University of Pittsburgh

Applied Cryptography and Network Security
CS 1653

Summer 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Prof. Adam Lee's CS1653 slides.)

Announcements

- Peer Evaluation on Phase 2 due tomorrow
- Homework 7 due this Friday @ 11:59 pm
- Project Phase 3 Due next Monday 7/17 @ 11:59 pm
  - Your team must schedule a meeting with me on or before this Thursday

Public Key Infrastructure

What is a public key infrastructure (PKI)?

Some common PKI models

- Monopoly
- Delegated CAs
- Oligarchy
- Anarchy

Case study: PKIX/X.509

Case study: PGP

What a **digital certificate** anyway?

Keeping track of public keys is simply not enough!

For instance, what if

- A transient fault switches Alice's and Bob's keys
- You and Trent have different opinions about who "Alice" is
- Trent is actually malicious and provides you with incorrect information
- Trent becomes compromised and the attacker provides false data
- …

In reality, key distribution servers manage digital certificates, rather than simple (name, key) pairs

Digital certificates are verifiable and unforgeable bindings between a user, a public key, and a trusted certifier (a certificate authority, or CA)

Terminology

If Alice issues a certificate vouching for Bob's name and key, then Alice is considered the issuer, and Bob is the subject

If Alice is verifying a certificate or chain of certificates, then she is called the verifier or relying party

Any user, machine, or service that has a certificate is called a principal

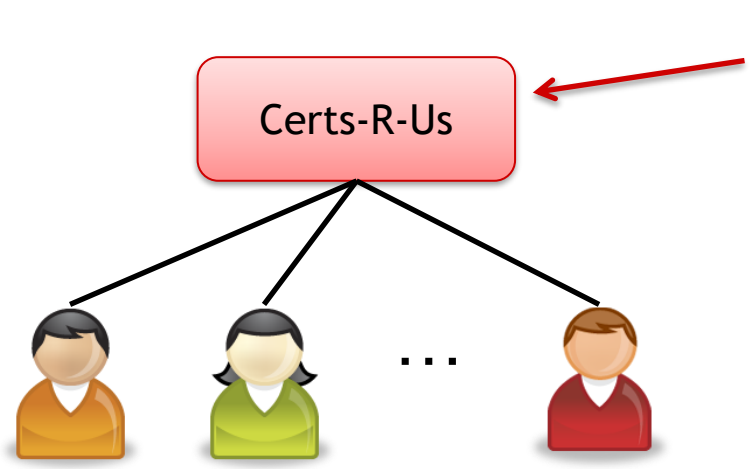
A trust anchor is a public key that is trusted by a verifier to certify the public keys of principals

---

Now that we have a basic vocabulary, let's explore a few ways that PKIs can be organized…

Most PKI models are hierarchical

A monopoly is the simplest form of PKI



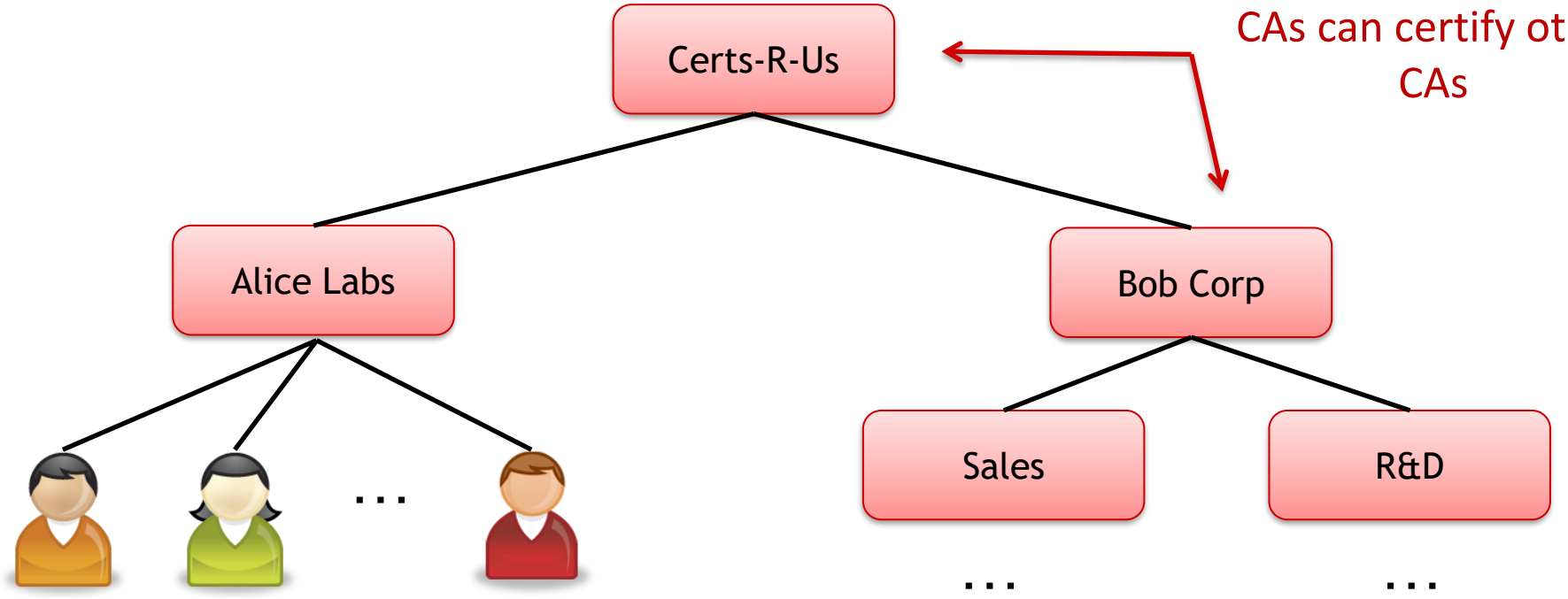
Pros:

- Very simple to model
- Only need to know a single public key to authenticate anyone

Cons:

- In real life, no organization is trusted by everyone
- Changing the key of that CA would be a nightmare
- How would a single CA verify the identity associated with every key?

Allowing delegation fixes some of these problems



Pros:

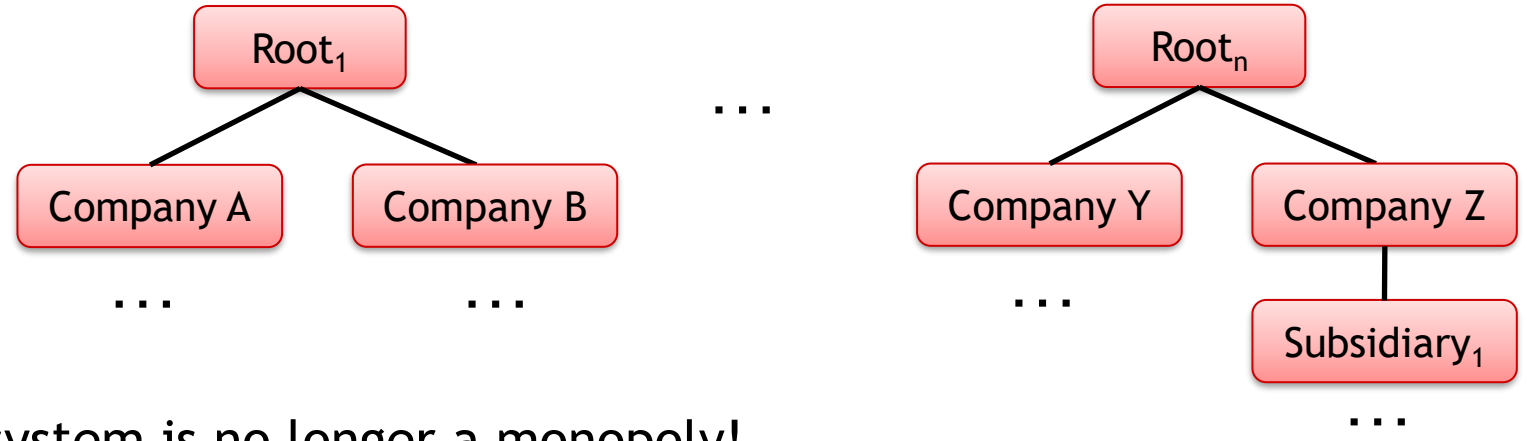
- Fairly simple to model
- Only need to know a single public key to verify a certificate chain that authenticates anyone
- Delegation of control makes identity verification more tractable

Cons:

- In real life, no organization is trusted by everyone
- Changing the key of that CA would be a nightmare

An oligarchy is a collection of hierarchies

Rather than pre-configuring a single trust anchor, allow applications to have a number of trust anchors



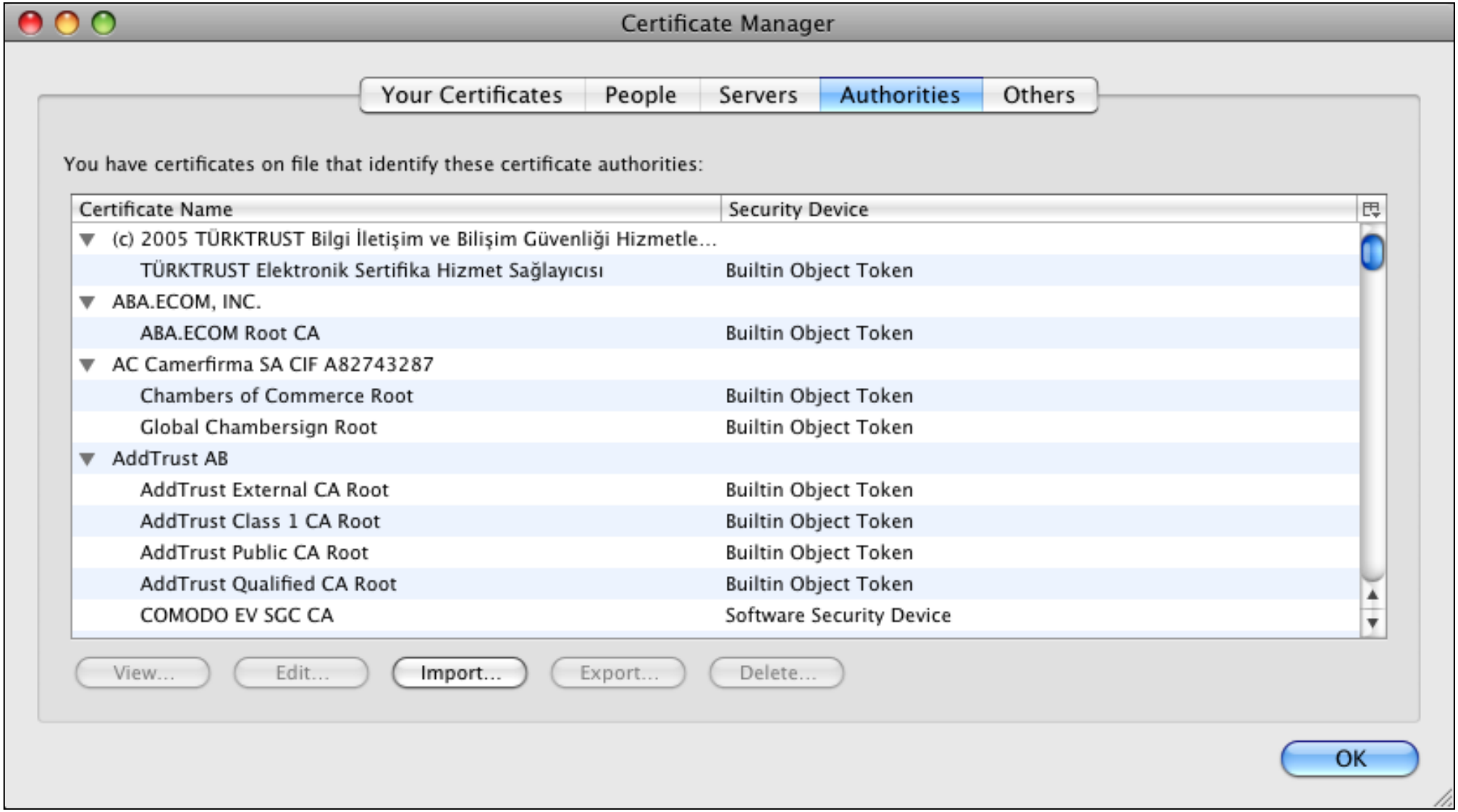
The system is no longer a monopoly!

- In a free market, this will make for more competitive certificate pricing
- There is no single point of attack

Unfortunately, this model is not without its problems

- Trust anchors are often chosen by application vendors, not users!
- If trust anchors are user configurable, it may be possible to trick users into adding a malicious root of trust!
- Managing a large set of trust anchors is complicated…

Your web browser uses the oligarchy model



In any PKI model, certificate issuance is a tricky issue

Say you find a certificate for Sherif Khattab, who is it *really* for?

- A teaching assistant professor at Pitt? (http://www.cs.pitt.edu/~skhattab/)
- A plastic surgeon? (https://www.Khattab.com)
- A pharmaceutical strategist? (https://www.linkedin.com/in/sherifkhattab/)
- many more …

- A certificate is only helpful if it unambiguously vouches for an identity
  - Am I really talking to Amazon.com, or is this a phishing site?
  - Did I just take plastic surgery advice from a CS professor?

Question: How can we *unambiguously* specify identities?

In our case studies, we'll see how this can be accomplished using unique identifiers and social connections

How can we deal with revoked certificates?

Over time, it may become necessary to revoke a certificate. For example,

- The private key becomes compromised (and the owner finds out)
- The binding between the name and certificate becomes invalidated

The revocation process can be handled in an offline manner using a certificate revocation list (CRL)

- Digitally-signed list of revoked certificates
- Issued periodically by the CA
- Always consulted prior to accepting a certificate

Note: This is how credit cards were checked for revocation in the "old" days

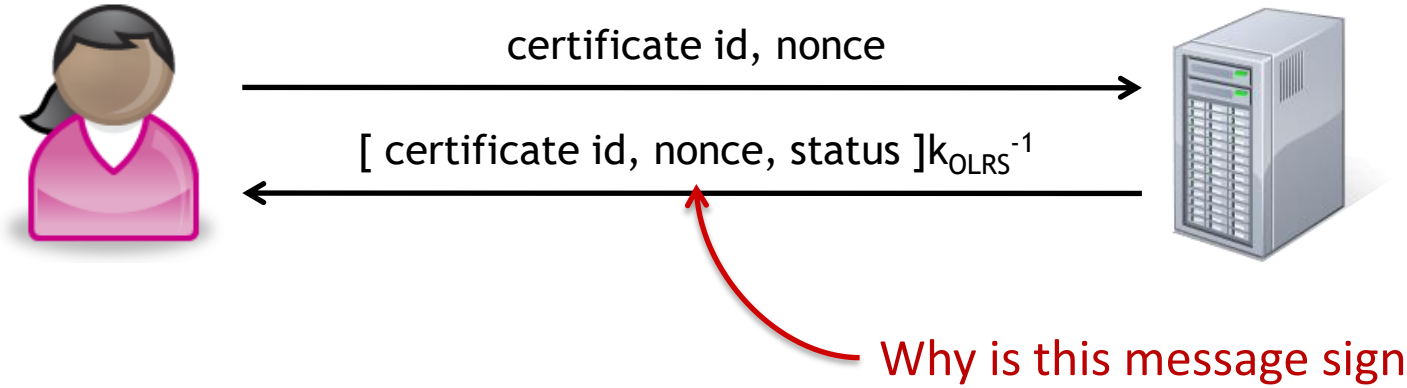
---

Question: Why do CRLs need to be signed?

Question: How can we make the distribution of large CRLs more efficient?

If we assume a reliable online service, we can implement a timelier revocation system

An online revocation server (OLRS) is an online service that is queried to check the validity of certificates



Usually, the OLRS is not the CA itself, but instead a service operating on behalf of the CA (Why?)

Alternatively, some OLRS pre-compute validity responses offline

- E.g., "Valid as of 14:00 on 07/10/2023"

Question: What are the tradeoffs between these two approaches?

Case study: X.509

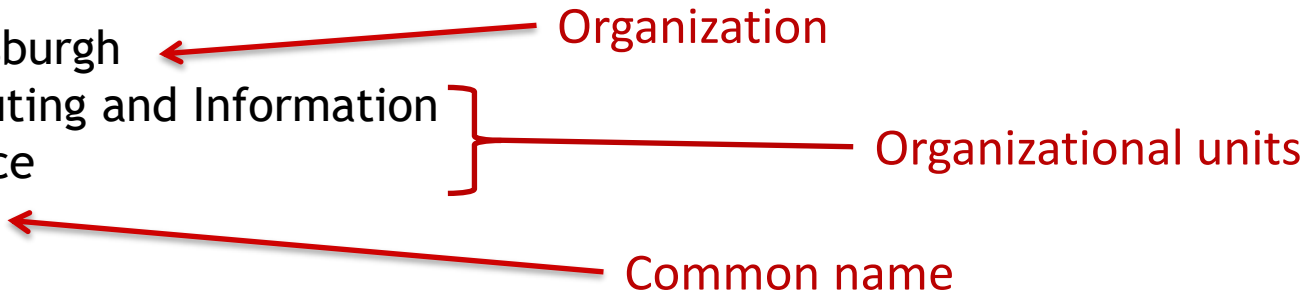
The IETF's PKIX working group is tasked with developing standards for deploying public key infrastructures based on the X.509 standard

This group has released a number of RFCs describing things like

- Certificate contents (RFC 5280)
- Certificate path validity checking (RFC 3280)
- Handling revocation lists (RFC 5280)
- Online certificate status protocols (RFC 2560)
- …

X.509 certificates uniquely identify principals through the use of X.500 distinguished names (DNs)

- /O=University of Pittsburgh
  /OU=School of Computing and Information
  /OU=Computer Science
  /CN=Sherif Khattab



X.509 Certificates

An X.509 certificate contains quite a bit of information

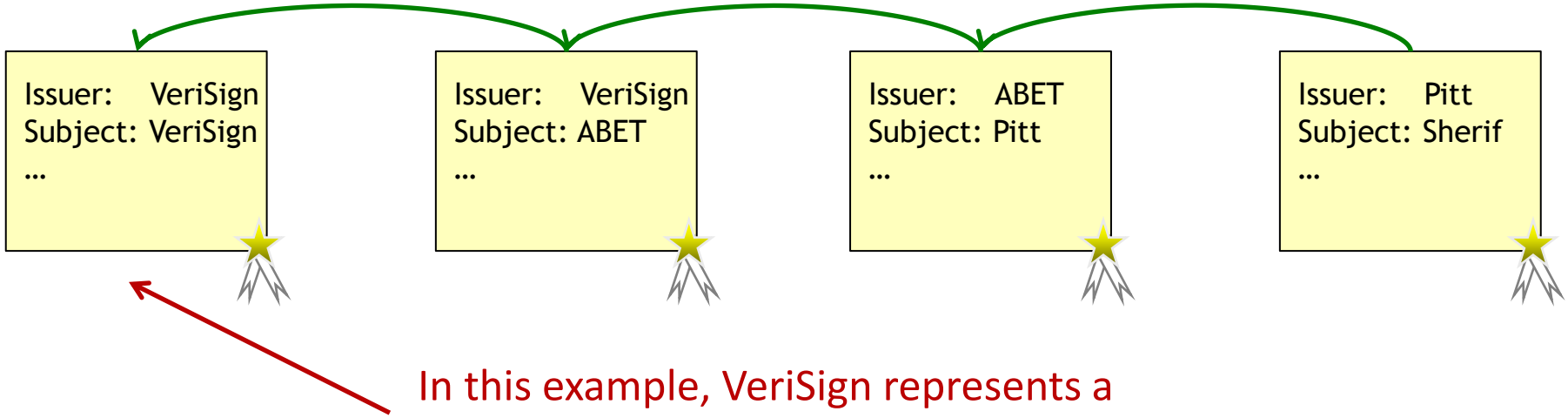
- Version
- Serial number: Must be unique amongst all certificates issued by the same CA
- Signature algorithm ID: What algorithm was used to sign this certificate?
- Issuer's distinguished name (DN): Who signed this certificate
- Validity interval: Start and end of certificate validity
- Subject's DN: Who is this certificate for?
- Subject's public key information: The public key of the subject
- Issuer's unique ID: Used to disambiguate issuers with the same DN
- Subjects unique ID: Used to disambiguate subjects with the same DN
- Extensions: Typically used for key and policy information
- Signature: A digital signature of (a hash of) all other fields

Note that each certificate has exactly one issuer

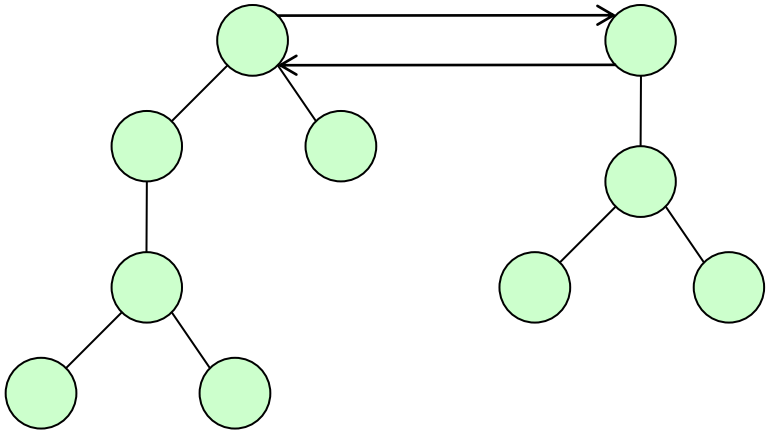
Question: What if you wanted a certificate to be authorized by more than one entity, but didn't want to deviate from the X.509 standard?

How can we validate X.509 certificates?

Since each certificate has exactly one issuer, we can build chains of trust



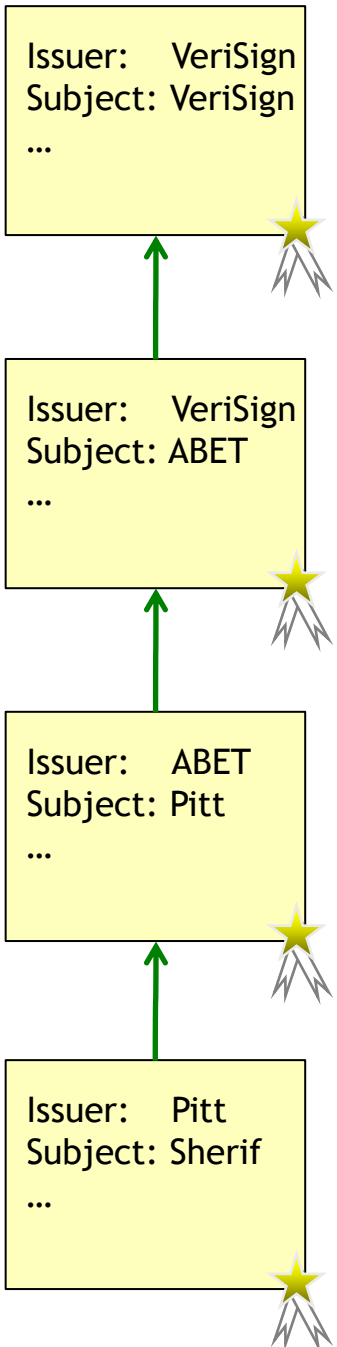
As you might expect, X.509 CAs form a hierarchy



Note: Cross-certification allows certificates from one domain to be interpreted in another

Question: Is there any value to allowing the use of self-signed client certificates within the X.509 model?

RFC 3280 describes how X.509 certificate chains are supposed to be validated



Checks run on certificate 1:

- Is this certificate a trust anchor?
- Is this certificate valid at the current time?
- Has this certificate been revoked?
- Is the self signature on this certificate valid?

Checks run on certificates 2 through *n*-1:

- Are these certificates CA certificates?
- Are these certificates currently valid?
- Have any of these certificate been revoked?
- Was certificate *i* signed by certificate *i*-1?

Checks run on certificate n:

- Is this certificate valid at the current time?
- Has this certificate been revoked?
- Was this certificate signed by certificate *n*-1?

Question: How do we find certificate chains in the first place?

Case study: PGP

Pretty Good Privacy (PGP) is a hybrid encryption program that was first released by Phil Zimmerman in 1991

In the PGP model

- Users are typically identified by their email address
- Users create and manage their own digital certificates
- Certificates can be posted and discovered by using volunteer "key servers"
- Key servers also serve a function similar to an OLRS

Email addresses are GUIDs, so they effectively disambiguate identities

However, note that there are no CAs in the system! As such, users can create certificates for any email address or identity that they want!

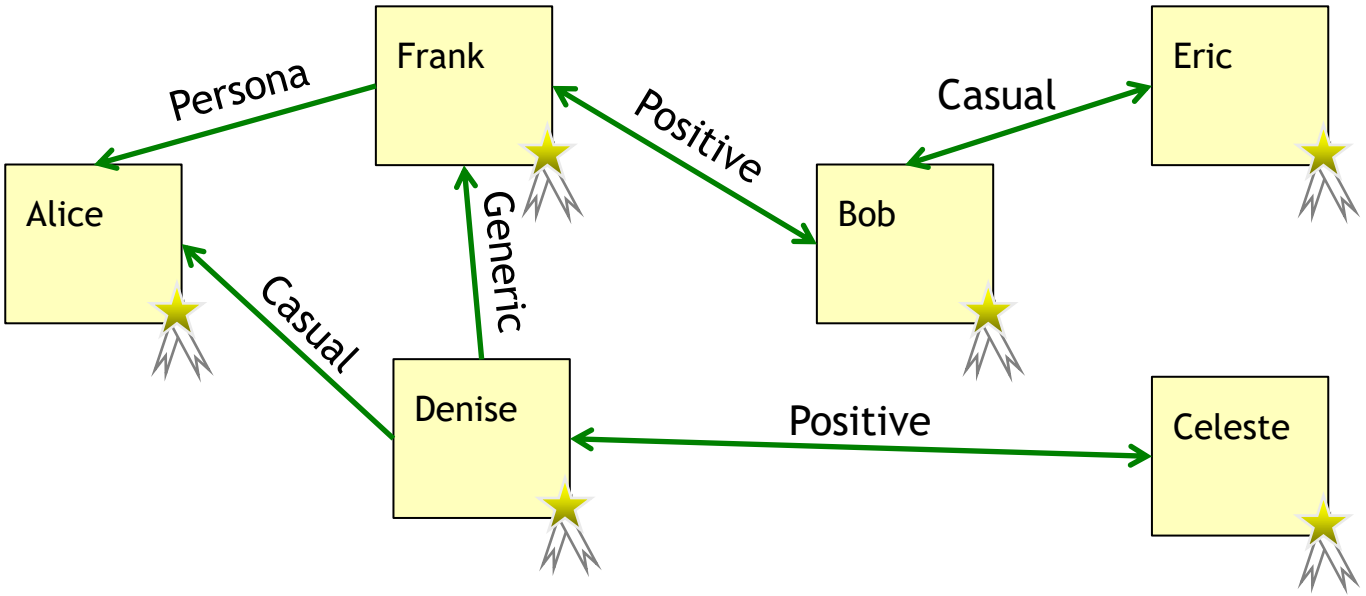
How can we deal with this anarchy?!

Unlike X.509, PGP takes a more "grassroots" approach to certificate validation

Users create their own certificates, and sign the certificates of others

- "I am Alice and I have verified that this certificate belongs to Bob"
- Four levels of certification: Generic, Persona, Casual, and Positive

This is essentially a cryptographic social network!



Question: Can Celeste trust Frank? Can Eric trust Frank?

PGP can handle two types of certificate formats

In addition to supporting X.509 certificates, PGP also has its own native certificate format

A PGP certificate contains the following information:

- PGP version number
- Public key: e.g., an RSA, DH, or DSA public key
- The certificate holder's "information": Free-form contents
- Signature: A self-signature on the certificate
- Validity period: When was this certificate created? How long is it valid?
- Preferred algorithms: What cipher suites should be used?

Question: If PGP keys are self-issued, why is a signature needed?

Discussion

Discussion Question

Which approach do you think is better? Why?

So Far …

Digital certificates act as verifiable and unforgeable bindings between an identity and a public key

There are many PKI models out there

- Some based on hierarchies (e.g., X.509)
- Some more ad-hoc (e.g., PGP)

In any deployment model, it is essential to unambiguously identify users

Handling revoked certificates can be done either online or offline

Next: Real-time communication security

Real-time communication security

Now we're going to talk about real-time security issues

Issues related to session key disclosure

- Perfect forward secrecy
- Forward secrecy
- Backward secrecy

Deniable authentication protocols

Denial of service

- Computational puzzles
- Guided tour puzzles

Real-time what?

Real-time security is a "catch all" term

- i.e., any interactive security protocol
- Basically: most authentication protocols

Note that the security protocols that we've discussed to date can be viewed as "soft" real-time protocols

- We require liveness (read: progress) in the system
- However, we do not usually have hard deadlines

We'll look at two classes of "real-time" properties today:

Key negotiation properties

Timeliness properties

Session keys: recap

Parties in cryptographic protocols typically have long-lived secrets

- Kerberos: K_A shared between Alice and the KDC
- PKI: Public/private key pairs (k_A, k_A^{-1})

Even so, it is good practice to generate new session keys regularly (Why?)

- Help limit compromises due to overuse
- Prevent breaks of one key from compromising all sessions
- …

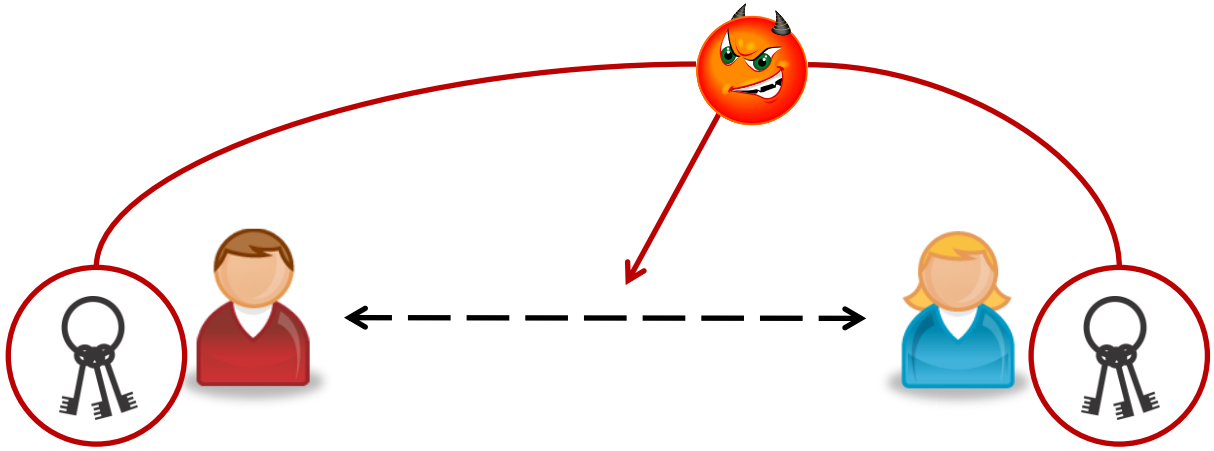
In general, session keys should be used for only a single session

Typically both parties contribute to the session keys for a protocol (Why?)

- If either party can generate "good" pseudorandom numbers, the derived key will be reasonably secure
- Less likely to have replay attacks lead to successful impersonations

Ideally, the session keys that we choose should have a property known as perfect forward secrecy

A key exchange protocol is said to have perfect forward secrecy if an adversary that (i) watches all messages exchanged and (ii) later compromises both parties cannot recover the agreed-upon session key



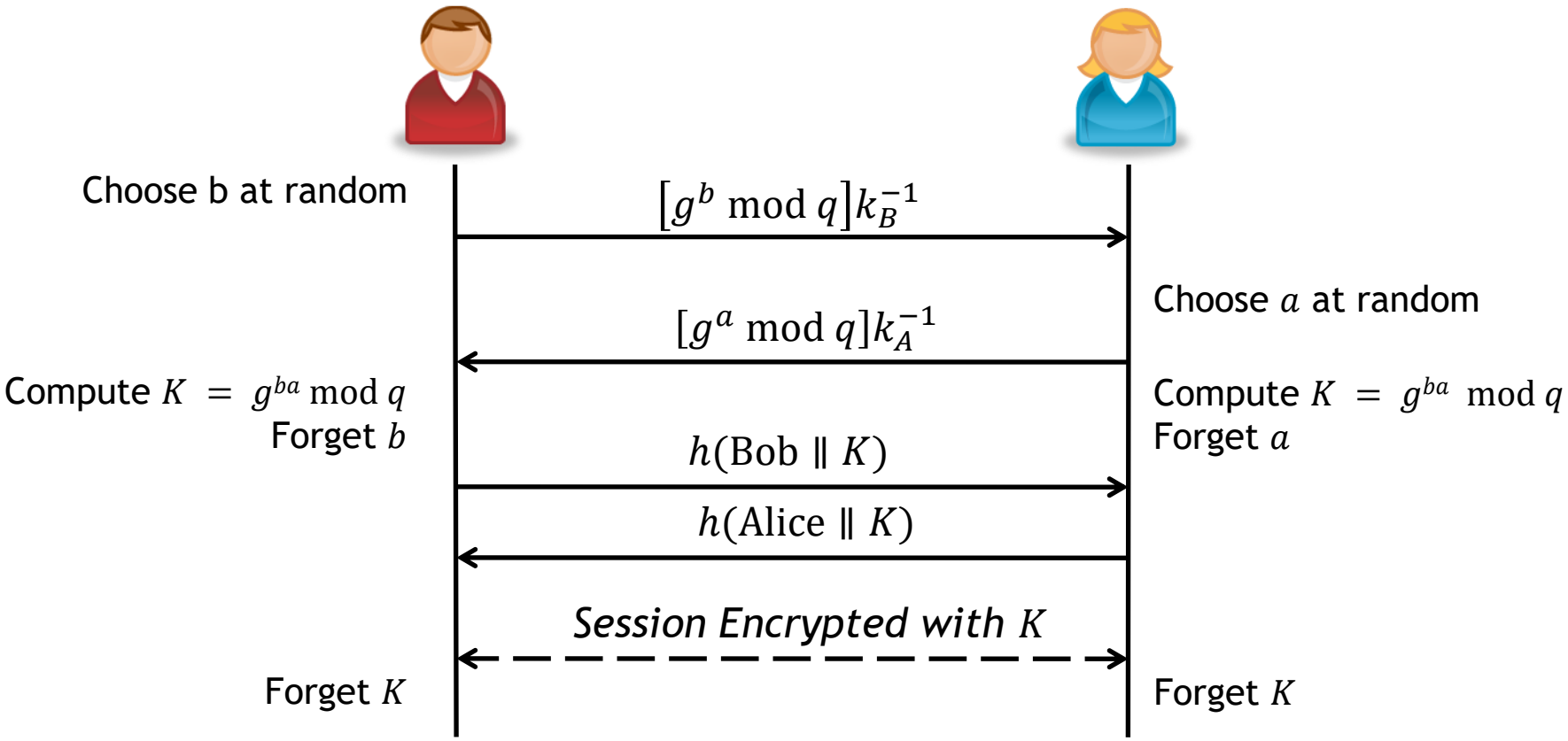
How do we achieve perfect forward secrecy?

- Generate a temporary secret
- Use only information that is not stored at the node long-term

---

This sounds good, but what protocols actually give us this property?

Signed Diffie-Hellman key exchange!



Question: Why does this protocol provide perfect forward secrecy?

- Adversary learns $g^a \bmod q$ and $g^b \bmod q$ by snooping
- Adversary learns k_B^{-1} and k_A^{-1} by compromise
- The important pieces are a, b, and K
  - These are not stored in the long term