People want to send messages to each other.

Confidentiality and/or authenticity may be necessary.

People have generated themselves asymmetric encryption / signature keys.

How can we find the public key that belongs to a certain person?

Something like a phone book is necessary…

Assume that each person has a (unique) name and when we want to talk about some person, we want to use his/her name.

- This assumption is probably wrong more often than it is correct.
- Still, the techniques we are going to see are generally applicable to other cases as well.

We want to bind names and public keys.

Let K_A be Alice's public key.

A certificate is a message

$$K_A \text{ is Alice's public key}$$

signed by the Certificate Authority (CA).

To get a certificate, Alice has to show up at CA in person and present some ID and her new public key.

Everybody trusts that CA is honest and performs the checks well.

The certificates

$$(K_A, A, attrs, sig_{K_{CA}}(K_A, A, attrs))$$

are published somehow.

The attributes *attrs* may include

- whether K_A is encryption or verification key;
- if K_A is verification key, then what kind of statements may be signed with K_A^{-1}.
- Validity period.
- ...

Different signature keys for different purposes help achieving reasonable security-availability tradeoffs.

The secret keys may leak before the validity period of the certificate is over.

Then we need to revoke the certificate.

Such certificates are included in Certificate revocation lists (CRL), also signed by the CA.

The CRL-s are periodically published. When checking a certificate, the latest CRL must be checked, too.

Over time, CRL-s may get large.

These days, everything is on-line. Instead of relying on CRL-s we may just ask CA for the validity of the certificate.

Let us have $sig_{K_A}(m)$ and the certificate binding K_A to Alice.

Sometime in the future the certificate will expire (or is revoked). What happens with the signature?

Will it cease to be valid?

That would be bad...

Given $sig_{K_A}(m)$ and $sig_{K_{CA}}(K_A, A, attrs)$, what happened before:

- signing m, or
- expiration of the certificate?

The goal of time-stamping is to associate a bit-string with a time-moment when that bit-string already "existed".

Originally, time-stamping was assumed to involve a trusted time-stamping authority (TSA).

On input m, TSA would return $(\tau, sig_{K_{TSA}}(m, \tau))$ where τ is the current time.

When K_{TSA} expires, all time-stamps lose validity. This is bad.

Beforehand, we were interested, which of the two events happened earlier.

Or, which of the two bit-strings existed earlier:

- $sig_{K_A}(m)$;
- revocation notice for K_A.

We were not interested in their actual times of creation.

Let $m_1, m_2, \ldots$ be the bit-strings that we want to time-stamp.

The i-th time-stamp L_i is $h(m_i, L_{i-1})$.

If $i < j$ then we can show that L_j was computed from L_i. The "proof" is $m_{i+1}, m_{i+2}, \ldots, m_j$.

L_i can be considered to be "later" than L_{i-1} because no other original can be computed for L_{i-1}.

We still need some time-stamping principal to facilitate the communication between different people generating m_i-s, but he does not have to be trusted.

This gives us an ordering on L_i-s.

This is not the same as the ordering on m_i-s yet.

However, if m_i is significant to some people, then somebody is interested to time-stamp it as soon as possible.

A message m_j can also be shown to be later than some L_i by incorporating L_i in m_j.

The proof that L_j is later than L_i has the size $\Theta(j - i)$. It can be reduced to $O(\log j)$ be defining $L_i = h(m_i, L_{i-1}, L_{p(i)})$ for a suitable chosen function p.

Could we get rid of certificates?

Could a person's name also be his/her public key? I.e. $K_A = A$?.

Yes.

In such systems, only the CA has an explicit public key.

The CA generates K_A^{-1} from A and K_{CA}^{-1}.

In such systems, the CA knows everybody's secret keys.

Is that bad? Yes, definitely.

But in usual systems, CA is trusted to give out certificates. This ability is (almost) as powerful...

Fiat-Feige-Shamir identification system

Let $n = pq$ be an RSA modulus, such that $p \equiv q \equiv 3 \pmod 4$. Let $t \in \mathbb{N}$ be fixed.

Then $\left(\frac{-1}{p}\right) = \left(\frac{-1}{q}\right) = -1$, hence -1 is a quadratic non-residue modulo n with $\left(\frac{-1}{n}\right) = 1$.

Key generation. pick $s_1, \ldots, s_t \in \mathbb{Z}_n$ and $b_1, \ldots, b_t \in \{0, 1\}$. Let $v_i = (-1)^{b_i} \cdot s_i^2 \bmod n$.

Public key: $(n, (v_1, \ldots, v_t))$. Secret key: $(s_1, \ldots, s_t)$.

n may be global. Prover may know that factorization, Verifier may not.

Commitment. Prover picks $r \in \mathbb{Z}_n$ and sends $x = r^2$ to the Verifier.

Challenge. Verifier picks $b_1, \ldots, b_t \in \{0, 1\}$ and sends them to the Prover.

Response. Prover sends $y = r \cdot \prod_{i=1}^{t} s_i^{b_i}$ to the Verifier.

Verification. Verifier checks that $y^2 = \pm x \cdot \prod_{i=1}^{t} v_i^{b_i}$.

(All computations in $\mathbb{Z}_n$)

If t is small then the protocol is zero-knowledge.

We are doing t rounds of Fiat-Shamir “in parallel”.

Signature schemes from identification systems

Let the messages in an identification system be

- Commitment $x(r)$ (computed from some random r which is kept secret),
- Challenge $b_1, \ldots, b_t$,
- Response $y(r, b_1 \cdots b_t, s)$.

Let $h : \{0,1\}^* \longrightarrow \{0,1\}^t$ be a collision-resistant hash function.

To sign m, generate a random r.
$sig_v(m) = (x(r), y(r, h(m \| x(r)), s))$.

Identity-based Fiat-Feige-Shamir signature scheme

Let CA generate the modulus $n = pq$ and keep p and q secret.

Let $J_n = \{m \in \mathbb{Z}_n \mid \left(\frac{m}{n}\right) = 1\}$.

Let $f : \{0,1\}^* \longrightarrow J_n$ be a fixed public function.

Key generation. For principal A, let $v_i = f(A, i)$. Everybody can compute v_i from A.

Let s_i be one of the square roots of either v_i or $-v_i$ mod n.

- Exactly one of v_i, $-v_i$ is quadratic residue modulo n.
- CA can compute the square roots.

There also exist asymmetric encryption schemes where the public key can be any bit-string.

(Google for identity-based encryption)

Again, the CA is used to create the secret decryption key. Hence CA can decrypt everything.

With such schemes, Alice can send encrypted messages to Bob even before Bob has contacted CA.