Data Protection

Jorge L. Villar

MCYBERS, UPC, Fall 2025

Outline

Digital Signatures

The public key version of a MAC:

- Only the intended signer can sign.
  Signing a public message requires the knowledge of a secret key.
- Everybody can verify a signature.
  Verifying a signature only requires the message and the public key.

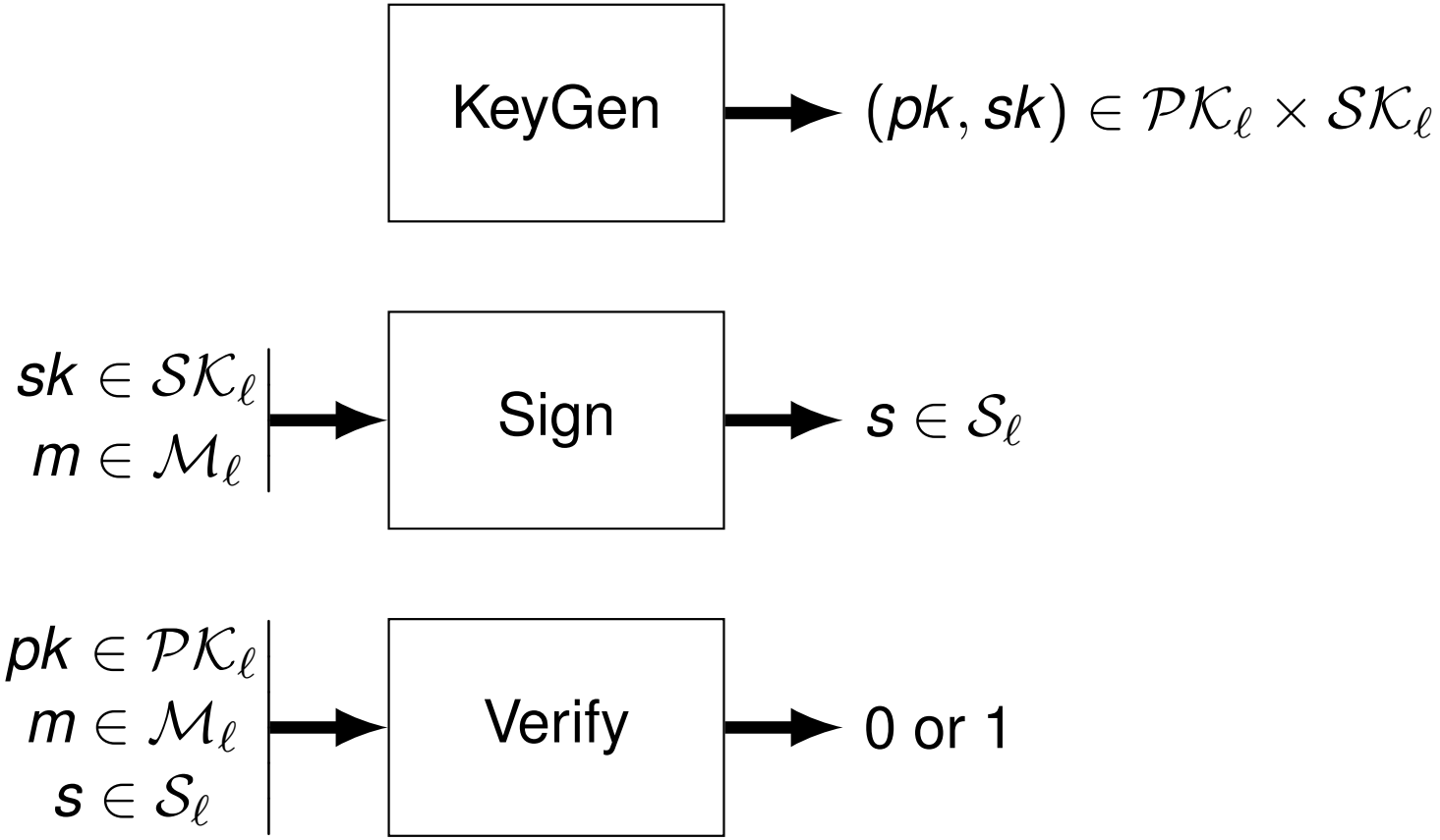
Digital Signatures

The public key version of a MAC:

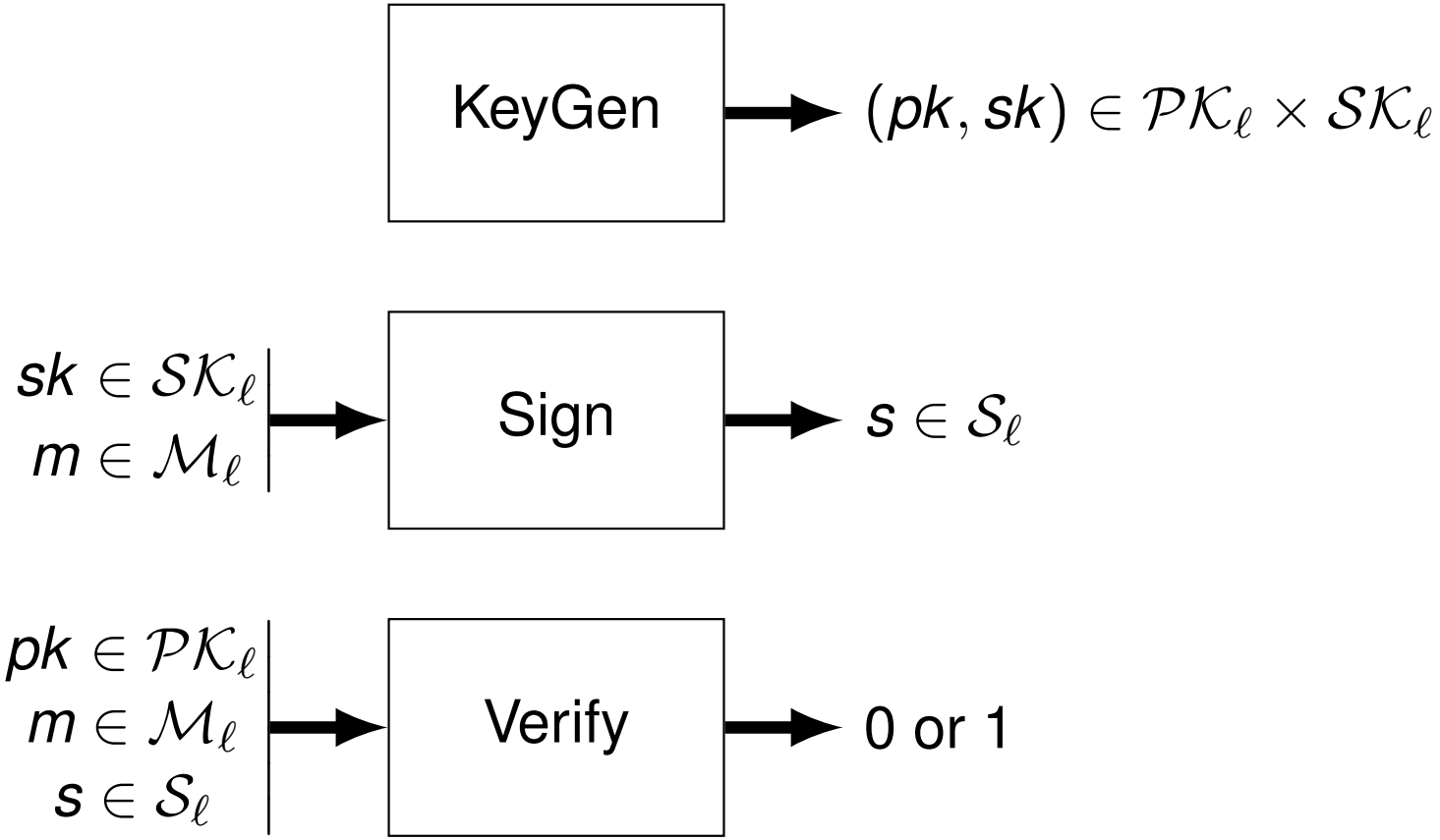
- Only the intended signer can sign.
  Signing a public message requires the knowledge of a secret key.
- Everybody can verify a signature.
  Verifying a signature only requires the message and the public key.

The man-in-the-middle impersonation attack also works for digital signatures!

Digital Signatures: Syntax



Digital Signatures: Correctness



$\forall m \in \mathcal{M}_\ell,\ \forall (pk, sk) \leftarrow \text{KeyGen}(\ell),$
$\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1$

Plain RSA Signature

Use RSA decryption for signing, and RSA encryption for verification.

KeyGen(ℓ) : (Same as RSA PKE.)
  output $((n, e), (n, d))$

Sign(n, d, m) :
  output $m^d \bmod n$

Verify(n, e, m, s) :
  output 1 **if** $m = s^e \bmod n$

Plain RSA Signature

Use RSA decryption for signing, and RSA encryption for verification.

KeyGen(ℓ) : (Same as RSA PKE.)
  output $((n, e), (n, d))$

Sign(n, d, m) :
  output $m^d \bmod n$

Verify(n, e, m, s) :
  output 1 **if** $m = s^e \bmod n$

Correctness:
$s^e \bmod n = (m^d)^e \bmod n = m^{de} \bmod n = m$.

Plain RSA Signature

Use RSA decryption for signing, and RSA encryption for verification.

KeyGen(ℓ) : (Same as RSA PKE.)
output $((n, e), (n, d))$

Sign(n, d, m) :
output $m^d \bmod n$

Verify(n, e, m, s) :
output 1 **if** $m = s^e \bmod n$

Correctness:
$s^e \bmod n = (m^d)^e \bmod n = m^{de} \bmod n = m$.

- It cannot be used to sign long messages!
- Known forgery: s is a valid signature of $m = s^e \bmod n$.

RSA-FDH Signature

Use RSA decryption for signing, and RSA encryption for verification.

KeyGen(ℓ) : (Same as RSA PKE.)
output $((n, H, e), (n, H, d))$ // H is a hash function

Sign(n, H, d, m) :
output $H(m)^d \bmod n$

Verify(n, H, e, m, s) :
output 1 **if** $H(m) = s^e \bmod n$

Correctness:
$s^e \bmod n = (H(m)^d)^e \bmod n = H(m)^{de} \bmod n = H(m)$.

- It supports arbitrary message lengths
- No forgery mechanism is known.

ElGamal Signature

Built on a cyclic group G of order q with a hard DLOG problem.

E.g., p, q primes, g a nontrivial solution of $g^q = 1 \bmod p$.

KeyGen(ℓ) : (Same as ElGamal PKE.)
  $param \leftarrow (p, q, g)$
  output $(y = g^x \bmod p, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^\times$; $r \leftarrow g^k \bmod p$ // r computed mod p
  $s = (m - rx)k^{-1} \bmod q$ // r used mod q!!!
  output (r, s)

Verify($param, y, m, (r, s)$) :
  output 1 **if** $g^m = r^s y^r \bmod p$ // r mod p and mod q

ElGamal Signature

Built on a cyclic group G of order q with a hard DLOG problem.

E.g., p, q primes, g a nontrivial solution of $g^q = 1 \bmod p$.

KeyGen(ℓ) : (Same as ElGamal PKE.)
  $param \leftarrow (p, q, g)$
  output $(y = g^x \bmod p, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^\times$; $r \leftarrow g^k \bmod p$ // r computed mod p
  $s = (m - rx)k^{-1} \bmod q$ // r used mod q!!!
  output (r, s)

Verify($param, y, m, (r, s)$) :
  output 1 **if** $g^m = r^s y^r \bmod p$ // r mod p and mod q

Correctness:
$r^s y^r = g^{ks+xr} = g^{m-rx+rx} = g^m \bmod p$.

Forgery: $r = y^\alpha g^\beta \bmod p$, $s = -r\alpha^{-1} \bmod q$, $m = s\beta \bmod p$.

Pointcheval-Stern Signature

Built on a cyclic group G of order q with a hard DLOG problem.

E.g., p, q primes, g a nontrivial solution of $g^q = 1 \bmod p$.

KeyGen(ℓ) : (Same as ElGamal PKE.)
  $param \leftarrow (p, q, g, H)$ // H is a hash function
  output $(y = g^x \bmod p, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^\times$; $r \leftarrow g^k \bmod p$ // r computed mod p
  $s = (H(r, m) - rx)k^{-1} \bmod q$ // r used mod q!!!
  output (r, s)

Verify($param, y, m, (r, s)$) :
  output 1 **if** $g^{H(r,m)} = r^s y^r \bmod p$ // r mod p and mod q

Correctness:
$r^s y^r = g^{ks+xr} = g^{H(r,m)-rx+rx} = g^{H(r,m)} \bmod p$.

No forgery mechanism is known.

Variant: DSA

p, q primes, g a nontrivial solution of $g^q = 1 \bmod p$.

KeyGen(ℓ) : (Same as ElGamal PKE.)
  $param \leftarrow (p, q, g, H)$ // H is a hash function
  output $(y = g^x \bmod p, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^\times$; $r \leftarrow (g^k \bmod p) \bmod q$
  if $r = 0$ **then** try a different k
  $s = (H(r, m) + rx)k^{-1} \bmod q$
  if $s = 0$ **then** try a different k
  output (r, s)

Verify($param, y, m, (r, s)$) :
  output 1 **if** $r \neq 0$ **and** $s \neq 0$ **and**
  $\left((g^{H(r,m)} y^r)^{s^{-1} \bmod q} \bmod p\right) \bmod q = r$

Variant: DSA

p, q primes, g a nontrivial solution of $g^q = 1 \bmod p$.

KeyGen(ℓ) : (Same as ElGamal PKE.)
  $param \leftarrow (p, q, g, H)$ // H is a hash function
  output $(y = g^x \bmod p, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^\times$; $r \leftarrow (g^k \bmod p) \bmod q$
  if $r = 0$ **then** try a different k
  $s = (H(r, m) + rx)k^{-1} \bmod q$
  if $s = 0$ **then** try a different k
  output (r, s)

Verify($param, y, m, (r, s)$) :
  output 1 **if** $r \neq 0$ **and** $s \neq 0$ **and**
  $\left((g^{H(r,m)} y^r)^{s^{-1} \bmod q} \bmod p\right) \bmod q = r$

Correctness:
$(g^{H(r,m)} y^r)^{s^{-1} \bmod q} = g^{(H(r,m)+rx)s^{-1} \bmod q} = g^k \bmod p.$

Variant: ECDSA

$E_{a,b}$ elliptic curve, q prime, B a nontrivial point s.t. $qB = O$.

KeyGen(ℓ) : (Same as ElGamal PKE on $E_{a,b}$.)
  $param \leftarrow (E_{a,b}, q, B, H)$ // H is a hash function
  output $(Y = xB, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^{\times}$; $r \leftarrow (kB)_x \bmod q$ // x-coordinate of the point kB, mod q
  if $r = 0$ **then** try a different k
  $s = (H(r, m) + rx)k^{-1} \bmod q$
  if $s = 0$ **then** try a different k
  output (r, s)

Verify($param, Y, m, (r, s)$) :
  output 1 **if** $r \neq 0$ **and** $s \neq 0$ **and**
    $\left((s^{-1} \bmod q)(H(r, m)B + rY)\right)_x \bmod q = r$

Variant: ECDSA

$E_{a,b}$ elliptic curve, q prime, B a nontrivial point s.t. $qB = O$.

KeyGen(ℓ) : (Same as ElGamal PKE on $E_{a,b}$.)
  $param \leftarrow (E_{a,b}, q, B, H)$ // H is a hash function
  output $(Y = xB, x)$ for a random x

Sign($param, x, m$) :
  $k \leftarrow \mathbb{Z}_q^\times$; $r \leftarrow (kB)_x \bmod q$ // x-coordinate of the point kB, mod q
  if $r = 0$ **then** try a different k
  $s = (H(r, m) + rx)k^{-1} \bmod q$
  if $s = 0$ **then** try a different k
  output (r, s)

Verify($param, Y, m, (r, s)$) :
  output 1 **if** $r \neq 0$ **and** $s \neq 0$ **and**
    $\left((s^{-1} \bmod q)(H(r, m)B + rY)\right)_x \bmod q = r$

Correctness: $(s^{-1} \bmod q)(H(r, m)B + rY) =$
$= (s^{-1}(H(r, m) + rx) \bmod q)B = kB.$

Summary of Digital Signature Schemes

- Plain RSA and ElGamal are insecure (known forgeries)
- RSA-FDH, Pointcheval-Stern, DSA and ECDSA can sign messages with arbitrary lengths
- RSA-FDH is deterministic
- Poor randomness of k in Pointcheval-Stern, DSA or ECDSA can leak the secret key

Summary of Digital Signature Schemes

- Plain RSA and ElGamal are insecure (known forgeries)
- RSA-FDH, Pointcheval-Stern, DSA and ECDSA can sign messages with arbitrary lengths
- RSA-FDH is deterministic
- Poor randomness of k in Pointcheval-Stern, DSA or ECDSA can leak the secret key

Typical sizes (in bits):

	pk	sk	signature
RSA-FDH	2048	2048	2048
Pointcheval-Stern	6368	224	2272
DSA	6368	224	448
ECDSA	896	224	448

Outline

Identification vs. Signature

Digital Signature: Token or protocol execution that convinces someone that “Who knows *sk* endorses document *m*”.

Digital Identification: Token or protocol execution that convinces someone that “I’m the person who knows *sk*”.

Identification vs. Signature

Digital Signature: Token or protocol execution that convinces someone that "Who knows *sk* endorses document *m*".

Digital Identification: Token or protocol execution that convinces someone that "I'm the person who knows *sk*".

Any signature scheme can be used as an identification scheme (e.g., by asking for a valid signature on a random document).

Some identification schemes can be easily upgraded to be a digital signature scheme.

Interactive Schnorr Identification Scheme

Prover	$\leftarrow (G, q, g) \rightarrow$	Verifier
	Setup:	
$x \leftarrow \mathbb{Z}_q;\ y = g^x$		
send(y)		**receive**(y)
	Prove:	
$r \leftarrow \mathbb{Z}_q;\ a = g^r$		
send(a)		**receive**(a)
		$c \leftarrow \mathbb{Z}_q$
receive(c)		**send**(c)
$t = r + cx \bmod q$		
send(t)		**receive**(t)
	Verify:	
		accept if $g^t = ay^c$

Correctness of Schnorr Identification

Prover	$\leftarrow (G, q, g) \rightarrow$	Verifier
$x \leftarrow \mathbb{Z}_q;\ y = g^x; \textbf{send}(y)$		$\textbf{receive}(y)$
$r \leftarrow \mathbb{Z}_q;\ a = g^r; \textbf{send}(a)$ $\textbf{receive}(c)$ $t = r + cx \bmod q; \textbf{send}(t)$		$\textbf{receive}(a)$ $c \leftarrow \mathbb{Z}_q; \textbf{send}(c)$ $\textbf{receive}(t)$
		$\textbf{accept if } g^t = ay^c$

Correctness: $g^t = g^{r+cx} = g^r(g^x)^c = ay^c$.

Soundness of Schnorr Identification

Prover'	$\leftarrow (G, q, g) \rightarrow$	Verifier
??? **send**(y)		**receive**(y)
??? **send**(a)		**receive**(a)
receive(c_1)		$c_1 \leftarrow \mathbb{Z}_q$; **send**$(c_1)$
??? **send**(t_1)		**receive**(t_1)
		accept if $g^{t_1} = a y^{c_1}$

$\Pr(\text{Accept}) > 1/q \Rightarrow$ two different accepting conversations

Soundness of Schnorr Identification

Prover'	$\leftarrow (G, q, g) \rightarrow$	Verifier
??? **send**(y)		**receive**(y)
??? **send**(a)		**receive**(a)
receive(c_1)		$c_1 \leftarrow \mathbb{Z}_q$; **send**$(c_1)$
??? **send**(t_1)		**receive**(t_1)
		accept if $g^{t_1} = ay^{c_1}$

$\Pr(\text{Accept}) > 1/q \Rightarrow$ two different accepting conversations

(*rewind*)	
receive(c_2)	$c_2 \leftarrow \mathbb{Z}_q \setminus \{c_1\}$; **send**$(c_2)$
??? **send**(t_2)	**receive**(t_2)
	accept if $g^{t_2} = ay^{c_2}$

Soundness: $g^{t_2 - t_1} = ay^{c_2}/ay^{c_1} = y^{c_2 - c_1} \Rightarrow y = g^x$ with $x = (t_2 - t_1)(c_2 - c_1)^{-1} \bmod q$. (The Prover "knows" it!)

Privacy of Schnorr Identification

Simulator	Prover	Verifier
receive(y)	$y = g^x$; **send** y	**receive**(y)
$t_S \leftarrow \mathbb{Z}_q$ $c \leftarrow \mathbb{Z}_q$ $a_S = g^{t_S} y^{-c}$	$a = g^r$; **send**(a) **receive**(c) $t = r + cx$; **send**(t)	**receive**(a) $c \leftarrow \mathbb{Z}_q$; **send**$(c)$ **receive**(t)
$g^{t_S} = a_S y^c$ **output**(a_S, c, t_S)		**accept if** $g^t = a y^c$ **output**(a, c, t)

Privacy: The outputs of the Simulator and the Prover are identically distributed.

Anything learnt by the verifier can also be learnt without interaction with the prover.

Fiat-Shamir: Non-Interactive Identification

Remove interaction by using a hash function:

Prover	$\leftarrow (G, q, g, H) \rightarrow$	Verifier
$x \leftarrow \mathbb{Z}_q;\ y = g^x$; **send**$(y)$		**receive**(y)
$r \leftarrow \mathbb{Z}_q;\ a = g^r$ $c = H(G, q, g, y, a)$ $t = r + cx \bmod q$; **send**(a, c, t)		**receive**(a, c, t)
		$a = g^t y^{-c}$ **accept if** $c = H(G, q, g, y, a)$

Fiat-Shamir: Non-Interactive Identification

Remove interaction by using a hash function:

Prover	$\leftarrow (G, q, g, H) \rightarrow$	Verifier
$x \leftarrow \mathbb{Z}_q;\ y = g^x$; **send**$(y)$		**receive**(y)
$r \leftarrow \mathbb{Z}_q;\ a = g^r$ $c = H(G, q, g, y, a)$ $t = r + cx \bmod q$; **send**(a, c, t)		**receive**(a, c, t)
		$a = g^t y^{-c}$ **accept if** $c = H(G, q, g, y, a)$

Correctness: As in the interactive case.

Soundness and Privacy: Require that H is a random function.
(Heuristic security argument using the Random Oracle Model.)

Schnorr Signature

Add the message to the hash in the non-interactive protocol:

KeyGen(ℓ) : (Same as ElGamal PKE.)
  $param \leftarrow (G, q, g, H)$ // H is a hash function
  output $(y = g^x, x)$ for a random x

Sign($param, x, m$) :
  $r \leftarrow \mathbb{Z}_q^\times$; $a \leftarrow g^r$; $c = H(param, y, a, m)$; $t = r + cx$
  output (c, t)

Verify($param, y, m, (c, t)$) :
  $a = g^t y^{-c}$
  output 1 **if** $c = H(param, y, a, m)$

Sizes and performance are comparable to DSA (or to ECDSA for a elliptic curve based Schnorr signature).

Outline

Public Key Certificates

To prevent the Man-in-the-middle attack, public keys need to be certified.

A certificate is a digital signature of a document containing at least:

- The public key owner's identity
- The public key algorithm identifier for which the public key is intended
- The public key itself
- The validity period

The document is signed by a trusted authority (CA).

Instead of trusting the public key of each user, you only need to trust the public key of the CA.

Certificate Chains

In practice, a single CA have limited trust (e.g., restricted to a particular company or institution).

The public keys of several CA's can be certified by another CA at a higher level.

The upper most CA in the trust tree is called a **root CA**. Root CA's certificates are self signed, and they are typically preinstalled in internet browsers and other network tools.

The sequence of public key certificates from the root CA to the final user is the **certificate chain**. It can be verified sequentially, just verifying all the digital signatures contained in them.

X.509 Certificate Specification

Practical public key certificates are widely used for secure networking, secure email, code signing, document signing, electronic ID, etc.

X.509 (v3) Certificate format (simplified):

- Certificate format version number
- (unique) Serial number
- Signature algorithm (used to sign the certificate)
- Issuer identity
- Validity period
- Subject identity
- Subject public key algorithm
- Subject public key data
- Extensions (key usage, policies)
- The certificate signature

Certificate Example

```
Data:
    Version: 3 (0x2)
    Serial Number:
        71:b0:65:39:7c:8e:07:d2:54:1a:96:7f:75:59:37:92
Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=ES, O=Agencia Cat..., OU=Serv..., OU=Veg..., OU=Jer..., CN=EC-ACC
    Validity
        Not Before: Sep 18 08:23:27 2014 GMT
        Not After : Sep 18 08:23:27 2030 GMT
    Subject: C=ES, O=CONSOR..., OU=Serveis..., CN=EC-SectorPublic
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:cb:ae:4e:31:31:bc:f1:db:11:f1:89:dc:c5:3d:
                ...
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:TRUE, pathlen:0
        X509v3 Key Usage: critical
            Certificate Sign, CRL Sign
        X509v3 Subject Key Identifier:
            47:3C:DE:14:77:BB:6A:4F:47:91:A9:02:FF:D4:06:E1:73:DC:E2:D9
        X509v3 Authority Key Identifier:
            keyid:A0:C3:8B:44:AA:37:A5:45:BF:97:80:5A:D1:F1:78:A2:9B:E9:5D:8D
        ...
Signature Algorithm: sha256WithRSAEncryption
     33:11:3b:47:aa:1a:2a:94:74:81:45:6f:1a:b0:29:78:7e:2b:
     ...
```

Certificate Revocation Lists

In a Public Key Infrastructure (PKI), a CLR adds the ability to invalidate some certificates (e.g., because the secret key has been compromised).

A CRL is maintained and authenticated by the CA itself.

Using CRLs is mandatory in any certificate validation process.

Contents of a CRL:

- CRL format version number
- Signature algorithm (used to sign the CRL)
- Issuer identity
- Validity period
- Extensions (optional)
- List of revoked certificates
- The CRL signature

CRL Example

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=ES/O=Agencia Cat.../OU=Serveis Pub.../OU=Vegeu https://.../CN=EC-ACC
        Last Update: Jul  1 06:26:29 2020 GMT
        Next Update: Jul  1 06:26:29 2025 GMT
        CRL extensions:
            X509v3 CRL Number:
                28
            X509v3 Authority Key Identifier:
                keyid:A0:C3:8B:44:AA:37:A5:45:BF:97:80:5A:D1:F1:78:A2:9B:E9:5D:8D
                DirName:/C=ES/O=Age.../OU=Serv.../OU=Vege.../CN=EC-ACC
                serial:11:D4:C2:14:2B:DE:21:EB:57:9D:53:FB:0C:22:3B:FF
            X509v3 Issuing Distribution Point:
                Full Name:
                  URI:http://epscd.catcert.net/crl/ec-acc.crl
                  URI:http://epscd2.catcert.net/crl/ec-acc.crl
Revoked Certificates:
    Serial Number: 20BD3DE069043D253E1C0B9C8252F422
        Revocation Date: Feb 21 09:04:25 2020 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Cessation Of Operation
    ...
    Signature Algorithm: sha1WithRSAEncryption
         19:b6:a3:db:20:01:6c:aa:62:1c:0c:7a:03:1a:2e:d7:94:89:
         ...
```

Data Protection

Jorge L. Villar

MCYBERS, UPC, Fall 2025

END