So far, we have only considered passive adversaries.

They keep their ears open and mouth shut.

For example, if

- A wants to send a secret message to B, and
- B thus sends his public key to A,

then the adversary does not attempt to replace that public key while in transit.

An active adversary might replace that public key by its own, then he can read what A has encrypted.

Digital signatures are a basic means to verify the source of a message.

A digital signature scheme is a tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where

- $\mathcal{P}$ is the set of possible messages;
- $\mathcal{A}$ is the set of possible signatures;
- $\mathcal{K}$ is the set of possible pairs of signature and verification keys (or "secret" and "public" keys);
- $\mathcal{S}$ contains a signature generation algorithm sig_K for each signature key;
- $\mathcal{V}$ contains a signature verification algorithm ver_K for each verification key.

For each $(K_s, K_v) \in \mathcal{K}$:

- $sig_{K_s} : \mathcal{P} \longrightarrow \mathcal{A}$;
- $ver_{K_v} : \mathcal{P} \times \mathcal{A} \longrightarrow \{\mathsf{true}, \mathsf{false}\}$.
- For each message $m \in \mathcal{P}$:

$$ver_{K_v}(m, sig_{K_s}(m)) = \mathsf{true} \quad .$$

The algorithms in $\mathcal{S}$ and $\mathcal{V}$ must be efficient.

Given m and K_v, it should be infeasible to generate such s, that $ver_{K_v}(m, s) = \mathsf{true}$.

A signature scheme is with appendix (*lisaga*) if $ver_{K_v}(m, s)$ actually uses m. Otherwise it is with message recovery (*sõnumit taastav*).

RSA signature scheme.

Generate large primes p, q. Let $n = pq$. Let $e \in \mathbb{Z}^*_{\varphi(n)}$. Let $d = e^{-1} \pmod{\varphi(n)}$.

Signature key: (n, d). Verification key: (n, e). $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$.

$$sig_{(n,d)}(m) = m^d \bmod n$$

$$ver_{(n,e)}(m, s) = \left[s^e \equiv m \pmod{n} \right]$$

Attacks:

Signatures to random messages The advesary may take an $s \in \mathcal{A}$ and compute $m = s^e \bmod n$. Then $ver_{(n,e)}(m, s) = \text{true}$.
Existential forgery.

Homomorphic properties If $ver_{(n,e)}(m_1, s_1) = \text{true}$ and $ver_{(n,e)}(m_2, s_2) = \text{true}$ then also

$$ver_{(n,e)}(m_1 m_2 \bmod n, s_1 s_2 \bmod n) = \text{true} \ .$$

Both attacks can be thwarted by adding redundancy to messages.

How to sign arbitrary messages?

Use collision-resistant hash functions.

A hash function is a mapping $h : \{0,1\}^* \longrightarrow \{0,1\}^n$ for some small n (typically $160 \leqslant n \leqslant 512$). It is

- one-way if given $y \in \{0,1\}^n$ it is infeasible to find any x, such that $h(x) = y$;
- 2nd preimage resistant if given x it is infeasible to find any $x' \neq x$, such that $h(x) = h(x')$;
- collision-resistant if it is infeasible to find any (x, x'), $x \neq x'$, such that $h(x) = h(x')$.

Note that we said "resistant" (*kindel*), not "free".

There exist hash functions which are (were) believed to be collision-resistant.

Collision-resistant hash functions can be constructed from secure symmetric encryption systems.

There exist hash functions which are collision-resistant under standard number-theoretic complexity assumptions.

We take a look at them in the next (or current?) lecture.

A collision-resistant hash function is also one-way.

Fix a collision-resistant hash function h whose output length is smaller than the length of the RSA modulus.

RSA signature generation is then

$$sig_{(n,d)}(m) = h(m)^d \bmod n$$

and verification is

$$ver_{(n,e)}(m, s) = \left[s^e \equiv h(m) \pmod{n}\right] \ .$$

The attacks described before would require inverting h to find a suitable message to go with the generated signature.

ElGamal signature scheme.

Fix a group G with hard discrete logarithm problem, $m = |G|$, g is a generator of G. We need a collision-resistant hash function $h : \{0,1\}^* \longrightarrow \mathbb{Z}_m$.

$\mathcal{P} = \{0,1\}^*$. $\mathcal{A} = G \times \mathbb{Z}_m$.

Key generation: randomly generate $\alpha \in \mathbb{Z}_m$. α is the signature key and $\chi = g^\alpha$ is the verification key.

To sign, generate a random $r \in \mathbb{Z}_m^*$.

$$sig_\alpha(m) = \mathbf{let}\ \tau = g^r\ \mathbf{in}\ (\tau, (h(m) - \alpha h(\tau))r^{-1} \bmod m)$$

$$ver_\chi(m,(\tau,s)) = \left[\chi^{h(\tau)} \cdot \tau^s = g^{h(m)}\right]$$

Signature verification works:

$$ver_\chi(m, (g^r, (h(m) - \alpha h(g^r))r^{-1})) =$$
$$\left[\chi^{h(g^r)} \cdot (g^r)^{h(m)-\alpha h(g^r))r^{-1}} = g^{h(m)}\right]$$

and

- $\chi^{h(g^r)} = g^{\alpha h(g^r)}$;
- $(g^r)^{h(m)-\alpha h(g^r))r^{-1}} = g^{h(m)}/g^{\alpha h(g^r)}$.

The ElGamal signing thus

- Generates a secret r and commits to it: publishes $\tau = g^r$.
- Somehow combines $h(m)$, $h(\tau)$, α and r. Publishes the result s.
  - We have $h(m) = \alpha \cdot h(\tau) + rs$.
- The equation involving $h(m)$, $h(\tau)$, α, r and s must be verifiable using only public data.

Exercise. Consider various ways of computing s. Judge the security of the system.

Security considerations:

The adversary cannot forge the signature by generating a random r and computing $\tau = g^r$.

Indeed, the adversary must then compute

$$s = (h(m) - \alpha h(\tau))r^{-1} \bmod m$$

and if he succeeds, he can also find α. This is equivalent to finding the discrete logarithm $\log_g \chi$.

Security considerations:

The random r must be kept secret. Otherwise the secret key α will be found from

$$s = (h(m) - \alpha h(\tau))r^{-1} \bmod m$$

Here m is the message and (τ, s) is the signature.

$$\alpha = (h(m) - rs)h(\tau)^{-1} \pmod m$$

Security considerations:

Different signatures must use different random r-s. Indeed, if m_1 and m_2 have signatures (τ, s_1) and (τ, s_2), where $\tau = g^r$ and

$$s_1 = (h(m_1) - \alpha h(\tau))r^{-1}$$
$$s_2 = (h(m_2) - \alpha h(\tau))r^{-1}$$

then we have a simple system of equations with two equations and two unknowns (r and α).

We get $r = (h(m_1) - h(m_2)) \cdot (s_1 - s_2)^{-1} \pmod{m}$ and find α as in the previous slide.

If $G = \mathbb{Z}_p^*$ (and then $m = p-1$) then $h(\tau)$ is usually taken to be τ. (or $\tau \bmod (p-1)$). In this case

$$sig_\alpha(m) = \mathbf{let}\ \tau = g^r\ \mathbf{in}\ (\tau, (h(m) - \alpha\tau) r^{-1} \bmod (p-1))$$

$$ver_\chi(m, (\tau, s)) = \left[\chi^\tau \cdot \tau^s \equiv g^{h(m)} \pmod p\right] \wedge \left[1 \leqslant \tau \leqslant p-1\right]$$

Security considerations:

The check $1 \leqslant \tau \leqslant p-1$ is necessary. Otherwise...

Suppose that the adversary knows the signature (τ, s) for the message m. It wants to sign the message m'. Let

- $u = h(m') \cdot (h(m))^{-1} \pmod{p-1}$;
- $s' = su \bmod (p-1)$;
- $\tau' \in \mathbb{Z}_{p(p-1)}$ satisfies (use CRT to find it)

$$\begin{cases} \tau' \equiv \tau u \pmod{p-1} \\ \tau' \equiv \tau \pmod{p} \end{cases}$$

Then (τ', s') will be accepted as a signature of m'.

Indeed,

$$\chi^{\tau'} \cdot \tau'^{s'} \equiv \chi^{\tau u} \cdot \tau^{su} =$$

$$(\chi^{\tau} \cdot \tau^{s})^{u} \equiv (g^{h(m)})^{\frac{h(m')}{h(m)}} = g^{h(m')} \pmod{p}$$

If $1 \leqslant \tau' \leqslant p-1$ then $\tau = \tau u$, hence $u = 1$ and $h(m) = h(m')$. If $m \neq m'$ then we have a collision.

Security considerations: If $\mathcal{P} = \mathbb{Z}_p^*$ and $h(m) = m$ then existential forgeries are possible.

The adversary has to generate (m, τ, s), such that

$$\chi^\tau \cdot \tau^s \equiv g^m \pmod{p} \ .$$

It generates $u \in \mathbb{Z}$, $v \in \mathbb{Z}_{p-1}^*$ and defines $\tau = g^u \chi^v$, $s = -\tau v^{-1} \pmod{p-1}$ and $m = su \bmod (p-1)$. Then

$$\chi^\tau \cdot \tau^s = \chi^\tau (g^u \chi^v)^{-\tau v^{-1}} = \chi^\tau g^{-u\tau v^{-1}} \chi^{-v\tau v^{-1}} = g^{-u\tau v^{-1}}$$

and $-u\tau v^{-1} = us \equiv m \pmod{p-1}$.

Efficiency considerations. The signing with ElGamal is fast (requires just a couple of multiplications).

- $\tau = g^r$ can be precomputed.

To verify, we must compute $\chi^{h(\tau)}$, τ^s and $g^{h(m)}$ — three exponentiations.

Fortunately, computing $a_1^{e_1} \cdots a_k^{e_k}$ (product of powers) can be done faster than k exponentiations.

We can then compute $\chi^{h(\tau)} \cdot \tau^s \cdot (g^{-1})^{h(m)}$ and compare it to $1 \in G$.

We want to compute $a_1^{e_1} \cdots a_k^{e_k}$.

Let $e_i = \sum_{j=0}^{n} e_{ij} 2^j$, where $e_{ij} \in \{0, 1\}$ and $n \geqslant \max_i \log_2 e_i$.

Define $b_{n+1} = 1$ and

$$b_r = b_{r+1}^2 \cdot \prod_{\substack{1 \leqslant i \leqslant k \\ e_{ir}=1}} a_i \quad .$$

Then

$$b_r = \prod_{i=1}^{k} a_i^{\sum_{j=r}^{n} e_{ij} 2^{j-r}} \quad .$$

And b_0 is the product that we are looking for.

We precompute all products $\prod_{i \in X} a_i$ for $X \subseteq \{1, \ldots, k\}$. This requires $2^k - k - 1$ multiplications.

Computing b_r from b_{r+1} requires two multiplications (sometimes one). Computing b_0 requires $\approx 2n$ multiplications (without precomputation).

A usual exponentiation requires n to $2n$ multiplications.

If k is small, such that also 2^k is small then the simultaneous exponentiation is almost as fast as a simple exponentiation.

Exercise. Consider ElGamal signature scheme in the group $\mathbb{Z}_p^*$ where p and $q = \frac{p-1}{2}$ are prime numbers. Show that if the adversary can choose the generator g of $\mathbb{Z}_p^*$ then it can also forge signatures.

Hint: The adversary chooses some $t \in \mathbb{Z}_{p-1}^*$ and fixes $g = q^t \bmod p$ (if q does not generate $\mathbb{Z}_p^*$ then the attack is unsuccessful). In a forged signature (τ, s), the adversary sets $\tau = q$.

Digital signature algorithm.

Proposed by the U.S. National Institute of Standards and Technology.

A simple variant of ElGamal signature scheme in a subgroup of $\mathbb{Z}_p^*$.

Let q be a 160-bit prime number and p a 512-(or 768-, or 1024-)bit prime number, such that $q \mid (p-1)$.

I.e. we consider only numbers of the form $2tq+1$ for suitably sized t when doing the prime number generation for p.

Let $g \in \mathbb{Z}_p^*$ have the order q. Let G be generated by g.

I.e. raise a generator of $\mathbb{Z}_p^*$ to the power $\frac{p-1}{q}$.

Exercise. What if we do not know a generator of $\mathbb{Z}_p^*$?

The group G should have hard-to-compute discrete logarithms.

Indeed, it is too large to use generic algorithms, and p is too large to use algorithms specifically for $\mathbb{Z}_p^*$.

The quantities q, p and g may global, or may be chosen for each key separately.

Key generation: randomly generate $\alpha \in \mathbb{Z}_q$. α is the signature key and $\chi = g^\alpha \in \mathbb{Z}_p^*$ is the verification key.

To sign m, choose a random $r \in \mathbb{Z}_q^*$.

- Let $\tau = (g^r \bmod p) \bmod q \in \mathbb{Z}_q$.
- Let $s = r^{-1}(h(m) + \alpha\tau) \bmod q \in \mathbb{Z}_q$.
- Return (τ, s).

To verify that (τ, s) is a signature of m,

- Verify that $0 < \tau, s < q$.
- Let $u_1 = s^{-1}h(m) \bmod q$ and $u_2 = \tau s^{-1} \bmod q$.
- Verify that $\tau = (g^{u_1}\chi^{u_2} \bmod p) \bmod q$.

Exercise. Verify that signature verification works, unless $\tau = 0$ or $s = 0$, which should occur extremely rarely.

Exercise. Consider the following signature scheme: Let p be a prime and g a generator of $\mathbb{Z}_p^*$ (both public). Let h be a collision-resistant hash function from $\{0,1\}^*$ to $\mathbb{Z}_{p-1}^*$.

Secret key is some $\alpha \in \mathbb{Z}_{p-1}$. The public key is $\chi = g^\alpha \bmod p$.

To sign a message m, compute $z \in \mathbb{Z}_{p-1}$ so, that $z \cdot h(m) \cong \alpha \pmod{p-1}$. The signature s of m is g^z. To verify the signature s of m, check that $s^{h(m)} = \chi$.

Show that the scheme works.

Is that scheme secure?

A hash function is a function $h : \{0,1\}^* \to \{0,1\}^n$ for some fixed n, such that h is easy to compute.

A compression function is a function $h : \{0,1\}^m \to \{0,1\}^n$ for some fixed m and n, such that $m > n$ and h is easy to compute.

Before we listed the properties "one-wayness", "2nd preimage resistance" and "collision-resistance".

Theorem. If a hash or compression function $h : X \to Z$ (here $|X| \geqslant 2|Z|$) is not one-way (in certain sense), then it is not collision-resistant.

Proof. Let $\mathcal{A}$ be an algorithm, such that $\mathcal{A}(y)$ returns some $x \in h^{-1}(y)$.

I.e. for any $y \in Z$, the probability $\Pr[h(\mathcal{A}(y)) = y]$ is significant.

To generate a collision,

- pick a random $x \in X$.
- Let $x' = \mathcal{A}(h(x))$.
- If $x \neq x'$ then output (x, x'), else fail.

The probability of failure is $|Z|/|X| \leqslant 1/2$.

We required from $\mathcal{A}$ that for all $y \in Z$, the probability

$$\Pr[h(x) = y \mid x \leftarrow \mathcal{A}(y)]$$

is significant.

Alternatively, we might have required that just

$$\Pr[h(x) = y \mid y \in_R Z, x \leftarrow \mathcal{A}(y)]$$

is significant.

The non-existence of such $\mathcal{A}$ is a more reasonable definition of one-wayness...

But then the theorem on previous slide no longer holds.

Indeed, let $h : \{0,1\}^* \to \{0,1\}^n$ be a collision-resistant hash function.

Define $h' : \{0,1\}^* \to \{0,1\}^{n+1}$ by

$$h'(x) = \begin{cases} 1 \,||\, x, & \text{if } |x| = n \\ 0 \,||\, h(x), & \text{otherwise} \end{cases} .$$

then h' is collision-resistant, but for half of the values $y \in \{0,1\}^{n+1}$, it is very easy to find an element of ${h'}^{-1}(y)$.

Exercise. Show that if a hash function is not 2nd preimage resistant, then it is not collision-resistant.

A generic way to find a collision of a hash function h is to compute the values $h(x)$ for random x-s until a collision is found.

If the values $h(x)$ are n bits long then $O(2^{n/2})$ attempts are necessary, by the birthday paradox.

This attack is called the birthday attack.

That's why the output of modern hash functions (MD5, SHA-1, etc.) are at least 128, and preferably 160 bits long.

A generalization of the birthday paradox: let X be a set, $|X| = n$. Let $x_1, \ldots, x_k$ and $y_1, \ldots, y_l$ be mutually independent uniformly distributed random variables over X. The probability that there exist such i and j, that $x_i = y_j$, is about

$$1 - \prod_{i=1}^{k} \prod_{j=1}^{l} \frac{n-1}{n} = 1 - (1 - \frac{1}{n})^{kl} \geqslant 1 - e^{-\frac{kl}{n}} \geqslant \frac{1}{2}$$

if $e^{-\frac{kl}{n}} \leqslant \frac{1}{2}$, i.e. $kl \leqslant n \ln 2 = O(n)$.

Let x and x' be two meaningful documents. The attacker may choose $n/2$ "places" in both of them where it may or may not make a modification that does not change the meaning of the document.

We get $2^{n/2}$ variants of the document x and $2^{n/2}$ variants of the document x'. With significant probability, $h(\bar{x}) = h(\bar{x'})$ for some variant $\bar{x}$ of x and $\bar{x'}$ of x'.

(Yuval's attack.)

Assume that $h : \{0,1\}^m \to \{0,1\}^n$ is a collision-resistant compression function. Let $r = m - n$. We can construct a collision-resistant hash function $h^* : \{0,1\}^* \to \{0,1\}^n$ as follows.

Let $\kappa : \{0,1\}^* \to (\{0,1\}^r)^*$ be an encoding function that is

- easily computable and easily invertible;
- suffix-free — if $x \neq x'$ then neither of $\kappa(x)$ and $\kappa(x')$ is a suffix of the other.

Exercise. Construct such κ. Try to keep the increase in length as small as possible.

Let $x \in \{0,1\}^*$ and let $(x_1, \ldots, x_t) = \kappa(x)$, where $x_1, \ldots, x_t \in \{0,1\}^r$.

Let H_0 be the string of n bits 0. Construct $H_1, \ldots, H_t$ as follows:

$$H_i = h(H_{i-1} \,||\, x_i) \ .$$

We define $h^*(x) = H_t$.

This is called the Merkle-Damgård construction.

Theorem. If h is a collision-resistant compression function then h^* is a collision-resistant hash function.

Proof. We show how to efficiently construct a collision of h from a collision of h^*.

Let $x = x'$ but $h^*(x) = h^*(x')$. Let $(x_1, \ldots, x_t) = \kappa(x)$ and $(x'_1, \ldots, x'_{t'}) = \kappa(x')$. Assume w.l.o.g. that $t \leqslant t'$.

Compute $H_0, \ldots, H_t$ (from $\kappa(x)$) and $H'_0, \ldots, H'_{t'}$ (from $\kappa(x')$). We have $H_t = H'_{t'}$. There are two cases:

1. There exists an $i \in \{1, \ldots, t\}$, such that $H_{t-i} \neq H'_{t'-i}$. Let i be the smallest with such property.
2. $H_t = H'_{t'}$, $H_{t-1} = H'_{t'-1}$,$\ldots$, $H_0 = H'_{t'-t}$.

In the first case we have

- $H_{t-i} \,||\, x_{t-i+1} \neq H'_{t'-i} \,||\, x'_{t'-i+1}$;
- $h(H_{t-i}||x_{t-i+1}) = H_{t-i+1} = H'_{t'-i+1} = h(H'_{t'-i}||x'_{t'-i+1})$.

a collision for h.

In the second case there are again two cases:

1. There exists an $i \in \{0, \ldots, t-1\}$, such that $x_{t-i} \neq x'_{t'-i}$. Let i be the smallest with such property.
2. $x_t = x'_{t'}$, $x_{t-1} = x'_{t'-1}$,$\ldots$, $x_1 = x'_{t'-t+1}$.

In the first case we have

- $H_{t-i-1} \,||\, x_{t-i} \neq H'_{t'-i-1} \,||\, x'_{t'-i}$;
- $h(H_{t-i-1} \,||\, x_{t-i}) = H_{t-i} = H'_{t'-i} = h(H'_{t'-i-1} \,||\, x'_{t'-i})$.

a collision for h.

In the second case $\kappa(x)$ is a suffix of $\kappa(x')$. This is impossible by the construction of κ.

Exercise. Let $h : \{0,1\}^{2n} \to \{0,1\}^n$ be a collision-resistant compression function. Show that $h' : \{0,1\}^{4n} \to \{0,1\}^n$, where

$$h'(x_1 \| x_2 \| x_3 \| x_4) = h(h(x_1 \| x_2) \| h(x_3 \| x_4))$$

for $x_1, x_2, x_3, x_4 \in \{0,1\}^n$, is collision-resistant, too.

Exercise. Let T be a binary tree with k leaves (and no vertices with exactly one child). Let $h : \{0,1\}^{2n} \to \{0,1\}^n$ be collision-resistant. Define $h_T : \{0,1\}^{kn} \to \{0,1\}^n$ by

$$h_\bullet(x) = x$$

$$h_{T_L \to \bullet \leftarrow T_R}(x_1 \| \cdots \| x_k) = h(h_{T_L}(x_1 \| \cdots \| x_{k_L}) \| h_{T_R}(x_{k_L+1} \| \cdots \| x_k)) \ .$$

Show that $h_T : \{0,1\}^{kn} \to \{0,1\}^n$ is collision-resistant.

Exercise. Define a function $h^* : (\{0,1\}^n)^* \to \mathbf{BTree} \times \{0,1\}^n$ as follows:

- On input $x_1 \| \cdots \| x_k$, pick a binary tree T with k vertices.
- Return $(T, h_T(x_1 \| \cdots \| x_k))$.

Show that h^* is not collision-resistant.

Exercise. For each $k \in \mathbb{N}$, let $\mathcal{T}(k)$ be some binary tree with k leaves. Let $h^* : (\{0,1\}^n)^* \to \{0,1\}^n$ be defined by

$$h^*(x_1 \| \cdots \| x_k) = h_{\mathcal{T}(k)}(x_1 \| \cdots \| x_k) \ .$$

Show that h^* is collision-resistant.

Chaum - van Heijst - Pfitzmann compression function is defined as follows:

Let $p \in \mathbb{P}$ be a strong prime (i.e. $q = \frac{p-1}{2}$ is also a prime). Let g be a generator of $\mathbb{Z}_p^*$ and let χ be a random element of $\mathbb{Z}_p^*$.

Define $h : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_p^*$ by

$$h(x, y) = g^x \chi^y \bmod p \ .$$

Theorem. If the discrete logarithm problem is hard in $\mathbb{Z}_p^*$ then h is collision-resistant.

Proof. Assume that we know a collision for h. Then we can find $z = \log_g \chi$ as follows.

Let $h(x_1, y_1) = h(x_2, y_2)$, but $(x_1, y_1) \neq (x_2, y_2)$. We have

$$g^{x_1 - x_2} \equiv \chi^{y_2 - y_1} \pmod{p}$$

or

$$x_1 - x_2 \equiv z(y_2 - y_1) \pmod{p - 1}$$

We solve this congruence for z (it must have at least one solution) and try out all possible solutions (raise g to that power and compare the result to χ).

The procedure on the previous slide fails if the congruence

$$x_1 - x_2 \equiv z(y_2 - y_1) \pmod{p-1}$$

has too many solutions.

It has $\gcd(y_2 - y_1, p-1) = \gcd(y_2 - y_1, 2q)$ solutions. As $y_1, y_2 < q$, then also $|y_2 - y_1| < q$ and this gcd can be either 1 or 2.

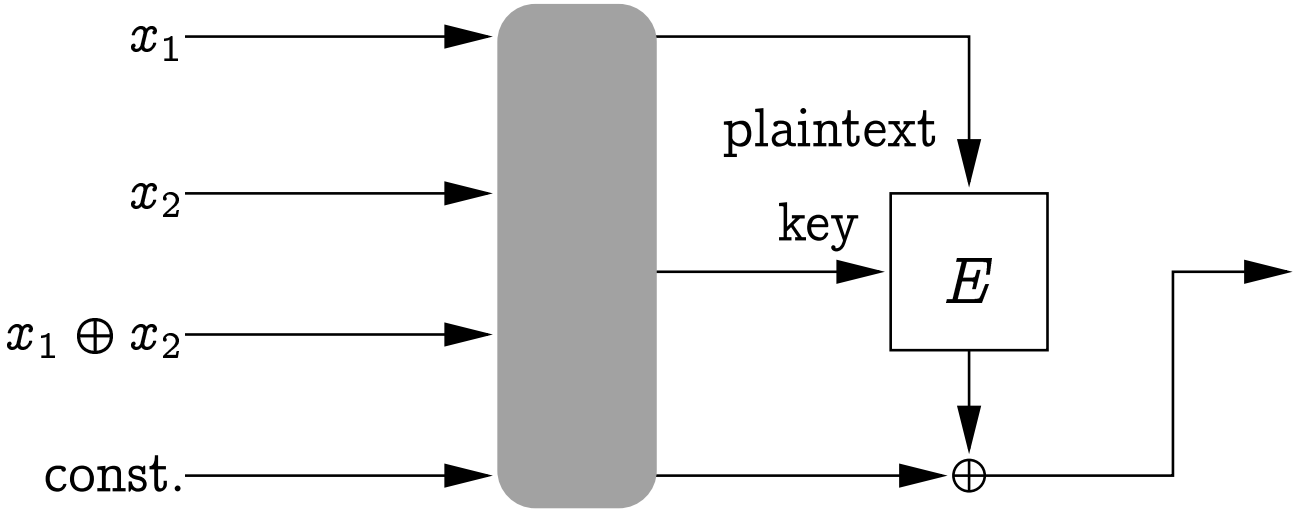
Unfortunately, the Chaum - van Heijst - Pfitzmann compression function is slow.

Exercise. Suppose that a group of principals wants to use the Chaum - van Heijst - Pfitzmann compression function among each other. They choose a strong prime p and a generator g of $\mathbb{Z}_p^*$. How can they pick χ, such that none of them knows $\log_g \chi$?

Exercise. Let n be an RSA-modulus (with unknown factorization). Let $g \in \mathbb{Z}_n^*$ be of maximum order. Let $h : \{1, \ldots, n^2\} \to \mathbb{Z}_n^*$ be defined by $h(x) = g^x \bmod n$. Show that h is a collision-resistant hash function.

Compression functions may be constructed from block ciphers.

Let a block cipher be given, with $\mathcal{P} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$ for some n. Then we can construct
$h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ as follows. $h(x_1, x_2)$ is



I.e. there are 64 possibilities. Most of them are not collision-resistant.

Among those 64 functions, there are

- 12 collision-resistant functions;
- 8 functions, which are not collision resistant, but a hash function, constructed from it using the Merkle-Damgård construction, is secure;
- 44 “useless functions”.

The security proof assumes that E is a *randomly chosen* block cipher.

Let $\mathbf{E}$ be the set of all functions E of type

$$\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$$

such that $E(k,\cdot)$ is a permutation of $\{0,1\}^n$ for each $k \in \{0,1\}^n$.

The security proof assumes that E is uniformly randomly chosen from $\mathbf{E}$.

This is a strong assumption.

That also means that the attacker may not "look inside" E. It only has oracle access to E and D (decryption).

Collision-resistance of a construction means that

- for any efficient algorithm $\mathcal{A}$
  - which may call E and D, but has no further description of them;
- for a randomly chosen E
  - which also determines D
- the probability that $\mathcal{A}^{E,D}$ outputs a collision of h is negligible.

```
interface BlockCipher {
    block encrypt(block k, block pt);
    block decrypt(block k, block ct);
}
```

```
class StrongRandPerm {
    private SetOfBlockPairs S;
    StrongRandPerm() { S := ∅ }
    block enc(block pt) {
        if ∃ct : (pt, ct) ∈ S then {
            return ct;
        } else {
            do{
                ct := random_block();
            }while(∃pt' : (pt', ct) ∈ S);
            S := S ∪ {(pt, ct)};
            return ct;
        } // if
    } // enc
    block dec(block ct) {
        if ∃pt : (pt, ct) ∈ S then {
            return pt;
        } else {
            do{
                pt := random_block();
            }while(∃ct' : (pt, ct') ∈ S);
            S := S ∪ {(pt, ct)};
            return pt;
        } // if
    } // dec
} // StrongRandPerm
```

```
class RandomCipher implements BlockCipher {
    FiniteMap⟨block → StrongRandPerm⟩ F;
    RandomCipher() { F := {}; }
    block encrypt(block k, block pt) { return getPerm(k).enc(pt) }
    block decrypt(block k, block ct) { return getPerm(k).dec(pt) }
    StrongRandPerm getPerm(block k) {
        if k ∉ domain(F) then {
            F := F{k ↦ new StrongRandPerm()};
        }
        return F(k);
    }
}
```

These 12 collision-resistant compression functions are

$$E(x_1, x_2) \oplus x_2 \qquad E(x_1 \oplus x_2, x_2) \oplus x_2$$
$$E(x_1, x_1 \oplus x_2) \oplus x_2 \qquad E(x_1, x_2) \oplus x_1 \oplus x_2$$
$$E(x_1 \oplus x_2, x_2) \oplus x_1 \qquad E(x_1, x_1 \oplus x_2) \oplus x_1 \oplus x_2$$

and six others, where we swap x_1 and x_2.

Exercise. Show that other 52 compression functions are not collision-resistant.

These 8 non-collision-resistant compression functions giving collision-resistant hash functions are

$$E(x_1 \oplus x_2, x_2) \oplus c \qquad E(x_1 \oplus x_2, x_2) \oplus x_1 \oplus x_2$$

$$E(x_2, x_1) \oplus c \qquad E(x_1 \oplus x_2, x_1) \oplus c$$

$$E(x_2, x_1) \oplus x_2 \qquad E(x_1 \oplus x_2, x_1) \oplus x_1 \oplus x_2$$

$$E(x_2, x_1 \oplus x_2) \oplus c \qquad E(x_2, x_1 \oplus x_2) \oplus x_2 \ .$$

In Merkle-Damgård construction, x_1 is the accumulated value and x_2 the next message block.

PROOF of the collision-resistance of $E(x_1, x_2) \oplus x_2$:

Consider the execution of $\mathcal{A}$.run(**new** RandomCipher()) where we record all queries and answers to the methods "encrypt" and "decrypt".

Let the plaintexts, keys and ciphertexts be (x_i, k_i, y_i), $1 \leqslant i \leqslant q$. Assume that all these triples are different.

A bad event happens if $x_i \oplus y_i = x_j \oplus y_j$ for some $i < j$.

At j-th query, x_j or y_j was randomly chosen from a set of size $2^n - (j-1)$.

The probability that it equals $x_i \oplus y_i \oplus y_j$ or $x_i \oplus y_i \oplus x_j$ is $\frac{1}{2^n - j + 1}$.

The sum (over i and j) of these probabilities is $O(q^2/2^n)$.

Existing dedicated hash functions (MD5, SHA-1, RIPEMD, their longer versions) are also constructed by Merkle-Damgård construction. One has to specify

- the encoding (or padding) function κ;
- the compression function.

Exercise. Let h_1 and h_2 be two hash functions. Let $h(x) = h_1(h_2(x))$. Judge the 2nd preimage resistance of h if

- both h_1 and h_2 are second preimage resistant;
- only h_1 is second preimage resistant;
- only h_2 is second preimage resistant.

In the symmetric setting we have seen encryption.

In the asymmetric setting we have seen encryption and signing.

What is the analogue to digital signatures in the symmetric setting?

Message authentication codes (MACs).

- Two parties share a secret key.
- One party can use that key to prove to the other one that the message was not modified during transit.
- This is hopefully more efficient than signing the message.

A MAC has the following components

- Plaintext space $\mathcal{P}$;
- Authentication code space $\mathcal{A}$;
- Key space $\mathcal{K}$;
- tagging algorithm $sig : \mathcal{K} \times \mathcal{P} \to \mathcal{A}$;
- verification algorithm $ver : \mathcal{K} \times \mathcal{P} \times \mathcal{A} \to \{\mathsf{true}, \mathsf{false}\}$.

$$ver_K(m, sig_K(m)) = \mathsf{true}$$

must hold.

If sig is deterministic then ver is already specified, too.

Security — the adversary (not knowing the key) cannot produce message-tag pairs that are accepted by ver.

```
interface MAC {
    Tag mkTag(Plaintext m);
    bool verify(Plaintext m, Tag t);
}
```

A message authentication code is (t, ε)-existentially unforgeable under chosen message attacks if no adversary working in time at most t can distinguish the following two implementations of the interface MAC with the advantage larger than ε:

```
class MACExperiment0 implements MAC {
    Key k;
    MACExperiment0() { k := 𝒦(); }
    Tag mkTag(Plaintext m) { return sig(k, m); }
    bool verify(Plaintext m, Tag t) { return ver(k, m, t); }
}
```

```
class MACExperiment1 implements MAC {
    Key k;
    SetOfTexts S;
    MACExperiment1() { k := 𝒦(); S := ∅; }
    Tag mkTag(Plaintext m) {
        S := S ∪ {m};
        return sig(k, m);
    }
    bool verify(Plaintext m, Tag t) {
        return m ∈ S ∧ ver(k, m, t);
    }
}
```

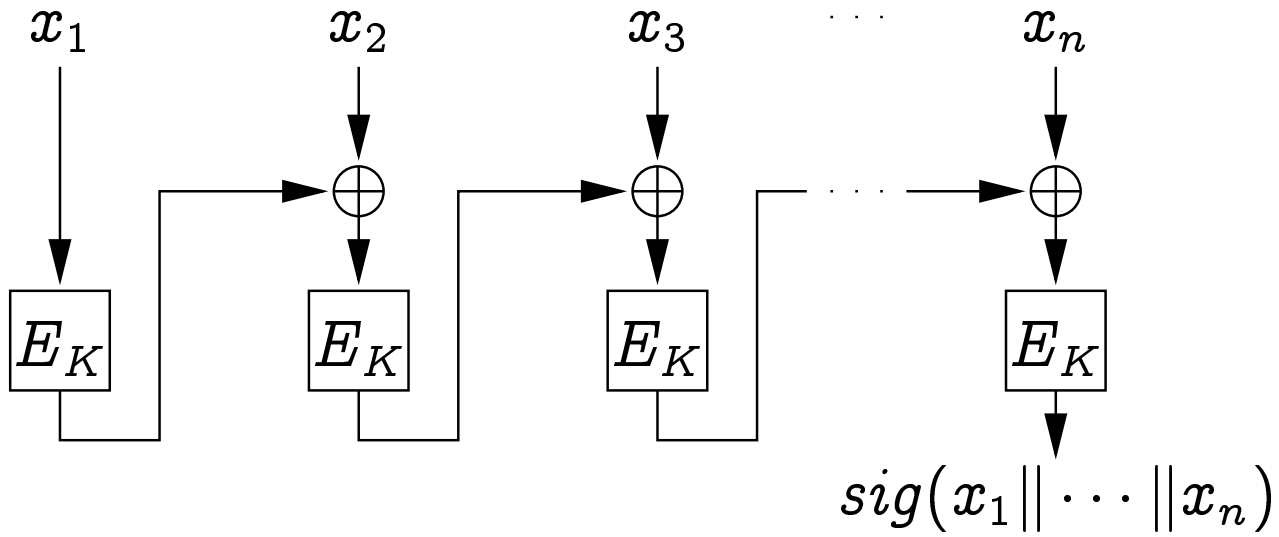
Exercise. Show that a random function is a secure MAC.

A random function — like the class RandFunc we had before, but the argument of the function may be any bit-string.

CBC-MAC: Let a block cipher E be given. Let l be the block size.

$\mathcal{K}$ of MAC is $\mathcal{K}$ of the block cipher.

$\mathcal{P} = (\{0,1\}^l)^*$. $\mathcal{A} = \{0,1\}^l$. Let $x_i \in \{0,1\}^l$.



Exercise. Break it.

The attack used the fact that when computing the MAC of a message, we sort of also compute the MACs of its suffixes.

The construction would have been secure if the length of messages had been fixed.

For variable-length messages x, we may start by computing

$$x' = \kappa(x)$$

where κ is a suffix-free function, and then apply the CBC-MAC construction to x'.

Does it yield a secure MAC?

That depends on κ.

It is not sufficient to define

$$\kappa(x_1 \cdots x_n) = (x_1, \ldots, x_n, n)$$

where $x_1, \ldots, x_n \in \{0,1\}^l$.

Let $b, b', c \in \{0,1\}^*$. Then

$$sig_K(b) = E_K(1 \oplus E_K(b))$$
$$sig_K(b') = E_K(1 \oplus E_K(b'))$$

$$sig_K(b\|1\|c) = E_K(3 \oplus E_K(c \oplus E_K(1 \oplus E_K(b)))) =$$
$$E_K(3 \oplus E_K(c \oplus sig_K(b))) =$$
$$E_K(3 \oplus E_K(c \oplus E_K(c \oplus sig_K(b) \oplus sig_K(b') \oplus sig_K(b'))) =$$
$$E_K(3 \oplus E_K(c \oplus E_K(c \oplus sig_K(b) \oplus sig_K(b') \oplus E_K(1 \oplus E_K(b')))) =$$
$$sig_K(b'\|1\|c \oplus sig_K(b) \oplus sig_K(b'))$$

If sig denotes the ordinary CBC-MAC (without κ) then the following variants are also secure wrt. variable-length messages:

- $sig_K(n \| x_1 \| \cdots \| x_n)$;
- $sig_{sig_K(n)}(x_1 \| \cdots \| x_n)$;
- $sig_{K'}(sig_K(x_1 \| \cdots \| x_n))$.

The last one is attractive in that we do not have to know the length n of the message in advance.

Birthday attack (for CBC-MAC with message length $n \cdot l$)

Let $a_1, \ldots, a_m \in \{0,1\}^l$ be distinct. Let $r_1, \ldots, r_m \in \{0,1\}^l$ be independent, uniformly distributed random variables.

If $m \approx 2^{l/2}$ then with significant probability there exist i and j, such that $i \neq j$ and $E_K(a_i) \oplus r_i = E_K(a_j) \oplus r_j$.

Then

$$sig_K(a_i \| (r_i \oplus c) \| x_3 \| \cdots \| x_n) = sig_K(a_j \| (r_j \oplus c) \| x_3 \| \cdots \| x_n)$$

for any $c, x_3, \ldots, x_n \in \{0,1\}^l$. These two messages differ because $a_i \neq a_j$.

How to verify that $E_K(a_i) \oplus r_i = E_K(a_j) \oplus r_j$?

Check that $sig_K(a_i \| r_i \| 0^{(n-2)l}) = sig_K(a_j \| r_j \| 0^{(n-2)l})$.

(Our adversary is active)

XOR-MAC:

- Split the message into blocks
- (do something with each block)
- Encrypt the blocks
- XOR the result together.

Simplest version:

$$sig_K(x_1 \| \cdots \| x_n) = E_K(x_1) \oplus \cdots \oplus E_K(x_n)$$

Exercise. Break it.

Next version:

Let $m < l$ where l was the block size. Let $\overline{i}$ denote the representation of the integer i as a m-bit string.

Split the message x to $(l-m)$-bit blocks $x_1, \ldots, x_n$.

$$sig_K(x_1 \| \cdots \| x_n) = E_K(\overline{1} \| x_1) \oplus E_K(\overline{2} \| x_2) \oplus \cdots \oplus E_K(\overline{n} \| x_n)$$

Exercise. Break it.

A secure version:

- Split the message x to $(l-m-1)$-bit blocks $x_1, \ldots, x_n$.
- Generate a random $r \in \{0,1\}^{l-1}$.
  - Actually, r only has to be fresh.
- Let

$$\tau = E_K(0\|r) \oplus E_K(1\|\overline{1}\|x_1) \oplus E_K(1\|\overline{2}\|x_2) \oplus \cdots \oplus E_K(1\|\overline{n}\|x_n) \ .$$

- $sig_K(x) = (r, \tau)$.

Exercise. How does the verification algorithm look like?

Obviously we may not reuse a generated r.

A universal one-way hash function (*universaalne ühesuunaline paiskfunktsioon*) is a finite family $\mathcal{H}$ of functions $h : D \longrightarrow R$ (for certain sets D and R), such that

- for every $x, x' \in D$, where $x \neq x'$
- for a uniformly randomly chosen h from $\mathcal{H}$
- $\Pr[h(x) = h(x')] \leqslant 1/|R|$.

Usually, there are keys to refer to the elements of $\mathcal{H}$.

$$\mathcal{H} = \{h_K \mid K \in \mathcal{K}_{\mathcal{H}}\}$$

Let $E_K : \{0,1\}^l \to \{0,1\}^l$ be a block cipher, let the set of keys be $\mathcal{K}_E$.

Let $\mathcal{H}$ be a universal one-way hash function from $\{0,1\}^n$ to $\{0,1\}^l$ $(n \geqslant l)$.

The following is a secure MAC:

- $\mathcal{P} = \{0,1\}^n$;
- $\mathcal{A} = \{0,1\}^l$;
- $\mathcal{K} = \mathcal{K}_\mathcal{H} \times \mathcal{K}_E$;
- $sig_{K_1,K_2}(x) = E_{K_2}(h_{K_1}(x))$.

Example of a universal one-way hash function from D to R:

- Let D be a finite field.
  - If $D = \{0,1\}^n$, then consider D as the set of polynomials of degree less than n over $\mathbb{Z}_2$, *modulo* some irreducible n-th degree polynomial.
- Let $g : D \to R$ be a mapping, such that the sets $g^{-1}(r)$ for $r \in R$ all have the same size.
  - If $R = \{0,1\}^l$, where $l \leqslant n$, then pick certain l bits out of n.
- Let $\mathcal{K} = D^* \times D$.
- Let $h_{a,b}(x) = g(ax + b)$.

Encryption and MAC together are used to create secure (confidential and authentic) channels.

Initially two parties exchange the encryption key K_e and the MAC key K_a.

Then, to transmit x, one sends $sig_{K_a}(E_{K_e}(x))$.

Certain other ways, e.g. $(E_{K_e}(x), sig_{K_a}(x))$ and $(E_{K_e}(sig_{K_a}(x))$ are insecure in general, but secure for specific MACs.

There exist block ciphers' modes of operation that provide both confidentiality and authenticity.

Use a single key.

They need $n + O(1)$ block cipher invocations to encrypt and authenticate a n-block message.

See `http://www.cs.ucdavis.edu/~rogaway/ocb`

Also see signcryption for asymmetric primitives giving both confidentiality and integrity.