Announcements

Homework 2 will be released today

- Available on the **course website**
  - If you cannot see it, try refreshing the page…
- Due ***in two weeks***: 11/20/19 11:59pm
- Submit through **GradeScope**

Reflection Attack and a Fix

- Original Protocol
  1. $A \rightarrow B$: r_A
  2. $B \rightarrow A$: $\{ r_A, r_B \}_K$
  3. $A \rightarrow B$: r_B

- Attack
  1. $A \rightarrow E$: r_A
  2. $E \rightarrow A$: r_A : Starting a new session
  3. $A \rightarrow E$: $\{ r_A, r_A' \}_K$: Reply to (2)
  2. $E \rightarrow A$: $\{ r_A, r_A' \}_K$: Reply to (1)
  3. $A \rightarrow E$: r_A'

Solutions?

- Use 2 different uni-directional keys k" (A→B) and k' (B→A)
- Remove symmetry (direction, msg identifiers)

Interleaving Attacks

- Protocol for Mutual Authentication
  1. $A \rightarrow B:$ $A, r_{A,}$
  2. $B \rightarrow A:$ $r_B, \{ r_B, r_A, A \}_{SK_B}$
  3. $A \rightarrow B:$ $r_A', \{ r_A', r_B, B \}_{SK_A}$

- Attack (E impersonates A):
  1. $E \rightarrow B:$ A, r_A
  2. $B \rightarrow E:$ $r_B, \{ r_B, r_A, A \}_{SK_B}$

  1. $E \rightarrow A:$ B, r_B

  2. $A \rightarrow E:$ $r_A', \{ r_A', r_B, B \}_{SK_A}$

  3. $E \rightarrow B:$ $r_A', \{ r_A', r_B, B \}_{SK_A}$

- Attack due to symmetric messages (2), (3)

Lessons learned?

- Designing **secure** protocols is hard. There are **many** documented failures in the literature.

- Good protocols are already standardized (e.g., ISO 9798, X.509, …) – use them!
  - In other words, don't invent your own!

- The problem of verifying (proving) protocol security gets much harder as protocols get more complex: more parties, messages and rounds.

If interested to learn further, read this paper:

“Programming Satan’s Computer”
by R. Anderson and R. Needham

available at:

http://www.cl.cam.ac.uk/~rja14/Papers/satan.pdf

Secure Protocol Examples

Authenticated Public-Key-based Key Exchange (Station-to-Station or STS Protocol)

Choose random v

$$y_a = a^v \bmod p$$

Choose random w,
Compute

$$K_{ba} = (y_a)^w \bmod p$$
$$y_b = a^w \bmod p$$
$$SIG_{bob} = \{y_b, y_a\}^{Bob}$$

$$CERT_{bob}, y_b, SIG_{bob}$$

Compute

$$K_{ab} = (y_b)^v \bmod p$$
$$SIG_{alice} = \{y_a, y_b\}^{alice}$$

$$CERT_{alice}, SIG_{alice}$$

x.509 Authentication & Key Distribution Protocols

One-msg
A→B

$$\{1, t_a, r_a, B, other_a, [K_{ab}]_{PK_B}\}_{SK_A}$$

Two-msg
A→B

$$\{2, t_a, r_a, B, other_a, [K_{ab}]_{PK_B}\}_{SK_A}$$

$$\{2, t_b, r_b, A, r_a, other_b, [K_{ba}]_{PK_A}\}_{SK_B}$$

Three-msg
A←→B

$$\{3, t_a, r_a, B, other_a, [K_{ab}]_{PK_B}\}_{SK_A}$$

$$\{3, t_b, r_b, A, r_a, other_b, [K_{ba}]_{PK_A}\}_{SK_B}$$

$$\{3, r_b\}_{SK_A}$$

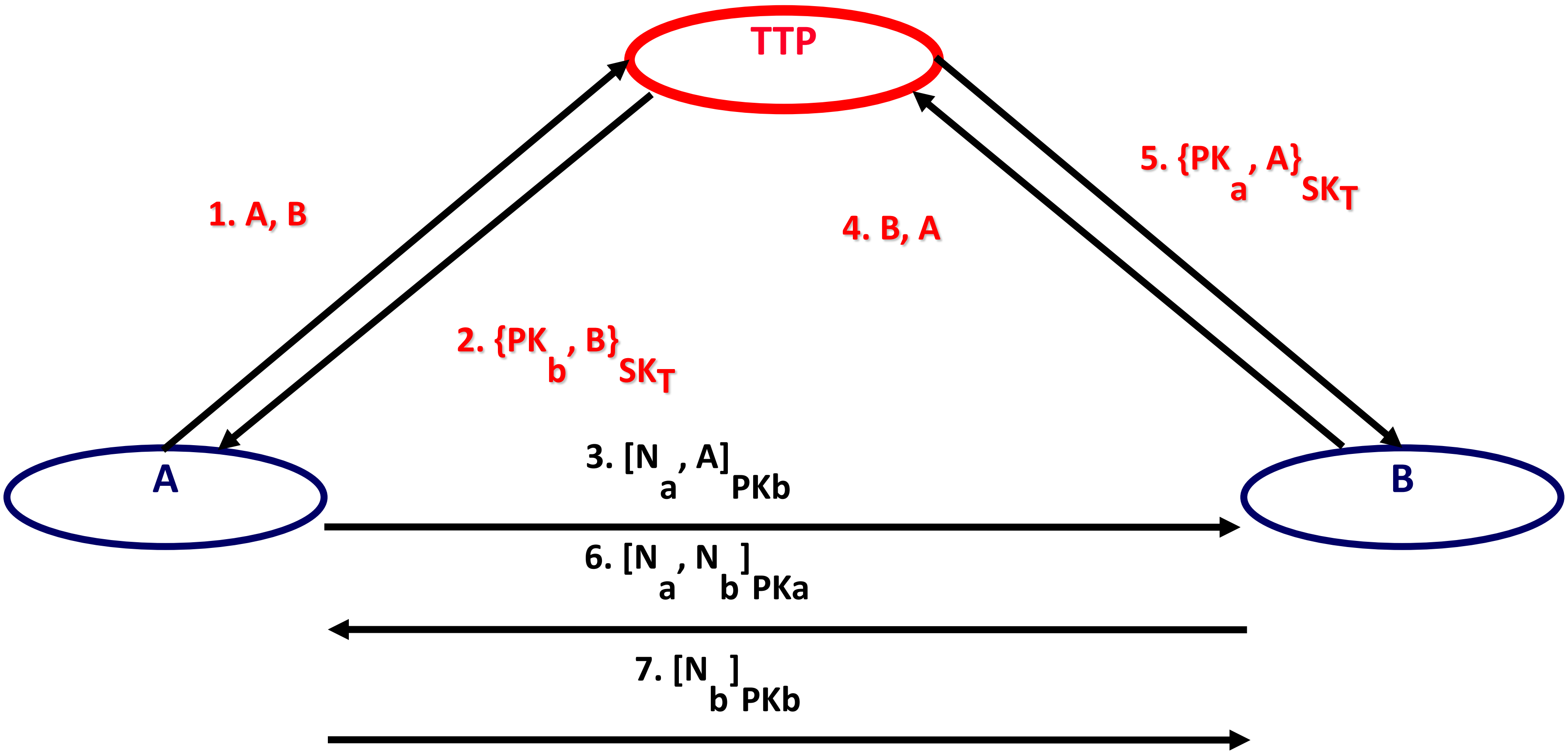
Lecture 11

Public Key Distribution (and Certification)

(Chapter 15 in KPS)

[lecture slides are adapted from previous slides by Prof. Gene Tsudik]

Recall PK-based Needham-Schroeder



Here, TTP acts as an "on-line" certification authority (CA) and takes care of revocation

What If?

- Alice and Bob have:
  - No common mutually trusted TTP(s)
    - and/or
  - No on-line TTP(s)

Public Key Infrastructure (Distribution)

- **Problem:** How to determine the correct public key of a given entity
  - Binding between IDENTITY and PUBLIC KEY

- Possible Attacks
  - Name spoofing: Eve associates Alice's name with Eve's public key
  - Key spoofing: Eve associates Alice's key with Eve's name
  - DoS: Eve associates Alice's name with a random key

- What happens in each case?

Public Key Distribution

- Popek - Kline (1979) proposed “trusted third parties” (TTPs) as a means of PK distribution:
  - Each org-n has a TTP that knows public keys of all of its constituent entities and distributes them on-demand
  - On-line protocol like the one we already saw
  - TTP = single point of failure
  - Denial-of-Service (DoS) attacks

Certificates

- Kohnfelder (BS Thesis, MIT, 1978) proposed “certificates” as yet another public-key distribution method
- Certificate = explicit binding between a public key and its owner’s (unique) name
- Must be issued (and signed) by a recognized trusted Certificate Authority (CA)
- Issuance is done off-line

Authenticated Public-Key-based Key Exchange (Station-to-Station or STS Protocol)

Choose random v

$$y_a = a^v \bmod p \longrightarrow$$

Choose random w, Compute

$$K_{ba} = (y_a)^w \bmod p$$

$$y_b = a^w \bmod p$$

$$SIG_{bob} = \{y_b, y_a\}^{Bob}$$

$$\longleftarrow CERT_{bob}, y_b, SIG_{bob}$$

Compute

$$K_{ab} = (y_b)^v \bmod p$$

$$SIG_{alice} = \{y_a, y_b\}^{alice}$$

$$CERT_{alice}, SIG_{alice} \longrightarrow$$

Certificates

- Procedure
  - Bob generated SK_B, PK_B, registers PK_B with CA
  - Bob receives his certificate:

$CERT_B$ = {PK_B, ID_B, issuance_time, expiration_time, etc.,...}SK_{CA}

  - Bob sends $CERT_B$ to Alice
  - Alice verifies CA's signature, checks expiration, ...
    - PK_{CA} hard-coded in Alice's (software or hardware)
  - Alice uses PK_B to encrypt for Bob and/or verifying Bob's signatures

Who Issues Certificates?

- CA: **<u>Certification Authority</u>**
  - e.g., GlobalSign, VeriSign, Thawte, etc.
  - look into your browser or smartphone...

- Trustworthy (at least to its users/clients)
- *<u>Off-line operation (usually)</u>*
- Has its own well-known long-term certificate
- May store (as backup) issued certificates
- Very secure: physically and electronically

How does it work?

- A public/private key-pair is generated by user
- User requests certificate via a local application (e.g., web browser, email, in person, etc.)
  - Good idea to prove knowledge of private key as part of the certificate request. Why?
- Public key and owner’s name are part of a certificate

- Private keys only used for small amount of data (signing, encryption of session keys)
- Symmetric keys (e.g., AES) used for bulk data encryption

Certification Authority (CA)

- CA must verify/authenticate the entity requesting a new certificate.
- CA’s own certificate is signed by a higher-level CA. Root CA’s certificate is self-signed and pre-installed in devices
- CA is a critical part of the system and must operate in a secure and predictable way according to some policy.

Who needs them?

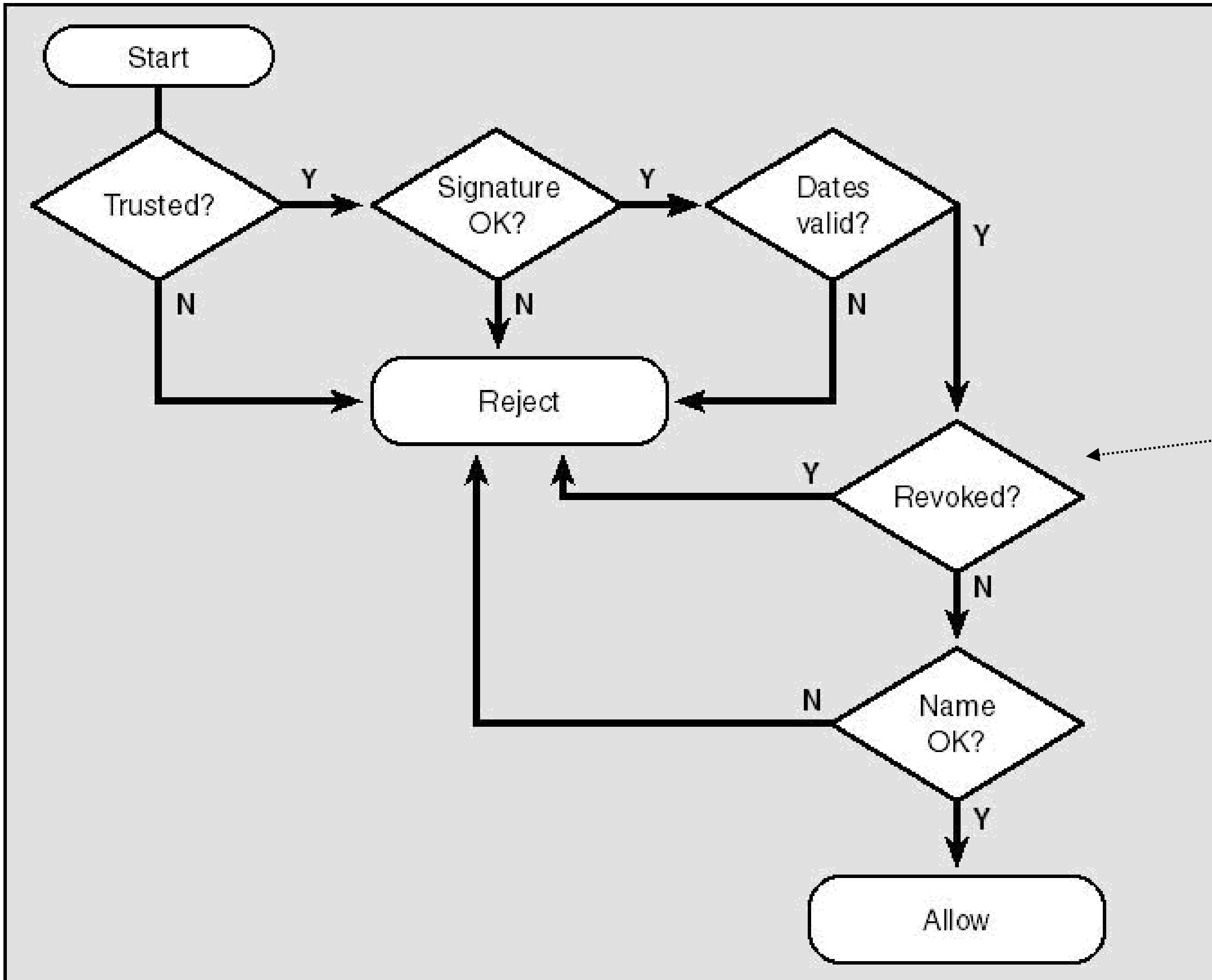
- Alice's certificate is checked by whomever wants to:
  1) verify her signatures, and/or 2) encrypt data for her.

- A signature verifier (or encryptor) must:
  - know the public key of the CA(s)
  - trust all CAs involved

- Certificate checking is: verification of the signature and validity

- Validity: expiration + revocation checking

Verifying a Certificate (assuming Common CA)



To be covered later

What are PK Certificates Good For?

- Secure channels in TLS / SSL for web servers
- Signed and/or encrypted email (PGP,S/MIME)
- Authentication (e.g., SSH with RSA)
- Code signing (for distribution, updates, etc.)
- Encrypting filesystems (EFS in Windows or IOS)
- IPSec: encryption/authentication at the network layer

Components of a Certification System

- Request and issue certificates (different categories) with verification of identity
- Storage of certificates
- Publishing/distribution of certificates (LDAP, HTTP)
- Pre-installation of root certificates in a trusted environment
- Support by OS platforms, applications and services
- Maintenance of database of issued certificates (no private keys!)
- Helpdesk (information, lost + compromised private keys)
- Advertising revoked certificates (and support for applications to perform revocation checking)
- Storage “guidelines” for private keys

CA Security

- Must minimize risk of CA private key being compromised
- Best to have an off-line CA
  - Requests may come in electronically but should not be processed in real time
- In addition, using tamper-resistant hardware for the CA would help (ideally, to make it impossible to extract CA's private key)

Storage of Private Key

- The problem of having the user to manage the private key (user support, key loss or compromise)
- Modern OS-s offer Protected Storage → saves private keys (encrypted).
- Applications take advantage of this; Browsers sometimes save private keys encrypted in their configuration directory
- Users who mix applications or platforms must manually import / export private keys via PFX files.

Key Lengths

- A CA should have an (RSA) modulus size of >= 3072 bits given its importance and typical lifetime

- A personal (end-user) RSA public key should have a modulus size of at least 2048 bits

Key Lengths

January 2016 Recommendation from the NSA

https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf

Algorithm	Usage
RSA 3072-bit or larger	Key Establishment, Digital Signature
Diffie-Hellman (DH) 3072-bit or larger	Key Establishment
ECDH with NIST P-384	Key Establishment
ECDSA with NIST P-384	Digital Signature
SHA-384	Integrity
AES-256	Confidentiality

Naming Comes First!

- Can not have certificates without a comprehensive naming scheme
- Can not have PKI without a comprehensive distribution/access method
- X.509 certificate format uses X.500 naming
- X.500 Distinguished Names (DNs) contain a subset of:
  - **C** **Country**
  - **SP** **State/Province**
  - **L** **Locality**
  - **O** **Organization**
  - **OU** **Organizational Unit**
  - **CN** **Common Name**

X.500

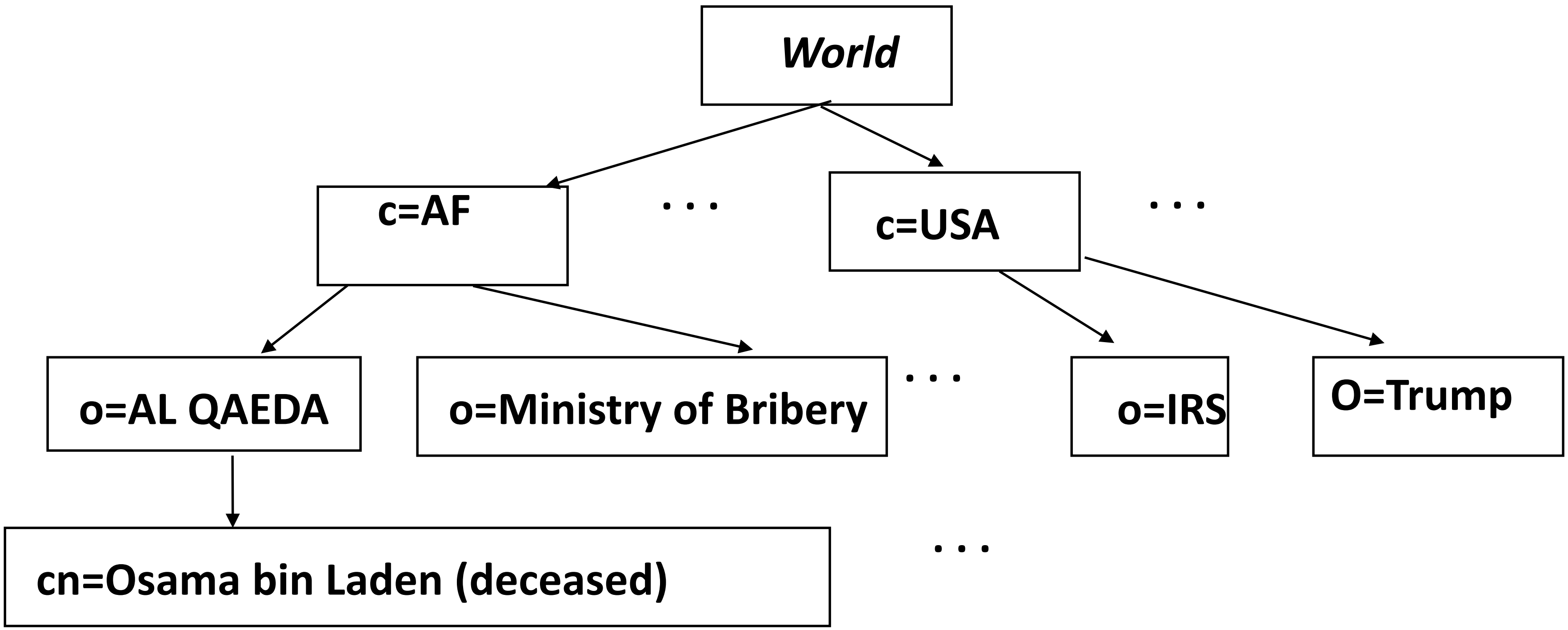
- ISO standard for directory services
- Global, distributed
- First solid version in 1988. (second in 1993.)
- Documentation - several Internet Standard Request for Comments (RFC)

X.500

- Data Model:
  - Based on hierarchical namespace
  - Directory Information Tree (DIT)
  - Geographically organized
  - Entry is defined with its **DN** (Distinguished Name)

- Searching:
  - You must select a location in DIT to base your search
  - A one-level search or a subtree search
  - Subtree search can be slow

X.500 - DIT



dn: cn=Osama bin Laden, o=Al Qaeda, c=AF

X.500

- Accessible through:
  - Telnet (client programs known as dua, dish, ...)
  - WWW interface

- Hard to use and very heavy …

  - … thus LDAP was developed

LDAP

- LDAP - **Lightweight Directory Access Protocol**
- LDAP v2 - RFC 1777, RFC 1778
- LDAP v3 - RFC 1779
- developed to make X.500 easier to use
- provides basic X.500 functions
- referral model instead original chaining
  - server informs client to ask another server (without asking question on the behalf of client)
- LDAP URL format:
  - ldap://server_address/dn
- (ldap://ldap.uci.edu/cn=Qi Alfred Chen,o=UCI,c=US)

Some Relevant Standards

- The IETF Reference Site
  - http://ietf.org/html.charters/wg-dir.html#Security_Area
- Public-Key Infrastructure (X.509, PKIX)
  - RFC 2459 (X.509 v3 + v2 CRL)
- LDAP v2 for Certificate and CRL Storage
  - RFC 2587
- Guidelines & Practices
  - RFC 2527
- S/MIME v3
  - RFC 2632 & 2633
- TLS 1.0 / SSL v3
  - RFC 2246