Authentication III

Computer and Network Security

Emilio Coppa

Authentication protocols based on public keys

Authentication using public key

Idea: use a public-key scheme to sign messages containing challenges/timestamps

- A to B: A, B, T (timestamp), N (nonce)
- B to A: K_{PR_B}(A, B, t, N+1)
- A checks (A, B, t, N) in message using K_{PB_B}

Problem: it is crucial to guarantee knowledge of public key of Bob. If Trudy can convince Alice that public key of Bob is K_{PB_T} then failure

Solution: Public Key Infrastructure (PKI), i.e., a system that integrates an authority able to guarantee correctness of public keys by signing them.

Needham-Schroeder public-key protocol

C is the trusted authority that signs public keys

Mutual Authentication:

1. A to C: A, B
2. C to A: B, K_{PB_B}, $Sig_C(K_{PB\_B}, B)$
3. A checks digital signature of C, generates nonce N and sends to B: $K_{PB\_B}(N, A)$
4. B decodes it and (now wants to check A's identity) sends to C: B, A
5. C to B: A, K_{PB_A}, $Sig_C(K_{PB\_A}, A)$
6. B checks C's digital signature, retrieves K_{PB_A}, generates nonce N' and sends to A: $K_{PB\_A}(N, N')$
7. A decodes, checks N, and sends to B: $K_{PB\_B}(N')$

N.S. public-key protocol: MITM attack

Trudy is a system user that can talk (being authenticated) to A, B and C

Two interleaved excerpts of the protocol:

S1: authentication between A and T

S2: authentication between T (like A) with B

Man in the middle attack (Lowe in 1995):

- 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

N.S. public-key protocol: MITM attack (2)

Steps 3, 6, 7 of S1 and S2:

- A to T [step 3 of S1]: $K_{PB\_T}(N, A)$
- T (like A) to B [step 3 of S2]: $K_{PB\_B}(N, A)$
- B to T (like A) [step 6 of S2]: $K_{PB\_A}(N, N')$
- T to A [step 6 of S1]: $K_{PB\_A}(N, N')$
- A to T [step 7 of S1]: $K_{PB\_T}(N')$
- T (like A) to B [step 7 of S2]: $K_{PB\_B}(N')$

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

Needham-Schroeder public-key protocol (fixed)

Mutual authentication:

1. A to C: A, B
2. C to A: B, K_{PB_B}, $Sig_C(K_{PB\_B}, B)$
3. A checks digital signature of C, generates nonce N and sends to B: $K_{PB\_B}(N, A)$
4. B decode (now wants to check A's identity) and sends to C: B, A
5. C to B: A, K_{PB_A}, $Sig_C(K_{PB\_A}, B)$
6. B checks C's digital signature, retrieves K_{PB_A}, generates nonce N' and sends to A: $K_{PB\_A}(\underline{B}, N, N')$
7. A decodes, checks N, and sends to B: $K_{PB\_B}(N')$

Why the previous attack fails now?

We focus on steps 3, 6, 7 of S1 and S2:

- A to T [step 3 of S1]: $K_{PB\_T}(N, A)$
- T (like A) to B [step 3 of S2]: $K_{PB\_B}(N, A)$
- B to T (like A) [step 6 of S2]: $K_{PB\_A}(B, N, N')$
- T to A [in step 6 of S1]:

  BEFORE: $K_{PB\_A}(N, N')$

  NOW: T CANNOT send $K_{PB\_A}(B, N', N)$ while is talking to A!
- A to T [step 7 of S1]: $K_{PB\_T}(N')$
- T (like A) to B [step 7 of S2]: $K_{PB\_B}(N')$

Standard: X.509 (RFC 5280)

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, e.g., [Stallings2005], or [ITU-T])
  - One-Way Authentication (most communications on the web are of this type!)
  - 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

A to B: $cert_A$, D_A, $Sig_A(D_A)$

where:

- $cert_A$ is the certificate of A’s public key, signed by an authority
- D_A is $(t, B, K_{PB\_B}(K_{AB}))$
- t is a timestamp
- K_{AB} is a shared secret key
- $Sig_A(D_A)$ is a signuture on D_A from A

X.509: one-way authentication (2)

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

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

Message includes at least a timestamp tA (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 authentication

1. A to B: $cert_A$, D_A, $Sig_A(D_A)$
2. B to A: $cert_B$, D_B, $Sig_B(D_B)$

where:

- $cert_X$ is the certificate of X's public key, signed by an authority
- D_A is $(t_A, N, B, K_{PB\_B}(K_{AB}))$, D_B is $(t_B, N, N', B, K_{PB\_A}(K_{BA}))$
- t_X is a timestamp from X, N/N' are nonces to prevent replay attacks
- K_{AB} and K_{BA} are shared secret keys
- $Sig_X(D_X)$ is a signature on D_X from X

Remark. There is no identity of A in D_A: source of problems in NS public-key protocol.

X.509: three-ways authentication

Used in case of unsynchronized clocks (t=0 in the protocol):

1. A to B: $cert_A$, D_A, $Sig_A(D_A)$
2. B to A: $cert_B$, D_B, $Sig_B(D_B)$
3. A to B: B, $Sig_A(N, N', B)$

where:

- $cert_X$ is the certificate of X’s public key, signed by an authority
- D_A is $(0, N, B, K_{PB\_B}(K_{AB}))$, D_B is $(0, N, N', B, K_{PB\_A}(K_{BA}))$
- N/N’ are nonces to prevent replay attacks
- K_{AB} and K_{BA} are shared secret keys
- $Sig_X(D_X)$ is a signature on D_X from X

Challenge-response: ISO/IEC 9798-3 Mutual authentication

1. B to A: N_B
2. A to B: $cert_A, N_A, N_B, B, Sig_A(N_A, N_B, B)$
3. B to A: $cert_B, N'_B, N_A, A, Sig_B(N'_B, N_A, A)$

Does it work?

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

1. B to A: N_B
2. A to B: $cert_A$, N_A, N_B, B, $Sig_A(N_A, N_B, B)$
3. B to A: $cert_B$, N'_B, N_A, A, $Sig_B(N'_B, N_A, A)$

Canadian Attack [Protocols for Authentication and Key Establishment]:

1. T (as B) to A: N_T
2. A to T (as B): $cert_A$, N_A, N_T, B, $Sig_A(N_A, N_T, B)$
3. T (as A) to B: N_A
4. B to T (as A): $cert_B$, N_B, N_A, A, $Sig_B(N_B, N_A, A)$
5. T (as B) to A: $cert_B$, N_B, N_A, A, $Sig_B(N_B, N_A, A)$ T is authenticated!

X509 authentication protocols

The three protocols are based on certificates that guarantee the correctness and validity of the public keys of Alice and Bob. Without this guarantee the protocol are unsecure.

Remark. The same requirement was discussed in Diffie-Hellman Key Exchange.

In X509, certificates are signed by a Certificate Authority (CA). The infrastructure to support interactions with a CA and user authentication is defined a Public Key Infrastructure (PKI).

Public Key Infrastructure (PKI)

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 only one public key (CA's public key)

Public Key Infrastructure (2)

Remarks:

- 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
- assumption behind a PKI: when a new user applies to a CA for a certificate, the CA can authenticate the identity of the applicant through other means.

Kinds of certificates

Three main kinds depending on the level of “identity assurance and authentication” that was carried out with regard to the applicant organization:

1. Extended Validation (EV): expensive, take several days
2. Organization Validation (OV): a few checks, reasonably fast
3. Domain Validation (DV): the applicant has the right to “use” a specific domain name. Very common and fast.

Browsers may show different UI element wrt EV vs OV vs DV.

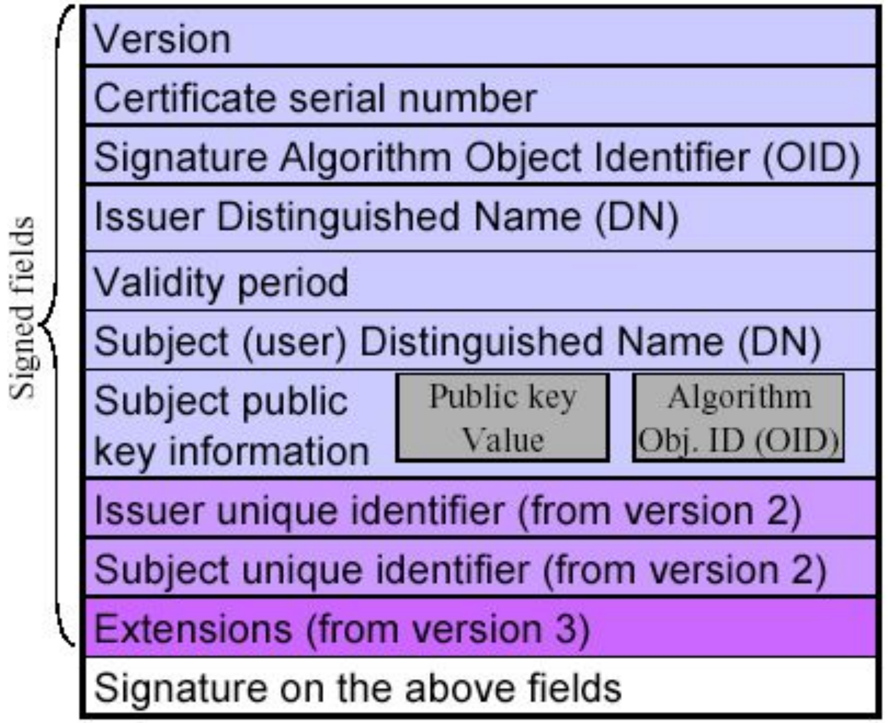
Let’s Encrypt: a non-profit CA

CA run by Internet Security Research Group (ISRG) that provides X.509 certificates for Transport Layer Security (TLS) encryption at no charge. Let's Encrypt certificates are valid for 90 days, during which renewal can take place at any time.

An automated process designed to overcome manual creation, validation, signing, installation, and renewal of certificates for secure websites:

- DV performed using Automatic Certificate Management Environment (ACME) protocol
- ACME is based on a challenge-response procedure:
  - your webserver has to respond to specific URLs with specific contents during a specific timeframe
  - (alternative) specific DNS records should be available during a specific timeframe
- Current initiatives of major browser developers such as Mozilla and Google to deprecate unencrypted HTTP are counting on the availability of Let's Encrypt.

X.509 Certificate



Signed fields
Version
Certificate serial number
Signature Algorithm Object Identifier (OID)
Issuer Distinguished Name (DN)
Validity period
Subject (user) Distinguished Name (DN)
Subject public key information
Public key Value
Algorithm Obj. ID (OID)
Issuer unique identifier (from version 2)
Subject unique identifier (from version 2)
Extensions (from version 3)
Signature on the above fields

X.509 Certificate: fields

- 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.
- SERIAL NUMBER. An integer that, together with the issuing CA's name, uniquely identifies this certificate.
- SIGNATURE ALGORITHM ID. 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.

X.509 Certificate: fields (2)

- ISSUER NAME. The X.500 name of the issuing CA. It follow a specific format and provides several details:
  - C: country, e.g., US
  - OU: organization unit, e.g., company name
  - CN: common name, e.g., *.google.com

  There is no standard on how to display these kinds of information.

- VALIDITY. This contains two subfields, the time the certificate becomes valid, and the last time for which it is valid.

X.509 Certificate: fields (3)

- SUBJECT. The X.500 name of the entity whose key is being certified. Again it follows specific rules.
- SUBJECT PUBLIC KEY INFO. This contains two subfields: (a) an algorithm identifier, and (b) the subject's public key.
- ISSUER UNIQUE IDENTIFIER. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the issuer of this certificate.

X.509 Certificate: fields (4)

- SUBJECT UNIQUE IDENTIFIER. Optional (permitted only in version 2 and version 3, but deprecated). Uniquely identifies the subject of this certificate.
- 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: properties

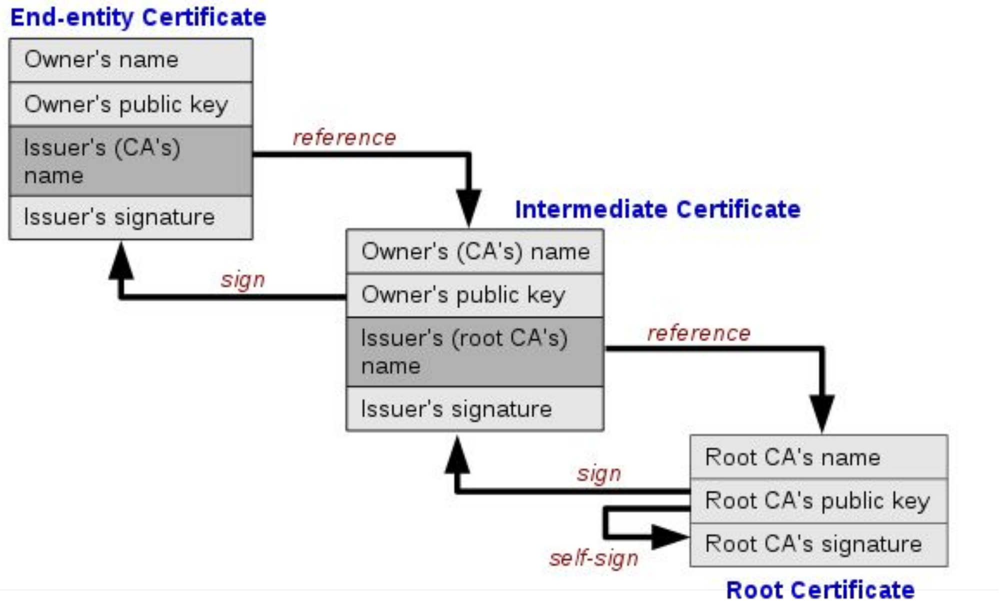
- They can be easily accessed
- Certificates are created and modified by CA
- Certificates “impossible” to falsify (RSA > 2000 bit)

Problem: where do we get the public key of a CA?

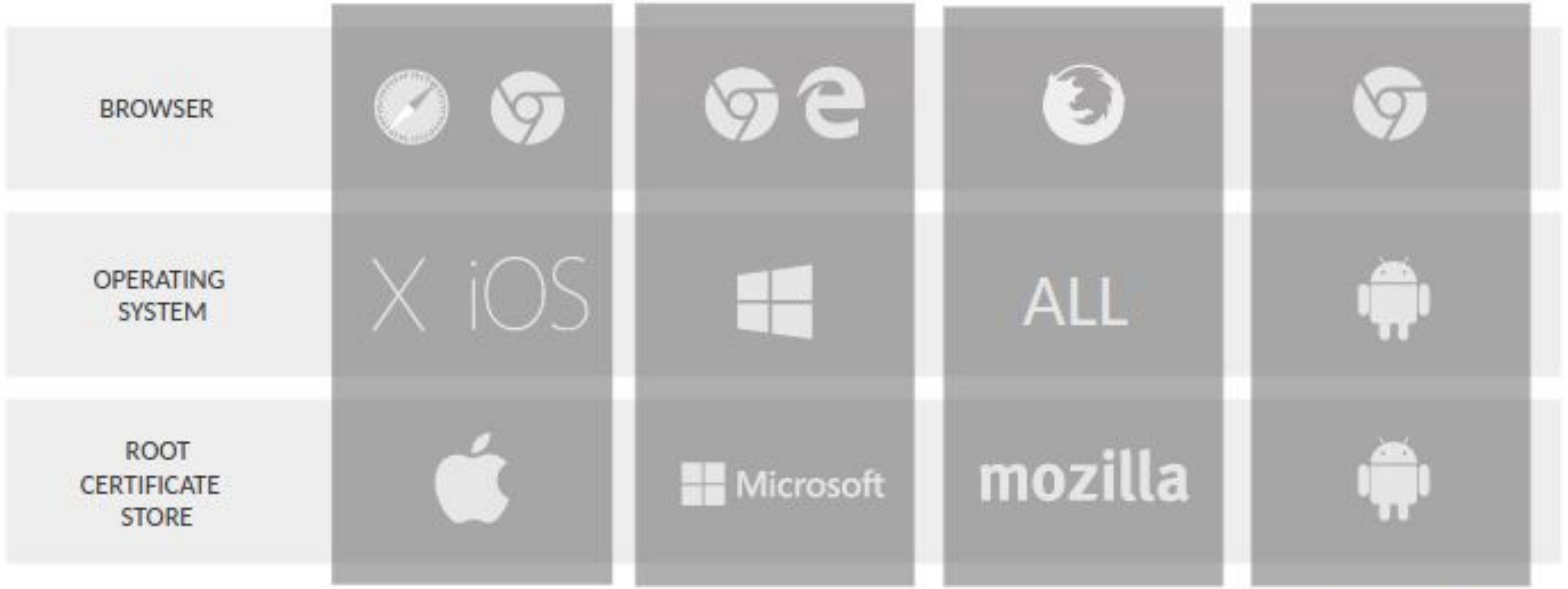
Chain of trust

Applications, e.g., browsers, contains public keys of a few and selected CAs, which are called “root store”, i.e., a list of trusted root CA certificates. X.509 defines a hierarchy that allows an application to trust a certificate from a CA whenever another CA trust it, proceeding recursively until a root CA is met along the path.

Root CAs self-sign their certificate. You can add or remove certificates from an application.



Root Certificate Stores



BROWSER
OPERATING SYSTEM
ROOT CERTIFICATE STORE
X iOS
ALL
Microsoft
mozilla

From http://certsimple.com

Root CAs in Linux

A (desktop) Linux distribution will often store root certificates under /etc/ssl/certs:

```
> ls /etc/ssl/certs/
…
Go_Daddy_Class_2_CA.pem
...
Microsoft_RSA_Root_Certificate_Authority_2017
...
SSL.com_EV_Root_Certification_Authority_ECC.pem
…
SwissSign_Gold_CA_-_G2.pem
...
VeriSign_Class_3_Public_Primary_Certification_Authority_-_G4.pem
```

```
> openssl x509 -text < /etc/ssl/certs/Go_Daddy_Root_Certificate_Authority_-_G2.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: sha256WithRSAEncryption <--- this means RSASSA-PKCS1-v1_5
        Issuer: C = US, ST = Arizona, L = Scottsdale, O = "GoDaddy.com, Inc.", CN = Go Daddy Root
Certificate Authority - G2
        Validity
            Not Before: Sep  1 00:00:00 2009 GMT
            Not After : Dec 31 23:59:59 2037 GMT
        Subject: C = US, ST = Arizona, L = Scottsdale, O = "GoDaddy.com, Inc.", CN = Go Daddy Root
Certificate Authority - G2
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:bf:71:62:08:f1:fa:59:34:f7:1b:c9:18:a3:f7:[...]
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                3A:9A:85:07:10:67:28:B6:EF:F6:BD:05:41:6E:20:C1:94:DA:0F:DE
    Signature Algorithm: sha256WithRSAEncryption
         99:db:5d:79:d5:f9:97:59:67:03:61:f1:7e:3b:06:31:75:2d:[....]
```

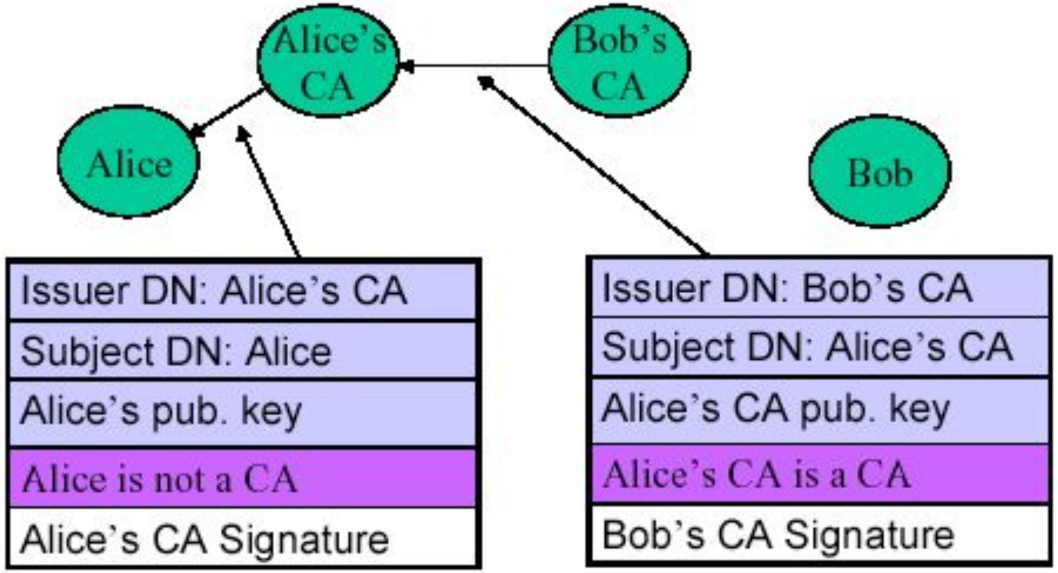
How to generate a certificate by yourself?

For testing purposes, you can generate a certificate by yourself:

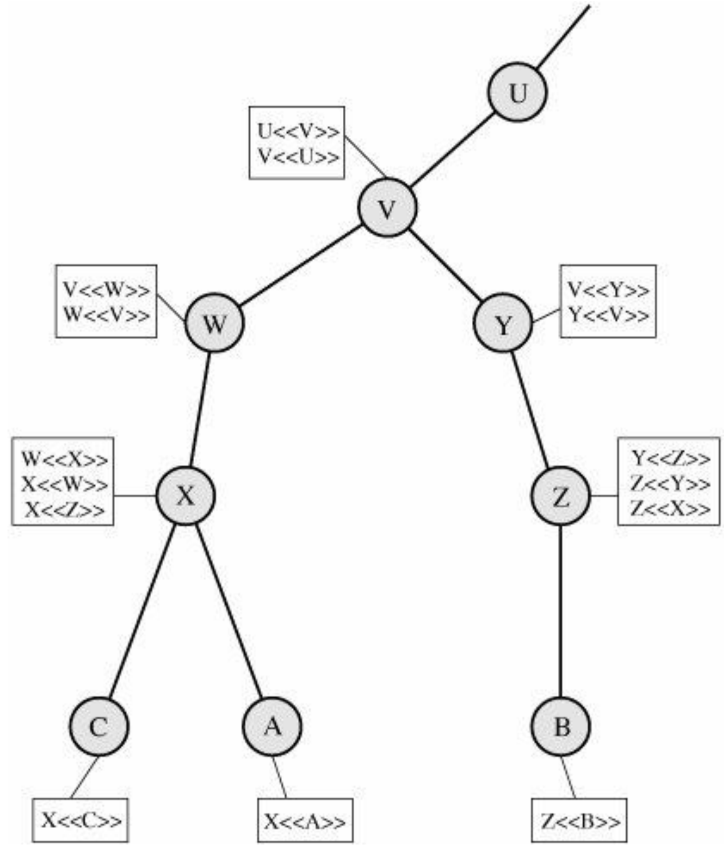
```
> openssl req -new -newkey rsa:1024 -days 365 -nodes -x509 -keyout test.pem -out test.cert
Generating a RSA private key
writing new private key to 'test.pem'
-----
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but
you can leave some blank. For some fields there will be a default value. If you enter '.', the field will be
left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Italy
Locality Name (eg, city) []:Rome
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Big Bang Theory
Organizational Unit Name (eg, section) []:Bazinga
Common Name (e.g. server FQDN or YOUR name) []:Sheldon Cooper
Email Address []:coppa@diag.uniroma1.it
```

Certificate test.cert will no be validated by any real-world CA!

Hierarchy of CAs



Hierarchy of CAs (2)



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)

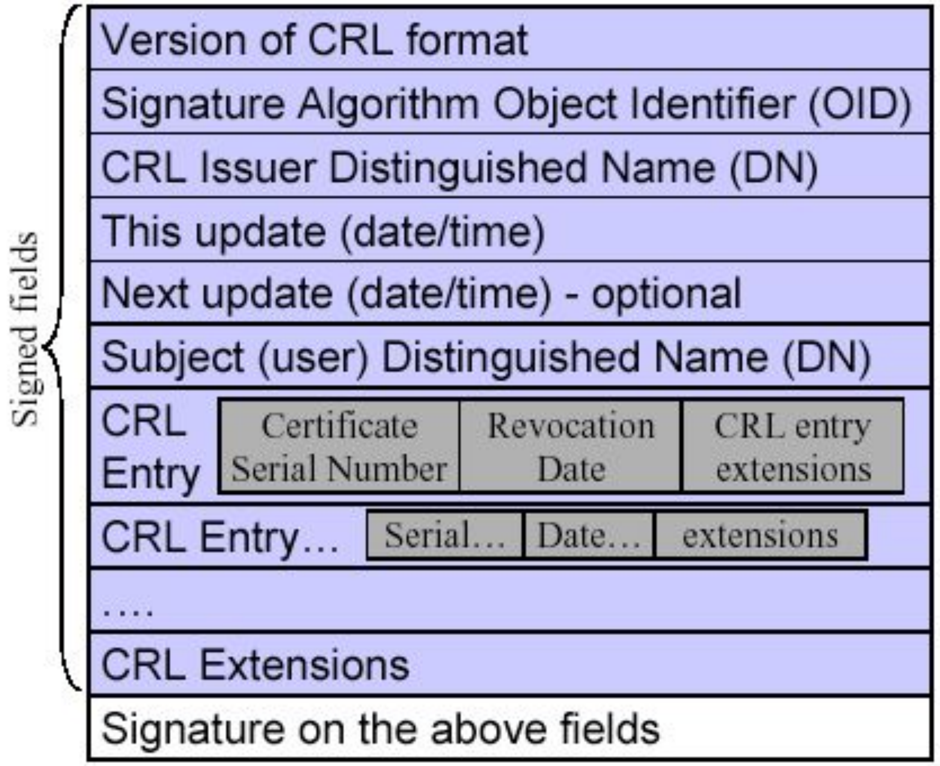
They can be revoked before the deadline:

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 authenticating a user

Certificate Revocation List



Signed fields

Version of CRL format
Signature Algorithm Object Identifier (OID)
CRL Issuer Distinguished Name (DN)
This update (date/time)
Next update (date/time) - optional
Subject (user) Distinguished Name (DN)
CRL Entry: Certificate Serial Number | Revocation Date | CRL entry extensions
CRL Entry…: Serial… | Date… | extensions
….
CRL Extensions
Signature on the above fields

Certificate Revocation List: 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.

Certificate Revocation List: fields (2)

- THIS UPDATE. This contains the time the CRL was issued.
- NEXT UPDATE. 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.

Certificate Revocation List: fields (3)

The following three fields repeat together, once for each revoked certificate:

- USER CERTIFICATE. This contains the serial number of the revoked certificate.
- REVOCATION DATE. This contains the time the certificate was revoked.
- CRL ENTRY EXTENSIONS. This contains various optional information such as a reason code for why the certificate was revoked.

Certificate Revocation List: fields (4)

- CRL EXTENSIONS. This contains various optional information.
- ALGORITHM IDENTIFIER. 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)

What if we do not want to trust a “centralized” root CA?

An alternative approach to a hierarchy of CA is proposed by Pretty Good Privacy (PGP):

- Trust model for E-mail certif. (Zimmerman)
- Idea: 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

Credits

These slides are based on material from:

- Slides of Prof. D’Amore from CNS 2019-2020
- Christof Paar and Jan Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer. http://www.crypto-textbook.com/
- Charlie Kaufman, Radia Perlman, and Mike Speciner. Network Security - Private Communication in a Public World. Prentice Hall.
- Wikipedia (english version)