Authentication based on public keys

public keys & X.509

Authentication using public key

- Idea: use signed messages containing challenges or timestamps; signatures can be verified using public key
- Problem: it is important to guarantee knowledge of public key. In fact

1. Alice wants to be authenticated by Bob;
2. Let K_{PT} Trudy's public key
3. If Trudy convinces B that the public key of A is K_{PT} then Trudy can be authenticated by Bob as Alice

Solution: Public Key Infrastructure: authority that guarantees correctness of public keys.

Authentication using public key Needham-Schroeder

K_{PX}: public key of X, Sig_C digital signature of C (trusted authority guarantees public keys)

Mutual authentication

1. A to C: <this is A, want to talk to B>
2. C to A: <B, K_{PB}, Sig_C(K_{PB},B)>
3. A checks digital signature of C, generates nonce N and sends to B: K_{PB}(N, A)
4. B decode (now wants to check A's identity) and sends to C: <B, A>
5. C to B: <A, K_{PA}, Sig_C(K_{PA}, A) >
6. B checks C's digital signature, retrieves K_{PA}, generates nonce N' and sends to A: K_{PA}(N, N')
7. A decodes, checks N, and sends to B: K_{PB}(N')

Attack to N.-S. public key

- Trudy is a system user that can talk (being authenticated) to A, B & C
- Two interleaved excerptions of the protocol:
  R1: authentication between A and T;
  R2: authentication between T (like A) with B
- Man in the middle attack
- *T must be able to induce A to start an authentication session with T*
- Steps 1, 2, 4, 5 allow to obtain public keys
- Steps 3, 6, 7 perform authentication

Attack to N.-S. public key

Steps 1, 2, 4 e 5 allow to know public keys
We focus on steps 3, 6, 7 of R1 and R2:

a) A-->T : step 3 of R1 sends $K_{PT}(N, A)$
b) T(like A)-->B: step 3 of R2 sends $K_{PB}(N, A)$
c) B-->T(like A): step 6 of R2 sends $K_{PA}(N', N)$
d) T-->A: step 6 of R1 sends $K_{PA}(N', N)$
e) A-->T: step 7 of R1 sends $K_{PT}(N')$
f) T(like A)-->B: step 7 of R2 sends $K_{PB}(N')$

B thinks that he is talking to A by sharing secret nonces!

Authentication using public key Needham-Schroeder (fixed)

K_{PX}: public key of X, Sig_C digital signature of C (trusted authority guarantees public keys)

Mutual authentication

1. A to C: <this is A, want to talk to B>
2. C to A: <B, K_{PB}, Sig_C(K_{PB},B)>
3. A checks digital signature of C, generates nonce N and sends to B: K_{PB}(N, A)
4. B decode (now wants to check A's identity) and sends to C: <B, A>
5. C to B: <A, K_{PA}, Sig_C(K_{PA}, A) >
6. B checks C's digital signature, retrieves K_{PA}, generates nonce N' and sends to A: K_{PA}(B, N, N')
7. A decodes, checks N, and sends to B: K_{PB}(N')

Why the previous attack fails

We focus on steps 3,6,7 of R1 and R2:

a) A-->T : step 3 of R1 sends $K_{PT}(N,A)$

b) T(like A)-->B: step 3 of R2 sends $K_{PB}(N,A)$

c) B-->T(like A): step 6 of R2 sends $K_{PA}(B, N', N)$

d) T-->A: BEFORE in step 6 of R1 T sends $K_{PA}(N', N)$; NOW T **CANNOT** send $K_{PA}(B, N', N)$ while is talking to A!!

e) A-->T: step 7 of R1 sends $K_{PT}(N')$

f) T(like A)-->B: step 7 of R2 sends $K_{PB}(N')$

X.509 Authentication standard

- Part of standard known as CCITT X.500
- Defined in 1988 and several times revised (until 2000)
  - version 3
- We need directory of public keys signed by certification authority
- Define authentication protocols (see for instance [Stallings2005], or https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201210-I!!PDF-E&type=items, p. 110)
  - One-Way Authentication
  - Two-Way Authentication
  - Three-Way Authentication
- Public key cryptography and digital signatures
  - Algorithms are not part of the standard (why?)

X.509 (one-way) authentication

Timestamp t_A

Session key K_{AB}

B's public key P_B

certA: certificate of A's public key, signed by certification authority

$A \rightarrow B : certA, D_A, Sig\_A(D_A)$

$D_A = <t_A, B, P_B(K_{AB})>$

One-way authentication (discussion)

Single transfer of information from A to B and establishes the following:

1. Identity of A and that message was generated by A
2. That message was intended for B
3. Integrity and originality (not sent multiple times) of message

- Message includes at least a timestamp t_A (a nonce could be included too) and identity of B and is signed with A's private key
  - Timestamp consists of (optional) generation time and expiration time. This prevents delayed delivery of messages.
  - Nonce can be used to detect replay attacks. Its value must be unique within the expiration time of the message. B can store nonce until message expires and reject any new messages with same nonce.
- For authentication, message used simply to present credentials to B
- Message may also include information, sgnData (not shown)
  - included within signature, guaranteeing authenticity and integrity.
- Message may also be used to convey session key to B, encrypted with B's public key

X.509 (two-ways) mutual authentication

Mutual authentication:

- $A \rightarrow B$

  $certA,\ D_A,\ Sig\_A(D_A)\ \ [D_A = \langle t_A,\ N,\ B,\ P_B(k)\rangle]$

  (how does A know P_B?)

- $B \rightarrow A$

  $certB,\ D_B,\ Sig\_B(D_B)\ [D_B = \langle t_B,\ N',\ A,\ N,\ P_A(k')\rangle]$

  (how does B know P_A?)

t_A, t_B = timestamps, to prevent delayed delivery of messages; k, k' session keys proposed by A and B; use of nonces avoids replay attacks

criticism: in D_A there is no identity of A - refer to the modified N.S. protocol

X.509 (three-ways) mutual authentication

Mutual authentication based on nonces, useful for unsynchronised clocks (0 denotes timestamp, optional)

three messages (A → B, B → A, A → B):

1. A → B: <certA, D_A, Sig_A(D_A)> [D_A = <0, N, B, P_B(k)>]
2. B → A: <certB, D_B, Sig_B(D_B)> [D_B= <0, N, A, N', P_A(k)>]
3. A → B: <B, Sig_A(N, N', B)>

Note: step 3 requires digital signature of nonces, making them tied (no replay attacks)

Challenge-response: ISO/IEC 9798-3 Mutual authentication (earlier version, bugged)

Why the following protocol does not work?

1. B to A: N_B
2. A to B: certA, N_A, N_B, B, Sig_A(N_A, N_B, B)
3. B to A: certB, N_B', N_A, A, Sig_B(N_B', N_A, A) [not predictable by A]

"Canadian" attack (from *Protocols for Authentication and Key Establishment*, C. Boyd and A. Mathuria, Springer 2003, p. 112)

1. T(B) to A : N_T
2. A to T(B): certA, N_A, N_T, B, Sig_A(N_A, N_T, B)
    1. T(A) to B: N_A
    2. B to T(A): certB, N_B, N_A, A, Sig_B(N_B, N_A, A)
3. T (B) to A: certB, N_B, N_A, A, Sig_B(N_B, N_A, A); T is authenticated!!

Note: use of N_B in step 3 (in place of N_B') has the same role as the use of Bob in step 6 of original N.-S. protocol

PKI: Public Key Infrastructure

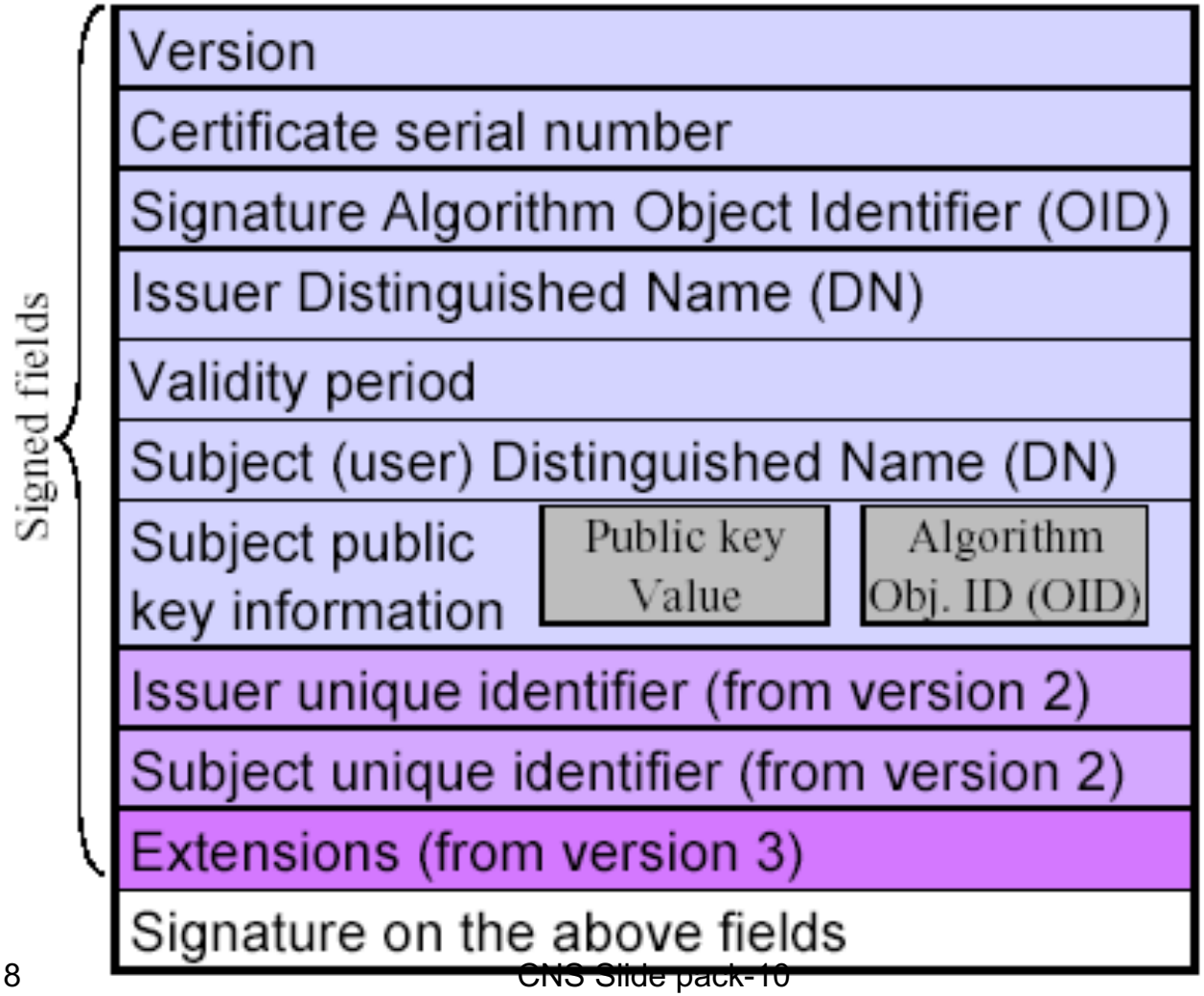
- Certificates are issued by a trusted Certification Authority (CA)
- The CA provides certificates of all users in domain
- When someone wants to know the public key of some user he/she asks the CA
  - CA provides user's public key, signing it by its own private key
- This implies it is sufficient to know one only public key (CA's public key)

Notice:

- If CA is not trusted, its certificates are useless
- Keys are not used forever, they are subject to changes
- Length of key is related to security level

X.509 Certificates

Certification authority CA guarantees public keys:



X.509 certificate's fields 1

- VERSION. There are currently three versions defined, version 1 for which the code is 0, version 2 for which the code is 1, and version 3 for which the code is 2.
- SERIALNUMBER. An integer that, together with the issuing CA's name, uniquely identifies this certificate.
- SIGNATURE. Specifies the algorithm used to compute the signature on this certificate. It consists of a subfield identifying the algorithm followed by optional parameters for the algorithm.
- ISSUER. The X.500 name of the issuing CA.

X.500 name

- X.500 names look like C=US, O=company name, OU=research, CN=Alice,
  where C means *country*, O means *organization*, OU means *organizational unit*, and CN is *common name*.
- There are rules about what types of name components are allowed to be under what others.
  - The encoding uses OIDs (Object IDentifiers) for each of the name component
- There is no standard for displaying X.500 names and different applications display them differently.

X.509 certificate's fields 2

- VALIDITY. This contains two subfields, the time the certificate becomes valid, and the last time for which it is valid.
- SUBJECT. The X.500 name of the entity whose key is being certified.
- SUBJECTPUBLICKEYINFO. This contains two subfields, an algorithm identifier, and the subject's public key.
- ISSUERUNIQUEIDENTIFIER. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the issuer of this certificate.

X.509 certificate's fields 3

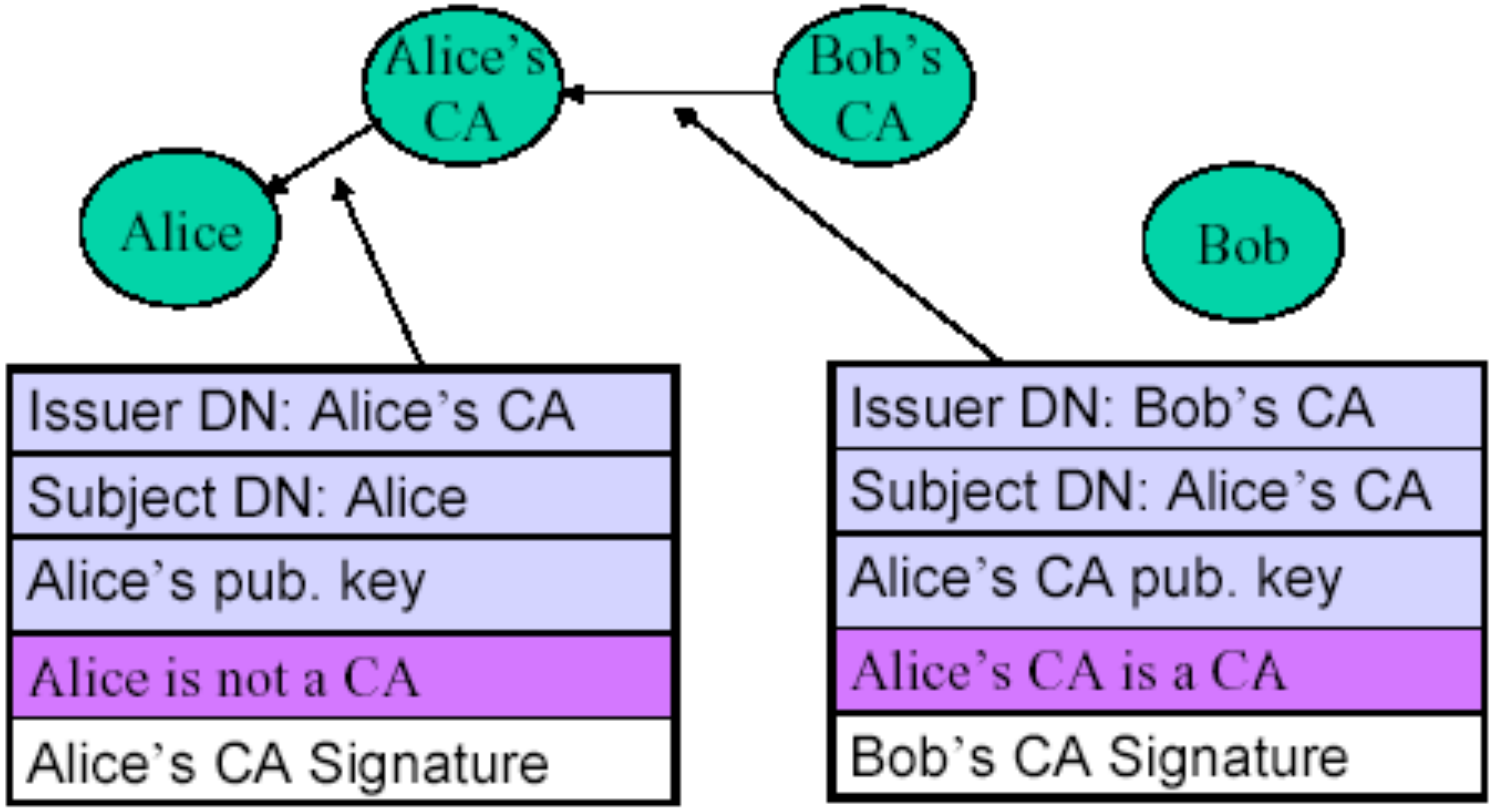
- SUBJECTUNIQUEIDENTIFIER. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the subject of this certificate.
- ALGORITHMIDENTIFIER. This repeats the SIGNATURE field. Redundant!
- EXTENSIONS. These are only in X.509 version 3. X.509 allows arbitrary extensions, since they are defined by OID.
- ENCRYPTED. This field contains the signature on all but the last of the above fields.

X.509 Certificates

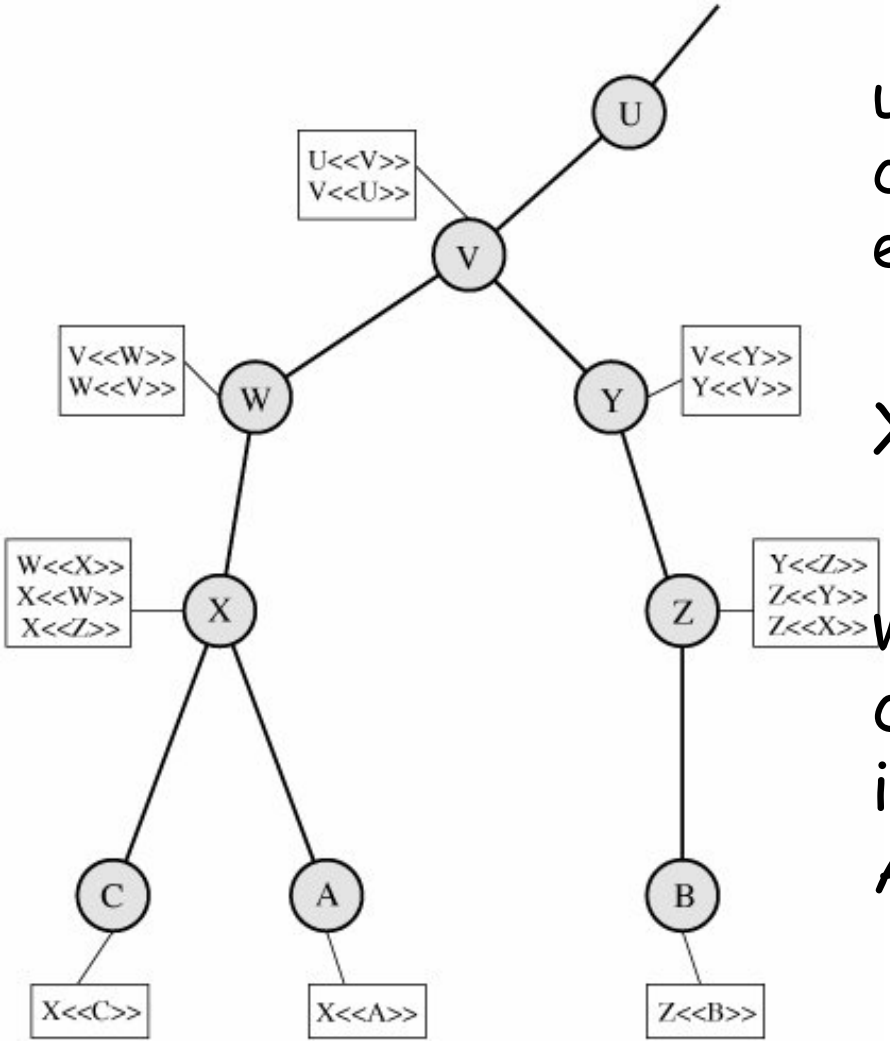
- They can be easily accessed
- Certificates are modified by CA
- Certificates impossible to falsify (RSA > 2000 bit)
- If Alice and Bob share the same CA then they can know each other Public Key
- Otherwise CA form a hierarchy

Hierarchy of CAs

- How Bob gets Alice's certificate if they refer to different CAs?



Hierarchy of CAs



user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W<<V>> V<<Y>> Y<<Z>> Z<<B>>

where A<<B>> means "the certificate of user B has been issued by certification authority A"

Certificate revocation

Certificates are valid for a limited time (CAs want to be paid)

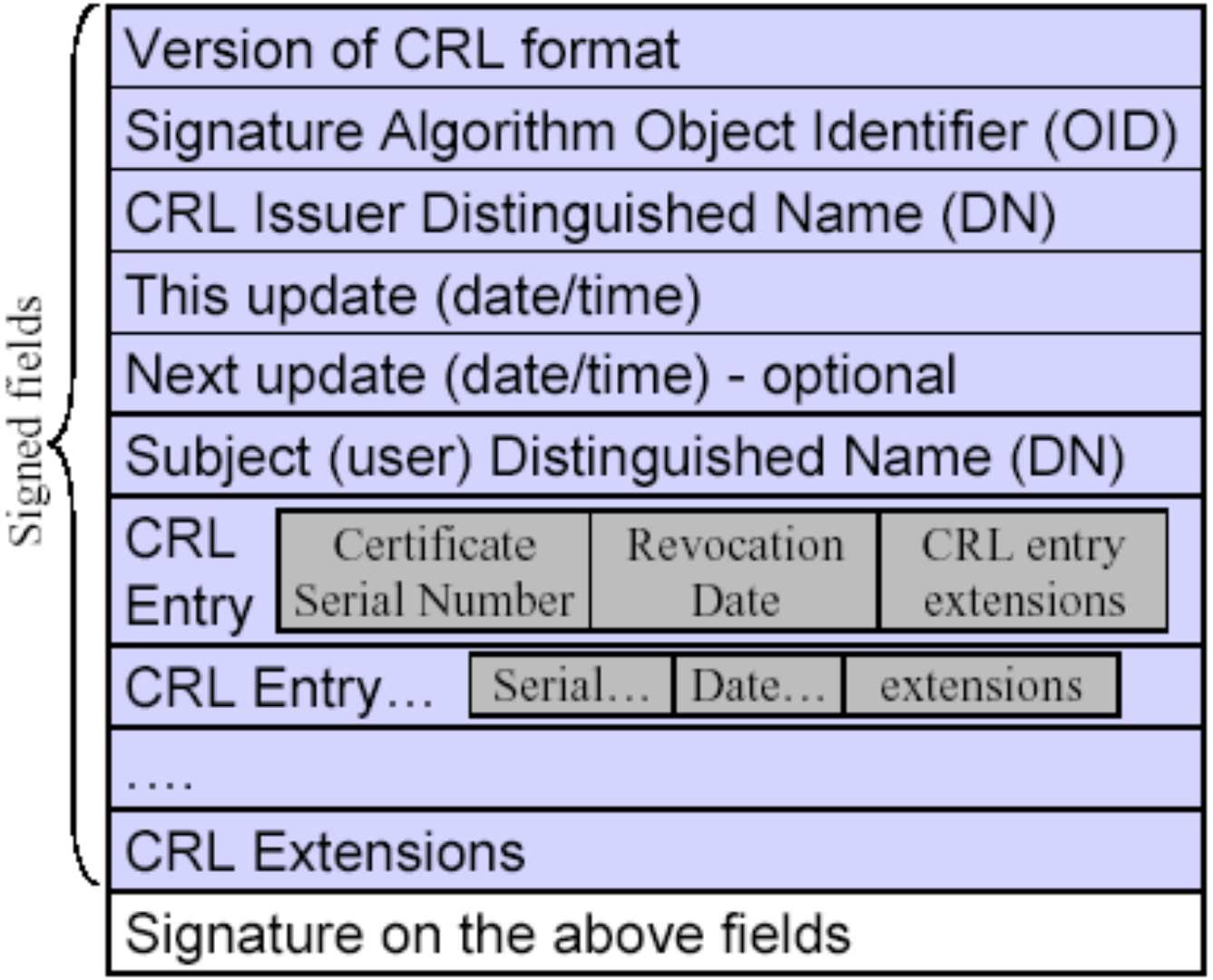
They can be revoked before the deadline:

1. User's secret key is not considered safe anymore
2. User is not certified by CA
3. CA's secret key is compromised

CRL: certificate revocation list

- Must be used before accessing a user

Certificate revocation



CRL's fields

- SIGNATURE. Identical to the SIGNATURE field in certificates, this specifies the algorithm used to compute the signature on this CRL.
- ISSUER. Identical to the ISSUER field in certificates, this is the X.500 name of the issuing CA.

CRL's fields

- THISUPDATE. This contains the time the CRL was issued.
- NEXTUPDATE. Optional. This contains the time the next CRL is expected to be issued. A reasonable policy is to treat as suspect any certificate issued by a CA whose current CRL has NEXTUPDATE time in the past.

CRL's fields

- The following three fields repeat together, once for each revoked certificate:
  - USERCERTIFICATE. This contains the serial number of the revoked certificate.
  - REVOCATIONDATE. This contains the time the certificate was revoked.
  - CRLENTRYEXTENSIONS. This contains various optional information such as a reason code for why the certificate was revoked.

CRL's fields

- CRLEXTENSIONS. This contains various optional information.
- ALGORITHMIDENTIFIER. As for certificates, this repeats the SIGNATURE field.
- ENCRYPTED. This field contains the signature on all but the last of the above fields.

X.509 Version 3

- In version 3 certificates have much more information:
  - email/URL, possible limitations in the use of the certificate
- Instead of adding fields for every possible new information define extensions
- Extensions:
  - Which kind of extension
  - Specification about the extension

OCSP

- Online Certificate Status Protocol used for obtaining the revocation status of an X.509 digital certificate (RFC 6960)
  - alternative to CRL, more agile
  - cert. status provided in TLS handshake (OCSP stapling: response on revocation check, signed by legit CA)
  - URL for check provided within the cert. (Certificate Extensions -> Authority Information Access, vers. 3 only)

PGP: Pretty Good Privacy

Trust model for E-mail certif. (Zimmerman)

- There are no trusted CA
- Each user acts like a CA and decides for himself
- Certificates contain email addresses and public keys
- Certificates are signed by one or many users
- If you trust a sufficient number of the user signing a certificate then you assume the certificate is good
- Each user keeps info on public keys of other users and signatures of these keys - together with trust value of the key