Cryptography

5 – Digital signatures

G. Chênevert

November 4, 2019

ISEN
ALL IS DIGITAL!
LILLE
yncréa

Today

Concept

Implementation

Certificates

Recall: message authentication

To autenticate a message m with a shared secret key k,

- Alice appends to it a tag $t = \mathrm{MAC}(k, m)$;
- upon reception of (m, t), Bob checks whether

$$t \stackrel{?}{=} \mathrm{MAC}(k, m).$$

Provides security against *forgery* by a malicious third party.

The problem with MACs

Alice and Bob share the exact same capabilities, so this system cannot protect them *against one another*.

Forgery:

Bob: "My name is Alice and I will give 100€ to Bob." ✗

Repudiation:

Alice: "My name is Alice and I will give 100€ to Bob." ✓

Alice: "Hey I never said that! It was Bob!" ✗

Digital signature provides

- message integrity
- sender authentication
- binding between message and sender
- non-falsification / forgery
- non-repudiation

Applications

- authenticity of official documents
- approval/agreement (contracts)
- software distribution
- financial transactions
- IoT
- ...

The (cryptographic) notion of *digital signature* should not be confused with the closely related (legal) notion of *electronic signature* (*cf.* European eIDAS regulation)

Construction idea

Use public-key encryption "in reverse"!

- private encryption (signing) key $k_{\text{priv}} = k_e$
- public decryption (verification) key $k_{\text{pub}} = k_d$

only Alice can sign, anyone can verify

Protocol (1^{st} try)

To sign a message m with private key k_e:

- Alice appends to it $s = E(k_e, m)$.

Upon reception of a pair (m, s):

- Bob checks with associated public key whether

$$D(k_d, s) \stackrel{?}{=} m.$$

Problems

- Asymmetric ciphers are inherently slow:

  problematic for long messages
- Need to use "multiple blocks" version of encryption
- Signed message is twice as long as original message!

Solution: sign a hash

To sign a message m with private key k_e:

- Alice computes $h = H(m)$;
- appends $s = E(k_e, h)$ to m.

Upon reception of a pair (m, s):

- Bob checks with associated public key whether

$$D(k_d, s) \stackrel{?}{=} H(m).$$

To sum up:

Digital signatures are (usually) built from a hash function + asymmetric encryption.

- Only Alice can sign with private k_e.
- Anyone can check that the signature is genuine using public k_d.
- Footprint is minimal (computation time + size of signed message).

Note: any weakness in the hash *or* encryption directly impacts the security of the signature.

Today

Concept

Implementation

Certificates

In practice

Most digital signature schemes in use today are based on either RSA or DLP ciphers.

Warning 1

Signatures do nothing to conceal the content of the message; encryption needs to be used as well.

Warning 2

Never use the same key pairs for encryption and signature!

RSA Probabilistic Signature Standard

Specified in PKCS #1

- H is taken to be one of the flavors of SHA-2
- The actual value that is signed incorporates some random salt
- Signature (encryption) can be sped up using CRT
- Verification (decryption) can be sped up by using a small Fermat prime

RSA-PSS protocol (1/2)

To sign a message m with public n, d, private e, Alice:

- computes $h = \mathrm{SHA2}(m)$
- chooses random salt k
- applies padding $M = T(k, h) \in [\![0, n[\![$
- appends $s :\underset{n}{\equiv} M^e$

RSA-PSS protocol (2/2)

Upon reception of a pair (m, s), Bob:

- computes $h = \mathrm{SHA2}(m)$
- decrypts $M \underset{n}{\equiv} s^d$
- recovers random salt k from M
- checks whether

$$M \stackrel{?}{=} T(k, h)$$

Digital Signature Standard

In the US, NIST specifies two other signature schemes in the Digital Signature Standard:

- **DSA** (variant of mod n ElGamal)
- **ECDSA** (using elliptic curves)

(Probabilistic padding not needed).

DSA (1/2)

Public parameters: (can be reused)

- a medium-sized prime $q \approx 2^{256}$
- a large prime $p \approx 2^{2048}$ such that $q \mid p - 1$
- an integer g of multiplicative order q mod p:

$$g^q \underset{p}{\equiv} 1$$

Keys:

- private $x \in \rrbracket 0, q \llbracket$
- public $y \underset{p}{\equiv} g^x$

DSA (2/2)

Signature:

- Choose random $k \in \,]\!]0, q[\![$
- Compute $r = (g^k \,\%\, p) \,\%\, q$
- Compute $s \underset{q}{\equiv} k^{-1} \cdot (H(m) + xr)$

Verification: upon reception of $(m, (r, s))$,

- Compute $t \underset{q}{\equiv} s^{-1}$
- Verify if

$$((g^{H(m)} y^r)^t \,\%\, p) \,\%\, q \stackrel{?}{=} r.$$

Schnorr signatures

(EC)DSA signatures are more compact than RSA

but: no formal security proof exists!

The crypto community today favors using some form of *Schnorr signature*

e.g. Edwards-curve Digital Signature Algorithm (Ed25519)

with actual formal reduction to hardness of DLP (Seurin 2012).

Schnorr signature algorithm (1/2)

Parameters:

- a group $\mathcal{G}$ of prime order q for which the DLP is hard
- a generator g of $\mathcal{G}$
- a secure hash function $H : \{0,1\}^* \to [\![0, q[\![$

Keys:

- private $x \in]\!]0, q[\![$
- public $y = g^x$.

Schnorr signature algorithm (2/2)

Signature of a message m:

- choose random $k \in \rrbracket 0, q \llbracket$
- compute $e = H(g^k \| m)$, $s = k - xe$
- signature is (s, e)

Verification:

- Check if $H(g^s y^e \| m) \stackrel{?}{=} e$

Today

Concept

Implementation

Certificates

The problem with public keys

Bob can check Alice's signature provided he has her public key.

Alice can broadcast it publicly...

...but how to prevent man-in-the-middle attacks?

Back to square one! (again)

Certificate

A trusted third party (Trent) *certifies* the pair (Alice, k_d) by broadcasting:

"I, Trent, certify that Alice's public key is k_d."

Revised protocol

To sign a message m, Alice:

- computes $s = S(k_{\text{priv}}, m)$
- sends (m, s) along with her certificate for k_{pub}

Bob:

- checks that the certificate is valid
- verifies the signature using k_{pub}

Trust management

Two main approaches:

- web of trust (*e.g.* PGP)
- public-key infrastructure (*e.g.* X.509):

  chain of certification authorities (CAs), revocation lists . . .

NB: Certain fundamental problems remain (WYSIWYS?) for electronic signature

(Cours légal en France depuis 2000)

Trusted top level CAs

Linux: `/etc/ssl/certs`

Win10: `certmgr.msc`

Browsers: