# Computer and Network Security

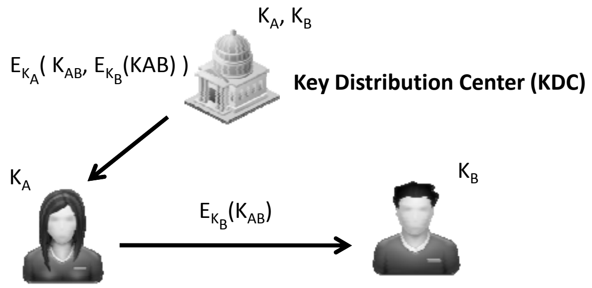
Lecture 10

Certificates and Revocation

# Outline

- Key Distribution
- Certification Authorities
- Certificate revocation

# Key Distribution


$K_A, K_B$
$E_{K_A}( K_{AB}, E_{K_B}(KAB) )$
**Key Distribution Center (KDC)**
$K_A$
$K_B$
$E_{K_B}(K_{AB})$

- KDC knows user secret keys
- What if…
  - Alice and Bob have no (mutually) trusted KDC
  - and / or have no online KDC

# Public Key Infrastructure

- 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?

# Diffie-Hellman

- Diffie-Hellman (1976) proposed the “public file” concept
  - universally accessible
  - no unauthorized modification
  - poor idea → not scalable!

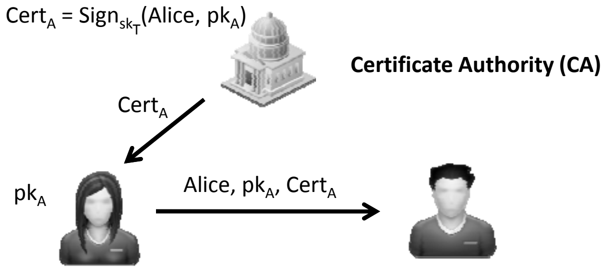
# Popek-Kline

- Popek-Kline (1979) proposed “trusted third parties” (TTPs)
  - TTPs know public keys of the entities and distribute them on-demand basis
  - on-line protocol (a disadvantage)

# Kohnfelder

- Kohnfelder (BS Thesis, MIT, 1978) proposed "certificates" as yet another public-key distribution method
- Explicit binding between the public-key and its owner/name
- Issued (digitally signed) by the Certificate Authority (CA)
- Issuance is done off-line

# Certificates



$\text{Cert}_A = \text{Sign}_{sk_T}(\text{Alice}, pk_A)$

**Certificate Authority (CA)**

$\text{Cert}_A$

$pk_A$

Alice, $pk_A$, $\text{Cert}_A$

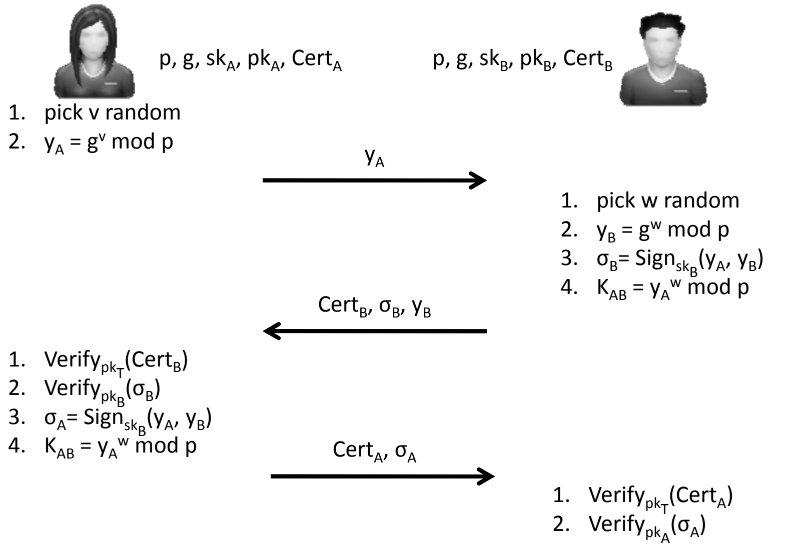
- User are issued certificates
  - Offline
- CA does not know user secret key
  - It only certifies (binds) identities and public keys

# Certificates

- Procedure
  - Alice registers at local CA
  - Alice receives her certificate:

    {σ, $pk_A$, $ID_A$, issuance_time, expiration_time, ...}
    σ = $Sign_{pk_T}${$pk_A$, $ID_A$, issuance_time, expiration_time, ...}
  - Alice sends her certificate to Bob
  - Bob verifies CA's signature on the certificate
    - $pk_T$ hard-coded in software (browser)
  - Bob uses $pk_A$ for encryption and/or verifying signatures with Alice

# Station-to-station protocol Authenticated PK-based key exchange



p, g, $sk_A$, $pk_A$, $Cert_A$

p, g, $sk_B$, $pk_B$, $Cert_B$

1. pick v random
2. $y_A = g^v \bmod p$

$y_A$ →

1. pick w random
2. $y_B = g^w \bmod p$
3. $\sigma_B = Sign_{sk_B}(y_A, y_B)$
4. $K_{AB} = y_A^w \bmod p$

← $Cert_B$, $\sigma_B$, $y_B$

1. $Verify_{pk_T}(Cert_B)$
2. $Verify_{pk_B}(\sigma_B)$
3. $\sigma_A = Sign_{sk_B}(y_A, y_B)$
4. $K_{AB} = y_A^w \bmod p$

$Cert_A$, $\sigma_A$ →

1. $Verify_{pk_T}(Cert_A)$
2. $Verify_{pk_A}(\sigma_A)$

# Who issues certificates?

- Certification Authority
  - e.g. GlobalSign, VeriSign, Thawte, etc.
  - look into your browser...
- Trustworthy (at least to its users/clients)
- Off-line operation (usually)
- Has a well-known long-term certificate
- Very secure: physically and electronically

# How does it work? 1/2

- A public/private key-pair is generated by user
- User requests certificate via local application (e.g., web browser)
  - Good idea to prove knowledge of private key as part of the certificate request. Why?
- Public key and “name” usually part of a PK 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

# How does it work? 2/2

- CA checks that requesting user is who he claims to be (in the certificate request)
- CA's own certificate  is signed by a higher-level
- Root CA's certificate is self-signed and his identity/name is "well-known"

Spain CA (X) Madrid CA (Y) UPM CA (W) ALICE (A)

$\text{Cert}_Y=\text{Sign}_{sk_X}(Y, pk_Y, \ldots)$ $\text{Cert}_W=\text{Sign}_{sk_Y}(W, pk_W, \ldots)$ $\text{Cert}_A=\text{Sign}_{sk_W}(A, pk_A, \ldots)$

$\text{Cert}_X=\text{Sign}_{sk_X}(X, pk_X, \ldots)$

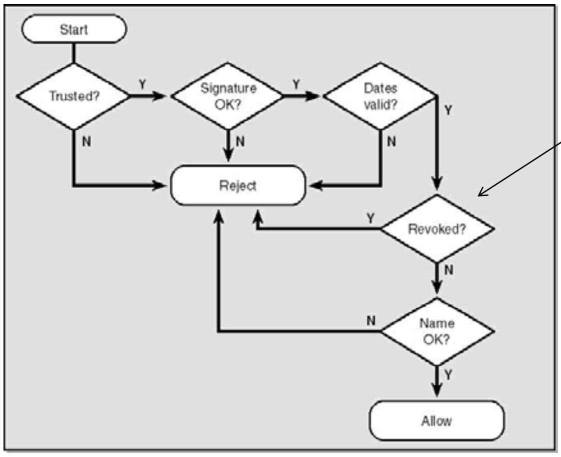
# Who needs (Alice's) certificate

- Any party wishing to
  - Send encrypted messages to Alice
  - Verify signature issued by Alice

- A verifier must
  - Know the public key(s) of the CA(s)
  - Trust all CA(s) involved
  - Verify signature and "validity"

- Validity
  - Expiration date > Signing date
  - Revocation checking = FAIL

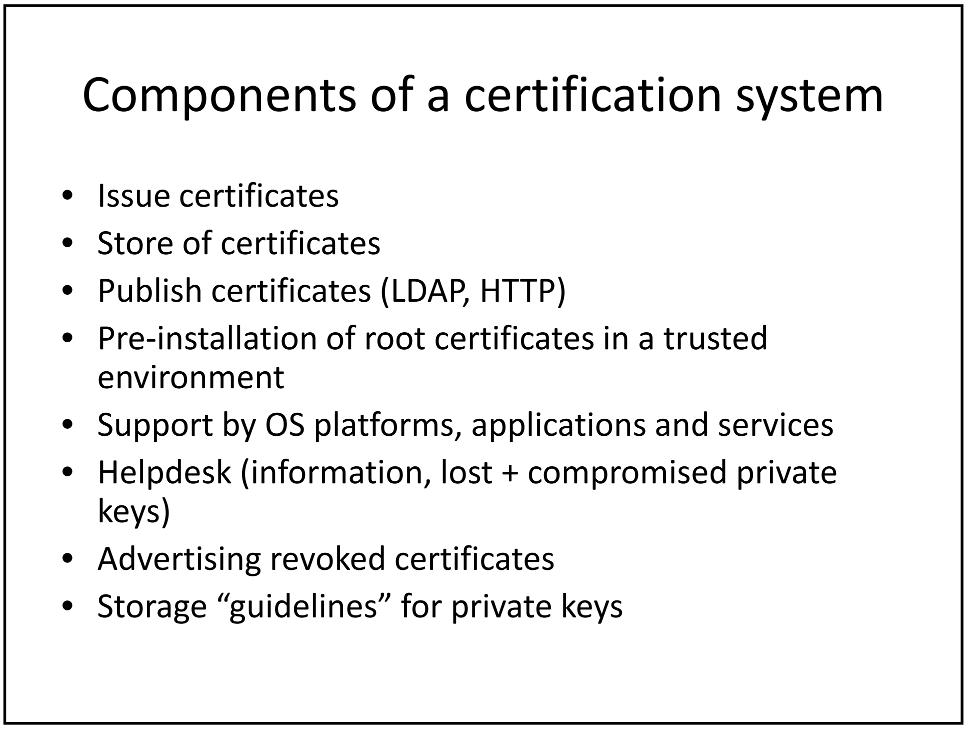
# Certificate verification



To be covered…

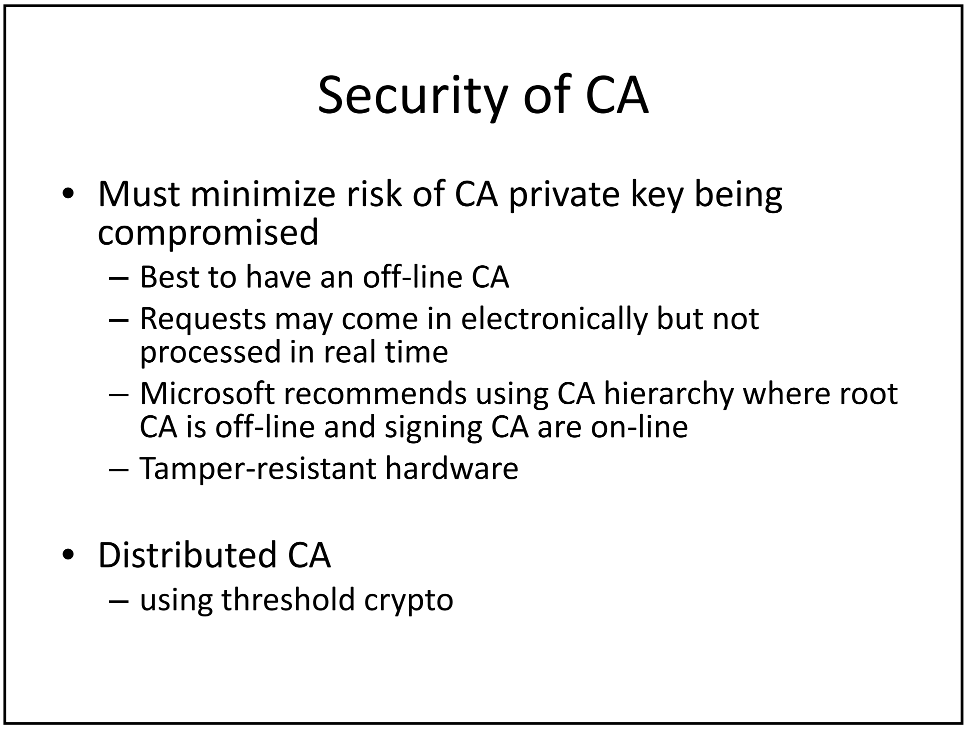
# Certificate applications

- 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/2000)
- IPSec: encryption/authentication at the network layer



## Components of a certification system

- Issue certificates
- Store of certificates
- Publish certificates (LDAP, HTTP)
- Pre-installation of root certificates in a trusted environment
- Support by OS platforms, applications and services
- Helpdesk (information, lost + compromised private keys)
- Advertising revoked certificates
- Storage “guidelines” for private keys



## Security of CA

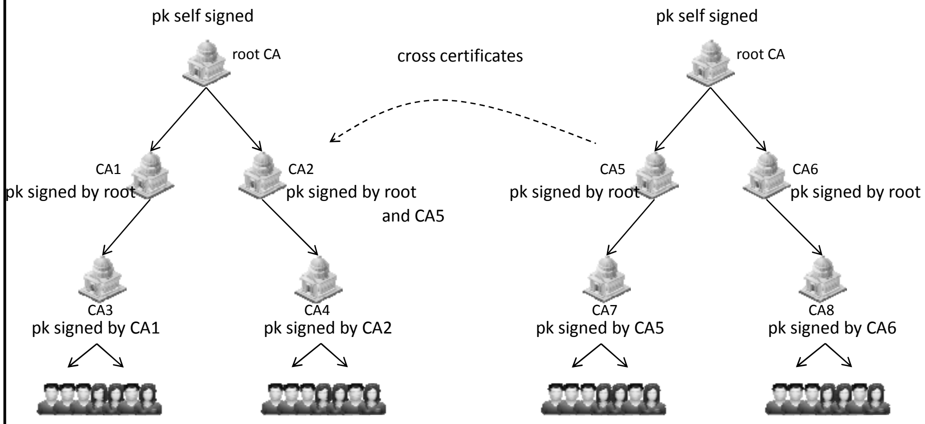
- Must minimize risk of CA private key being compromised
  – Best to have an off-line CA
  – Requests may come in electronically but not processed in real time
  – Microsoft recommends using CA hierarchy where root CA is off-line and signing CA are on-line
  – Tamper-resistant hardware
- Distributed CA
  – using threshold crypto

# Key Lengths

- Strong encryption has been adopted since the relaxation of US export laws
- 512-bit RSA and 56-bit DES are not safe
- Root CA should have an (RSA) key length of ≥ 2048 bits
  - 3-to-5 years lifetime
- A personal (RSA) certificate should have key length of ≥ 1536 bits

- Security requirements are constantly increasing!

# Certificate hierarchy



# Revocation

- Certificate have expiration date

- What if
  - Bob’s CA goes berserk?
  - Bob forgets his private key?
  - Someone steals Bob’s private key?
  - Bob looses his private key?
  - Bob willingly discloses his private key?
    - Eve can decrypt/sign while Bob’s certificate is still valid…
    - Bob reports key loss to CA (or CA finds out somehow)

- CA issues a Certificate Revocation List (CRL)
  - Distributed in public announcements
  - Published in public databases

- When verifying Bob’s signature or encrypting a message for Bob, Alice first checks if Bob’s certificate is still valid!

# Generally, certificate = capability

- Certificate revocation needs to occur when
  - certificate holder key compromise/loss
  - CA key compromise
  - early end of contract
- Certificate Revocation Lists (CRLs) hold the list of certificates that are not yet naturally expired but revoked
  - Reissued periodically (even if no activity!)
  - More on revocation later…

## Requirement for revocation

- Timeliness
  - Must check most recent revocation status
- Efficiency
- Computation
- Bandwidth and storage
- Availability
- Security

## Types of Revocation

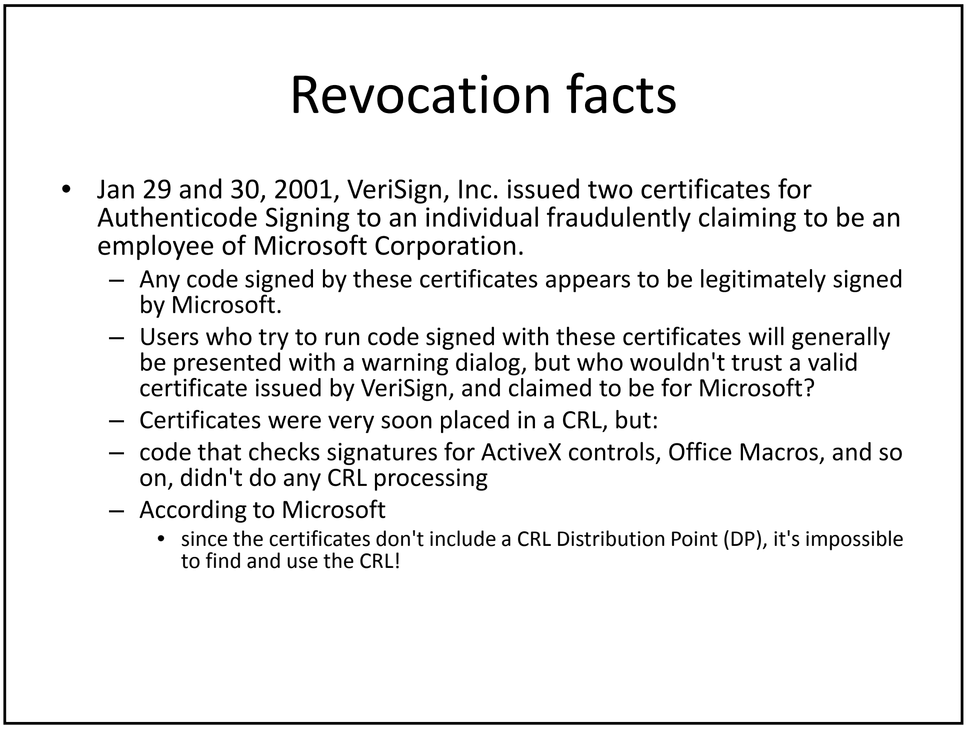
- Implicit
  - Each certificate is periodically re-issued
  - Alice has a fresh certificate → Alice not revoked
  - No need to distribute/publish revocation info
- Explicit
  - Only revoked certificates are periodically announced
  - Alice’s certificate not listed among the revoked onse → Alice not revoked
  - Need to distribute/publish revocation info

# Revocation Methods

- CRL - Certificate Revocation List
  - CRL-DP, indirect CRL, dynamic CRL-DP,
  - delta-CRL, windowed CRL, etc.
  - CRT and other Authenticated Data Structures
- OCSP – On-line Certificate Status Protocol
- CRS  - Certificate Revocation System

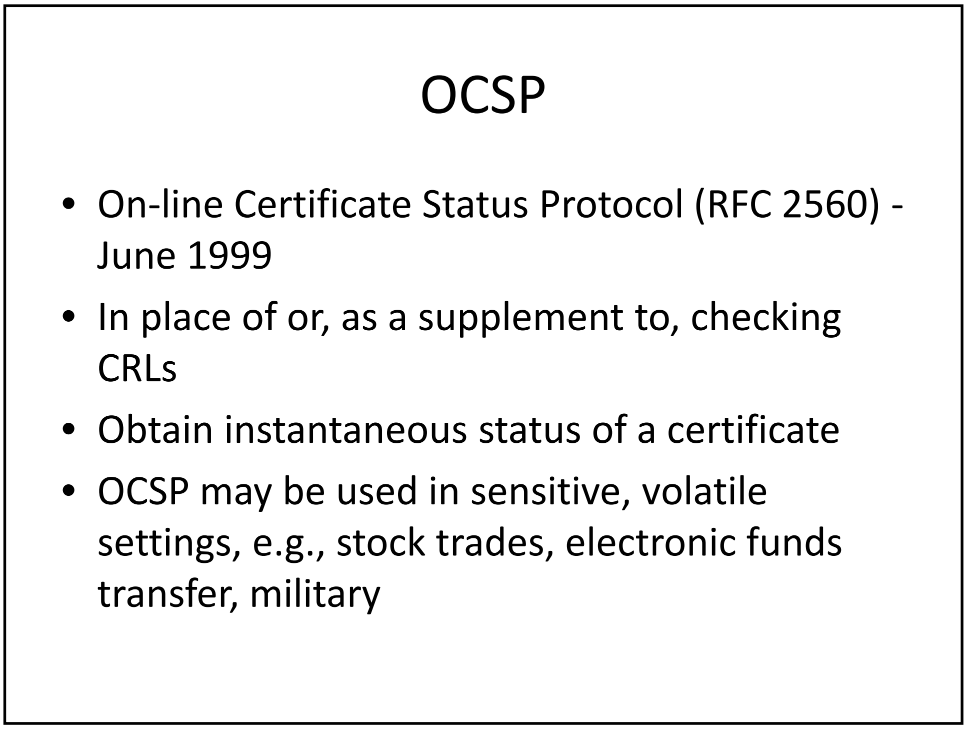
# CRL

- Off-line mechanism
- CRL = list of revoked certificates (e.g., SNs) signed by a revocation authority (RA)
- RA not always CA that issued the revoked certificates
- Periodically issued: daily, weekly, monthly, etc.
- Pros
  - Simple
  - Don’t need secure channels for CRL distribution
- Cons
  - Timeliness: “window of vulnerability”
  - CRLs can be huge



# Revocation facts

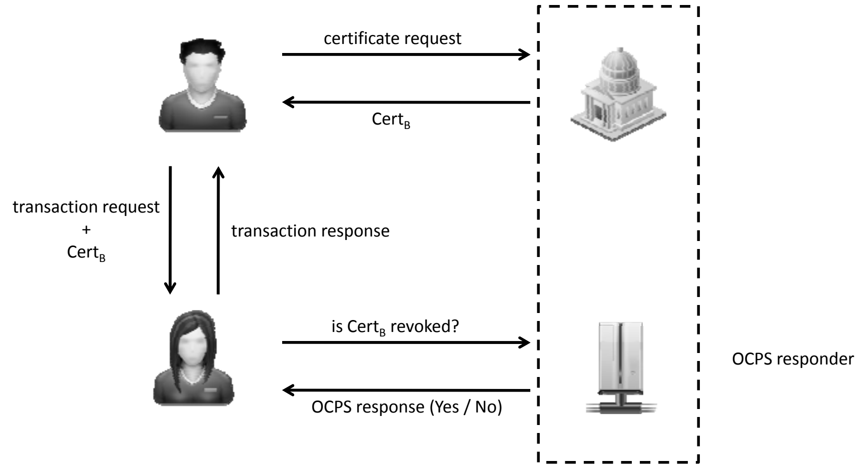
- Jan 29 and 30, 2001, VeriSign, Inc. issued two certificates for Authenticode Signing to an individual fraudulently claiming to be an employee of Microsoft Corporation.
  - Any code signed by these certificates appears to be legitimately signed by Microsoft.
  - Users who try to run code signed with these certificates will generally be presented with a warning dialog, but who wouldn't trust a valid certificate issued by VeriSign, and claimed to be for Microsoft?
  - Certificates were very soon placed in a CRL, but:
  - code that checks signatures for ActiveX controls, Office Macros, and so on, didn't do any CRL processing
  - According to Microsoft
    - since the certificates don't include a CRL Distribution Point (DP), it's impossible to find and use the CRL!



# OCSP

- On-line Certificate Status Protocol (RFC 2560) - June 1999
- In place of or, as a supplement to, checking CRLs
- Obtain instantaneous status of a certificate
- OCSP may be used in sensitive, volatile settings, e.g., stock trades, electronic funds transfer, military

# OCSP Players



certificate request

$\text{Cert}_B$

transaction request
+
$\text{Cert}_B$

transaction response

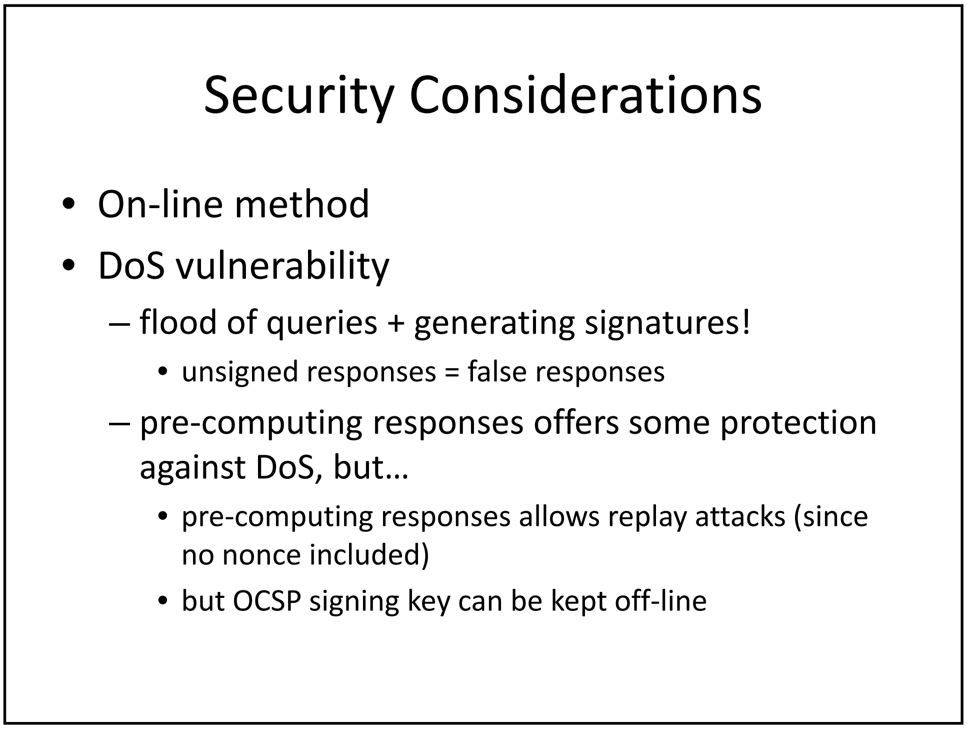
is $\text{Cert}_B$ revoked?

OCPS response (Yes / No)

OCPS responder

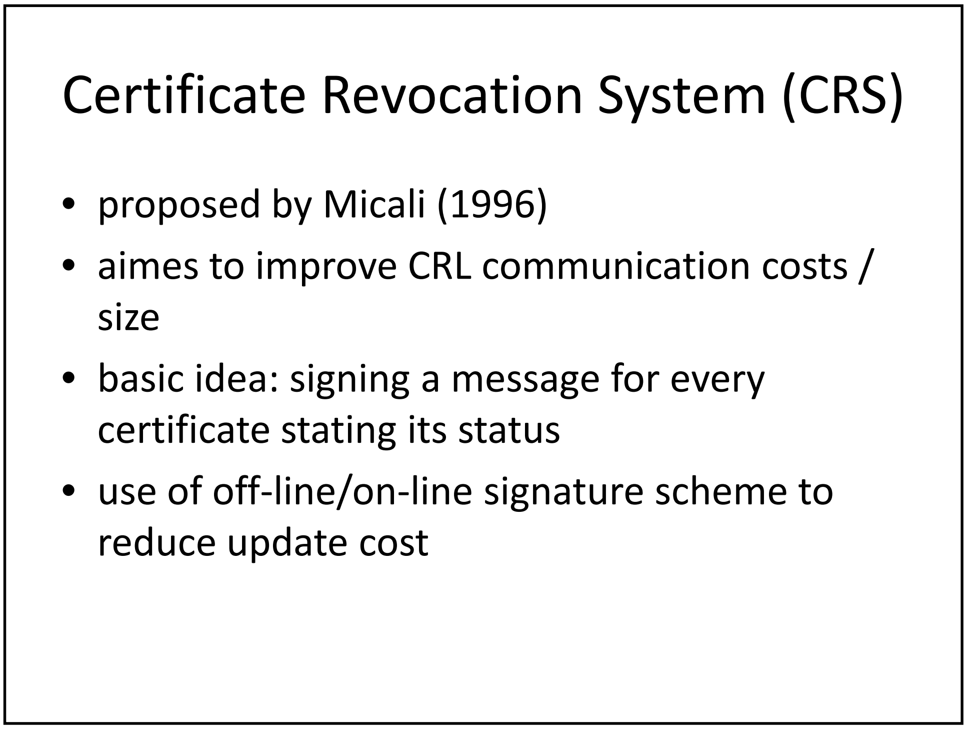
# Who signs OCPS responses?

- The CA
  - Has to be online
- Trusted OCPS responder
  - Authorized by the CA
  - Has a special certificate that says
    - “Responder can sign OCPS responses for Certificates issued by CA”



## Security Considerations

- On-line method
- DoS vulnerability
  - – flood of queries + generating signatures!
    - unsigned responses = false responses
  - – pre-computing responses offers some protection against DoS, but…
    - pre-computing responses allows replay attacks (since no nonce included)
    - but OCSP signing key can be kept off-line



## Certificate Revocation System (CRS)

- proposed by Micali (1996)
- aimes to improve CRL communication costs / size
- basic idea: signing a message for every certificate stating its status
- use of off-line/on-line signature scheme to reduce update cost

## CRS: Creation of a certificate

- Two new parameters in Cert: $Y_{MAX}$ and N
  - $Y_{MAX} = H_{MAX}(Y_0)$
  - $N=H(N_0)$
- $Y_0$, $N_0$
  - unique per certificate
  - securely stored at the CA
- H()
  - public one-way function

## CRS: creation of a certificate

- Two new parameters in PKC: $Y_{MAX}$ and N

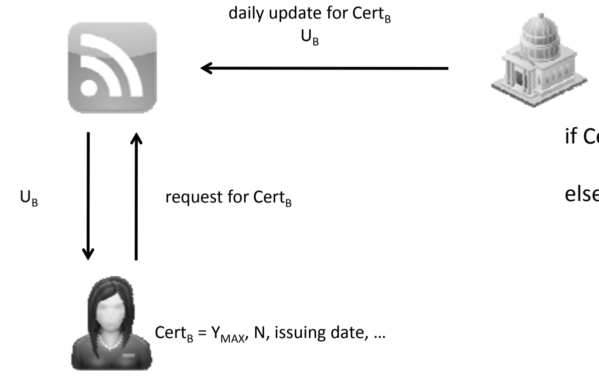
$$Y_{MAX} = H^{MAX}(Y_0)$$

$$N=H(N_0)$$

- [ $Y_0$, $N_0$ ] -- per-PKC secrets stored by CA
- H() -- public one-way function

# How CRS works

1. MAX = 365
2. at the issuance date, i=0



daily update for $\text{Cert}_B$

$U_B$

$U_B$

request for $\text{Cert}_B$

$\text{Cert}_B = Y_{MAX}$, N, issuing date, …

if $\text{Cert}_B$ is valid
    $U_B = Y_{365-i} = H^{365-i}(Y_0)$
else
    $U_B = N_0$

```
if $H(U_B)=N$
    $\text{Cert}_B$ is not valid
elseif $H^i(U_B)=Y_{MAX}$
    $\text{Cert}_B$ is valid
```

# Example, MAX = 3

$\text{Cert}_B = Y_3$, N

Pre-computed signatures:
$Y_1 = H(Y_0)$
$Y_2 = H(Y_1) = H(H(Y_0))$
$Y_3 = H(Y_2) = H(H(Y_1)) = H(H(H(Y_0)))$

Day 1    $U_B = Y_1$

Alice (max-i = 2)
    Is $H(U_B) = N$?
    Is $H^2(U_B) = Y_3$?

Day 2    $U_B = Y_2$

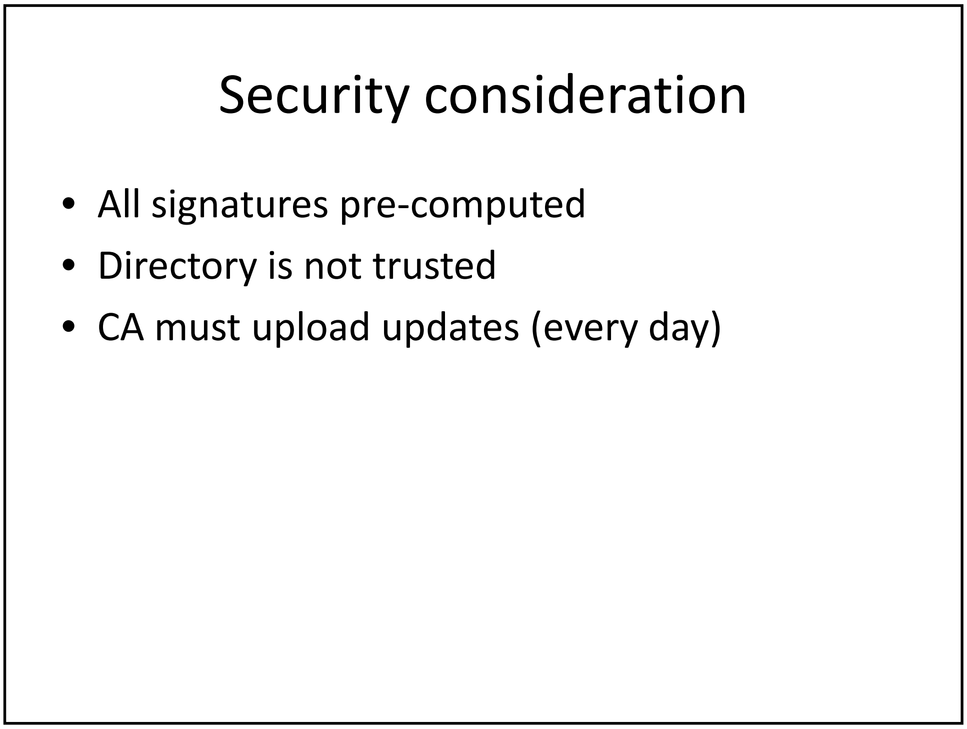
Alice (max-i = 1)
    Is $H(U_B) = N$?
    Is $H(U_B) = Y_3$?

Secret = $Y_0$, $N_0$

Pre-computed signatures:
$Y_1 = H(Y_0)$
$Y_2 = H(Y_1) = H(H(Y_0))$
$Y_3 = H(Y_2) = H(H(Y_1)) = H(H(H(Y_0)))$

Revocation token:
$N = H(N_0)$

- Since H one-way
  - Nobody can guess $N_0$
  - Nobody can guess $Y_j$ from $Y_1,\ldots,Y_{j-1}$



## Security consideration

- All signatures pre-computed
- Directory is not trusted
- CA must upload updates (every day)