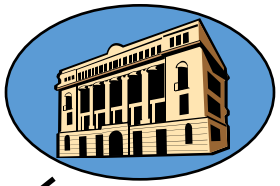
# Lecture 12

## Public Key Distribution (and Certifications)

## (Chapter 15 in KPS)

# A Typical KDC-based Key Distribution Scenario



KDC = Key Distribution Center

$E_K[X]$ = Encryption of X with key K

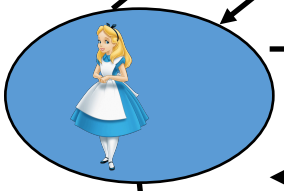
**(1) Request|B|$N_1$**

**(2) $E_{K_a}[K_s|\text{Request}|N_1|E_{K_b}(K_s,A)]$**

**(3) $E_{K_b}[K_s,A]$**

**(4) $E_{K_s}[A,N_2]$**

**(5) $E_{K_s}[f(N_2)]$**



Notes:

- Msg2 is tied to Msg1
- Msg2 is fresh/new
- Msg3 is possibly old *
- Msg1 is possibly old (KDC doesn't authenticate Alice)
- Bob authenticates Alice
- Bob authenticates KDC
- Alice DOES NOT authenticate Bob

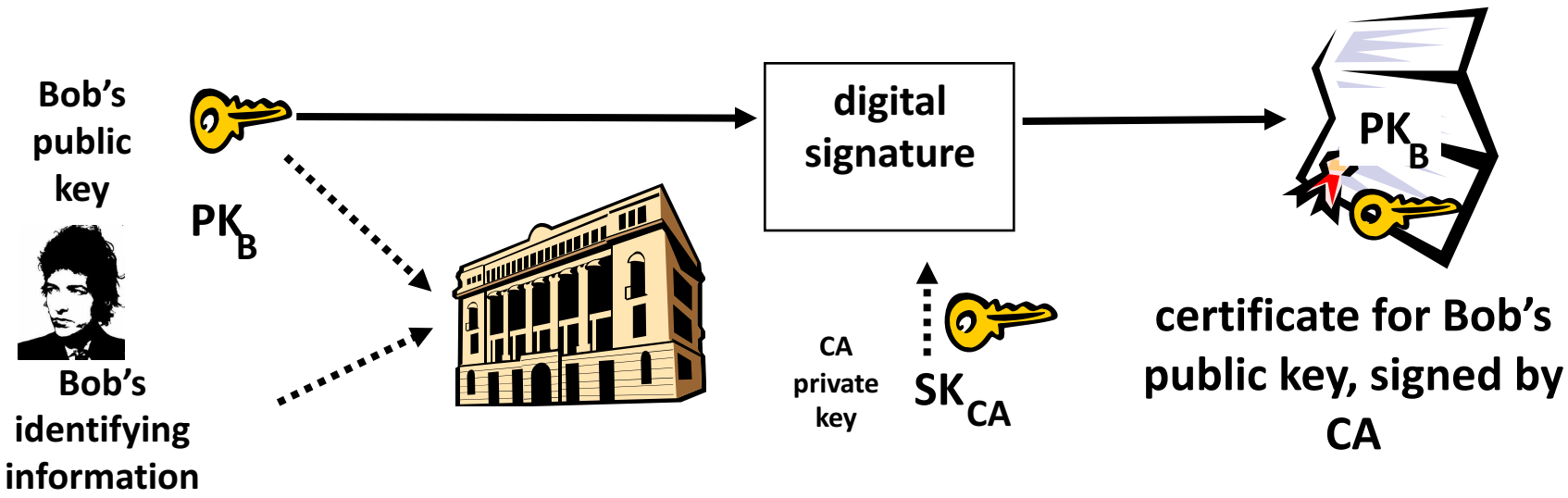
# Public Key Distribution

- General Schemes:
  - Public announcement (e.g., in a newsgroup or email message)
    - Can be forged
  - Publicly available directory
    - Can be tampered with
  - Public-key certificates (PKCs) issued by trusted off-line Certification Authorities (CAs)

# Certification Authorities

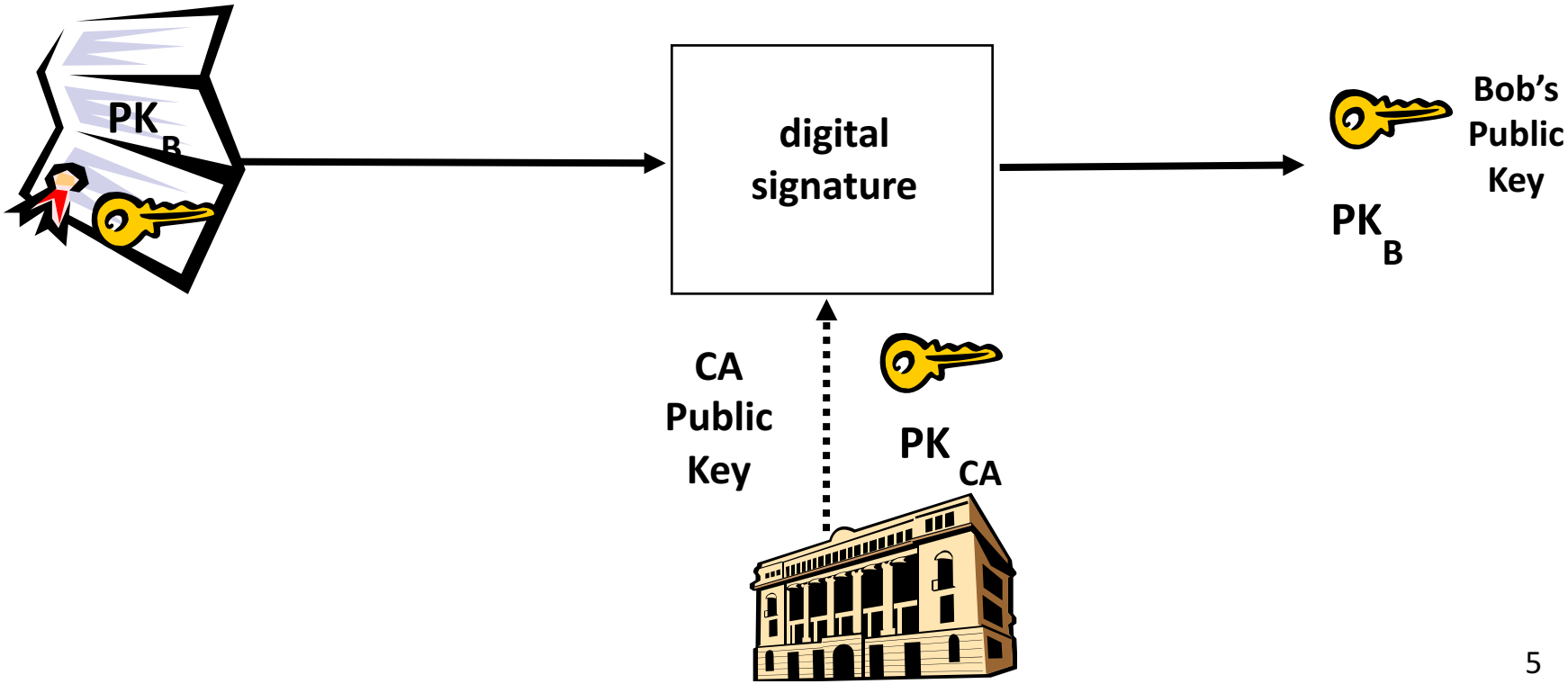
- Certification Authority (CA): binds public key to a specific entity
- Each entity (user, host, etc.) registers its public key with CA.
  - Bob provides "proof of identity" to CA.
  - CA creates certificate binding Bob to this public key.
  - Certificate containing Bob's public key digitally signed by CA:

**CA says: "this is Bob's public key"**



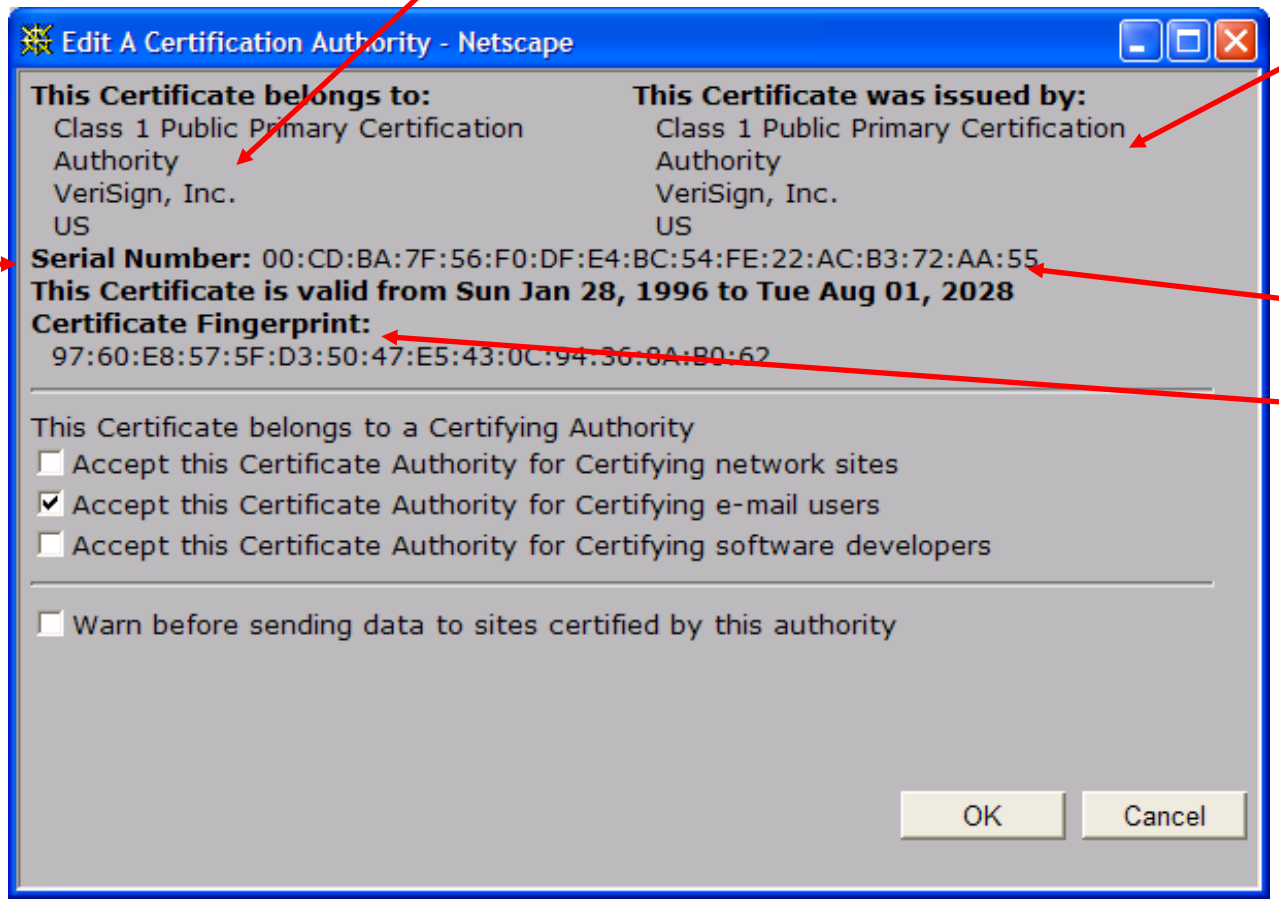
# Certification Authority

- **When Alice wants to get Bob's public key:**
  - **Get Bob's certificate (from Bob or elsewhere)**
  - **Using CA's public key verify the signature on Bob's certificate**
  - **Check for expiration**
  - **Check for revocation (we'll talk about this later)**
  - **Extract Bob's public key**



# A Certificate Contains

- Serial number (unique to issuer)
- Info about certificate owner, including algorithm and key value itself (not shown)




- info about certificate issuer
- valid dates
- digital signature by issuer

# Reflection Attack and a Fix

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

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

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
  1. $E \rightarrow B :$ $A, r_A$
  2. $B \rightarrow E :$ $r_B, \{ r_B, r_A, A \}_{SK_B}$
  3. $E \rightarrow A :$ $B, r_B$
  4. $A \rightarrow E :$ $r_A', \{ r_A', r_B, B \}_{SK_A}$
  5. $E \rightarrow B :$ $r_A', \{ r_A', r_B, B \}_{SK_A}$

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

# x.509 Authentication & Key Distribution Protocols

One-way
A→B

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

Two-way
A→B

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

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

Tree-way
A←→B

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

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

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

# 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!
- The problem of verifying security gets much harder as protocols get more complex (more parties, messages, rounds).

# Merkle's Puzzles (1974)

$0 < i < 2^n = N$

$X_i, Y_i$ -- *random secret keys*

$index_i$ = *random (secret) value*

*Puzzle* $P_i = \{index_i, X_i, S\}^{Y_i}$

$S$ -- *fixed string, e.g., "Alice to Bob"*

$\{P_i \mid 0 < i < 2^n\}$

*Pick random* $j$, $0 < j < 2^n$

*Select* $P_j$

$index_j$

*Break* $Y_j$ *by brute force*

*Look up* $index_j$

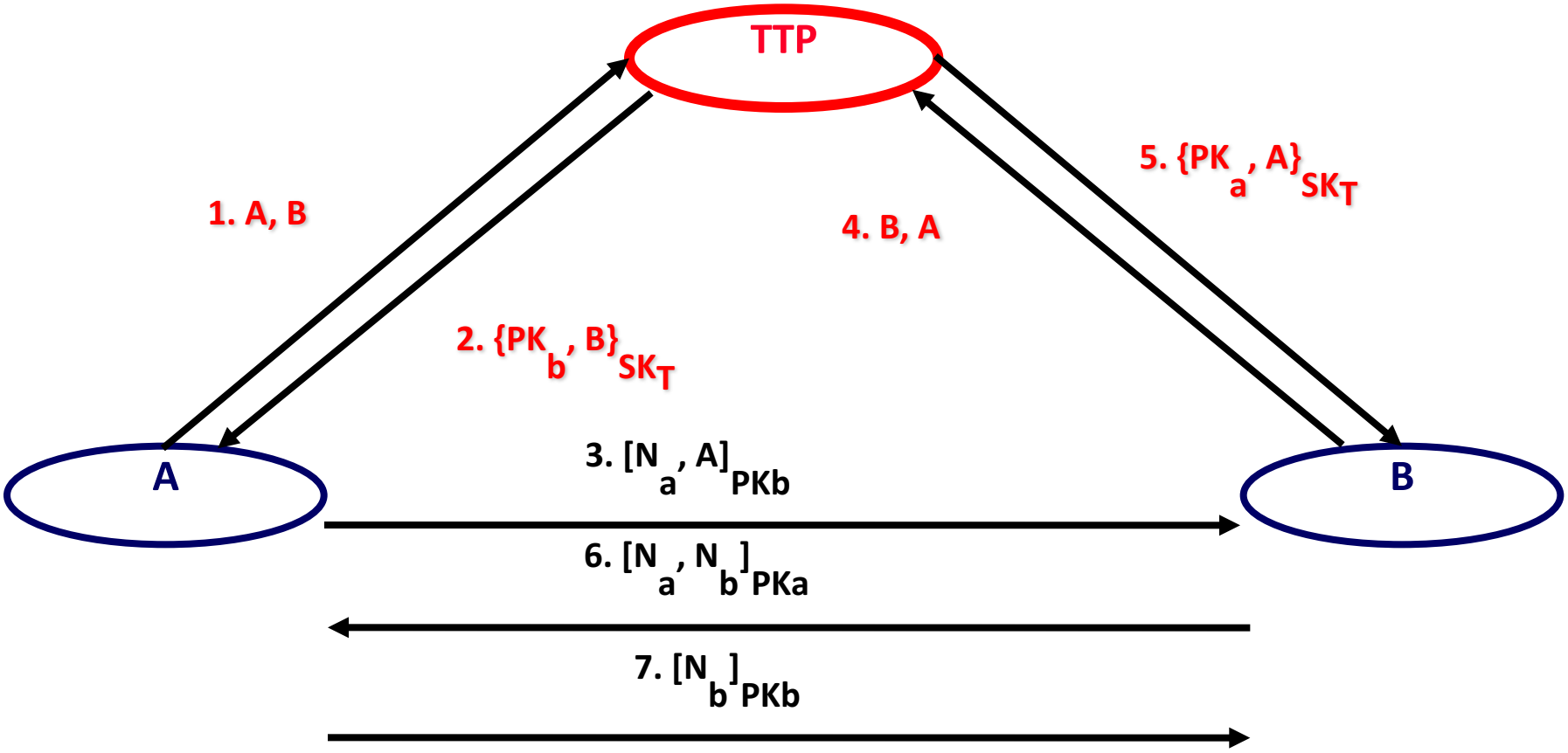
*Obtain* $\{index_j, X_j, S\}$

*Obtain* $X_j$

Encrypted communication with $X_j$


Is security computational or information theoretic?

# 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 nonsensical (bogus) key
- What happens in each case?

# Public Key Distribution

- Diffie - Hellman (1976) proposed the "public file" concept
  - universally accessible
  - no unauthorized modification
  - not scalable!

# 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 done off-line

# 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

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

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

$CERT_{alice}, SIG_{alice}$

# Certificates

- Procedure
  - Bob registers at local CA
  - Bob receives his certificate:

    { $PK_B$, IDB, issuance_time, expiration_time, etc.,...}$SK_{CA}$

  - Bob sends certificate to Alice
  - Alice verifies CA's signature
    - $PK_{CA}$ hard-coded in software
  - Alice uses $PK_B$ for encryption and/or verifying signatures

# Who Issues Certificates?

- CA: **Certification Authority**
  - e.g., GlobalSign, VeriSign, Thawte, etc.
  - look into your browser ...

- Trustworthy (at least to its users/clients)
- *Off-line operation (usually)*
- 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)
  - Good idea to prove knowledge of private key as part of the certificate request. Why?
- Public key and owner's name are usually part of a certificate

- Private keys only used for small amount of data (signing, encryption of session keys)
- Symmetric keys (e.g., RC5, 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 its name is "well-known."
- CA is a critical part of the system and must operate in a secure and predictable way according to some policy.

# Who needs them?

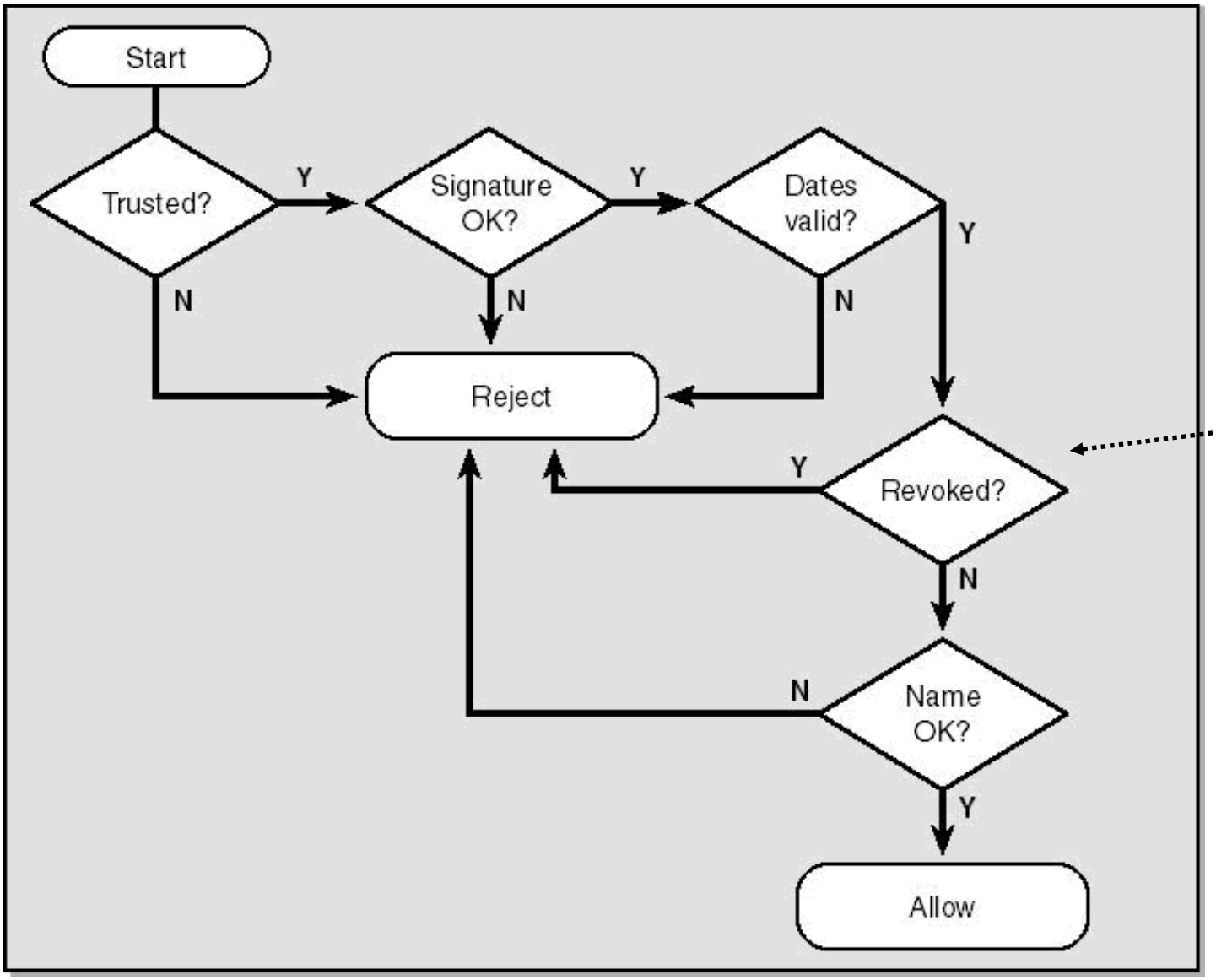
- Alice's certificate is checked by whomever wants to: 1) verify her signatures, and/or 2) encrypt data for her.

- 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

# BTW:

- Certificate Types
  - PK (Identity) certificates
    - Bind PK to some identity string
  - Attribute certificates
    - Bind PK to arbitrary attribute information, e.g., authorization, group membership
- We concentrate on former

# 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!
- Encrypting files (EFS in Windows)
- 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 not processed in real time
- In addition, using tamper-resistant hardware for the CA would help (should be impossible to extract private key)

# Mapping Personal Certificates into Accounts/Names

- Certificate must map “one-to-one” into an account/name for the sake of authentication
- In some systems, mapping are based upon X.509 naming attributes from the Subject field
- Example: Verisign issues certificate as CN=Full Name (account)
- Account/name is local to the issuing domain

# Storage of Private Key

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

# Key Lengths

- Strong encryption has been adopted since the relaxation of US export laws
- E.g., 512- and 1024-bit RSA is not safe anymore
- Root CA should have an (RSA) key length of >= 2048 bits given its importance and typical lifetime of 3-5 years
- A personal (RSA) certificate should have key length of at least 1536 bits

# Key Lengths

January 2016 Recommendation from National Security Agency (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!

- Cannot have certificates without a comprehensive naming scheme
- Cannot have PKI without a comprehensive distribution/access method
- X.509 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

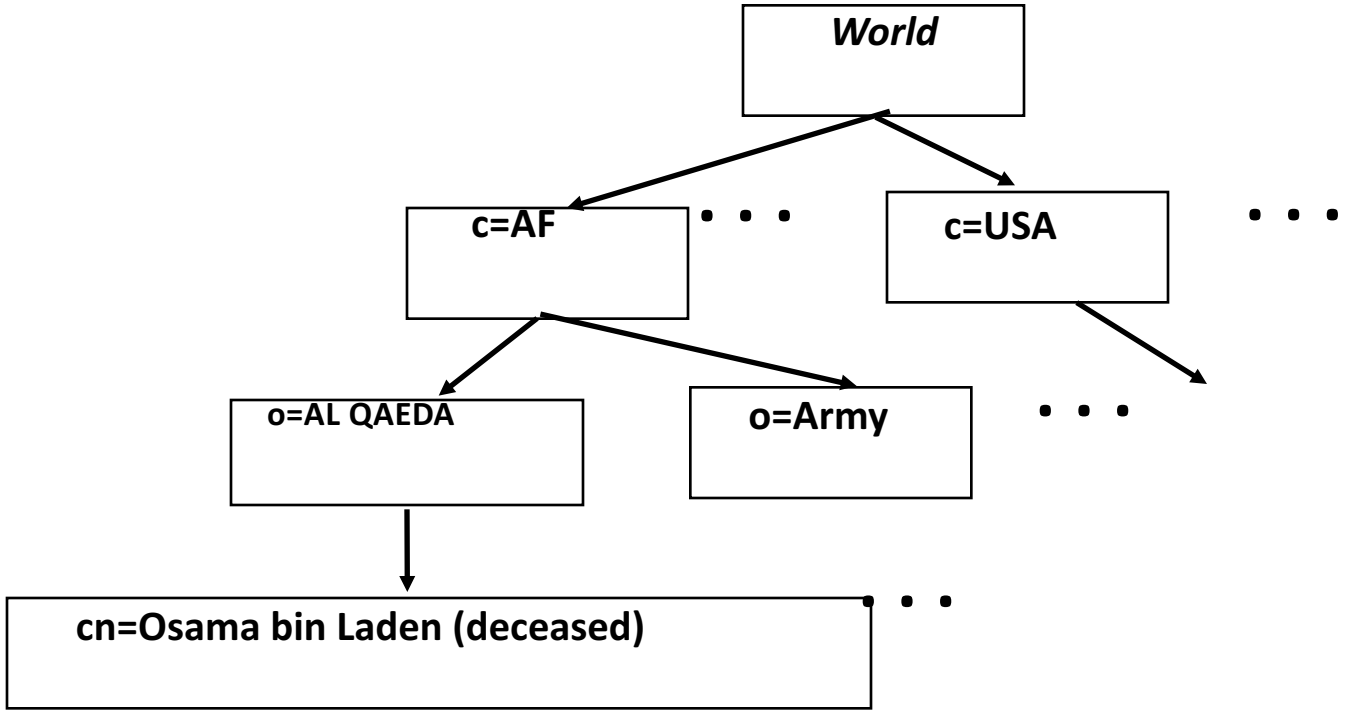
- ISO standard for directory services
- Global, distributed
- First solid version in 1988. (second in 1993.)
- Documentation - several Internet Standard Request for Comments (RFC)

# 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
  - For example: http://www.dante.net:8888/
- 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=Kasper Rasmussen,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