Practical Aspects of Modern Cryptography

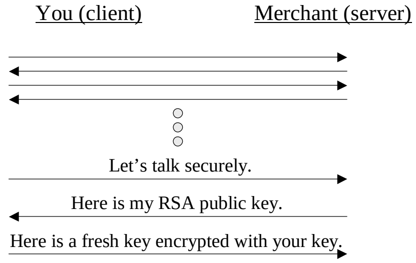
Josh Benaloh & Brian LaMacchia

Lecture 5: SSL/TLS in depth, Certificates & Trust

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

SSL/TLS



You (client) Merchant (server)

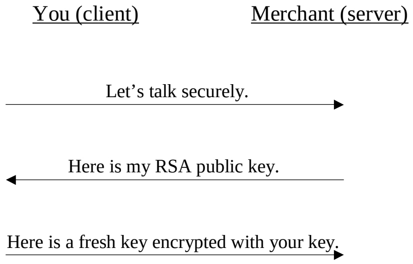
○
○
○

Let's talk securely.

Here is my RSA public key.

Here is a fresh key encrypted with your key.

SSL/TLS



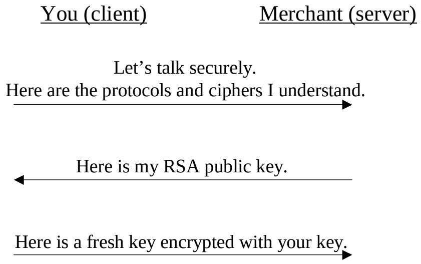
You (client) Merchant (server)

Let's talk securely.

Here is my RSA public key.

Here is a fresh key encrypted with your key.

SSL/TLS

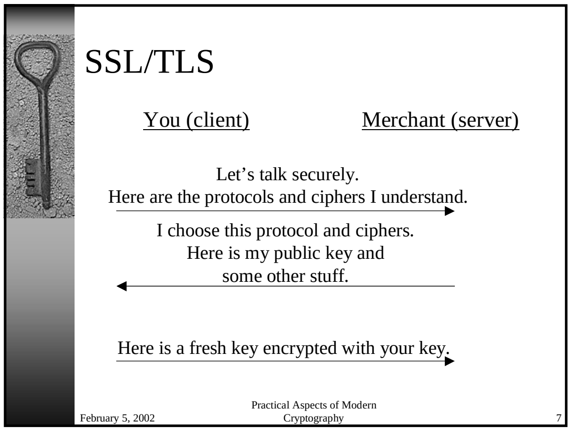


You (client) Merchant (server)

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

Here is my RSA public key.

Here is a fresh key encrypted with your key.



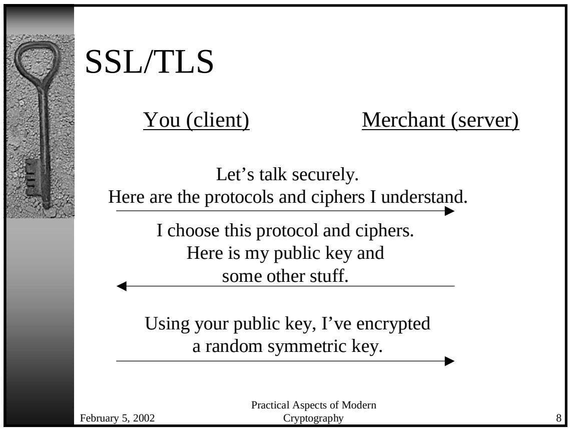
SSL/TLS

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 fresh key encrypted with your key.



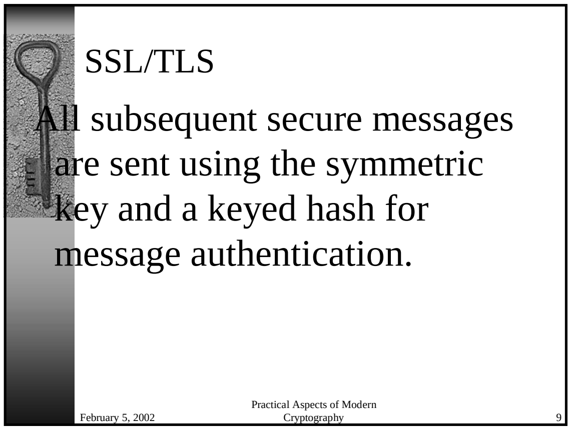
SSL/TLS

You (client) Merchant (server)

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

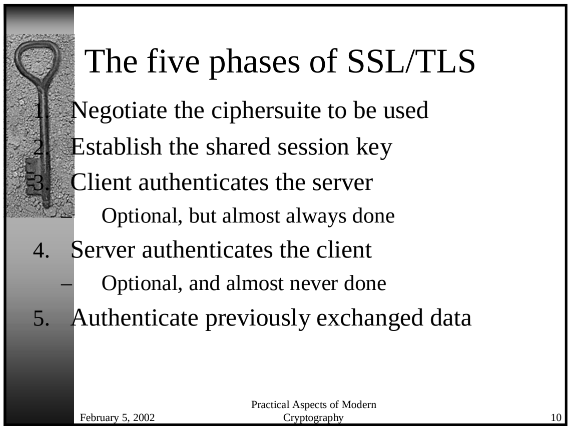
I choose this protocol and ciphers.
Here is my public key and
some other stuff.

Using your public key, I’ve encrypted
a random symmetric key.



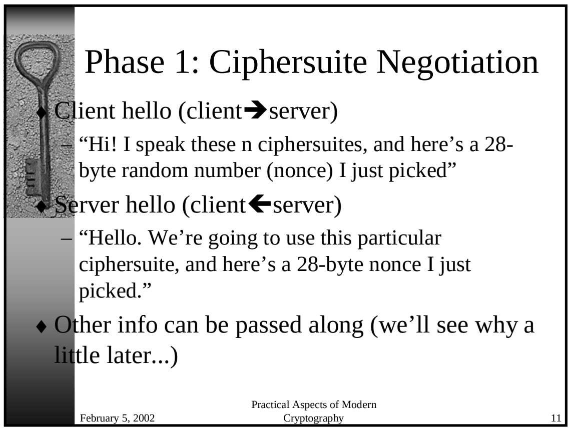
SSL/TLS

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



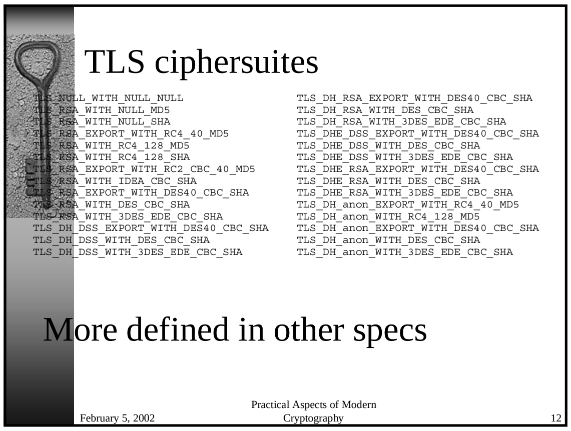
The five phases of SSL/TLS

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



Phase 1: Ciphersuite Negotiation

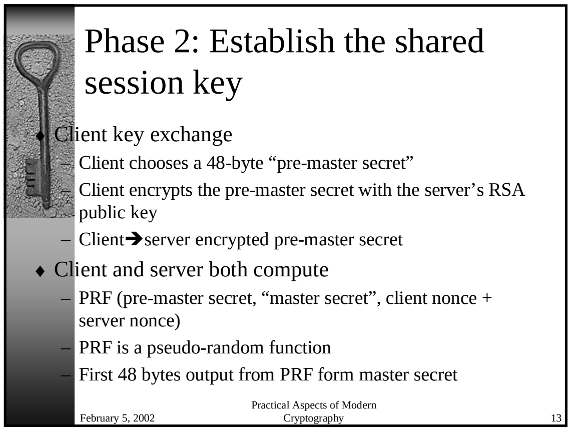
- Client hello (client➔server)
  - “Hi! I speak these n ciphersuites, and here’s a 28-byte random number (nonce) I just picked”
- Server hello (client🡐server)
  - “Hello. We’re going to use this particular ciphersuite, and here’s a 28-byte nonce I just picked.”
- Other info can be passed along (we’ll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



TLS ciphersuites

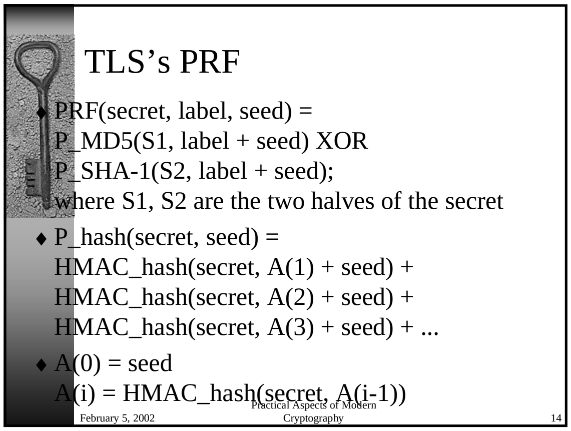
```
TLS_NULL_WITH_NULL_NULL
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
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
```

More defined in other specs



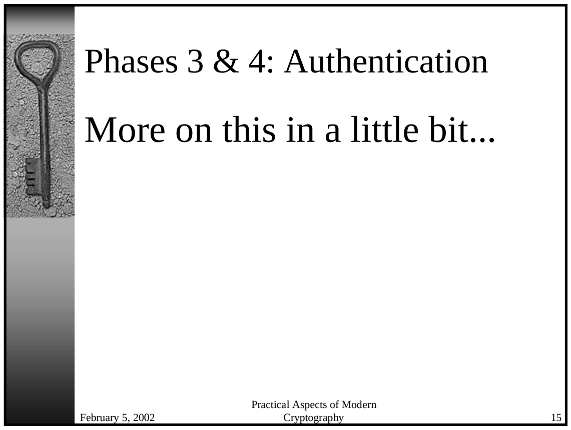
Phase 2: Establish the shared session key

- 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



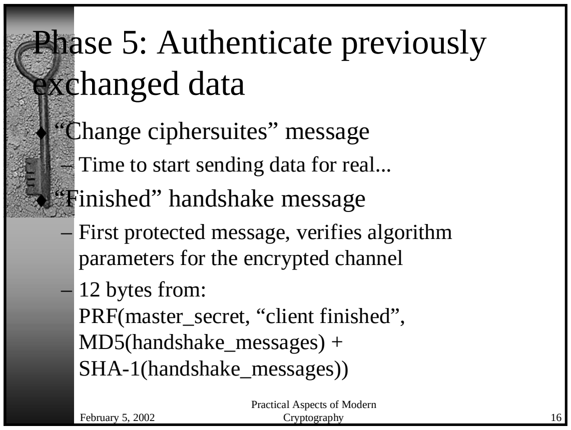
TLS’s PRF

- 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))



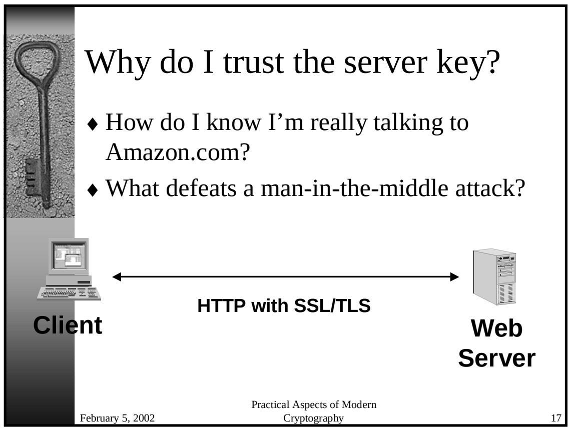
Phases 3 & 4: Authentication

More on this in a little bit...



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))



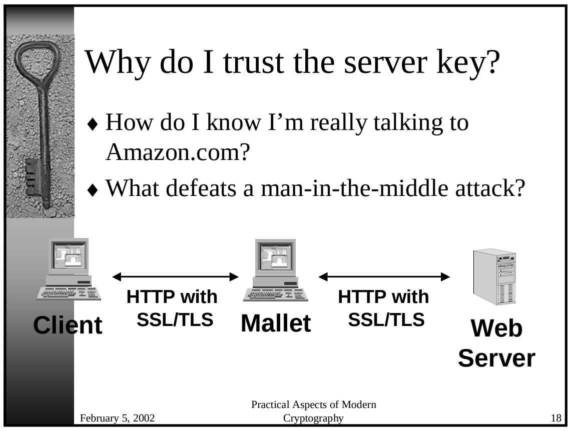
Why do I trust the server key?

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

Client

HTTP with SSL/TLS

Web Server



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?

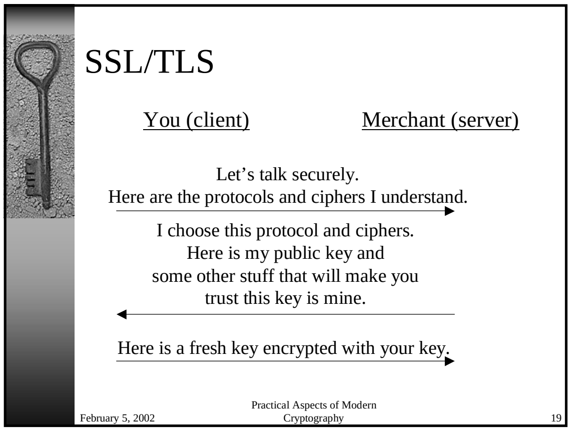
Client

HTTP with SSL/TLS

Mallet

HTTP with SSL/TLS

Web Server



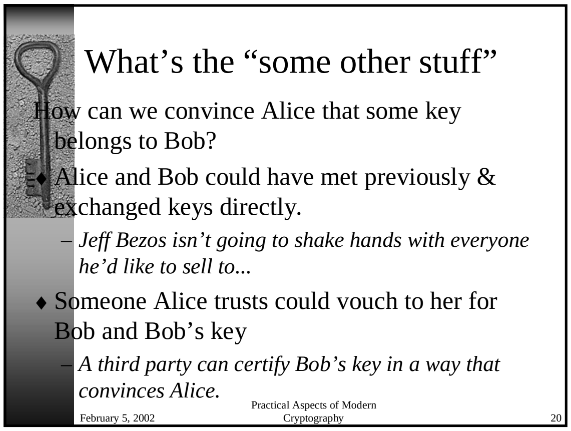
SSL/TLS

<u>You (client)</u> <u>Merchant (server)</u>

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.
←

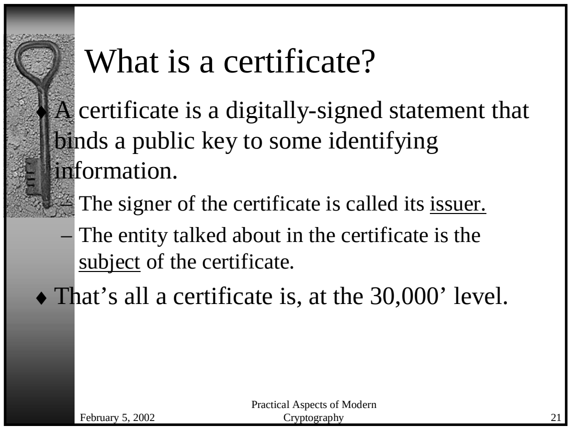
Here is a fresh key encrypted with your key.
→



What's the "some other stuff"

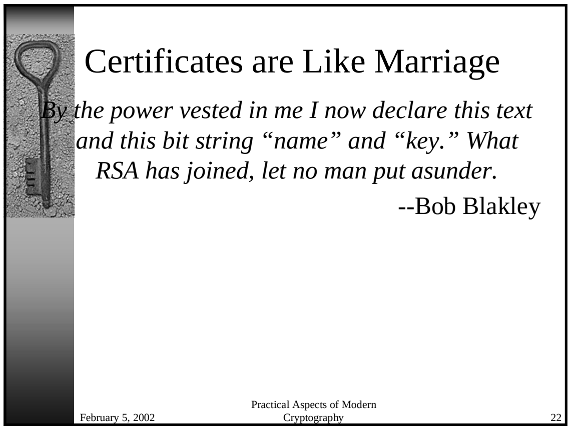
How can we convince Alice that some key belongs to Bob?

- ♦ 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?

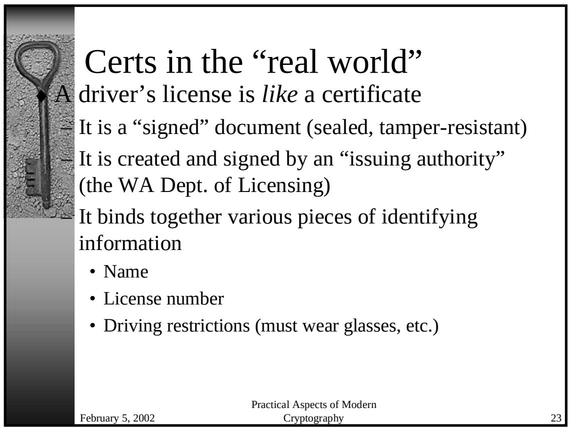
- ♦ A certificate is a digitally-signed statement that binds a public key to some identifying information.
  - – The signer of the certificate is called its <u>issuer.</u>
  - – The entity talked about in the certificate is the <u>subject</u> of the certificate.
- ♦ That's all a certificate is, at the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



Certificates are Like Marriage

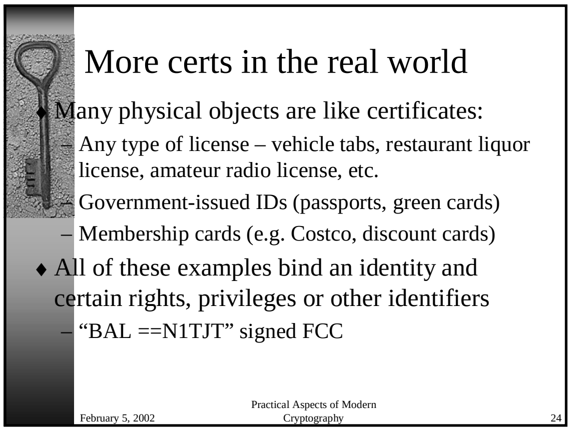
By the power vested in me I now declare this text and this bit string "name" and "key." What RSA has joined, let no man put asunder.

--Bob Blakley



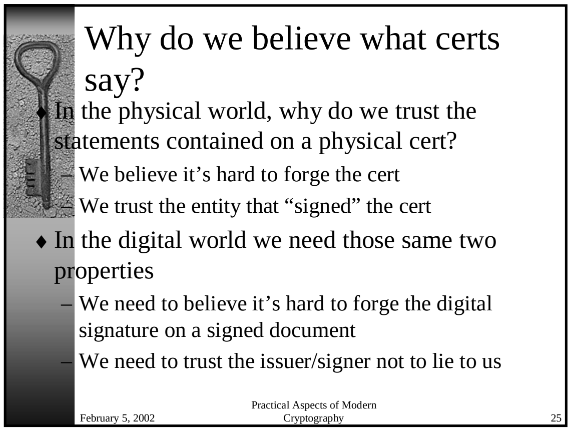
Certs in the "real world"

- ♦ 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.)



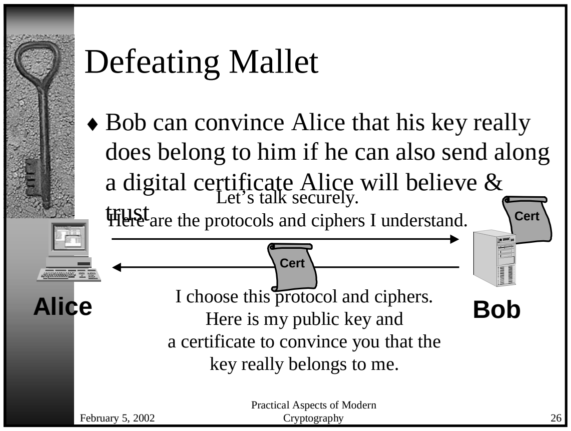
More certs in the real world

- ♦ 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

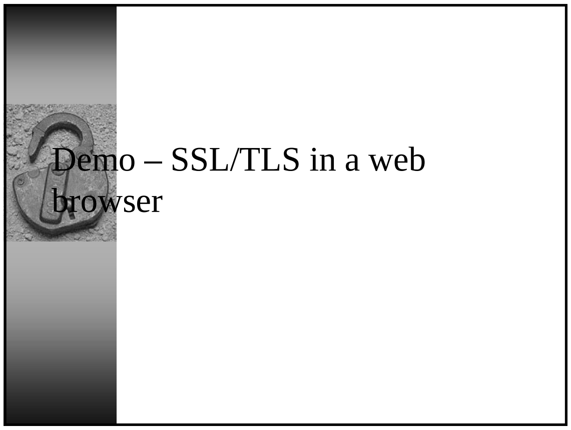


Defeating Mallet

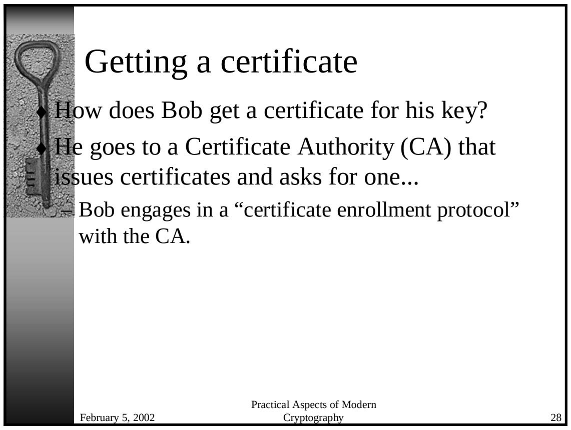
- Bob can convince Alice that his key really does belong to him if he can also send along a digital certificate Alice will believe & trust

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

I choose this protocol and ciphers.
Here is my public key and
a certificate to convince you that the
key really belongs to me.

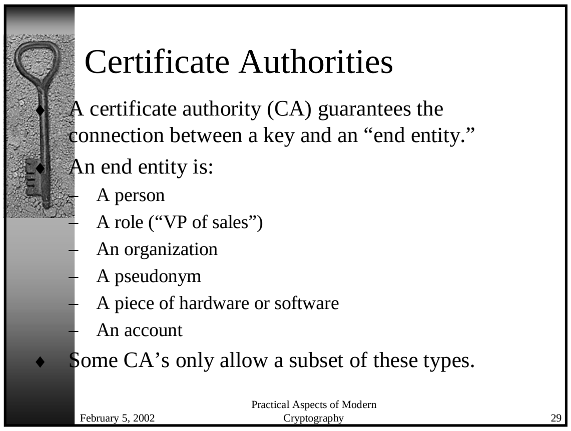


Demo – SSL/TLS in a web browser



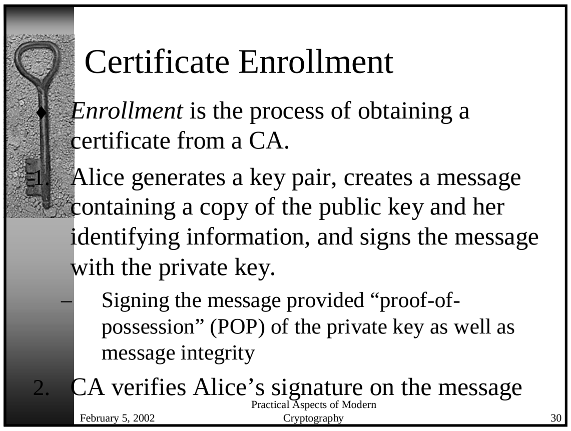
Getting a certificate

- How does Bob get a certificate for his key?
- He goes to a Certificate Authority (CA) that issues certificates and asks for one...
  - Bob engages in a "certificate enrollment protocol" with the CA.



Certificate Authorities

- A certificate authority (CA) guarantees the connection between a key and 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.



Certificate Enrollment

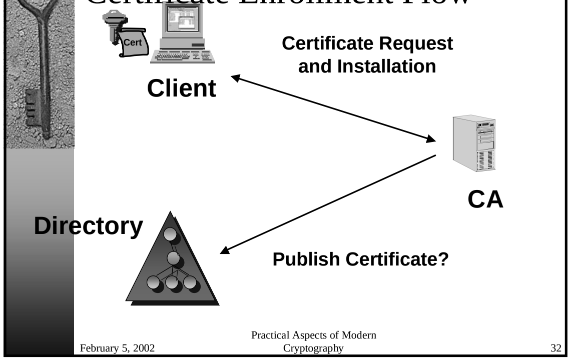
- *Enrollment* is the process of obtaining a certificate from a CA.

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.
   - 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

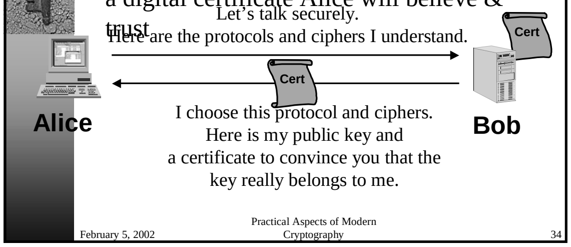


Using Certificates

- ♦ 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?

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


Let’s talk securely.
Here are the protocols and ciphers I understand.
Cert
Cert
Alice
Bob
I choose this protocol and ciphers.
Here is my public key and
a certificate to convince you that the
key really belongs to me.

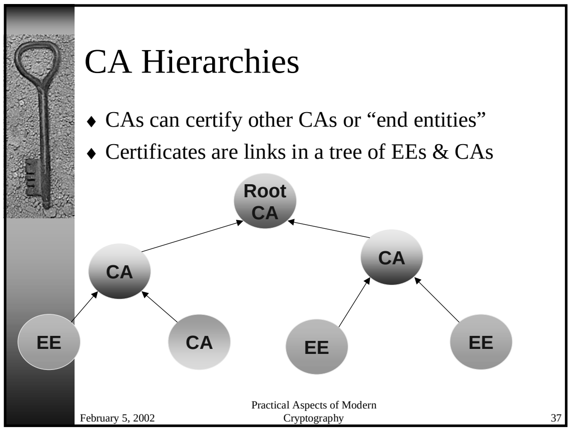
Does Alice Trust Bob’s CA?

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...

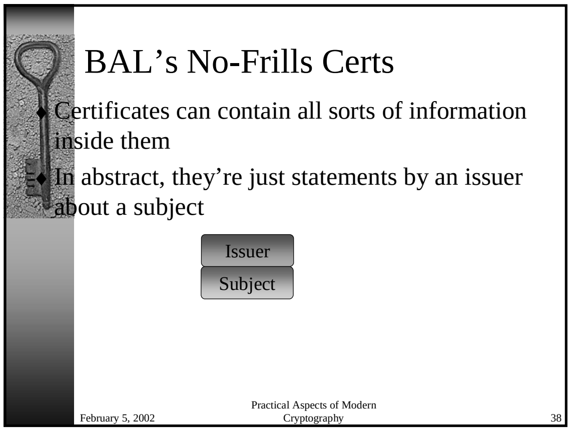
What’s Alice’s Trust Model

- ♦ 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 (next class)



CA Hierarchies

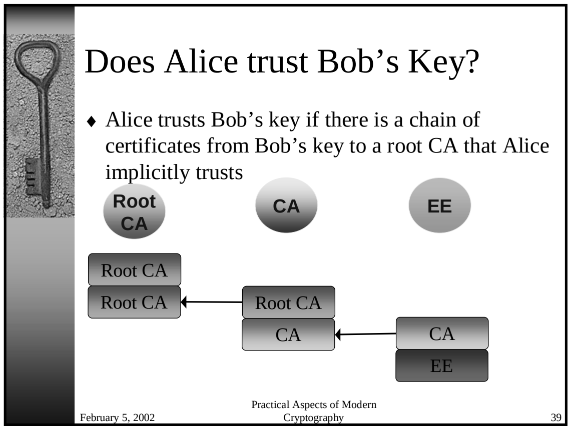
- ♦ CAs can certify other CAs or “end entities”
- ♦ Certificates are links in a tree of EEs & CAs



BAL’s No-Frills Certs

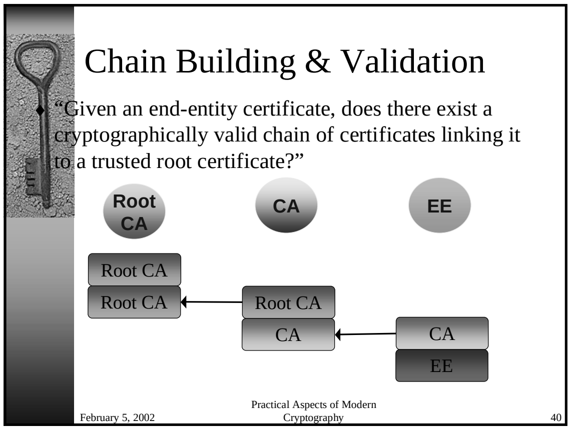
- ♦ Certificates can contain all sorts of information inside them
- ♦ In abstract, they’re just statements by an issuer about a subject

Issuer
Subject



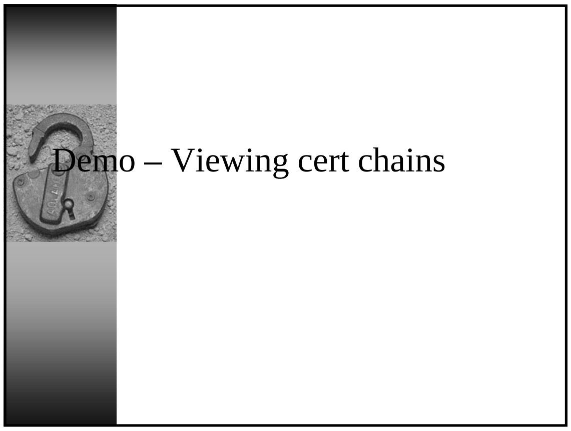
Does Alice trust Bob’s Key?

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

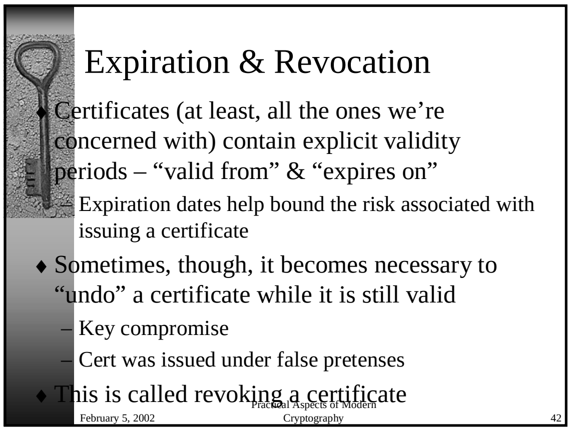


Chain Building & Validation

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



Demo – Viewing cert chains



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

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 <u>retroactive</u> 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)

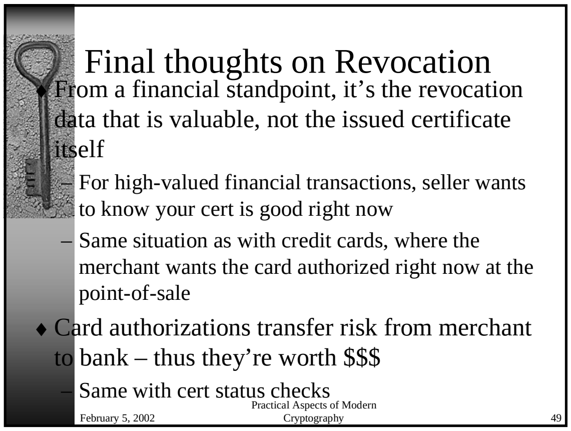
- ♦ Revoking a CA cert is more problematic than revoking an end-entity cert
  - – When you revoke a CA cert, you potentially take out the entire subordinate structure, depending on what chaining logic you use
- ♦ How do you revoke a self-signed cert?
  - – “The cert revokes itself.”
    - • *Huh?*
  - – 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

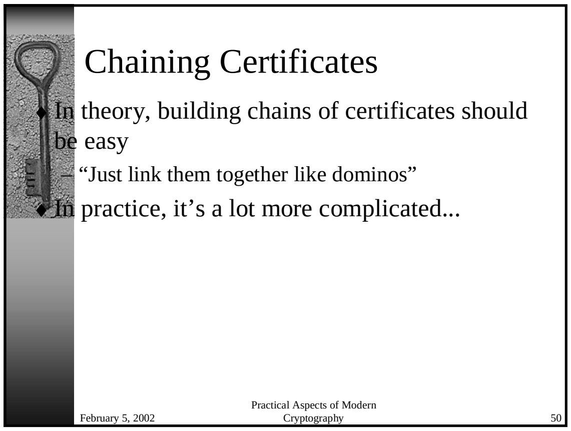
Online Status Checking

- ♦ 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



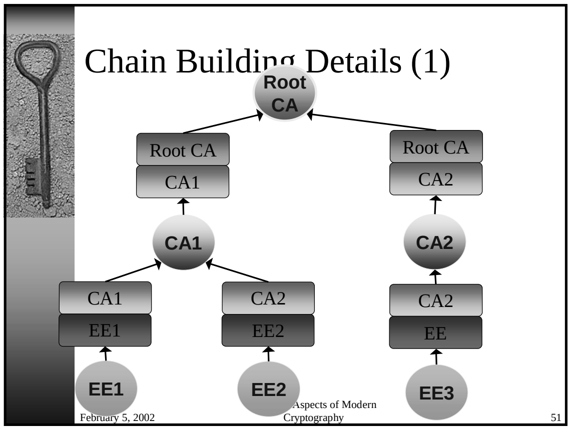
Final thoughts on Revocation

- 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

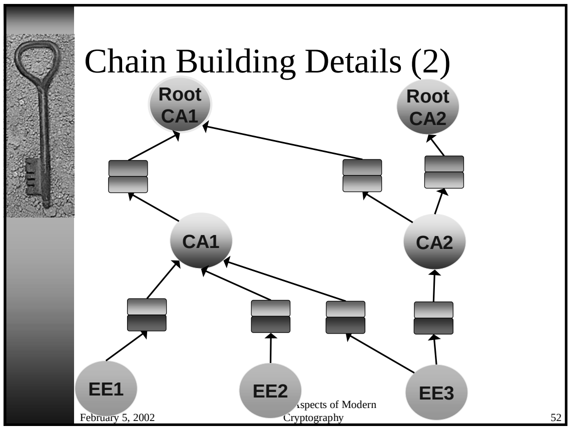


Chaining Certificates

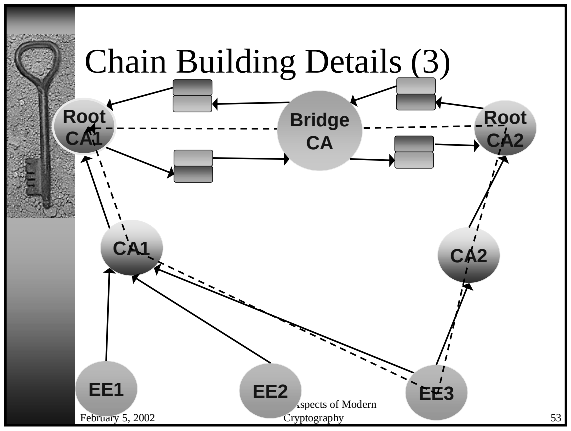
- In theory, building chains of certificates should be easy
  - "Just link them together like dominos"
- In practice, it's a lot more complicated...



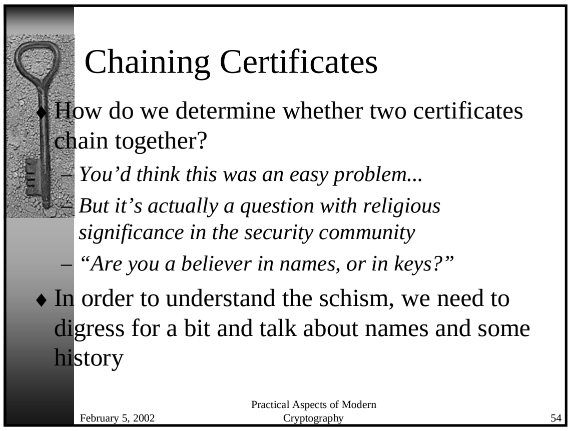
Chain Building Details (1)



Chain Building Details (2)

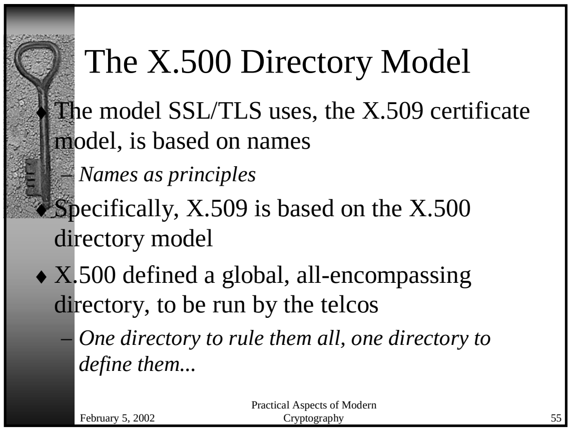


Chain Building Details (3)



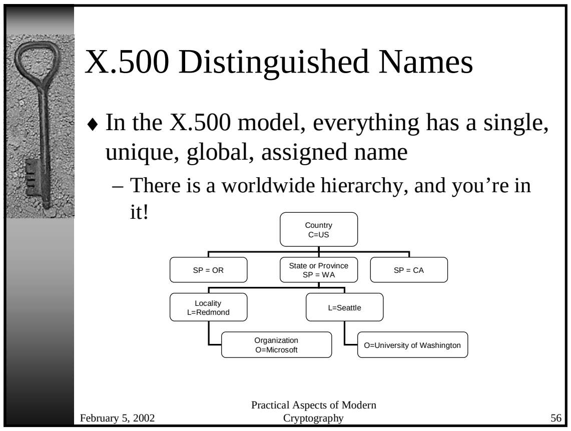
Chaining Certificates

- 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



The X.500 Directory Model

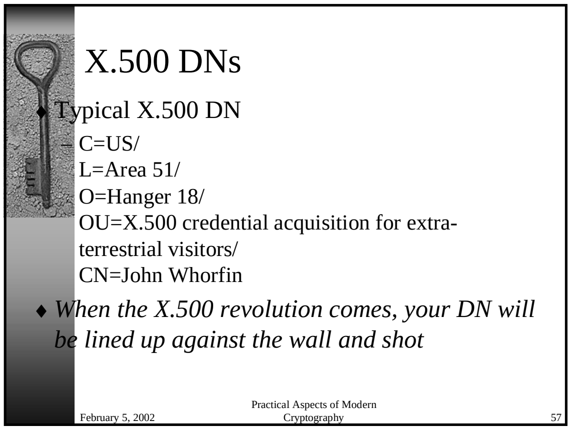
- The model SSL/TLS uses, the X.509 certificate model, is based on names
  - *Names as principles*
- Specifically, X.509 is based on the X.500 directory model
- X.500 defined a global, all-encompassing directory, to be run by the telcos
  - *One directory to rule them all, one directory to define them...*



X.500 Distinguished Names

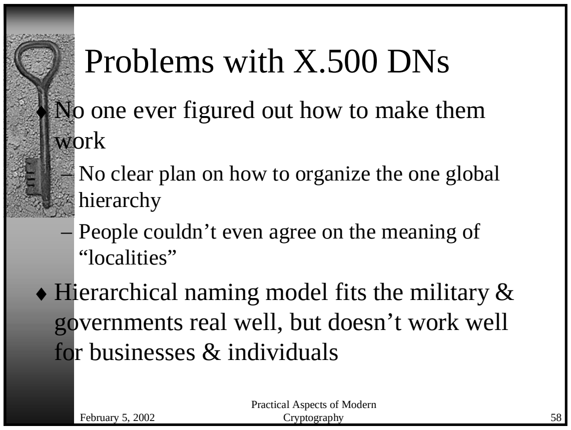
- In the X.500 model, everything has a single, unique, global, assigned name
  - There is a worldwide hierarchy, and you're in it!

Country
C=US
SP = OR
State or Province
SP = WA
SP = CA
Locality
L=Redmond
L=Seattle
Organization
O=Microsoft
O=University of Washington



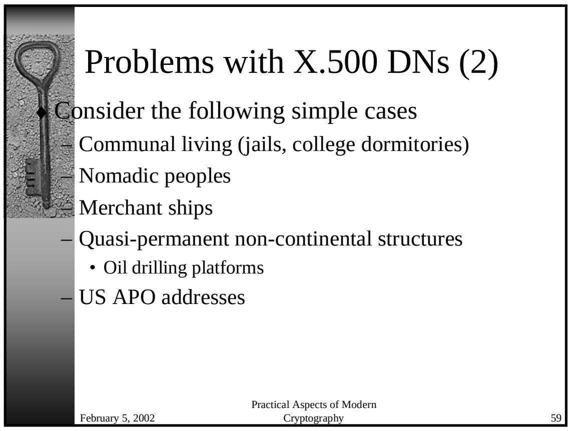
X.500 DNs

- Typical X.500 DN
  - C=US/
    L=Area 51/
    O=Hanger 18/
    OU=X.500 credential acquisition for extra-terrestrial visitors/
    CN=John Whorfin
- *When the X.500 revolution comes, your DN will be lined up against the wall and shot*



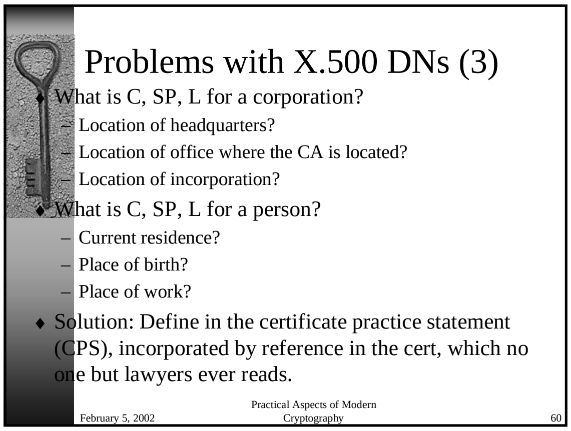
Problems with X.500 DNs

- No one ever figured out how to make them work
  - No clear plan on how to organize the one global hierarchy
  - People couldn't even agree on the meaning of "localities"
- Hierarchical naming model fits the military & governments real well, but doesn't work well for businesses & individuals



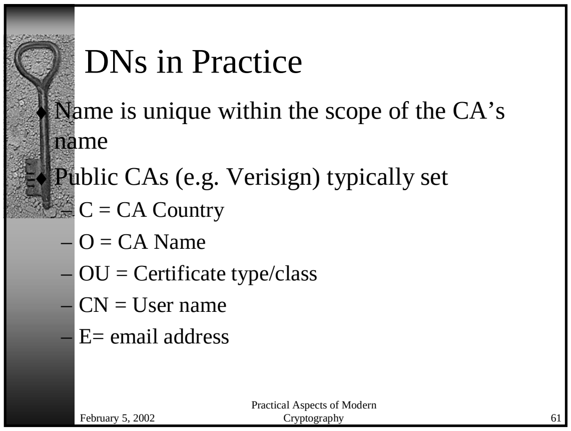
Problems with X.500 DNs (2)

- Consider the following simple cases
  - Communal living (jails, college dormitories)
  - Nomadic peoples
  - Merchant ships
  - Quasi-permanent non-continental structures
    - Oil drilling platforms
  - US APO addresses



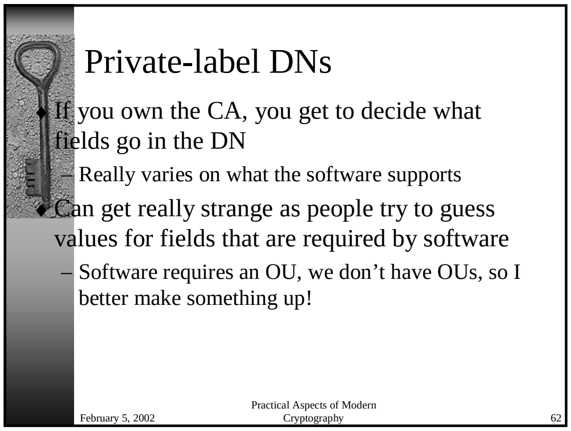
Problems with X.500 DNs (3)

- What is C, SP, L for a corporation?
  - Location of headquarters?
  - Location of office where the CA is located?
  - Location of incorporation?
- What is C, SP, L for a person?
  - Current residence?
  - Place of birth?
  - Place of work?
- Solution: Define in the certificate practice statement (CPS), incorporated by reference in the cert, which no one but lawyers ever reads.



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!