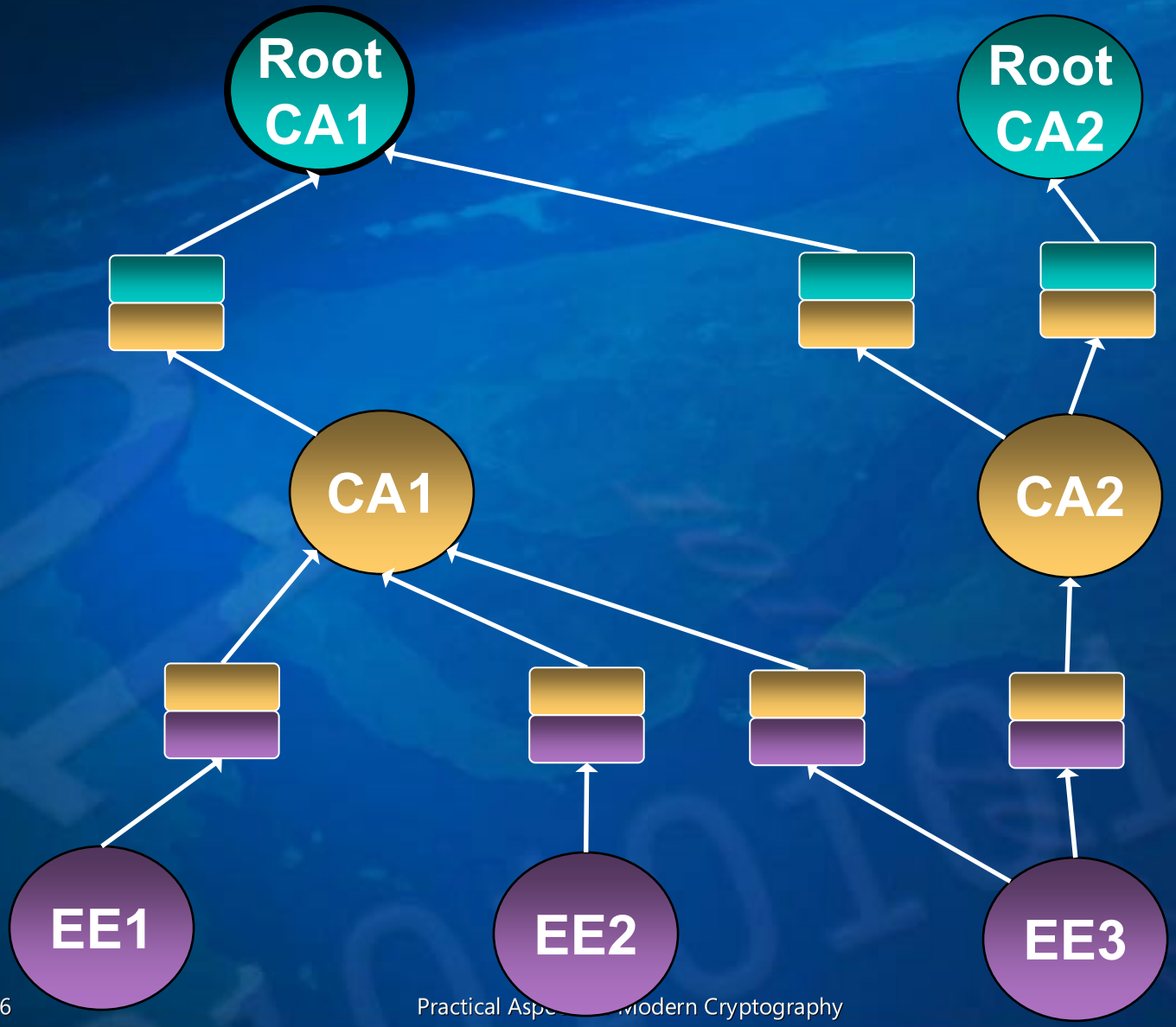
Chaining Certificates

- 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)

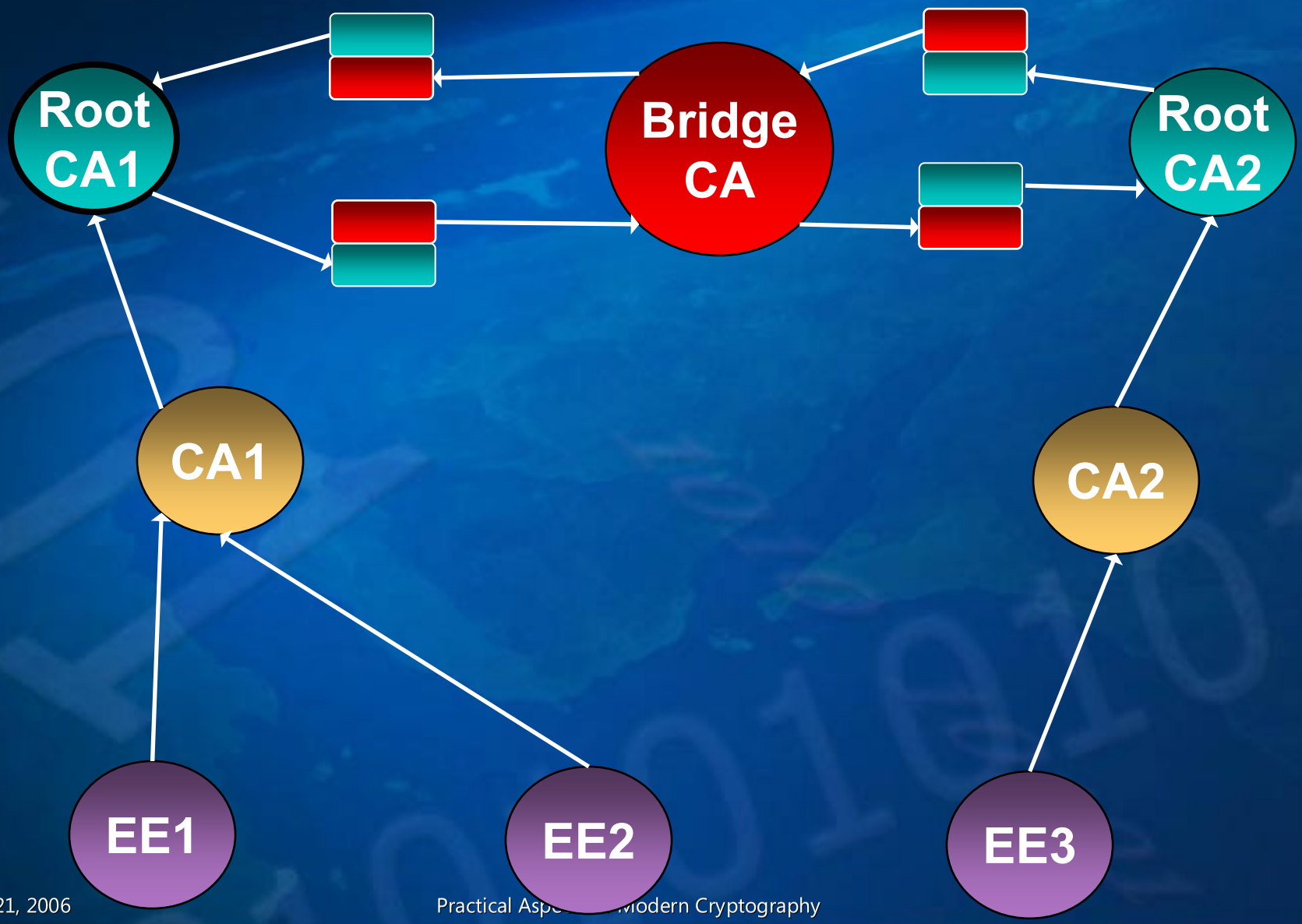


Chain Building Details (2)

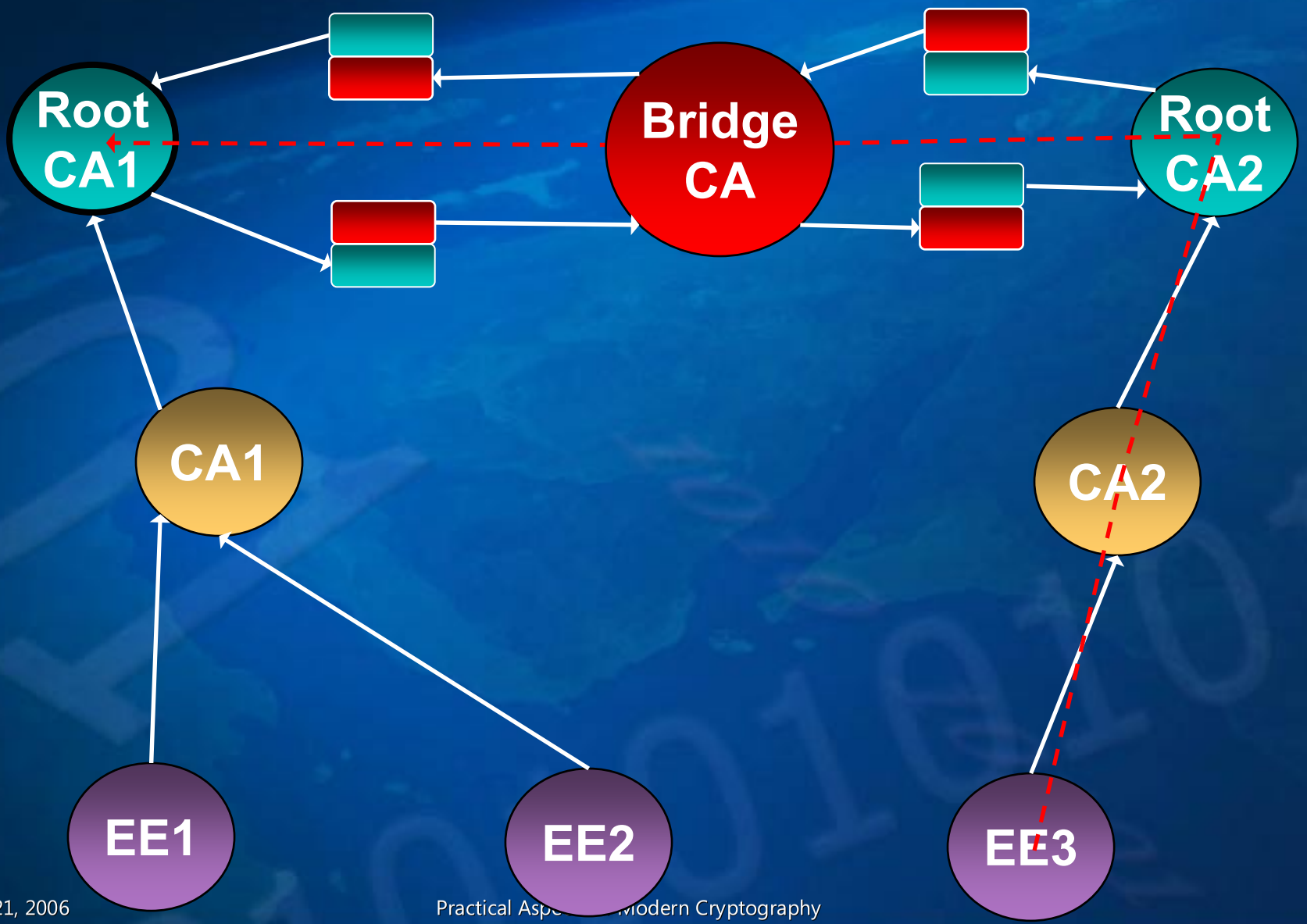


Chain Building Details (3)

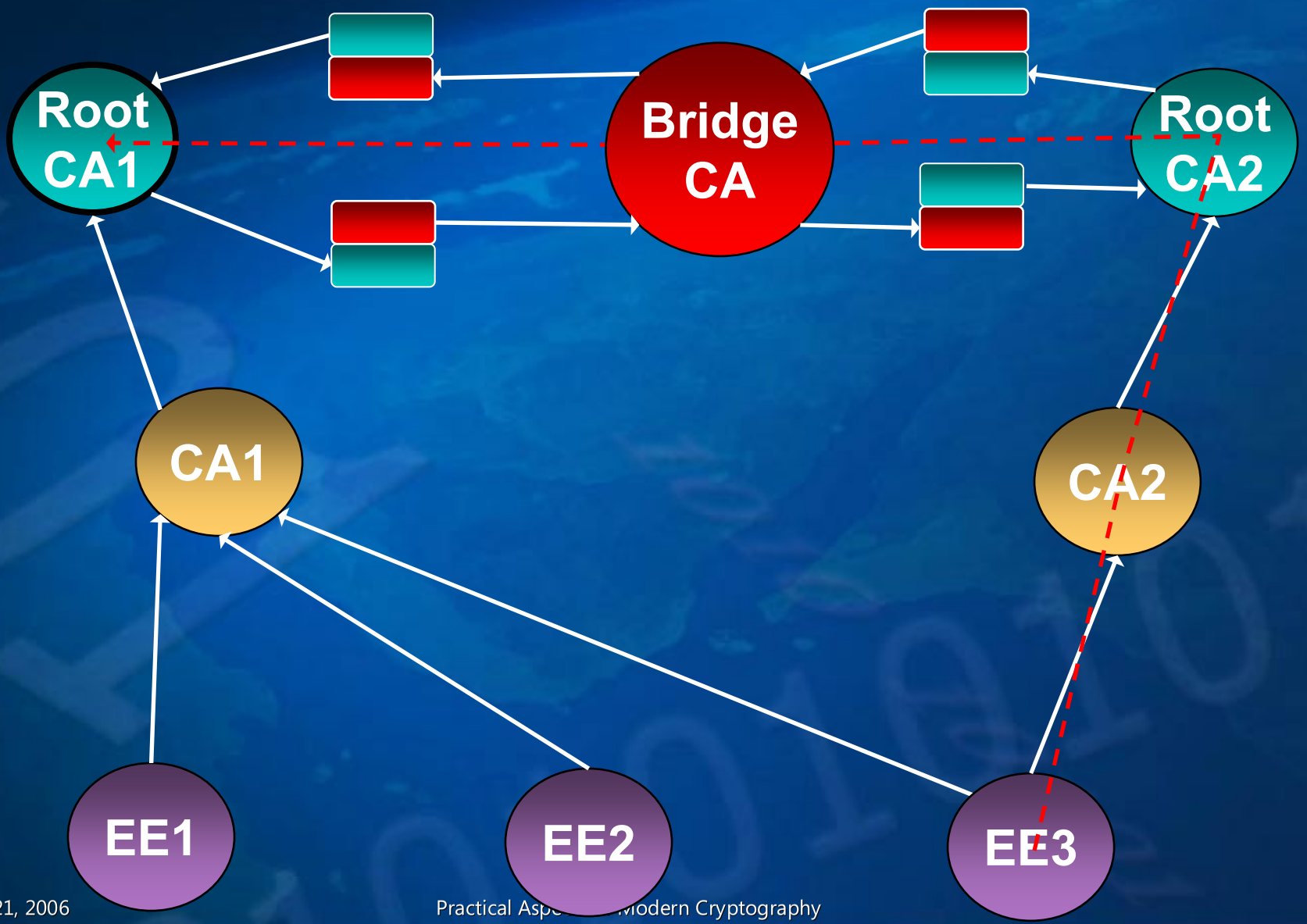
Chain Building Details (3)



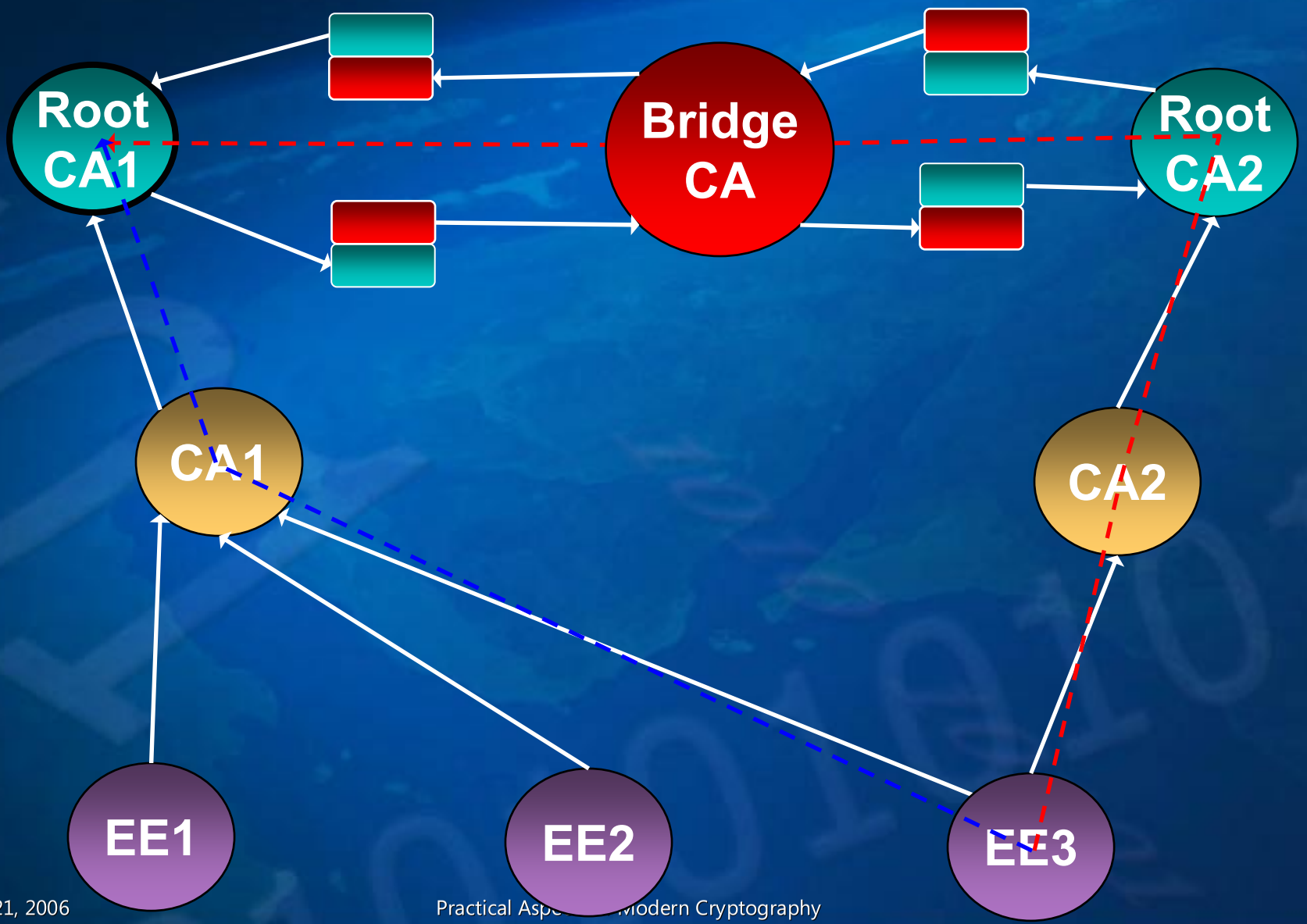
Chain Building Details (3)



Chain Building Details (3)



Chain Building Details (3)



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?"*
- In order to understand the schism, we need to digress for a bit and talk about names and some history

PKI Alphabet Soup

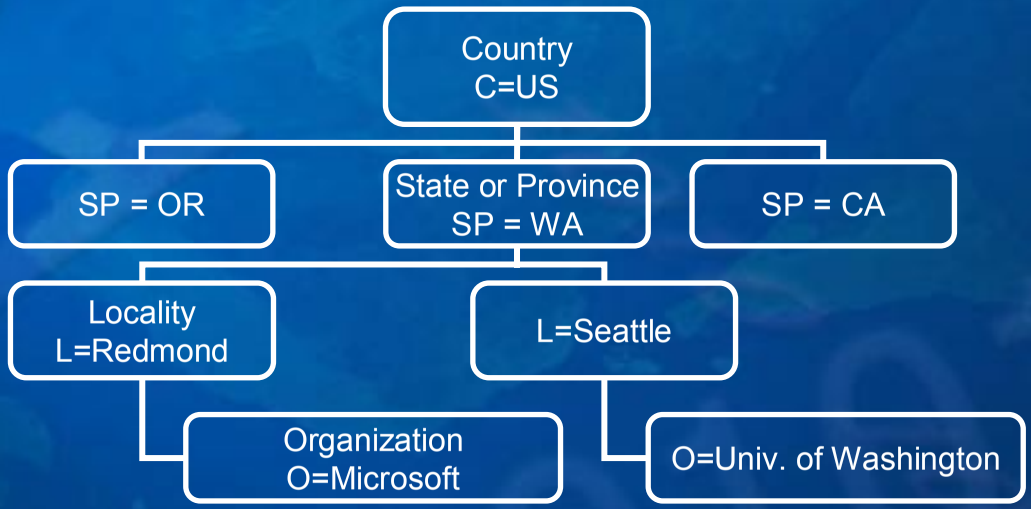
- X.509v3 - standard content of a certificate
- PKIX – IETF Working Group on PKI interoperability
  - PKIX == Public Key Infrastructure using X.509v3 certificates
- ASN.1 - Abstract Syntax Notation, exact description of a certificate format
- DER - Distinguished Encoding Rules, how to physically package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

The X.500 Directory Model

- The model SSL/TLS uses, the X.509 certificate model, is based on names
  - *Names as principles*
- Specifically, X.509 is based on the X.500 directory model
- X.500 defined a global, all-encompassing directory, to be run by the telcos
  - *One directory to rule them all, one directory to define them…*

X.500 Distinguished Names

- In the X.500 model, everything has a single, unique, global, assigned name
  - There is a worldwide hierarchy, and you're in it!



DNs in Practice

- Name is unique within the scope of the CA's name
- Public CAs (e.g. Verisign) typically set
  - C = CA Country
  - O = CA Name
  - OU = Certificate type/class
  - CN = User name
  - E= email address

Private-label DNs

- If you own the CA, you get to decide what fields go in the DN
  - Really varies on what the software supports
- Can get really strange as people try to guess values for fields that are required by software
  - Software requires an OU, we don't have OUs, so I better make something up!

DNs in X.509 Certificates

- The X.509 certificate standard began as a way to associate a certificate with a node in the directory.
- How is the subject of a cert identified?
  - By its DN.
- How is the issuer of a cert identified?
  - By its DN.
- How are certificates linked together?
  - By DNs.

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!

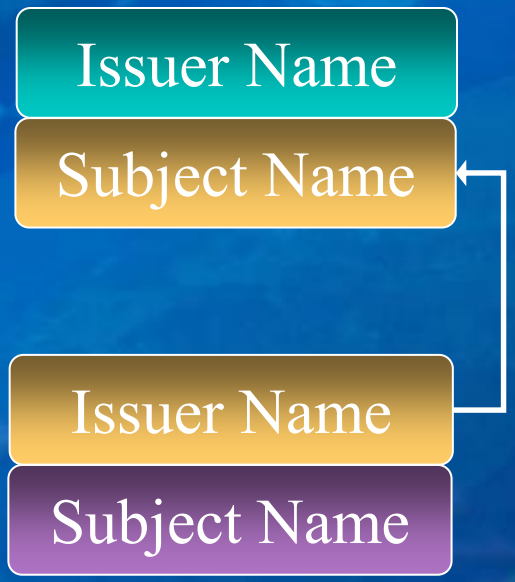
Back to Chain Building

- 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



Matching Names

- How do we determine if two DNs match?
  - "Use directory name matching rules!"
  - Try to be mildly smart about it
    - Remove spaces, case-fold, etc.
    - Disaster…
  - Try to be really dumb about it
    - Exact binary match
    - Less of a disaster, but there are still problems we can't work around…

Unicode Names

- Are these two character equal?

  é é

- They look equal…

Unicode Names

- Are these two character equal?

  é é

- They look equal…
- …but may not be
- In Unicode, you can compose characters, so:
  - "é" as one character
  - "é" as two characters – "e" followed by non-spacing accent
  - "é" as two characters – non-spacing accent followed by "e"
- Ick!

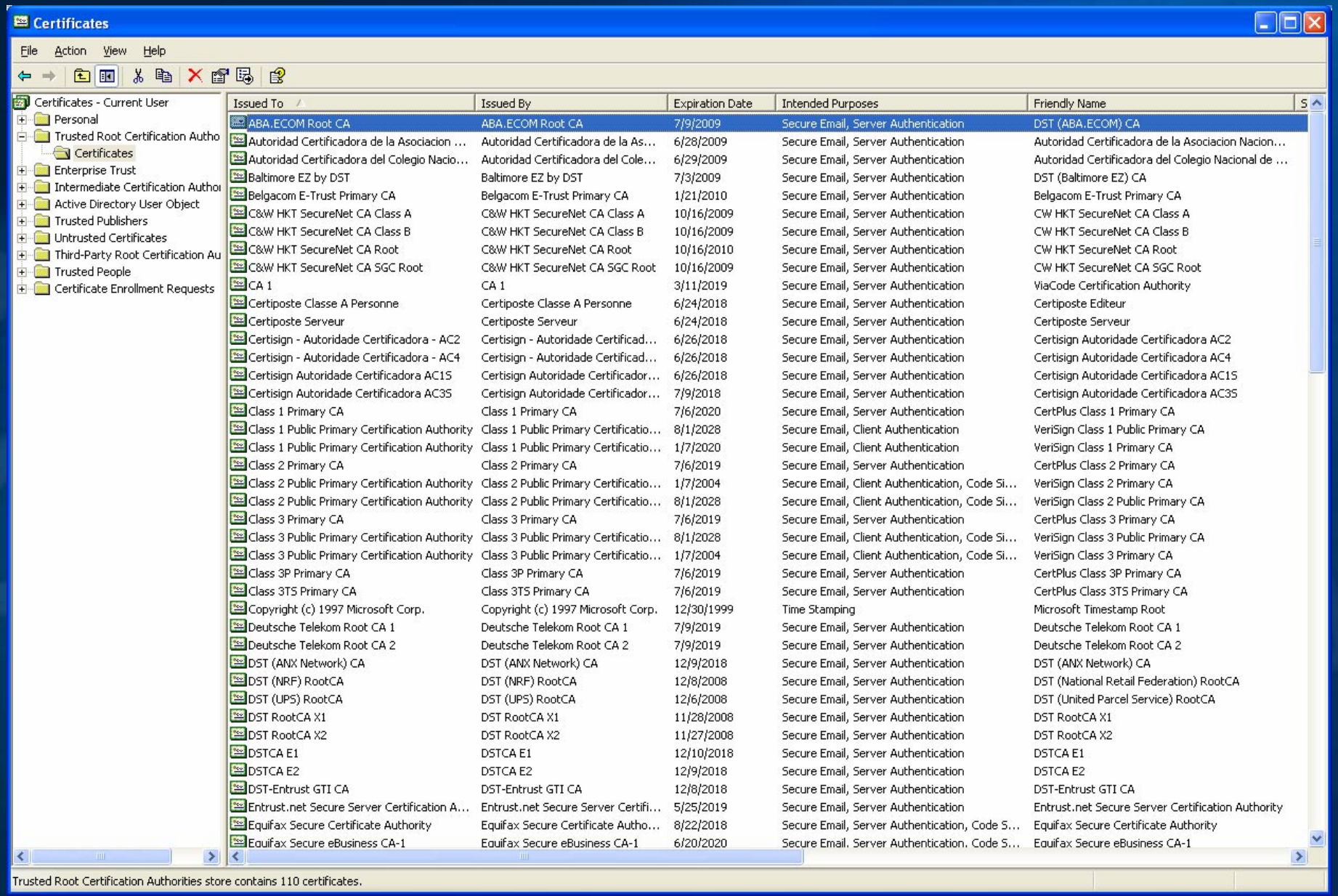
Even More Chain Building

- 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
  - *What other information is there in certs?*

Trusted Root Certificates

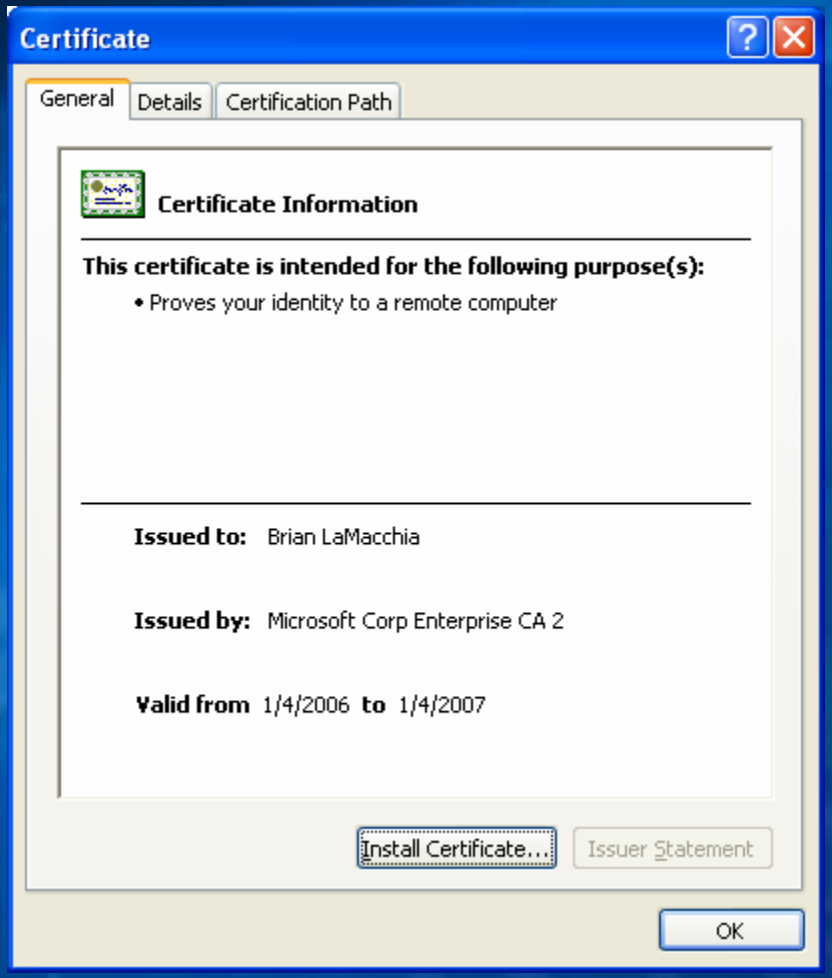
- Who do I trust to be roots at the top of the cert chain?
- In theory, "anyone you want"
- In practice, trusted roots come from two sources
  - They're baked into your web browser or operating system
  - They're pushed onto your "enterprise managed desktop"

Trusted Root Certificates

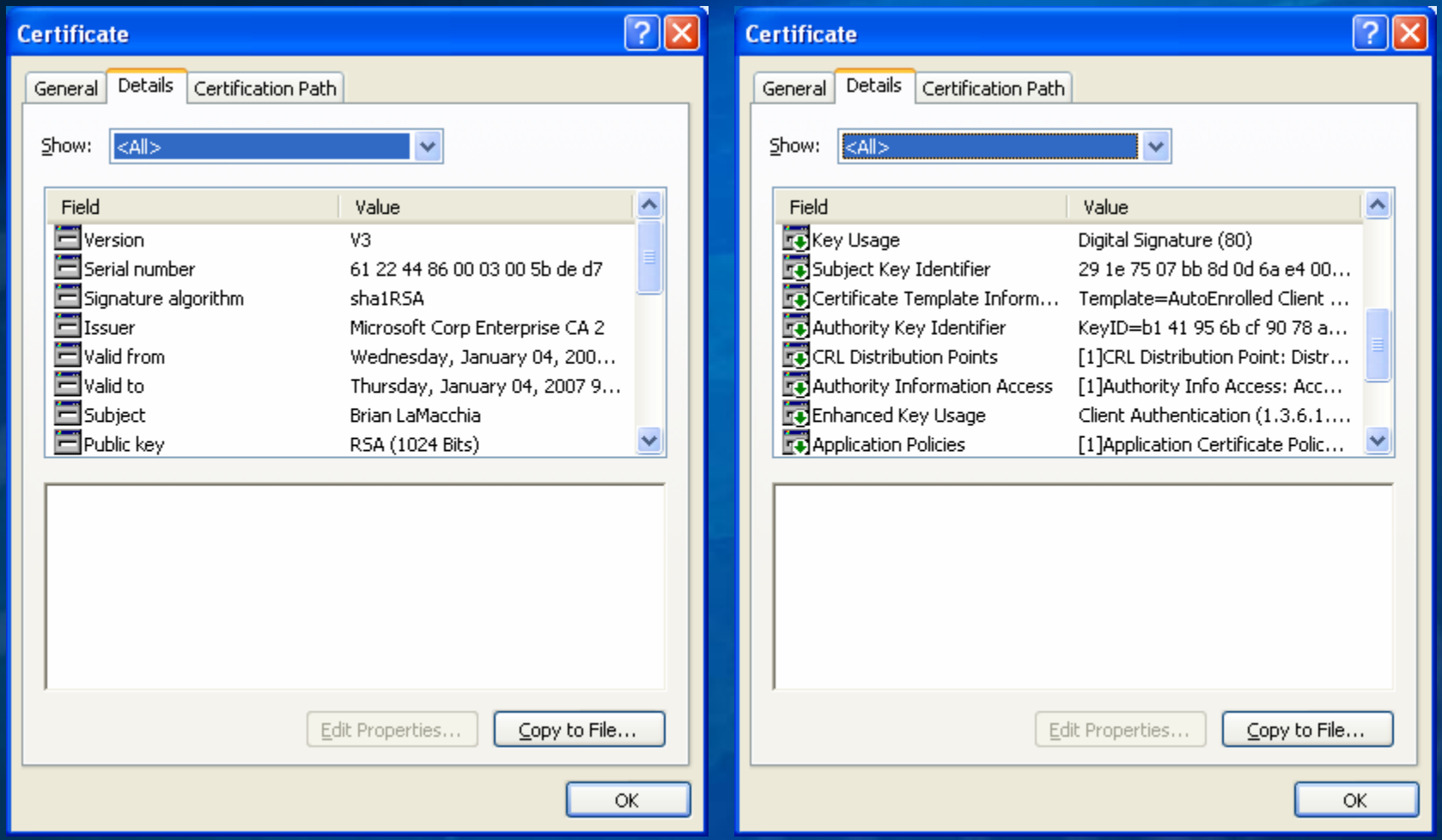


Certificate Extensions

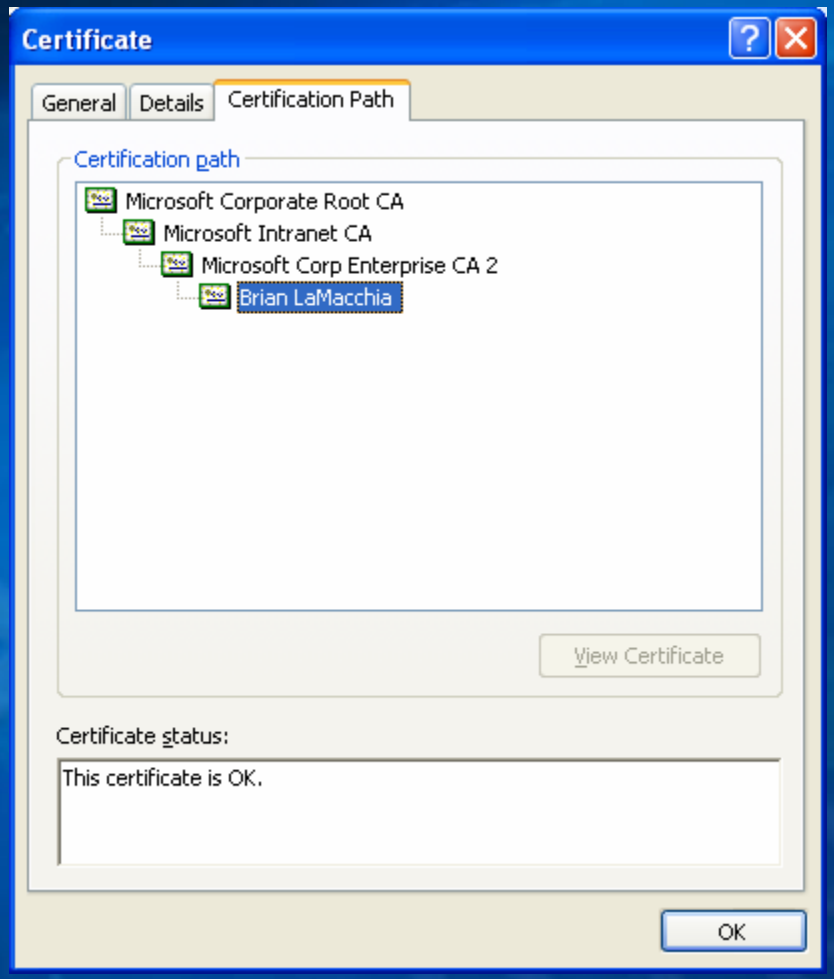
Exploring inside an X.509 Cert



Exploring inside an X.509 Cert



Exploring inside an X.509 Cert

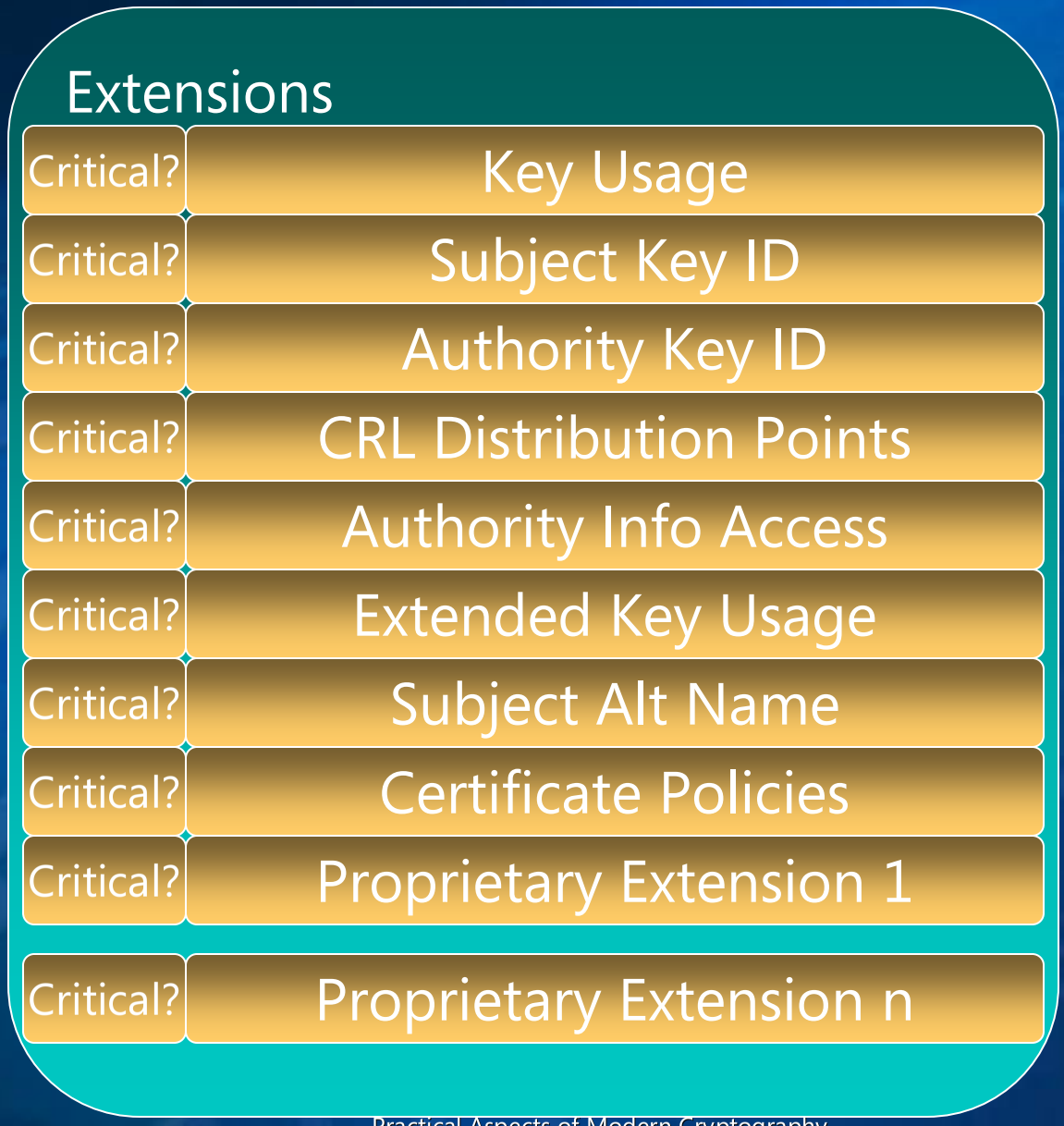


Inside an X.509v3 Certificate

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



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."

Certificate Lifecycle Management

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)

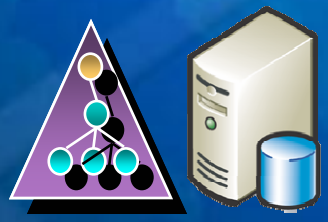
3. (Optional) CA verifies Alice's ID through out-of-band means.
4. CA creates a certificate containing the ID and public key, and signs it with the CA's own key
   - CA has certified the binding between key and ID
5. Alice verifies the key, ID & CA signature
6. Alice and/or the CA publish the certificate

Certificate Enrollment Flow

Client

Certificate Request
and Installation

CA



Directory

Publish Certificate?

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

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
  - CLRs 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

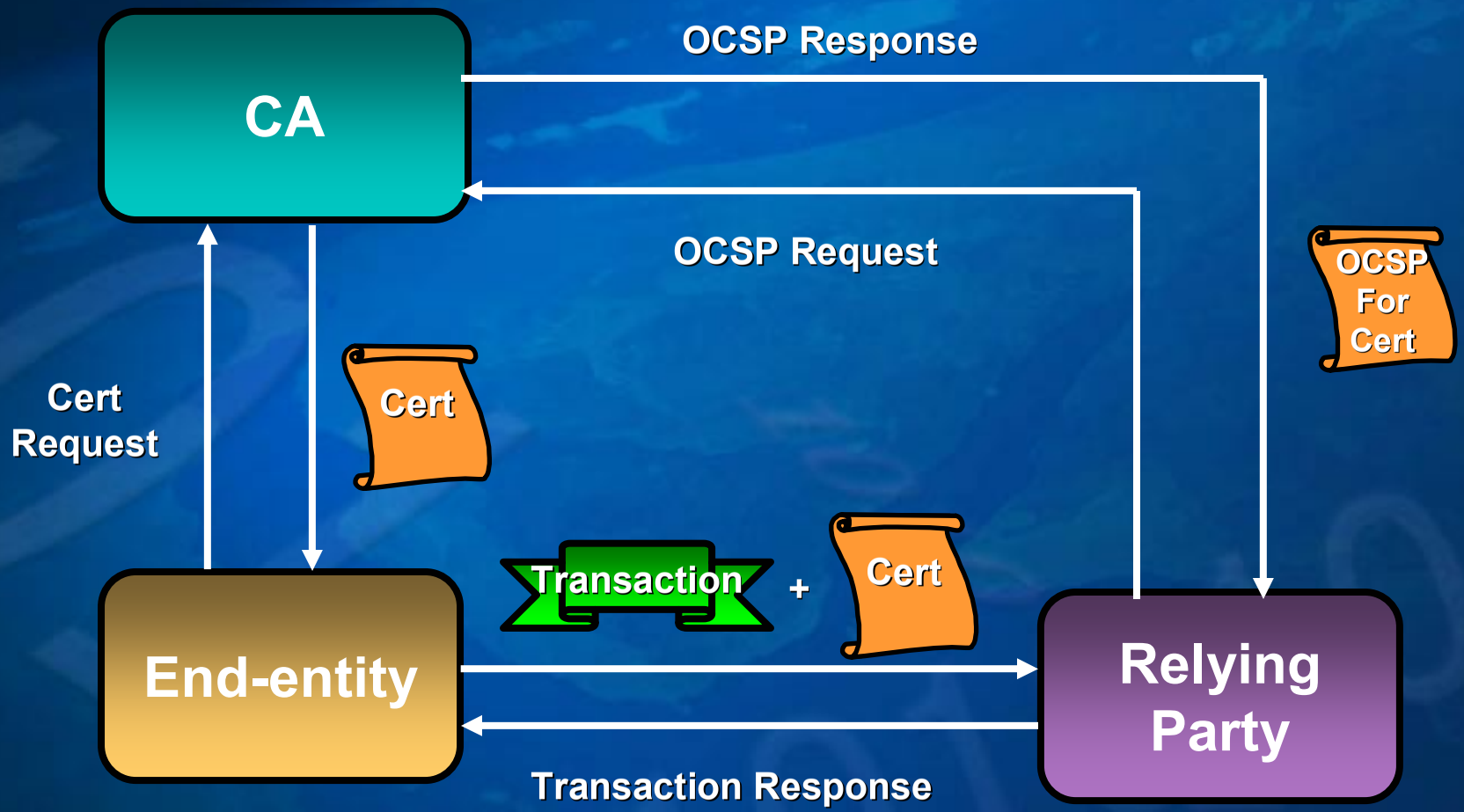
CRLs vs. OCSP Responses

- 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