Advanced Cryptography

Mihai Prunescu*

Contents

*University of Bucharest, Faculty of Mathematics and Informatics; and Simion Stoilow Institute of Mathematics of the Romanian Academy, Research unit 5, P. O. Box 1-764, RO-014700 Bucharest, Romania.
mihai.prunescu@imar.ro, mihai.prunescu@gmail.com

Part I
Algebra

1 Groups

Definition: The structure $(G, \cdot, 1)$ is a group if and only if it satisfies the following axioms:

1. Associativity: $\forall\, x, y, z\ (xy)z = x(yz)$.
2. Neutral element: $\forall\, x\ x1 = 1x = x$.
3. Inversibility: $\forall\, x\ \exists\, y\ xy = yx = 1$.

□

Definition: If $(G, \cdot, 1)$ is a group and $1 \in H \subseteq G$ a subset, H is a subgroup of G if $(H, \cdot, 1)$ is a group. We write $H \leq G$. □

One sees that if $x, y \in G$ and $H \leq G$ subgroup, then $xH = yH$ or $xH \cap yH = \emptyset$. So if G is finite, then $|H|\,|\,|G|$.

Fundamental example: Let A be a set and $S(A)$ be the set of all bijective functions $f : A \to A$. Then $(S(A), \circ, 1 = id_A)$ is a group, called group of all permutations of A. If $|A| = n$, we denote $S(A)$ with S_n. We observe that $|S_n| = 1 \cdot 2 \cdot \cdots \cdot n = n!$

Cayley's Theorem: *Let G be a group. Then $G \leq S(G)$.*

Proof: To some $g \in G$ we associate $f_g : G \to G$ given as $f_g(x) = gx$. The function f_g is bijective, so is in $S(G)$. One sees that $f_1 = id$, $f_g \circ f_h = f_{gh}$ and that if $f_{g_1} = f_{g_2}$ then $g_1 = g_2$. □

Definition: A subgroup $H \leq G$ is called normal if and only if $\forall\, x \in G\ xHx^{-1} = H$. A normal subgroup is denoted $H \trianglelefteq G$.

If $H \trianglelefteq G$ then the set of classes $\{xH \,|\, x \in G\}$ form a group with the operation $xH \cdot yH = (xy)H$, which proves to be well defined. This group is denoted G/H, the quotient group of G by H.

The embedding given by Cayley's Theorem does not generally embed G normally in $S(G)$. Such situations are very special.

Definition: Let G_1 and G_2 be two groups. A function $h : G_1 \to G_2$ is a group homomorphism if $h(1) = 1$, and for every $x, y \in G_1$, $h(xy) = h(x)h(y)$. The set $\{x \in G_1 \,|\, h(x) = 1\}$ is the kernel of h and is denoted $Ker\ h$.

Isomorphism Theorem: *If $h : G_1 \to G_2$ is a group homomorphism, then $Ker\ h \trianglelefteq G_1$ and:*

$$G_1/Ker\ h \simeq Im\ h \leq G_2.$$

The proof does not need other facts as the definitions given above. □

Definition: If $A \subset G$, the group generated by A, denoted $\langle A \rangle$, is the smallest subgroup of G which contains A:

$$\langle A \rangle = \bigcap_{A \subseteq H \leq G} H.$$

If $a \in G$, $\langle a \rangle$ is the subgroup generated by the element a. For the group $(\mathbb{Z}, +, 0)$, $\mathbb{Z} = \langle 1 \rangle$. One says that a has a finite order if $\langle a \rangle$ is finite. A group is called cyclic if and only if it is generated by one element. The finite cyclic groups are:

$$\langle a \rangle \simeq \mathbb{Z}/n\mathbb{Z},$$

where $n = \operatorname{ord} a$ is the least $k > 0$ such that $a^k = 1$.

Definition: G is abelian or commutative if for all $x, y \in G$, $xy = yx$.

Abelian groups are usually denoted $(G, +, 0)$. In such a group, every subgroup is normal. We observe also that if $H \leq \mathbb{Z}/n\mathbb{Z}$ then there is some $m \in \mathbb{N}$ such that $m|n$ and $H \simeq \mathbb{Z}/m\mathbb{Z}$.

Some other interesting results about the permutation group:

Definition: The function $\varepsilon : S_n \to (\{-1, 1\}, \cdot)$ given by:

$$\varepsilon(\sigma) = \prod_{1 \leq i < j \leq n} \frac{\sigma(i) - \sigma(j)}{i - j}$$

is called signature of permutations and is a homomorphism of groups. $Ker\ \varepsilon := A_n$ is the so called alternative group, or the group of even permutations. This group is normal in S_n and does not contain any own normal subgroups for $n \geq 5$, being a so called simple group. $|S_n/A_n| = 2$.

Definition: Let $a_1, \ldots, a_k$ be some distinct objects. A cycle $(a_1, a_2, \ldots, a_k)$ is the permutation c such that $c(a_i) = a_{i+1}$ and $c(a_k) = a_1$, but for every object x different from all $a_1, \ldots, a_k$, $c(x) = x$. A cycle of length 2 is called transposition.

Theorem: *Every permutation $\sigma \in S_n$ is a product of $\leq n - 1$ transpositions.*

Proof: Let $\sigma \in S_n$. If $\sigma(1) \neq 1$, consider the permutation $(1, \sigma(1))\sigma$. If not, consider again σ. In both cases, now we deal with a permutation τ such that $\tau(1) = 1$. If $\tau(2) \neq 2$, consider the permutation $(2, \tau(2))\tau$, and if not, consider τ. In both cases, now we deal with a permutation ρ fixing 1 and 2. After at most $n - 1$ such steps, we get a permutation ι that fixes 1, 2, ..., $n - 1$. This permutation is fixing n as well, so $\iota = id$. Hence $t_k t_{k-1} \ldots t_1 \sigma = id$, where t_i are transpositions. Finally $\sigma = t_1 t_2 \ldots t_k$. □

In the decomposition given above, the transpositions are in general not disjoint, and consequently they do not commute.

Theorem: *Every permutation is a product of disjoint cycles. Those cycles commute.*

Proof: For the element 1 we compute $\sigma(1)$, $\sigma^2(1)$ and so on, untill $\sigma^k(1) = 1$. We have already got a cycle $(1, \sigma(1), \ldots, \sigma^{k-1}(1))$. Let m be the least element in the sequence $1, \ldots, n$ that does not belong to the orbit of 1. We built the orbit of m. We go on like this, untill every element belongs to some cycle. The cycles are disjoint by construction. So they commute, because they act on disjoint sets respectively. □

Exercise: Prove the following identities with cycles and permutations:

$$(k, k+1) = (1, 2, \ldots, n)^{k-1}(1, 2)(1, 2, \ldots, n)^{1-k},$$

$$(i, j) = (j-1, j)(j-2, j-1)\ldots(i+1, i+2)(i, i+1)(i+1, i+2)\ldots(j-2, j-1)(j-1, j),$$

$$(a_1, a_2, \ldots, a_k) = (a_1, a_2)(a_2, a_3)\ldots(a_{k-1}, a_k),$$

where $i < j$.

Exercise: Using the identities given above, show that:

1. The transpositions $\{(k, k+1) \mid k = 1, \ldots, n-1\}$ generate together the whole group S_n.
2. The permutations $t = (1, 2)$ and $c = (1, 2, \ldots, n)$ generate the whole group S_n.

2 Rings

Definition: A structure $(R, +, \cdot, 0, 1)$ is a ring if and only if it models the following axiomes:

1. $(R, +, 0)$ is an abelian group.
2. The multiplication is associative: $\forall\, x, y, z\ x(yz) = (xy)z$.
3. The element 1 is neutral for multiplication: $\forall\, x\ 1x = x1 = x$.

4. The multiplication is distributive relatively to addition:
   $\forall\, x, y, z\ x(y+z) = xy + xz \wedge (y+z)x = yx + zx$.

Example: The ring of integers $(\mathbb{Z}, +, \cdot, 0, 1)$.

Definition: A subset $I \subseteq R$ is an ideal if and only if $\forall\, a, b \in I\ a+b \in I$ and $\forall\, x \in R\ \forall\, a \in I\ xa \in I$.

Definition: A function $h : R_1 \to R_2$ is a homomorphism of rings if and only if $h(1) = 1$ and $\forall\, a, b \in R\ h(a+b) = h(a) + h(b) \wedge h(ab) = h(a)h(b)$.

Isomorphism theorem: If $h : R_1 \to R_2$ is an homomorphism of rings, then $Ker\ h$ is an ideal in R_1 and

$$R_1/Ker\ h \simeq Im\ h \leq R_2 \text{ is a subring.}$$

Essential example: $I \subseteq \mathbb{Z}$ is an ideal if and only if $I = n\mathbb{Z}$ for some $n \in \mathbb{N}$. Indeed, if $I \neq \{0\}$ consider $n_I = \min\{x \in I \,|\, x > 0\}$ and show that $n_I\mathbb{Z} = I$. We also observe that:

$$\begin{aligned} n\mathbb{Z} + m\mathbb{Z} &= \gcd(m,n)\mathbb{Z} \\ n\mathbb{Z} \cap m\mathbb{Z} &= \operatorname{lcm}(m,n)\mathbb{Z} \end{aligned}$$

Essential example: $\mathbb{Z}/n\mathbb{Z}$ is a ring, also known as ring of remainders modulo n.

Indeed, $(a+b) \bmod n = (a \bmod n + b \bmod n) \bmod n$ and so on.

Definition: Let R be a ring. $R^\times = \{x \in R \,|\, \exists\, y \in R\ xy = 1\}$ is its multiplicative group of units.

Observe that $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ if and only if $\gcd(x, n) = 1$, (they are relatively prime, or have no common divisor). Exactly the same condition is equivalent with $\langle x \rangle_+ = \mathbb{Z}/n\mathbb{Z}$. $\mathbb{Z}/n\mathbb{Z}$ is also denoted with $\mathbb{Z}_n$.

Chinese Remainder Theorem: Let $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$ some natural number and its prime factor decomposition. Then the following rings are isomorphic:

$$\mathbb{Z}/n\mathbb{Z} \simeq \mathbb{Z}/p_1^{\alpha_1}\mathbb{Z} \times \dots \times \mathbb{Z}/p_k^{\alpha_k}\mathbb{Z}.$$

Proof: If $\gcd(m, n) = 1$ then $\mathbb{Z}_{mn} \simeq \mathbb{Z}_m \times \mathbb{Z}_n$ as rings. We see that:

$$p : \mathbb{Z} \to \mathbb{Z}_n \times \mathbb{Z}_m \ \ ;\ \ p(a) = (a \bmod n, a \bmod m)$$

is a homomorphism with $Ker\ p = mn\mathbb{Z}$ and is surjective. □

The converse problem is the following: given n, m, $\dots$, k all relatively prime, and remainders $a_1 \bmod n$, $a_2 \bmod m$ etc, find a representative $a \bmod nm \dots k$. This converse is more difficult then the direct problem and will be solved after introducing the notion of modular inverse.

For now, recall that $a \bmod n$ is invertible in $\mathbb{Z}_n$ if and only if $\gcd(a, n) = 1$.

Definition: Euler's function $\varphi : \mathbb{N} \to \mathbb{N}$ is defined such that $\varphi(0) = 0$, $\varphi(1) = 1$ and

$$\varphi(n) = \#\{k \in \mathbb{Z}_n \,|\, \gcd(k, n) = 1\} = |\mathbb{Z}_n^\times|$$

If $\gcd(m, n) = 1$ then $\mathbb{Z}_{mn}^\times \simeq \mathbb{Z}_m^\times \times \mathbb{Z}_n^\times$, so $\varphi(mn) = \varphi(m)\varphi(n)$.

Theorem: *If $n = p_1^{\alpha_1} \dots p_k^{\alpha_k}$ then:*

$$\varphi(n) = n(1 - \frac{1}{p_1}) \dots (1 - \frac{1}{p_k}).$$

Proof:

$$\begin{aligned} \varphi(n) = \varphi(p_1^{\alpha_1}) \dots \varphi(p_k^{\alpha_k}) = (p_1^{\alpha_1} - p_1^{\alpha_1 - 1}) \dots (p_k^{\alpha_k} - p_k^{\alpha_k - 1}) = \\ = p_1^{\alpha_1} \dots p_k^{\alpha_k}(1 - \frac{1}{p_1}) \dots (1 - \frac{1}{p_k}) = n(1 - \frac{1}{p_1}) \dots (1 - \frac{1}{p_k}). \end{aligned}$$

□

Euler's Theorem: *If* $a, n > 0$ *with* $\gcd(a, n) = 1$*, then:*

$$a^{\varphi(n)} = 1 \bmod n.$$

Proof: $|\mathbb{Z}_n^\times| = \varphi(n)$, $a \in \mathbb{Z}_n^\times$, $\mathrm{ord}(a) | \varphi(n)$. □

Fermat's Theorem: *If* p *is prime,* $p \not| a$*, then* $a^{p-1} = 1 \bmod p$.

This is a direct consequence of Euler's Theorem.

Observation: $\mathbb{Z}$ is an euclidian ring, i. e. there exists a norm function $f : \mathbb{Z} \to \mathbb{N}$ such that:

$$\forall\, a, b\ \exists\, r, q \quad q \geq 0 \wedge r \geq 0 \wedge 0 \leq f(r) < f(b) \wedge a = bq + r.$$

Indeed, $\mathbb{Z}$ satisfies this condition with the norm $f(b) = |b|$. We say that $\mathbb{Z}$ has division with remainder.

Euclid's Algorithm for the computation of the greatest common divisor $\gcd(a, b)$ uses repeated divisions with remainders. In the case that $\gcd(a, b) = 1$ we can use this algorithm (in its extended form) to compute the modular inverse $b^{-1} \bmod a$, i. e. the multiplicative inverse of b in the unit group $\mathbb{Z}_a^\times$.

□

There are many implementations of the old (unextended) Algorithm of Euclid. The following one is known as very fast:

$g = 1$;
while ($a \bmod 2 = 0$ **and** $b \bmod 2 = 0$) **do** $a = a/2$; $b = b/2$; $g = 2g$, **end**;
while $a \neq 0$ **do**
    while $a \bmod 2 = 0$ **do** $a = a/2$;
    while $b \bmod 2 = 0$ **do** $b = b/2$;
    (now they are both odd!)
    if $a \geq b$ **then** $a = (a - b)/2$ **else** $b = (b - a)/2$;
end
return $g \cdot b$;

Let $a = 100$ and $b = 17$. Euclid's Algorithm works as follows:

$$\begin{aligned} 100 &= 5 \cdot \underline{17} + \underline{15} \\ \underline{17} &= 1 \cdot \underline{15} + \underline{2} \\ \underline{15} &= 7 \cdot \underline{2} + \underline{1} \\ \underline{2} &= 2 \cdot \underline{1} + 0 \end{aligned}$$

The last non-zero remainder is $\gcd(a, b)$. In our case $\gcd(100, 17) = 1$. Now we will understand why the quotients and the remainders have been underlined. In the second part of the algorithm those numbers play the role of literal variables in a backwards substitution process. The following example is mod100.

$$1 = \underline{15} - 7 \cdot \underline{2} = \underline{15} - 7(\underline{17} - \underline{15}) = 8 \cdot \underline{15} - 7 \cdot \underline{17} = 8 \cdot (-5) \cdot \underline{17} - 7 \cdot \underline{17} = (-47) \cdot 17 = 53 \cdot 17.$$

So $17^{-1} \bmod 100 = 53$.

Instead of unwinding the algorithm backwards, one can do it also forwards. The computations are true modulo 100:

$$\underline{15} = (-5) \cdot \underline{17},$$

$$\underline{2} = \underline{17} - \underline{15} = \underline{17} - (-5) \cdot \underline{17} = 6 \cdot \underline{17},$$
$$\underline{1} = \underline{15} - 7 \cdot \underline{2} = (-5) \cdot \underline{17} - 7 \cdot 6 \cdot \underline{17} = (-47) \cdot 17.$$

What is happening here? In the algorithm of Euclid there is a descending sequence of remainders $r_1 > r_2 > \ldots$. In the case that a and m are relatively prime, this ends with an $r_k = 1$. The method of unwinding presented here writes stepwise the remainders r_n in the form $A_n a \bmod m$, and we observe that these coefficients respect the recurrence:

$$A_n = A_{n-2} - q_n A_{n-1}.$$

So at every step we need just the last two coefficients and the actual quotient. This leads to the following algorithm for the modular inverse, very easy to implement in any programming language:

input: a, m.

actual, r, q;

$m_0 = m$; older = 0; old = 1;

repeat:

$q = m/a$;

$r = m \bmod a$;

actual = older $- q *$ old;

if $r \leq 1$ **break**;

$m = a$;

$a = r$;

older = old;

old = actual;

if actual < 0 **do** actual $+ = m_0$;

return actual;

It is finally the time to state:

Chinese Reminder Theorem, effective version: *Let* $m_1, \ldots, m_r \geq 2$ *with* $\gcd(m_i, m_j) = 1$ *for* $i \neq j$. *One gets the conditions* $x \bmod m_i = a_i$ *for* $i = 1, \ldots, r$. *Let:*

$$\begin{aligned} M &= m_1 m_2 \ldots m_r, \\ M_i &= M/m_i, \\ y_i &= M_i^{-1} \bmod m_i, \\ x &= \sum_{i=1}^{r} a_i M_i y_i \bmod M. \end{aligned}$$

Then x *satisfies the given conditions, so it is a solution of the system of congruences.*

Proof: Indeed $x \bmod m_i = a_i M_i y_i \bmod m_i = a_i M_i M_i^{-1} \bmod m_i = a_i$. □

Example: Look for x such that:

$$\begin{aligned} x &= 5 \bmod 7, \\ x &= 3 \bmod 11, \\ x &= 10 \bmod 13. \end{aligned}$$

Then $M = 7 \cdot 11 \cdot 13 = 1001$, $M_1 = 11 \cdot 13 = 143$, $M_2 = 7 \cdot 13 = 91$, $M_3 = 7 \cdot 11 = 77$. Further $y_1 = 143^{-1} \bmod 7 = 3^{-1} \bmod 7 = 5$, $y_2 = 91^{-1} \bmod 11 = 3^{-1} \bmod 11 = 4$ and $y_3 = 77^{-1} \bmod 13 = 12^{-1} \bmod 13 = (-1)^{-1} \bmod 13 = -1 \bmod 13 = 12$. Finally:

$$x = (5 \cdot 143 \cdot 5 + 3 \cdot 91 \cdot 4 + 10 \cdot 77 \cdot 12) \bmod 1001 = 894.$$

□

The next algorithm is crucial for the lectures to come:

Fast exponentiation. For fast computing a^b use that:

$$a^b = a^{\left(\sum\limits_{b_i \in \{0,1\}} b_i 2^i\right)} = \prod_{b_i=1} a^{2^i}.$$

Elements of the form a^{2^i} are computed by successive squaring:

$$a \rightsquigarrow a^2 \rightsquigarrow a^4 \rightsquigarrow a^8 \rightsquigarrow \cdots \rightsquigarrow a^{2^i} \rightsquigarrow a^{2^{i+1}} \rightsquigarrow \ldots$$

This algorithm is even better when you compute $a^b \bmod c$ because then all numbers are bounded by c so the number of operations remains $\mathcal{O}(\log b)$.

Finally a very nice theorem, at the moment given without proof:

Theorem: *The multiplicative group $(\mathbb{Z}_n^\times, \cdot, 1)$ is cyclic if and only if $n \in \{2, 4, p^k, 2p^k\}$, where p is an odd prime number.*

Example: $\mathbb{Z}_9^\times = \{2, 4, 8, 7, 5, 1\}$, and all are successive powers of the generator $g = 2$.

3 Fields

Definition: A structure $(K, +, \cdot, 0, 1)$ is a field if and only if it is a ring and every element which is not 0 has a multiplicative inverse. So in this ring $K^\times = K \setminus \{0\}$.

Examples: $(\mathbb{C}, +, \cdot, 0, 1)$ is the field of complex numbers. The sets $\mathbb{R}$ and $\mathbb{Q}$ are subfields. The quaternion field $\mathbb{H}$ is not commutative. For every prime number p, the ring $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ is a field. Those are not all finite fields, but only the so called prime fields.

Definition: The characteristic of a field is the least natural number $n \in \mathbb{N}$ such that $n \cdot 1 = 0$ is true in this field. The infinite characteristic is called by tradition characteristic 0. The field $\mathbb{H}$, $\mathbb{C}$, $\mathbb{R}$ and $\mathbb{Q}$ have characteristic 0.

It is easy to observe that a finite characteristic is always a prime number. Let K be a field of characteristic $c = ab$ with $a \geq 2$ and $b \geq 2$. So:

$$(1 + \cdots + 1) + \cdots + (1 + \cdots + 1) = 0,$$

where each term in parantheses consists of a sum of a many 1's and there are exactly b such terms. Using distributivity:

$$(1 + \cdots + 1 \ a \text{ terms})(1 + \cdots + 1 \ b \text{ terms}) = 0.$$

In a field, this is possible only if one of the two factors is 0.

This contradicts the minimality of $c = ab$. □

The field $\mathbb{Z}_p$ has characteristic p.

Let K be a field of characteristic p. K contains the elements 1, $1+1$, $1+1+1$, ... So K contains $\mathbb{Z}_p$. As they are both fields, K is a vector space over $\mathbb{Z}_p$. Suppose now that K is finite. As a vector space over $\mathbb{Z}_p$, the additive group $(K, +, 0)$ is isomorphic with a finite product $\mathbb{Z}_p \times \cdots \times \mathbb{Z}_p$ and has p^k elements.

We have here some results about finite fields, without proof:

1. *Every finite field is commutative. (Wedderburn)*
2. *Two finite fields with the same number of elements are isomorphic. This fact justifies the notation $\mathbb{F}_{p^k}$ for the field with p^k elements. $\mathbb{F}_p = \mathbb{Z}_p$.*

3. $\mathbb{F}_{p^s} \subseteq \mathbb{F}_{p^r}$ *if and only if* $s \mid r$.

Example: The field $\mathbb{F}_4$. The polynomial $f \in \mathbb{F}_2[X]$, $f = X^2 + X + 1$ has no roots in $\mathbb{F}_2$ because $f(0) = f(1) = 1$. Being of degree 2, it is irreducible. Let ω be a symbol such that $\omega^2 + \omega + 1 = 0$. In characteristic 2, this means that $\omega^2 = \omega + 1$. In the polynomial ring $\mathbb{F}_2[X]$, f generates a prime ideal, which is also maximal, because the ring is euclidian. The field $\mathbb{F}_4$ is defined as follows:

$$\mathbb{F}_4 = \mathbb{F}_2[X]/(f) = \mathbb{F}_2[\omega] = \{0, 1, \omega, \omega + 1\}.$$

What multiplicative group is generated by ω?

$$\omega,\ \omega^2 = \omega + 1,\ \omega^3 = \omega(\omega + 1) = \omega + 1 + \omega = 1.$$

So $\langle\omega\rangle = \mathbb{F}_4 \setminus \{0\}$, and the multiplicative group of $\mathbb{F}_4$ proves to be cyclic. The following theorem shows that this fact is general. Moreover, this fact plays a crucial role in the present cryptography.

Theorem: *Let K be a commutative field and let G a finite subgroup of the group $K^\times = (K \setminus \{0\})$. Then G cyclic. In particular, if K is finite, its multiplicative group $K^\times$ is cyclic.*

Proof: Let h be the number of elements in G. We look for an element of order h in G. Let $h = p_1^{r_1} \dots p_k^{r_k}$ its decomposition in prime factors. We observe that for all $i = 1, \dots, k$ there is an $x_i \in G$ such that:

$$x_i^{\frac{h}{p_i}} \neq 1,$$

because otherways the polynomial $X^{h/p_i} - 1$ would have a number of roots bigger than its degree, and this is not possible in a field. Let:

$$h_i = \frac{h}{{p_i}^{r_i}}.$$

Define the element:

$$y_i = x_i^{h_i}.$$

We claim this $\operatorname{ord}(y_i) = p_i^{r_i}$. Indeed $y_i^{p_i^{r_i}} = x_i^h = 1$, so $\operatorname{ord}(y_i) \mid p_i^{r_i}$. If we suppose that $\operatorname{ord}(y_i) = p_i^s$ with $s < r_i$, as we know that

$$y_i^{p_i^{r_i - 1}} = x_i^{\frac{h}{p_i}} \neq 1,$$

or $p_i^s \mid p_i^{r_i - 1}$, we reach a contradiction.

Let $y = y_1 y_2 \dots y_k$. As the orders of the elements y_i are relatively prime, the order of the product is the product of the orders, so the order of y is h and y generates G. □

Example: $\mathbb{Z}_7^\times = \{1, 2, 3, 4, 5, 6\} = \langle 3\rangle$. The powers of $2 \bmod 7$ are 2, 4, 1, and then they repeat periodically. For 3, the period length is 6. The number of generators of this cyclic group is $\varphi(6) = \varphi(3)\varphi(2) = (3 - 1)(2 - 1) = 2$. But the generators are not the same elements which generate the additive group $\mathbb{Z}_6$. For example 3 does not additively generate $\mathbb{Z}_6$ but multiplicatively generates $\mathbb{Z}_7^\times$. □

Observation: Although the fields with p^k elements are isomorphic, they can be defined using different irreducible polynomials of degree k over $\mathbb{F}_p$. So $\mathbb{F}_{p^k}$ has different representations. Those representations are explicitely used in cryptography. So, just mentioning p^k is not sufficient to perform encryptions and decryptions, without explicitely giving the irreducible polynomial used to define the field multiplication.

4 Functions in finite fields

The simplest finite field is $\mathbb{F}_2 = \mathbb{Z}_2 = \{0, 1\}$, with the following operations:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

From the point of view of mathematical logic, the addition $+$ is the logic function XOR and the multiplication $\cdot$ is the logic conjunction $\wedge$. Other logic functions are the negation and the disjunction:

$\neg$	0	1
	1	0

$\vee$	0	1
0	0	1
1	1	1

Theorem: *Every function $f : \{0,1\}^k \to \{0,1\}$ can be expressed using only $\vee$, $\wedge$ and $\neg$.*

Proof:

$$f = \bigvee_{\substack{(\varepsilon_1,\ldots,\varepsilon_n)\\ f(\varepsilon_1,\ldots,\varepsilon_n)=1}} x_1^{\varepsilon_1} \wedge \cdots \wedge x_n^{\varepsilon_n},$$

where by convention $x^0 = \neg x$ and $x^1 = x$. □

Example: $x + y = x^0y^1 \vee x^1y^0 = (\neg x \wedge y) \vee (x \wedge \neg y)$.

Because $x \wedge y = \neg(\neg x \vee \neg y)$ one can represent every function using only negations and disjunctions. On the other hand $x \vee y = \neg(\neg x \wedge \neg y)$ so one can use only negations and conjunctions as well. One can even go further. Consider the operation $\mid$ also called NAND or Sheffer's Stroke:

$$x \mid y = x^0y^0 \vee x^0y^1 \vee x^1y^0.$$

Then:

$$\neg x = (x \mid x),$$
$$x \vee y = (x \mid x) \mid (y \mid y).$$

So every function can be expressed by a formula containing only Sheffer's Stroke.

Finally, as $x \wedge y = xy$ and $\neg x = 1 + x$, we get the following result: *Every function $f : \{0,1\}^k \to \{0,1\}$ can be expressed as a polynomial function.* This fact is generally true for all finite fields.

Theorem: *Let $\mathbb{F}$ be a finite field. Every function $f : \mathbb{F}^k \to \mathbb{F}$ is a polynomial function.*

The proofs use the property of interpolation. The interpolation is the method according to which, given a field K and k pairs of values (x_i, y_i) where all x_i are pairwise distinct, one finds a polynomial f of degree $k-1$ such that all conditions $f(x_i) = y_i$ are satisfied. This method can be generalized to any number of variables.

Proof 1: Let $\varepsilon \in \mathbb{F}$ be some element. We define the polynomial $g_\varepsilon : \mathbb{F} \to \mathbb{F}$ as:

$$g_\varepsilon(x) = \frac{\prod\limits_{a \neq \varepsilon} (x - a)}{\prod\limits_{a \neq \varepsilon} (\varepsilon - a)}.$$

This polynomial takes the following values:

$$g_\varepsilon(x) = \begin{cases} 1, & x = \varepsilon, \\ 0, & x \neq \varepsilon. \end{cases}$$

Let $(\varepsilon_1, \ldots, \varepsilon_k) \in \mathbb{F}^k$ be some tuple of elements. We define the polynomial $g_{(\varepsilon_1,\ldots,\varepsilon_k)}$ in the following way:

$$g_{(\varepsilon_1,\ldots,\varepsilon_k)}(x_1, \ldots, x_k) = g_{\varepsilon_1}(x_1) g_{\varepsilon_2}(x_2) \ldots g_{\varepsilon_k}(x_k).$$

This polynomial takes the value 1 only for $(x_1, \ldots, x_k) = (\varepsilon_1, \ldots, \varepsilon_k)$ and 0 for all other tuples. Finally we can represent the function f by the following polynomial:

$$f(x_1, \ldots, x_k) = \sum_{(\epsilon_1, \ldots, \epsilon_k)} g_{(\varepsilon_1, \ldots, \varepsilon_k)}(x_1, \ldots, x_k) \cdot f(\varepsilon_1, \ldots, \varepsilon_k).$$

Proof 2: Same idea, just that $g_\varepsilon(x)$ has a different construction. Let n be the number of elements of $\mathbb{F}$. The multiplicative group $\mathbb{F} \setminus \{0\}$ has $n-1$ elements, so:

$$x^{n-1} = \begin{cases} 1, & x \neq 0, \\ 0, & x = 0. \end{cases}$$

The function $g_\varepsilon(x)$ is defined as:

$$g_\varepsilon(x) = 1 - (x - \varepsilon)^{n-1}.$$

From here on, the proof is identical with Proof 1. □

5 Linear encryption

Let $\mathcal{A}$ be some alphabet with $|\mathcal{A}| = n$ elements. One can identify $\mathcal{A}$ with elements of a finite ring R. The only one condition is that the ring has at least so many elements as the alphabet. Let $k \geq 1$. A text in $\mathcal{A}$ is partitioned in blocks of length k and those blocks are linearly encrypted:

$$c : \mathcal{A}^k \to \mathcal{A}^k \text{ given by } c(a_1, \ldots, a_k) = M \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix}$$

where $M \in \mathcal{M}_{k \times k}(R)$ is invertible, i. e. there is a matrix M^{-1} such that $M^{-1}M = I_k$.

Recall that the inverse matrix has elements a_{ij} of the form:

$$a_{ij} = (-1)^{i+j} \det(M)^{-1} \det(M_{ij}).$$

Theorem: M is invertible if and only if $\det(M) \in R^\times$, i. e. is invertible in R.

Examples: let $|\mathcal{A}| = 26$ and blocks of length 2, so that the encryption reads $x_1x_2 \rightsquigarrow y_1y_2$. We identify $\mathcal{A}$ with $\mathbb{Z}_{26}$. The operation:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 6 & 2 \\ 5 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \bmod 26$$

is not good to perform a linear encryption because $\gcd(26, \det(M)) = 2$, so M is not invertible. Find different blocks x_1x_2 and $x_1'x_2'$ with the same encryption y_1y_2.

For the operation:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 6 & 1 \\ 5 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \bmod 26$$

find the rule of decryption.

6 Elliptic curves

Definition: Let $p \neq 2, 3$ be a prime number. An elliptic curve over $\mathbb{Z}_p$ is the set of pairs $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ with:

$$y^2 = x^3 + Ax + B \bmod p,$$

together with the point at infinity O. Here $A, B \in \mathbb{Z}_p$ and the discriminant:

$$4A^3 + 27B^2 \neq 0 \bmod p.$$

We denote this set $E(\mathbb{Z}_p)$.

One can define elliptic curves over any field, but the general definition is more complicated. The curves defined here are already put in the Weierstraß form. In the first figure you can see how the curve depends on parameters in the real case.

All elliptic curves have a group structure. In the real case, the group structure has a geometric definition: the line connecting P and Q intersects the degree 3 curve in a third point. The sum $P+Q$ is got from this third point of intersection, by mirroring it relatively to the symmetry axe of the curve. For computing $P + P$, the tangent in P is used instead of the line PQ. The point $-P$ lies symmetrical to P relatively to the symmetry axe of the curve. But as the line $P(-P)$ does not intersect the curve a third time, one considers that $P + (-P) = O$. So the point at infinity O is the neutral element of the group.

Using analytic geometry we deduce the following formulas for the group operations:

$$P + O = O + P = P,$$

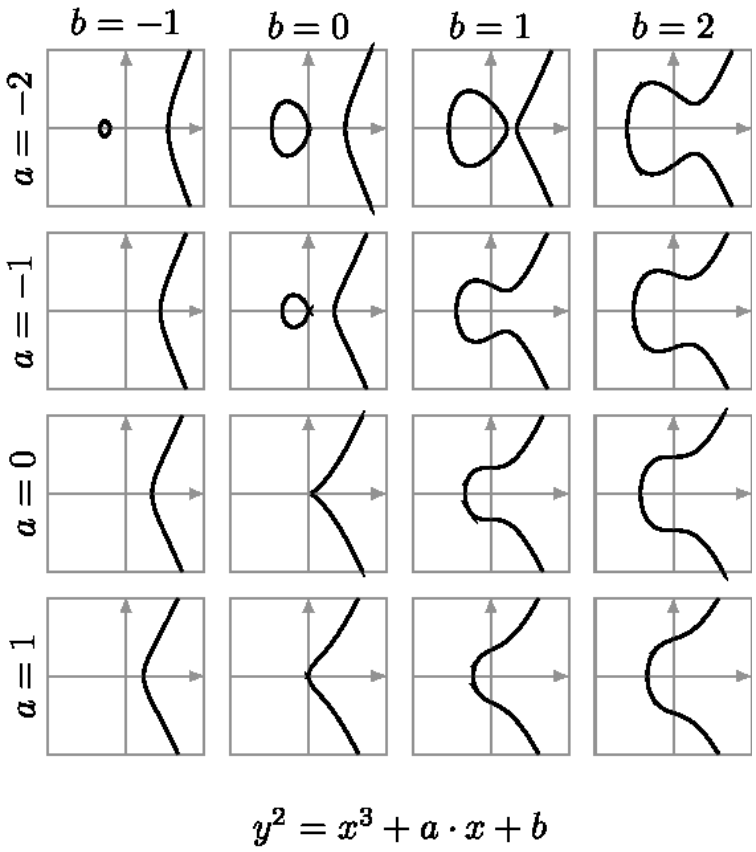
$$P = (x, y), \quad Q = (x, -y) \quad \longrightarrow \quad P + Q = O,$$

$$P + Q = R, \quad R = (x_3, y_3),$$

$$x_3 = (m^2 - x_1 - x_2) \bmod p,$$

$$y_3 = m(x_2 - x_3) - y_1 \bmod p,$$

$$m = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \bmod p, & P \neq Q, \\ \\ \frac{3x_1^2 + A}{2y_1} \bmod p, & P = Q. \end{cases}$$



Theorem: *For every field K, $(E(K), +, O)$ is an abelian group. In the case $K = \mathbb{Z}_p$, $p > 3$ prime, there are $n_1, n_2 > 1$ such that:*

$$E(\mathbb{Z}_p) \simeq \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2},$$

si $n_2 | n_1$, $n_2 | (p-1)$.

So for $n_2 = 1$, $E(\mathbb{Z}_p)$ is cyclic. In the practice one looks for a point P generating a subgroup of prime order. The following result is a first estimation of the cardinality of the curve:

Hasse's Theorem:

$$p + 1 - 2\sqrt{p} \leq |E(\mathbb{Z}_p)| \leq p + 1 + 2\sqrt{p}.$$

Example: Consider $\mathbb{Z}_7$ and the elliptic curve E given by $y^2 = x^3 + 2x + 1$. Make first a table with the powers modulo 7 of the elements:

x	0	1	2	3	4	5	6
x^2	0	1	4	2	2	4	1
x^3	0	1	1	6	1	6	6

For $x = 0$, $y^2 = 1$, so $y = 1$ or $y = 6$. If $x = 1$, $y^2 = 4$, so $y = 2$ or $y = 5$. For $x = 2, 3, 4, 5, 6$, $y^2 = 6, 3, 5$ and there are no other points. Finally:

$$E = \{O, (0,1), (0,6), (1,2), (1,5)\}.$$

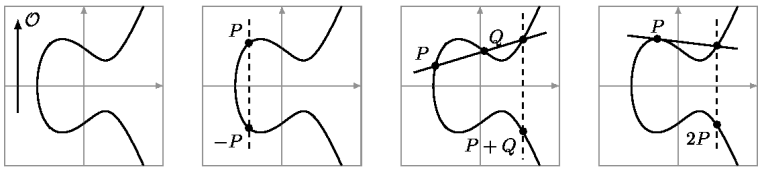
As E has 5 elements and there is only one group of 5 elements, which is cyclic, so every element different of O is generator of the group. □

Example: The curve E defined as $y^2 = x^3 + 2x + 2 \bmod 17$. The point $P = (5, 1)$ additively generates the following set:

$$\langle P \rangle = \{(5,1), (6,3), (10,6), (3,1), (9,16), (16,13), (0,6), (13,7), (7,6),$$

$$(7,11), (13,10), (0,11), (16,4), (9,1), (3,16), (10,11), (6,14), (5,16), O\}$$

This group has 19 elements and is cyclic. Every element different of O is a generator. For every point P in the first line, we can find $-P$ in the second line. □



Neutral element $\mathcal{O}$ | Inverse element $-P$ | Addition $P + Q$ "Chord rule" | Doubling $P + P$ "Tangent rule"

Part II
Symmetric cryptography

7 Perfect security

We start by introducing the conditional probabilities. $p(A|B)$ is the probability of the event A with the condition that event B also takes place. We know that:

$$p(A|B) = \frac{p(A \cap B)}{p(B)}.$$

Definition: Let M be the set of all plain texts, let K be the set of the encryption keys and let C the set of encrypted texts, also called cyphers.

Definition: The encryption is a function $Enc : M \times K \to C$. The decryption is a function $Dec : C \times K \to M$. Encryption and decryption with symmetric key mean:

$$\forall\, m \in M\ \forall\, k \in K\ Dec_k(Enc_k(m)) = m.$$

Definition: One says that the pair (Enc, Dec) has perfect security if:

$$\forall\, m \in M\ \forall\, c \in C\ p(m|c) = p(m).$$

With other words, seing c does not reveal any information about m.

In the following lines K, C and M are considered to be finite.

Lemma: In the case of perfect security,

$$|K| \geq |C| \geq |M|.$$

Proof: Enc_k is injective for every $k \in K$ fixed, so $|C| \geq |M|$.

For every $c \in C$, $p(c) > 0$, otherwise we can exclude c from the set. So for every $m \in M$ and $c \in C$,

$$p(m|c) = p(m) > 0.$$

So for every $m \in M$ and $c \in C$, there is a $k \in K$ with $Enc_k(m) = c$. So $|K| \geq |C|$. □

Shannon's Theorem: If $|M| = |C| = |K|$ then (Enc, Dec) has perfect security if and only if both following conditions are satisfied:

1. Every key is used with probability $1/|K|$.
2. $\forall\, m \in M\ \forall\, c \in C\ \exists!\, k \in K\ Enc_k(m) = c$.

Necessity: Suppose that (Enc, Dec) are perfectly secure. For all $m \in M$ and $c \in C$ there exists $k \in K$ such that $Enc_k(m) = c$. But $|C| = |K|$ so $|\{Enc_k(m) \,|\, k \in K\}| = |K|$. Hence for $m \in M$ and $c \in C$ there is a **unique** $k \in K$ such that $Enc_k(m) = c$.

Now we show that the keys are used with equal probability. Let $|K| = n$, $M = \{m_i \,|\, 1 \leq i \leq n\}$ and fix $c \in C$. Perfect security means $p(m_i|c) = p(m_i)$, so:

$$p(m_i) = p(m_i|c) = \frac{p(c|m_i)p(m_i)}{p(c)} = \frac{p(k_i)p(m_i)}{p(c)}.$$

For all i, $p(k_i) = p(c)$. The keys occur with equal probability.

Sufficience: Suppose that both conditions are fulfilled. We show that $p(m|c) = p(m)$. Fix $m \in M$ and $c \in C$.

The keys occur with equal probability, so:

$$p(c) = \sum_{k \in K, m \in M} p(k)p(m = Dec_k(c)) = \frac{1}{|K|} \sum_{k \in K, m \in M} p(m = Dec_k(c)).$$

But for all m and c there is a unique key k such that $Enc_k(m) = c$. So:

$$\sum_{k \in K, m \in M} p(m = Dec_k(c)) = \sum_{m \in M} p(m) = 1.$$

Let be again $n = |K|$. This means $p(c) = 1/n$ and for $c = Enc_k(m)$, $p(c|m) = p(k) = 1/n$. According to Bayes' formula for conditional probabilities:

$$p(m|c) = \frac{p(m)p(c|m)}{p(c)} = \frac{p(m) \cdot 1/n}{1/n} = p(m).$$

□

Example: $\mathcal{A} = \mathcal{A}_{26} = \{A, B, C, \ldots, Z\}$. Then $|K| = |M| = |C| = 26^n$ and

$$c = m + k \bmod 26,$$

with addition performed letterwise.

Example: Vernam's Code, OTP. (One Time Pad) $\mathcal{A} = \{0, 1\}$, $|K| = |M| = |C| = 2^n$ and

$$c = m \oplus k,$$

where $\oplus$ is the addition over $\mathbb{F}_2$ and is performed letterwise.

Observation: Do not use the same key twice. If you do so:

$$c_1 \oplus c_2 = m_1 \oplus k \oplus m_2 \oplus k = m_1 \oplus m_2,$$

so the security is no more perfect.

8 Entropy

Definition: Let X be a random variable, taking values $(x_i)_{1 \leq i \leq n}$ with probabilities $p_i = p(x_i)$. The entropy of X is the quantity:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i,$$

where we apply the convention that if $p = 0$ then $p \log_2 p = 0$.

Example: If I answer every question with *yes*, then $p_1 = 1$, $p_2 = 0$ and

$$H(X) = -1 \log_2 1 - 0 \log_2 0 = 0,$$

so this conversation does not reveal any information. If my answers are **yes** and **no** with probability $1/2$, then:

$$H(X) = (-\log_2 \frac{1}{2} - \log_2 \frac{1}{2})\frac{1}{2} = 1,$$

so every answer contains one bit of information.

Properties of the entropy:

1. $H(X) \geq 0$.
2. $H(X) = 0 \leftrightarrow \exists! i \;\; p_i = 1 \wedge \forall\, j \neq i \;\; p_j = 0$.
3. $\forall\, i \; p_i = \frac{1}{n} \rightarrow H(X) = \log_2 n$.

4. Jensen's inequality

$$\sum_{i=1}^{n} a_i = 1 \longrightarrow \sum_{i=1}^{n} a_i \log_2 x_i \leq \log_2 \Big(\sum_{i=1}^{n} a_i x_i \Big),$$

with equality for $x_1 = \cdots = x_n$. This inequality expresses the fact that the function log has a convex graph.

Theorem: *If X takes exactly n values,*

$$0 \leq H(X) \leq \log_2 n.$$

Proof:

$$H(X) = -\sum p_i \log_2 p_i = \sum p_i \log_2 \frac{1}{p_i} \leq \log_2 \sum p_i \frac{1}{p_i} = \log_2 n.$$

□

Definition: Let X and Y two random variables. We define the conditional entropy:

$$H(X|y) = -\sum_x p(X = x|Y = y) \log_2 p(X = x|Y = y),$$

$$H(X|Y) = \sum_y p(Y = y) H(X|y).$$

Definition: Let X and Y be two random variables.
We define the common entropy. If $r_{ij} = p(X = x_i \wedge Y = y_j)$, then:

$$H(X, Y) = -\sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \log_2 r_{ij}.$$

Properties of the common entropy and of the conditional entropy:

1. $H(X, Y) \leq H(X) + H(Y)$ with equality if and only if they are independent.
2. $H(X, Y) = H(Y) + H(X|Y)$.
3. $H(X|Y) \leq H(X)$ with equality if and only if they are independent.

In cryptography the following applications of the concept of informational entropy are important:
$H(M|K, C) = 0$. If we know the cypher and the key, we get the plain text.
$H(C|M, K) = 0$. If we know the plain text and the key, we get the cypher.

$$\begin{aligned} H(K, M, C) &= K(M, K) + H(C|M, K) =, \text{ but } H(C|M, K) = 0, \\ &= H(M, K) = \\ &= H(K) + H(M) \text{ because } K \text{ and } M \text{ are independent.} \end{aligned}$$

$$\begin{aligned} H(K, M, C) &= H(K, C) + H(M|K, C) =, \text{ but } H(M|K, C) = 0, \\ &= H(K, C) \end{aligned}$$

Finally:

$$H(K, C) = H(K) + H(M).$$

Definition: $H(K|C)$ is called *key equivocation* and represents the uncertainty about the key, if one gets only a cyphered message.

$$H(K|C) = H(K,C) - H(C) = H(K) + H(M) - H(C).$$

Example: $M = \{a,b,c,d\}$, $C = \{1,2,3,4\}$, $K = \{k_1,k_2,k_3\}$ where $p(a) = 0,25$, $p(b) = p(d) = 0,3$, $p(c) = 0,15$. Also $p(k_1) = p(k_3) = 0,25$, $p(k_2) = 0,5$. Let the encryption be:

	a	b	c	d
k_1	3	4	2	1
k_2	3	1	4	2
k_3	4	3	1	2

One computes:

$$\begin{aligned}
p(1) &= p(k_1)p(d) + p(k_2)p(b) + p(k_3)p(c) = 0,2625\\
p(2) &= p(k_1)p(c) + p(k_2)p(d) + p(k_3)p(d) = 0,2625\\
p(3) &= p(k_1)p(a) + p(k_2)p(a) + p(k_3)p(b) = 0,2625\\
p(4) &= p(k_1)p(b) + p(k_2)p(c) + p(k_3)p(a) = 0,2125\\
H(M) &\sim 1,9527\\
H(K) &\sim 1,5\\
H(C) &\sim 1,9944\\
H(K|C) &\sim 1,9527 + 1,5 - 1,9944 = 1,4583
\end{aligned}$$

So, if one sees a cyphered message, we still need about $1,45$ bits of information about the key. This is really not much, so the system is very insecure.

If every plain message occurs with a probability of $1/4$ and every key occurs with the probability of $1/3$, then every cypher occurs with the probability $3(1/3)(1/4) = 1/4$. In this case the entropies are:

$$\begin{aligned}
H(M) &= -4 \cdot \frac{1}{4}\log_2\frac{1}{4} = 2\\
H(K) &= -3 \cdot \frac{1}{3}\log_2\frac{1}{3} = \log_2 3\\
H(C) &= -4 \cdot \frac{1}{4}\log_2\frac{1}{4} = 2\\
H(K|C) &= 2 + \log_2 3 - 2 = \log_2 3 \sim 1,5849 > 1,4583
\end{aligned}$$

So with a uniform distribution of probabilities, the whole system is more secure. □

Let L be a natural language, and let H_L be the letter-wise entropy, i.e. the letter-wise amount of information. For random strings over the english alphabet one has $H = \log_2 26 \sim 4,70$. So $H_L \leq 4,70$. If we consider the occurrence probabilities in English:

$$p(A) = 0,082; \ldots; p(E) = 0,127; \ldots; p(Z) = 0,001;$$

then $H_L \leq H(p) = 4,14$ bits of information per letter. But the situation is in fact more delicate: Q is always followed by U, TH is very frequent, etc. So we should better consider the groups of two letters. Let M^2 be the random variable of the digrames MM'.

$$H(M^2) = \sum_{i,j} p(M = i, M' = j)\log(M = i, M' = j) \sim 7,12$$

and $H_L \leq H(M^2)/2 = 3,56$.

Definition: The entropy of the natural language L is:

$$H_L = \lim_{n\to\infty} \frac{H(M^n)}{n}$$

and for English is estimated as $1,0 \leq H_L \leq 1,5$.

So a letter in English needs 5 bits to be encoded but in context contains at most 1,5 bits of information. This means that natural languages are very redundant. An easy example (guess the original text):

*On** up** a t**e t**re **s a **rl **ll** Sn** Wh**e.*

Definition: Let's define the redundancy of a language:

$$R_L = 1 - \frac{H_L}{\log_2 |M|}$$

If we consider the mean value $H_L = 1,25$, the redundancy in English is about:

$$R_L = 1 - \frac{1,25}{\log_2 26} \sim 0,75.$$

So in general we can compress english texts of 10 MB to files of around 2,5 MB.

Some thoughts about false keys. Let $c \in C$ be an ecryped message of length $|c| = n$. Consider the set:

$$K(c) = \{k \in K \,|\, Dec_k(c) \text{ "makes sense"}\}.$$

Then $|K(c)| - 1$ is the number of false keys.

The mean number of false keys, or false ideas of decryption, is:

$$s_n = \sum_{c\in C} p(c)(|K(c)| - 1) = (\sum_{c\in C} p(c)|K(c)|) - 1.$$

Take n big and consider $|M| = |K|$. Apply Jensen's inequality:

$$\log_2(s_n + 1) = \log_2 \sum_{c\in C} p(c)|K(c)| \geq \sum_{c\in C} p(c) \log_2 |K(c)|.$$

So:

$$\log_2(s_n + 1) \geq \sum_{c\in C} p(c)H(K|c) = H(K|C^n) = H(K) + H(M^n) - H(C^n) \sim$$
$$\sim H(K) + nH_L - H(C^n) = H(K) - H(C^n) + n(1 - R_L) \log_2 |M| \geq$$
$$\geq H(K) - n \log_2 |C| + n(1 - R_L) \log_2 |M| = H(K) - nR_L \log_2 |M|.$$

because $H(C) \leq n \log_2 |C|$ and $|M| = |C|$. Finally:

$$s_n \geq \frac{|K|}{|M|^{nR_L}} - 1.$$

Definition: The unicity distance n_0 is the value of n for which the mean number of false keys becomes 0.

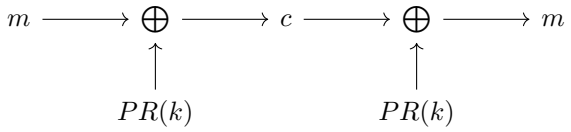
$$n_0 \sim \frac{\log_2 |K|}{R_L \log_2 |M|}.$$

For $|M| = 26$, $|K| = 26! \sim 4 \cdot 10^{26}$ and $R_L = 0,75$ we get $n_0 \sim 25$.

So for $|c| \geq 25$ we suppose that there is only one decryption that makes sense. If we compress data before transmission, then $n_0 \sim |k|/0 \sim \infty$ so the enemy has much more work to find a key that makes sense.

9 Linear feedback shift registers

The perfect security is not practical because the key has the same length as the message, must be randomly chosen and must be changed for every new message. Instead pseudo-random sequences are used. A pseudo-random generator gets as argument a key k, which is much shorter than the message m. The generator produces a pseudo-random sequence of needed length which is binarily added to m. The decryption is done in the same way:



How one can construct a pseudo-random generator? Every determinist program, with input k, produces a sequence that ultimately becomes periodic. So reasonable conditions will be:

- The period must be as long as possible.

- The sequence must be so "unpredictable" as possible.

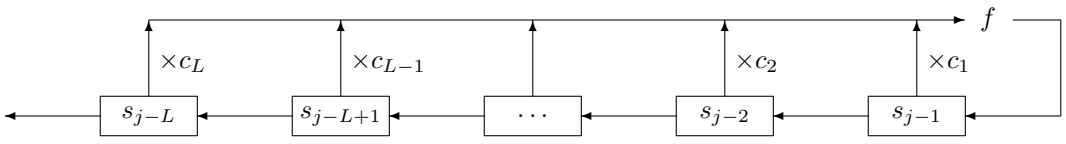
A first idea is to choose $p \in \mathbb{N}$ quite big and to observe that $\langle 10 \rangle_\times = \mathbb{Z}_p^\times$. In this case $1/p$ is a periodic fraction and the period has length p. The pseudo-random properties must be tested from case to case.

Another possibility is to choose a prime number p and a number k such that $\langle k \rangle_\times = \mathbb{Z}_p^\times$. The pseudo-random sequence is the sequence of successive powers $(k^n)_{n\in\mathbb{N}}$ with an appropiate encoding - for example write the numbers binarily and concatenate those binary words. Other pseudo-random sequences are got by concatenating the decimal encodings, or even the original sequence, with values in $\mathbb{Z}_p^\times$.

Now we deal with linear feed-back shift registers or LFSR.

Definition: Let $L \geq 1$ be the length of the register, $c_1, \ldots, c_L \in \{0,1\}$ be the coefficients of the register, $[s_0, \ldots, s_{L-1}]$ be the initial state of the register, where $s_i \in \{0,1\}$. The produced sequence is $(s_n)_{n\in\mathbb{N}}$ where:

$$s_j = f(s_{j-1}, \ldots, s_{j-L}) = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \cdots \oplus c_L s_{j-L}, \quad \forall\, j \geq L.$$



A LFSR has a period of length $N < 2^L - 1$. This results as follows:

A LFSR is associated with the following connection matrix $M \in \mathcal{M}_{L\times L}(\mathbb{F}_2)$ and the following state transition:

$$\begin{pmatrix} s_{j-L+1} \\ s_{j-L+2} \\ \vdots \\ s_{j-1} \\ s_j \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ c_L & c_{L-1} & c_{L-2} & \ldots & c_1 \end{pmatrix} \begin{pmatrix} s_{j-L} \\ s_{j-L+1} \\ \vdots \\ s_{j-2} \\ s_{j-1} \end{pmatrix}$$

Definition: The connection polynomial of a LFSR is the polynomial:

$$C(X) = 1 + c_1 X + \cdots + c_L X^L \in \mathbb{F}_2[X].$$

Definition: Let $\mathbb{F}$ be a finite field, $C(X) \in \mathbb{F}[X]$ be an irreducible polynomial and $\mathbb{G} = \mathbb{F}[X]/(C)$ be the decomposition field of $C(X)$. The polynomial $C(X)$ is called a primitive polynomial if a solution ω of $C(X)$ generates the cyclic multiplicative group $\mathbb{G}^\times$.

If one applies the connection matrix to vectors $\vec{v} \in \mathbb{F}_2^L$, this operation is isomorphic with the multiplication of corresponding elements from $\mathbb{F}_{2^N}$ with a special element ω^{-1}, where ω is a solution of the polynomial $C(X)$.

The initial state $\vec{s_0} = 0$ is excluded, because this state always leads to a constant sequence. In general following cases may occur:

The singular case: $c_L = 0$. For some initial states, the sequence becomes periodic later, and not from the beginning. There are periodic sequences with different periods of different lengths.

The reducible case: $c_L = 1$ but $C(X)$ is reducible over $\mathbb{F}_2$. In this case the sequence is periodic from the beginning for all initial states but there are disjoint periodic cycles of different length.

The irreducible non-primitive case: $c_L = 1$ and $C(X)$ is irreducible over $\mathbb{F}_2$ but is not a primitive polynomial. In this case the sequence is periodic for every initial state. There are disjoint cycles but they all have the same length.

The primitive case: $c_L = 1$ and $C(X)$ is primitive. In this case there is just one cycle of maximal length $|\mathbb{G}^\times| = 2^L - 1$.

Observation: LFSR is insecure for a plain text attack. If we know the number L and $2L$ many successive values of the sequence, we can deduce the coefficients using the following system over the field $\mathbb{F}_2$:

$$\begin{pmatrix} s_{L-1} & s_{L-2} & \dots & s_0 \\ s_L & s_{L-1} & \dots & s_1 \\ \vdots & \vdots & \ddots & \vdots \\ s_{2L-2} & s_{2L-3} & \dots & s_{L-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_L \end{pmatrix} = \begin{pmatrix} s_L \\ s_{L+1} \\ \vdots \\ s_{2L-1} \end{pmatrix}$$

To prevent such attacks, we combine some LSFR using non-linear functions, as like:

$$F(x_1, x_2, x_3, x_4, x_5) = 1 \oplus x_2 \oplus x_3 \oplus x_4 x_5 \oplus x_1 x_2 x_3 x_5.$$

In this particular case the length of the period will be $(2^{L_1} - 1)(2^{L_2} - 1) \dots (2^{L_5} - 1)$.

10 DES

DES means *data encryption standard* and was for many years the security standard established by NIST (National Institute of Standards in Technology, USA). The system was the result of a joint work of Horst Feistel and an IBM team. DES was never broken. However, because of the increased power of computation, some worries arose that the 64-bit block and the 56-bit key are too short for a secure system. So NIST organized a competion for a new security standard. The winner was AES, a system which will be studied in the next section.

Definition: A Feistel round is the function $f : \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined as follows. Let $F : \{0,1\}^n \to \{0,1\}^n$ an arbitrary function. One divides the words $x \in \{0,1\}^{2n}$ in equal halfs $x = (L, R)$ with $L, R \in \{0,1\}^n$. Then:

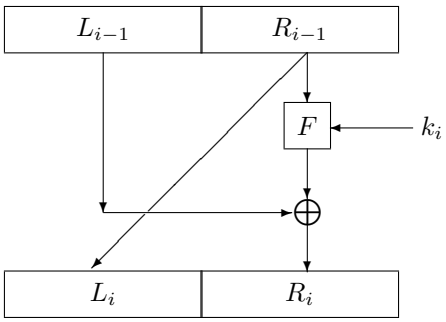
$$f(x) = f(L, R) = (R, L \oplus F(R)).$$

Theorem: The Feistel round is a bijection.

Proof: We observe that this happens without F being invertible. Indeed:

$$f^{-1}(A, B) = (B \oplus F(A), A).$$

□



Feistel rounds are serially connected to build Feistel nets. The pseudo-random function F depends on a round key k_i. The action of a Feistel round can be written down as:

$$\begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} \oplus F(k_i, R_{i-1}), \end{aligned}$$

and the decryption step is then:

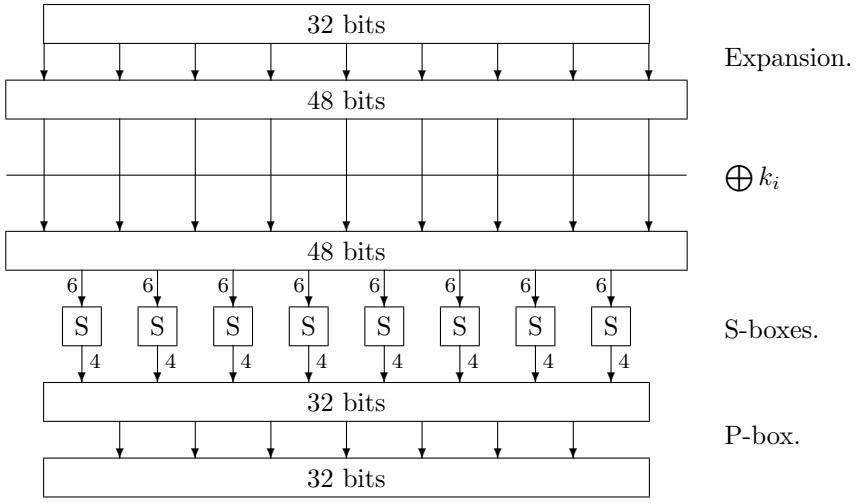
$$\begin{aligned} R_{i-1} &= L_i, \\ L_{i-1} &= F(k_i, L_i) \oplus R_i. \end{aligned}$$

DES consists of 16 Feistel rounds.

The rest of this section deals only with the function F.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Initial permutation, to be applied before the 16 Feistel rounds. The first 4 rows build the block L, the remaining 4 rows build the block R. After the 16 rounds, the inverse of this permutation is applied.



The function $F(k_i, \cdot)$ in DES.

Expansion. A function $E : \{0,1\}^{32} \to \{0,1\}^{48}$ acting as follows:

$$E(x_1, x_2, \ldots, x_{32}) = (x_{32}, x_1, x_2, x_3, x_4, x_5, x_4, x_5, \ldots, x_{32}, x_1).$$

Every bit-pair x_{4k}, x_{4k+1} is duplicated like $x_{4k}, x_{4k+1}, x_{4k}, x_{4k+1}$. This process takes place circularly. Finally, the 48 output bits are the 32 initial bits plus 16 duplicated bits.

The 48 bits block is binarily added with the round key. Every round key is computed from the main key using a function of key scheduling: $k_i = s(k, i)$.

The result of the binary addition is divided in eight 6-bit little blocks. Every 6-bit block is transformed by an S-box.

S-box. Every S-box is a function $S_i : \{0,1\}^6 \to \{0,1\}^4$. There are 8 S-boxes S_1, ..., S_8. Those functions are completely defined and standardized.

As an example, we describe S_1. In the left column there are the numbers encoded by the two outer 2 bits. On the first row are the numbers encoded by the 4 inner bits. To look for the value $S_1(010101)$ we look at the line $01_2 = 1_{10}$ intersected with the column $1010_2 = 10_{10}$. So we find $S_1(010101) = 12_{10} = 1100_2$.

All S-boxes are built such that every row is a permutation of the numbers 0, ..., 15. They all have the property that if one input bit is modified, then at least two output bits are modified. This behavior corresponds to the principle of **confusion** defined by Shannon: the radical modification of sub-blocks.

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-box S_1 given as an output list.

P-box. The final permutation of which the function F consists has the property that the bits produced by some S-box are directed at addresses corresponding to 4 other S-boxes. This corresponds to the principle of **diffusion** defined by Shannon: to strongly mix-up the sub-blocks obtained by confusion. Here is the P-box presented as output list:

$$\begin{pmatrix} 16 & 7 & 20 & 21 \\ 29 & 12 & 28 & 17 \\ 1 & 15 & 23 & 26 \\ 5 & 18 & 31 & 10 \\ 2 & 8 & 24 & 14 \\ 32 & 27 & 3 & 9 \\ 19 & 13 & 30 & 6 \\ 22 & 11 & 4 & 25 \end{pmatrix}$$

A possible advanced way to apply DES is the algorithm 3DES or Triple DES. A key is the concatenation of three DES keys. The encryption is done as follows:

$$c = \mathrm{DES}_{k_3}\mathrm{DES}_{k_2}^{-1}\mathrm{DES}_{k_1}(m),$$

and the decryption is the corresponding inverse function.

11 AES

AES (Advanced Encryption Standard) followed DES as NIST standard in October 2000. Its authors are Joan Daemen and Vincent Rijmen from Belgium, so the algorithm is also known as Rijndael.

This algorithm cannot be described without preliminary definitions and notations.

Definition: In the hexadecimal writing, i.e. basis 16, the digits 0, 1, 2, ..., 9 are completed with the new digits A, B, C, D, E, F. A hexadecimal number is always preceded by the prefix $0x$. Example:

$$0x83 = 8 \cdot 16 + 3 = 131 = 128 + 2 + 1 = 2^7 + 2 + 1 = X^7 + X + 1.$$

Binary strings are divided in 4-blocks which are translated in hexadecimal digits:

$$X^7 + X + 1 = 10000011 = 1000\ 0011 = 0x83.$$

Definition: The arithmetic over the field $\mathbb{F}_{2^8} = \mathbb{F}_{256}$ is defined by the irreducible polynomial:

$$X^8 + X^4 + X^3 + X + 1.$$

Definition: The 32 bit words are identified with polynomials of degree ≤ 3 from $\mathbb{F}_{256}[X]$. So, a block $a_0a_1a_2a_3$ with $a_i \in 2^8$ is identified with the polynomial:

$$a_3X^3 + a_2X^2 + a_1X + a_0.$$

The arithmetic in the ring $\mathbb{F}_{2^8}[X]$ is done modulo $X^4 + 1$. Observe that $X^4 + 1 = (X^2 + 1)^2 = (X + 1)^4$. For this reason, this ring is not a field as it contains divisors of 0. Indeed, $(X + 1)(X^3 + X^2 + X + 1) = 0$.

AES works with blocks of 128 bits using round keys of 128 bits.

Definition: The internal state of the algorithm is the block from the main register at a given moment. This state is represented by a 4×4 matrix of bytes:

$$S = \begin{pmatrix} s_{00} & \cdots & s_{03} \\ \vdots & \ddots & \vdots \\ s_{30} & \cdots & s_{33} \end{pmatrix}$$

The round key has a similar representation:

$$K_i = \begin{pmatrix} k_{00} & \dots & k_{03} \\ \vdots & \ddots & \vdots \\ k_{30} & \dots & k_{33} \end{pmatrix}$$

The algorithm is built up from the following elementary operations:

Sub-Bytes or **S-box**. This is the only one operation of **confusion** and is applied byte-wise. If we consider a byte as element of the field $\mathbb{F}_{256}$, this operation works as $x \rightsquigarrow Ax^{-1} + b$. More exactly, the operation $x \rightsquigarrow x^{-1}$ is the multiplicative inverse for $x \neq 0$ and is taken as $0 \rightsquigarrow 0$ otherwise. Then we apply the affine transformation $y \rightsquigarrow z = Ay + b$ in the affine space $\mathbb{F}_2^8$ as follows:

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

Sub-Bytes can be perform once for all possible bytes and saved in a list or dictionary, which is fast consulted using binary search. The list or dictionary can be used also to read the inverse of this permutation of the set $\{0, \dots, 255\}$.

Now to a operation of pure **diffusion**:

Shift Rows. This is a cyclic shift of the state matrix:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{pmatrix}$$

Mix Columns. Every column of the state matrix is considered to be a polynomial of degree ≤ 3 with coefficients in $\mathbb{F}_{256}$. The operation Mix Columns, which does both confusion and diffusion, can be defined as:

$$b_0 + b_1X + b_2X^2 + b_3X^3 = (a_0 + a_1X + a_2X^2 + a_3X^3)(2 + X + X^2 + 3X^3) \bmod (X^4 + 1),$$

where the coefficients 2 and 3 mean ω and $1 + \omega$, according to the interpretation given modulo a degree 8 irreducible polynomial used to define the arithmetic of the field $\mathbb{F}_{256}$.

This operation can be also written down as follows:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

As this matrix is invertible in the ring $\mathcal{M}_{4\times4}(\mathbb{F}_{256})$, the inverse operation consists also in multiplication with some matrix.

Add Round Key. This means $S \rightsquigarrow S \oplus K_i$.

Round key computation. Let k be the principal 128 bit key. We divide it in four equal blocks $k = (k_0, k_1, k_2, k_3)$. Let $RC_i = X^i \bmod (X^8 + X^4 + X^3 + X + 1)$ the round constant. The following algorithm produces the round keys as $k_i = (w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3})$. Define **Rot Bytes** to be

the rotation of the word left-wise with one bit. Sub Bytes, which has been previously defined, is applied to every byte in a binary word.

$w_0 = k_0$; $w_1 = k_1$; $w_2 = k_2$; $w_3 = k_3$;
for $i = 1$ **to** 10 **do**
    $T =$ SubBytes RotBytes w_{4i-1};
    $T = T \oplus RC_i$;
    $w_{4i} = w_{4i-4} \oplus T$;
    $w_{4i+1} = w_{4i-3} \oplus w_{4i}$;
    $w_{4i+2} = w_{4i-2} \oplus w_{4i+1}$;
    $w_{4i+3} = w_{4i-1} \oplus w_{4i+2}$;
end;

AES. Now we have all the necessary elements to present the encryption algorithm:

$S = S \oplus k_0$;
for $i = 1$ **to** 9 **do**
    $S =$ MixColumns ShiftRows SubBytes S;
    $S = S \oplus k_i$;
end;
$S =$ ShiftRows SubBytes S;
$S = S \oplus k_{10}$;

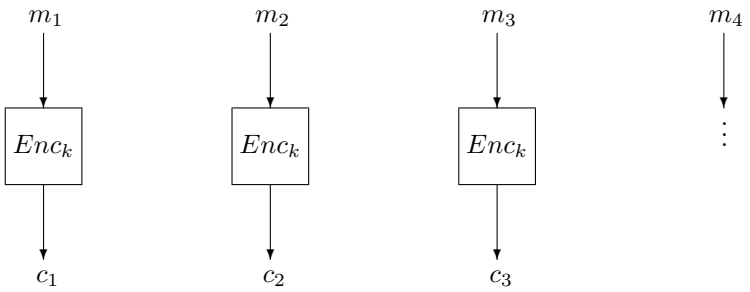
12 Modes of operation for blockcyphers

Let A be a finite alphabet and $Enc : K \times A^n \to A^n$ an encryption system for blocks of length n. How can we apply the system Enc to encrypt and decrypt messages of arbitrary length? From the practice of cryptography, 5 classical modes of operation crystallized and got proper names. Every mode of operation has its own advantages and disadvantages, and they will be shortly mentioned.

In general, a plain text m is divided in blocks $m_1||m_2||m_3 \ldots$, all of length n. If the last block is shorter than n, it will be completed with blanks or with other conventional character. Encrypted blocks are denoted c_1, c_2, c_3, ..., and the cypher is $c = c_1||c_2||c_3|| \ldots$.

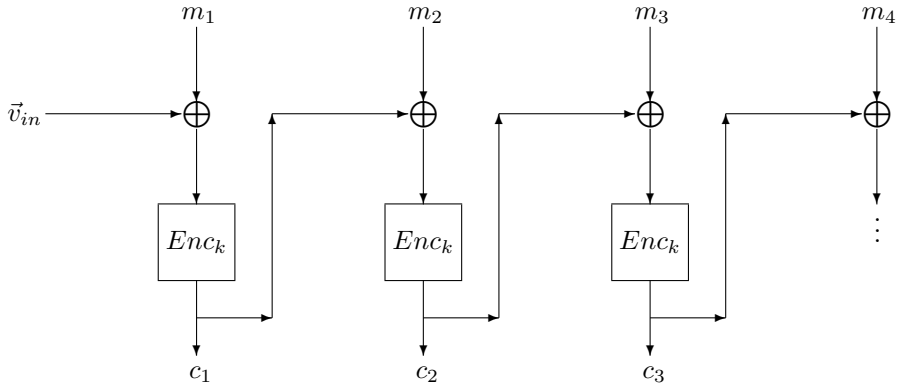
ECB Mode. Electronic Code-book Mode. Every block is encrypted for its own, and they are all encrypted with the same key. The blocks can be parallelly encrypted, and this leads to a fast encryption, respectively decryption. A mistake corrupts only the actual block. However, this mode presents only a low level of security.

$$\begin{aligned} c_i &= Enc_k(m_i), \\ m_i &= Dec_k(c_i). \end{aligned}$$



ECB mode.

CBC Mode. Cypher Blockchaining Mode. The first block is added binary with a random word, called **initialisation vector**. Then, the result is encrypted. Every other block, before being encrypted, is added with the encryption of the preceding block. Main disadvantage: if a block in the encrypted message is corrupted by transmission, it will corrupt the decryption for all the following blocks. Moreover, if there is a mistake during the encryption, this will corrupt the encryption for all following blocks.



CBC mode.

So the encryption is:

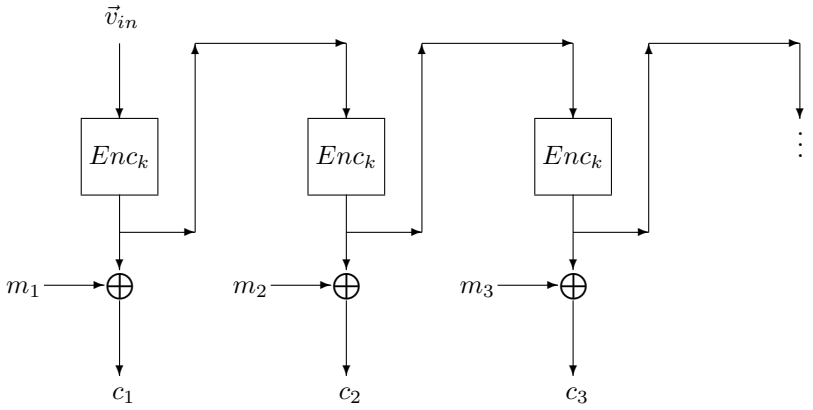
$$\begin{aligned} c_1 &= Enc_k(m_1 \oplus \vec{v}_{in}), \\ c_i &= Enc_k(m_i \oplus c_{i-1}), \end{aligned}$$

where $i \geq 2$, and the decryption is:

$$\begin{aligned} m_1 &= Dec_k(c_1) \oplus \vec{v}_{in}, \\ m_i &= Dec_k(c_i) \oplus c_{i-1}. \end{aligned}$$

OFB Mode. Output Feedback Mode. In this mode of operation, one encrypts only the initialisation vector once, twice, etc. The message block m_i is encrypted only by binary addition with the respective i-th encryption of the initialisation vector. So the encryption algorithm is used only as a pseudo-random generator. The advantage is that this pseudo-random generator

can be computed before the transmission takes place, and that we do not need the decryption function *Dec* to perform the decryption. The disadvantage is again that if by encryption some block from the pseudo-random sequence is corrupted by some mistake, the message cannot be decrypted anymore.



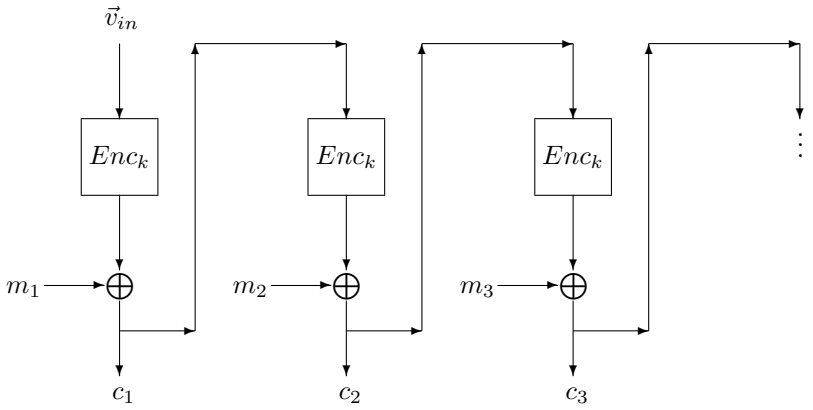
OFB mode.

So the encryption and the decryption are:

$$\begin{aligned} c_i &= Enc_k^i(\vec{v}_{in}) \oplus m_i, \\ m_i &= Enc_k^i(\vec{v}_{in}) \oplus c_i, \end{aligned}$$

unde $i \geq 1$.

CFB Mode. Cypher Feedback Mode. This mode starts similarly with OFB. The block m_1 is binarily added with the encryption of the initialisation vector, to get the first cypher block c_1. The second block is binarily added to the encryption of c_1, to compute c_2, and so on. CFB is a combination of OFB and CBC. This is a very secure method, but is much slower then OFB and ECB. CFB also does not use the function *Dec* for the decryption. It has the same problems with corrupted blocks, like all other chain methods.



CFB mode.

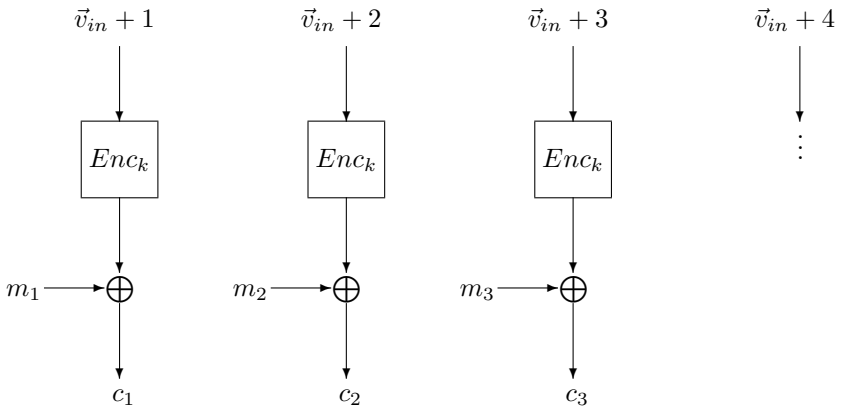
The encryption is:

$$\begin{aligned} c_1 &= m_1 \oplus Enc_k(\vec{v}_{in}), \\ c_i &= m_i \oplus Enc_k(c_{i-1}), \end{aligned}$$

where $i \geq 2$, and the decryption is:

$$\begin{aligned} m_1 &= c_1 \oplus Enc_k(\vec{v}_{in}), \\ m_i &= c_i \oplus Enc_k(c_{i-1}). \end{aligned}$$

CTR Mode. Counter Mode. In the counter mode, the initialisation vector is modified from block to bloc in some previsible way. It can be translated in a natural number, and for every new block one adds one unit to this number. The results are taken as strings and encrypted. Because the block methods work with confusion and diffusion, the results will be very unsimilar. So like in the OFB mode, the encryption is used only as a pseudo-random generator, and the decryption function is not used for decryption.



CTR mode.

Encryption and decryption are done using following rules:

$$\begin{aligned} c_i &= Enc_k(\vec{v}_{in} + i) \oplus m_i, \\ m_i &= Enc_k(\vec{v}_{in} + i) \oplus c_i, \end{aligned}$$

unde $i \geq 1$.

13 Hash functions

In this section, an object is said to be *difficult to find* if in order to compute it we need exponential time $\mathcal{O}(2^n)$, where n is a parameter of security. In most cases, the object is a string and n is its length.

Let A be some alphabet. A hash function $h : A^* \to A^n$ is a function relatively easy to compute that is expected to fulfill properties like:

1. *Resistance to pre-image.*

   Given $y \in A^n$, it is difficult to find x with $h(x) = y$.

2. *Resistance to the second pre-image.*

   Given $x \in A^*$, it is difficult to find $x' \neq x$ with $h(x) = h(x')$.

3. *Resistance to collisions.*

   It is difficult to find a pair $x, x' \in A^*$ with $x \neq x'$ and $h(x) = h(x')$.

Observe that:

$$\text{RC} \to \text{R2P} \to \text{RP}$$

Other properties, which are necessary in applications:

- Less collisions as possible, so good distribution of the values.

- Chaos. By chaos we understand that similar inputs produce very different outputs. Ideally, if inputs differ in one bit, the outputs should differ in at least half of their bits.

- Confusion. Given the value of the function, you have no hint about its argument.

- Surjectivity.

- Efficiency.

The Merkle-Damgard Construction. Supposing that we already have a function $f : \{0,1\}^s \to \{0,1\}^n$ with $s > n$, resistent to collisions, we construct a function $h : \{0,1\}^* \to \{0,1\}^n$.

input $m = m_1||m_2||\dots||m_t$;
$H = H_0$;
for $i = 1$ **to** t **do**
    $H = f(H||m_i)$;
end;
output H

This idea can be applied as follows. Let $d = s - n$. One completes m with zeros, untill one gets the length td, multiple of d. This input is divided in t blocks. If H_0 is a given sequence, of length n, then at every new step the function f gets an input of length $n + d = s$. □

Observation: Let $f : \{0,1\}^8 \to \{0,1\}^4$ some compression function which is resistent to collisions. If we apply the method given before to $m_1 = 010$ and $m_2 = 0100$, we get:

$$h(m_1) = f(0100\ 0000) = h(m_2).$$

A better method is to mark the end of the message with an 1. In this case:

$$\begin{aligned} h(m_1) &= f(0101\ 0000), \\ h(m_2) &= f(0100\ 1000), \end{aligned}$$

which are in general different.

To better understand a compression function, we present here the function **MD4**. For $u, v, w \in \{0,1\}^{32}$ define:

$$\begin{aligned} f(u,v,w) &= (u \wedge v) \vee (\neg u \wedge w) \\ g(u,v,w) &= (u \wedge v) \vee (u \wedge w) \vee (v \wedge w) \\ h(u,v,w) &= u \oplus v \oplus w \end{aligned}$$

component-wise. There is a current state (A, B, C, D) with $A, B, C, D \in \{0,1\}^{32}$. Take the following initial values:

$$\begin{aligned} H_1 &= 0x67452301, \\ H_2 &= 0xEFCDAB89, \\ H_3 &= 0x98BADCFE, \\ H_4 &= 0x10325476, \end{aligned}$$

and constants (y_j, z_j, s_j) which depend on every given round j. The input is divided in blocks:

$$X = X_0||X_1||\dots||X_{15},$$

and every block contains 32 bits. So $|X| = 16 \times 32 = 512$. For the first step, the internal state is initialized with the start value:

$$(A, B, C, D) = (H_1, H_2, H_3, H_4)$$

The algorithm consists of 3 packages of 16 rounds every. The first package is:

for $j = 0$ **to** 15 **do**

$t = A \oplus f(B, C, D) \oplus X_{z_j} \oplus y_j$;

$(A, B, C, D) = (D, t <<< s_j, B, C)$

end;

The second package, with $j = 16$ to 31, is identic with the first one, just that it uses g instead of f. The third package, with $j = 32$ to 47, uses h instead of g. At the end one computes:

$$(A, B, C, D) = (A, B, C, D) \oplus (H_1, H_2, H_3, H_4).$$

So the function MD4 : $\{0,1\}^{512} \to \{0,1\}^{128}$ is a succession of confusions and difusions. □

Merkle-Damgard is not the only one method to construct a general hash function. One can construct hash functions also using **block encryptions**, if they have good properties of diffusion and confusion. Those methods are generically called *one-way compression functions*. Here are some examples:

Matyas-Meyer-Oseas

$$H_i = f(x_i, H_{i-1}) = Enc_{H_{i-1}}(x_i) \oplus x_i$$

Davies-Meyer

$$H_i = f(x_i, H_{i-1}) = Enc_{x_i}(H_{i-1}) \oplus H_{i-1}$$

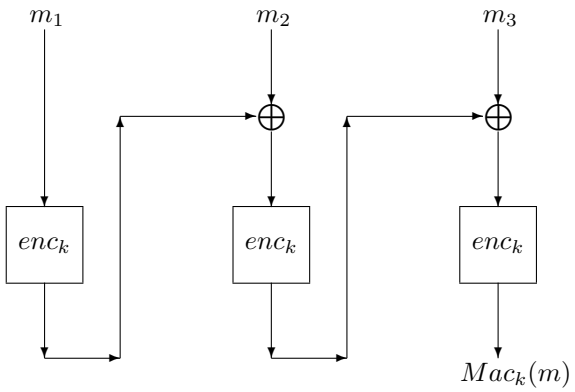
Miyaguchi-Preenel

$$H_i = f(x_i, H_{i-1}) = Enc_{H_{i-1}}(x_i) \oplus x_i \oplus H_{i-1}$$

One of the main applications of a hash function is the authentification of messages by methods also called MAC, *Message Authentification Codes*. For this protocols, the message is sent together with the value of a hash function. This function can be computed only if the sender knows a supplimentary authentification key, so this is a supplementary measure of security. This is done in automatised protocols, and the receiver does not decrypt the cypher anymore if it is not authentic. The best variant of authentificationis the following:

$$Enc_{k_1}(m)||Mac_{k_2}(Enc_{k_1}(m)).$$

Here is important that Mac is a different algorithm as Enc and the two keys have to be different as well. As function Mac you can use any performant hash function, or ever the following one, called CBC-Mac. Here the function enc is another encryption algorithm then the function Enc used for encryption. Example for 3 blocks:



CBC-Mac

Part III
Public key cryptography

14 Public key and private key

The public key cryptography is justified by the fact that sometimes two participants in a communication are not in a symmetric situation. Sometimes there is some authority (state agency, bank, etc) which is contacted by users, and this communication is not symmetric. The state agency receives requests and complaints, that cannot be immediately answered. The citizens do not want to make public their private requests, but they want the agency to decrypt their message. If the Ministry of Finance would have a symmetric key with every citizen, they should administrate around 22.000.000 keys. Such a system would be equally expensive, as slow and unsecure.

The idea is to replace the 22.000.000 symmetric keys (k_i) with one pair of keys (p, s). The public key p is known by all people. Everyone can compute an encrypted message:

$$c = Enc_p(m),$$

and can send it to the authority using a public channel. The secret (private) key s is known only by the authority. They can compute the plain text:

$$m = Dec_s(c),$$

but nobody else could compute this message without the secret key. Also, the knowledge of the public kee p, the observation of the encryption process for some chosen texts, and so on, do not help in finding out the secret key.

The public key cryptography is based on the existence of **one-way functions**. Those functions are easy to compute as direct function but difficult to compute as inverse function. A classical example of one-way function is factoring. If the product of two primes:

$$(p, q) \to pq,$$

is easy to compute, it is much more difficult to find back the primes, given the product:

$$pq \to (p, q).$$

15 Mathematical problems

In this section we list some mathematical problems. The security of public key cryptography is based on their (supposed) difficulty. Here p and q are unknown primes, $p \neq q$, and $N = pq$ is known.

FACTORING. Given $N = pq$, product of two primes, find p.

RSA. Given e such that $\gcd(e, (p-1)(q-1)) = 1$ and given c, find m and d such that $m^d = c \bmod N$.

QUADRES. Given a, determine if a is a square modulo N.

SQRROOT. Given a such that $a = x^2 \bmod N$, find x satisfying the equality.

If FACTORING would be solved by some determinist polynomial time algorithm, then the same would be true for RSA, QUADRES and SQRROOT. This is not evident, but we will give the reasons in the following sections.

DLP. Given $(G, \cdot)$ a finite abelian group, and given $g, h \in G$ such that $h \in \langle g \rangle$, find a natural number x such that $h = g^x$. This problem is called *Discrete Logarithm.*

DHP. Given $(G, \cdot)$ a finite abelian group, $g \in G$, and two elements $a, b \in G$ such that $a = g^x$ and $b = g^y$, find a $c \in G$ such that $c = g^{xy}$. This problem is called *Diffie-Hellman.*

DDH. Given $(G, \cdot)$ a finite abelian group, $g \in G$, $a = g^x$, $b = g^y$ and $c = g^z$, decide if $z = xy$.

If DLP would be solvable in deterministic polynomial time, the same would be true for DHP and DDH.

For the time being, for none of these problems deterministic algorithms in polynomial time are known. On the other side, it is known that those problems belong to NP, but none of them is known to be NP-complete.

16 Squares in prime fields

Consider the function $sq : \mathbb{F}_p \to \mathbb{F}_p$ given by $sq(x) = x^2$. Observe that $sq : \mathbb{F}_p^\times \to \mathbb{F}_p^\times$ is a homomorphism of groups. As $sq(x) = sq(-x)$, $sq(\mathbb{F}_p^\times) \leq \mathbb{F}_p^\times$ of index $[\mathbb{F}_p^\times : sq(\mathbb{F}_p^\times)] = 2$. So exactly $(p-1)/2$ elements are squares, for p odd prime.

Definition: Let p be a prime. The symbol of Legendre is the function:

$$\left(\frac{\cdot}{p}\right) : \mathbb{Z} \to \{-1, 0, 1\},$$

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p|a, \\ +1 & \text{if } a \bmod p \in sq(\mathbb{F}_p^\times), \\ -1 & \text{otherwise.} \end{cases}$$

Observe that:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \bmod p,$$

and this computation is fast if one uses the fast exponentiation algorithm. For composite numbers, the following computation rules can be applied:

Gauss' Quadratic Reciprocity. If p and q are odd primes, then:

$$\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)(-1)^{\frac{(p-1)(q-1)}{4}},$$

and explicitly:

$$\left(\frac{q}{p}\right) = \begin{cases} -\left(\frac{p}{q}\right) & p \bmod 4 = q \bmod 4 = 3, \\ +\left(\frac{p}{q}\right) & \text{otherwise.} \end{cases}$$

Other computation rules are:

$$\begin{aligned} \left(\frac{q}{p}\right) &= \left(\frac{q \bmod p}{p}\right), \\ \left(\frac{qr}{p}\right) &= \left(\frac{q}{p}\right)\left(\frac{r}{p}\right), \\ \left(\frac{2}{p}\right) &= (-1)^{\frac{p^2-1}{8}}. \end{aligned}$$

Example:

$$\left(\frac{15}{17}\right) = \left(\frac{3}{17}\right)\left(\frac{5}{17}\right) = \left(\frac{17}{3}\right)\left(\frac{17}{5}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = (-1)(-1)^3 = 1.$$

Observation: QUADRES is easier then SQRROOT. Indeed, a fast computation algorithm would be as well a fast decision algorithm.

We will show that for $N = pq$, FACTORING and SQRROOT have practically the same difficulty, speaking in terms of polynomial time calculation. The first remark is that there is a fast algorithm to compute the square root modulo a prime.

Shanks' Algorithm for the square root modulo p prime.

Inputs: p prime, $n \in \mathbb{Z}_p^\times$.

Find $Q = 1 \bmod 2$, S: $p - 1 = 2^S Q$.

Find $z \in \mathbb{Z}_p^\times$: $\left(\frac{z}{p}\right) = -1$.

$M = S$.

$c = z^Q$.

$t = n^Q$.

$R = n^{\frac{Q+1}{2}}$.

Loop: **if** $t = 1$ **return** R.

Find least i: $0 < i < M \wedge t^{2^i} = 1$ **or return NO**.

$b = c^{2^{M-i-1}}$.

$M = i$.

$c = b^2$.

$t = tb^2$.

$R = Rb$.

Theorem: *Shanks' Algorithm computes efficiently the square root modulo a prime.*

Proof: During the algorithm, the following quantities remain constant:

$$\begin{aligned} c^{2^{M-1}} &= -1, \\ t^{2^{M-1}} &= 1, \\ R^2 &= tn. \end{aligned}$$

At the beginning:

$$c^{2^{M-1}} = z^{2^{S-1}Q} = z^{\frac{p-1}{2}} = -1,$$

because z is not a quadratic remainder,

$$t^{2^{M-1}} = n^{2^{S-1}Q} = n^{\frac{p-1}{2}} = 1,$$

because n is a quadratic remainder,

$$R^2 = n^{Q+1} = nt.$$

At any new iteration, let (M', c', t', R') be the new values of the tuple (M, c, t, R).

$$c'^{2^{M'-1}} = (b^2)^{2^{i-1}} = c^{2^{M-i} \cdot 2^{i-1}} = c^{2^{M-1}} = -1.$$

$$t'^{2^{M'-1}} = (tb^2)^{2^{i-1}} = t^{2^{i-1}} b^{2^i} = (-1)(-1) = 1,$$

where $t^{2^{i-1}} = -1$ because $t^{2^i} = 1$ and $t^{2^{i-1}} \neq 1$, if not i would not be the smallest value with the given property.

$$b'^{2^i} = c^{2^{M-i-1} \cdot 2^i} = c^{2^{M-1}} = -1,$$

$$R'^2 = R^2 b^2 = tnb^2 = t'n.$$

But M becomes smaller any new iteration, so finally t will be equal with 1, and in this case $R^2 = n$. All computations are modulo p. □

Example: Let $p = 17$ and $n = 15$. All computations are modulo 17. As we have seen,

$$\left(\frac{15}{17}\right) = 1,$$

so it makes sense to look for the square root.

$$p - 1 = 16 = 2^4 \cdot 1,$$

so $Q = 1$ and $S = 4$. The number $z = 3$ is not a quadratic residuum, because:

$$3^8 = 9 \cdot 27^2 = 9 \cdot 10^2 = 9 \cdot 4 \cdot 25 = 9 \cdot 4 \cdot 8 = 18 \cdot 16 = 1 \cdot (-1) = -1.$$

So we start with:

$$\begin{aligned} M = S &= 4 \\ c = z^Q &= 3 \\ t = n^Q &= 15 \\ R = n^{\frac{1+1}{2}} &= 15 \end{aligned}$$

By successive squaring we find the smallest $i < 4$ such that $15^{2^i} = 1$.

$$15^2 = (-2)^2 = 4 \rightsquigarrow 4^2 = 16 = -1 \rightsquigarrow (-1)^2 = 1.$$

So $i = 3$.

$$b = c^{2^{M-i-1}} = 3^{2^{4-3-1}} = 3,$$

$$\begin{aligned} M = i &= 3 \\ c = b^2 &= 9 \\ t = tb^2 = 15 \cdot 9 &= 16 \\ R = Rb = 15 \cdot 3 &= 11 \end{aligned}$$

By successive squaring we find the smallest $i < 3$ such that $16^{2^i} = 1$.

$$16^2 = (-1)^2 = 1.$$

So $i = 1$.

$$b = c^{2^{M-i-1}} = 9^{2^{3-1-1}} = 3 \cdot 27 = 30 = 13 = -4,$$

$$\begin{aligned} M = i &= 1 \\ c = b^2 &= 16 \\ t = tb^2 = 16 \cdot 16 &= 1 \\ R = Rb = 11 \cdot (-4) &= -10 = 7 \end{aligned}$$

As $t = 1$ one gets the result 7. Indeed, $49 \bmod 17 = 15$, OK. The other solution is $-7 = 10$.

Observation: The maximal number of iterations in Shanks' Algorithm is the exponent of 2 in $p - 1$. This number is $< \log p$, so the algorithm works in polynomial time.

Theorem: *FACTORING and SQRROOT are polynomial time equivalent for $N = pq$, which are products of two primes.*

Proof: Suppose that we can solve FACTORING in polynomial time. So we find primes the p and q. Using Shanks we find s_1 and s_2 such that:

$$\begin{aligned} s_1 &= \sqrt{n} \bmod p, \\ s_2 &= \sqrt{n} \bmod q. \end{aligned}$$

Applying the Chinese Remainder Theorem, we find an x with $0 \leq x < pq$ such that:

$$\begin{aligned} s_1 &= x \bmod p, \\ s_2 &= x \bmod q. \end{aligned}$$

So:

$$\begin{aligned} x^2 \bmod p &= n, \\ x^2 \bmod q &= n, \end{aligned}$$

so $x^2 \bmod pq = n$ because p and q are relatively prime.

Now suppose that we can solve SQRROOT in polynomial time modulo $N = pq$ with $p \neq q$ prime. Choose randomly some $x \in \mathbb{Z}_N^\times$. Let $z = x^2 \bmod N$ and $y = \sqrt{z} \bmod N$. There are 4 such roots, because $N = pq$. With a probability of 50% we get $y \neq \pm x \bmod N$, or we repeat the procedure. From $x^2 = y^2 \bmod N$ we get $N|(x+y)(x-y)$. But $N \not| (x+y)$ and $N \not| (x-y)$ it results that the factors of N are distributed here. So $\gcd(N, x-y)$ is p or q. □

17 Cipolla's Algorithm

Cipolla's algorithm to compute a square root modulo p is easier to memorize then Shanks' Algorithm, and is also more suitable for exercises. It can be presented by a pseudo-program, as follows:

Input $n \in sq(\mathbb{F}_p^\times)$.

Find $a \in \mathbb{F}_p^\times$ **such that** $a^2 - n \notin sq(\mathbb{F}_p^\times)$.

$\omega = \sqrt{(a^2 - n)} \notin \mathbb{F}_p$

$x = (\omega + a)^{\frac{p+1}{2}}$.

Output x.

Theorem: *Cipolla's algorithm produces a* $x \in \mathbb{F}_p$ *with* $x^2 = n$.

Proof: The computation is done in the field $\mathbb{F}_p[\omega] \simeq \mathbb{F}_{p^2}$ built with the rule $\omega^2 = a^2 - n$, which is not a square in $\mathbb{F}_p$. As:

$$\omega^{p-1} = (\omega^2)^{\frac{p-1}{2}} = -1,$$

it is true that $\omega^p = -\omega$. So for $u, v \in \mathbb{F}_p$,

$$(u + \omega v)^p = (u - \omega v).$$

Hence:

$$x^2 = (a+\omega)^{p+1} = (a+\omega)(a+\omega)^p = (a+\omega)(a-\omega) = a^2 - \omega^2 = a^2 - \big(a^2 - n\big) = n.$$

But the polynomial $X^2 - n$ has two solutions x and $p - x$ in $\mathbb{F}_p$, it follows that the computed solution x belongs to $\mathbb{F}_p$. □

Example: Let us compute again $\sqrt{15}$ in $\mathbb{F}_{17}$. We choose $a = 1$ and $a^2 - n = 1 - 15 = 3 \bmod 17$. The element 3 is not a square, because the sequence 3^n is:

$$3, 9, 10, 13, 5, 15, 11, -1,$$

so $3^8 = -1$. If 3 would have been a square, then $3^8 = z^{16} = 1$, because $\mathbb{Z}_{17}^\times$ has 16 elements. So $\omega^2 = 3$. We compute:

$$x = (1+\omega)^9.$$

As $9 = 8 + 1$,

$$\begin{aligned} (1+\omega)^2 &= 4 + 2\omega, \\ (1+\omega)^4 &= 11 - \omega, \\ (1+\omega)^8 &= 5 - 5\omega, \\ (1+\omega)^9 &= (1+\omega)(5-5\omega) = 5(1-3) = 7. \end{aligned}$$

Indeed $x = 7$ and $x = 17 - 7 = 10$ are the square roots. □

18 RSA

RSA (Rivest - Shamir - Adleman) is the oldest encryption algorithm with public key, and is still used. We will present first the variant which was originally published and the original proof. In a second step we make a deeper algebraic analysis. The algorithm is presented as a dialogue between two sides: the authority Alice and the user Bob.

Setup.

Alice:

1. Chooses big prime numbers $p \neq q$.
2. Computes $N = pq$ and $\varphi(N) = pq(1 - \frac{1}{p})(1 - \frac{1}{q}) = (p-1)(q-1) = |\mathbb{Z}_N^\times|$.
3. Chooses e such that $\gcd(e, \varphi(N)) = 1$.
4. Announces the public key (N, e).
5. Finds d such that $d = e^{-1} \bmod \varphi(N)$.
6. Keeps the secret key d.

Protocol.

Bob: Wants to encrypt a message $m \in \mathbb{Z}_N$. Sends $c = m^e \bmod N$.

Alice: Computes $m = c^d \bmod N$.

Theorem: *If $p \neq q$ are prime numbers, $N = pq$, $\varphi(N) = (p-1)(q-1)$, $\gcd(e, \varphi(N)) = 1$, $d = e^{-1} \bmod \varphi(N)$ and $0 \leq m < N$, then:*

$$(m^e \bmod N)^d \bmod N = m.$$

Proof: Because $ed = 1 \bmod (p-1)(q-1)$, it follows that:

$$\begin{aligned} ed &= 1 \bmod (p-1), \\ ed &= 1 \bmod (q-1), \end{aligned}$$

hence for some $k, h \in \mathbb{N}$,

$$\begin{aligned} ed &= k(p-1) + 1, \\ ed &= h(q-1) + 1. \end{aligned}$$

We observe that:

$$(m^e \bmod N)^d \bmod N = m^{ed} \bmod N.$$

If $\gcd(m, p) = 1$ then $m^{p-1} = 1 \bmod p$ because $\mathbb{Z}_p$ is a field. So:

$$m^{ed} = m^{k(p-1)+1} = (m^{p-1})^k m = 1^k m = m \bmod p.$$

If $p|m$, then:

$$m^{ed} = 0 = m \bmod p.$$

But in both cases:

$$m^{ed} = m \bmod p.$$

Analogously one proves that:

$$m^{ed} = m \bmod q.$$

As p and q are different prime numbers, one has $\gcd(p,q)=1$, and from $N=pq$ it follows:

$$m^{ed} = m \bmod N.$$

□

Example: $N = 91 = 7 \cdot 13$, $\varphi(91) = 6 \cdot 12 = 72$. Choose $e = 5$. To compute $d = e^{-1} \bmod 72$, apply the extended algorithm of Euclid:

$$\begin{aligned} 72 &= 14 \cdot \underline{5} + \underline{2}, \\ \underline{5} &= 2 \cdot \underline{2} + 1. \end{aligned}$$

$$1 = \underline{5} - 2 \cdot \underline{2} = \underline{5} - 2 \cdot (-14 \cdot \underline{5}) = 29 \cdot 5 \bmod 72,$$

so $d = 29$. If $m = 10$, $c = 10^5 = 10^4 \cdot 10 \bmod 91$. Successive squaring yields:

$$10 \rightsquigarrow 100 = 9 \rightsquigarrow 81,$$

so $c = 81 \cdot 10 \bmod 91 = -100 \bmod 91 = 82$. For decryption, compute $m = 82^{29} \bmod 91 = 82^{16+8+4+1} \bmod 91$. Again successive squaring yields:

$$82^2 = (-9)^2 = 81 \rightsquigarrow 81^2 = (-10)^2 = 9 \rightsquigarrow 81 \rightsquigarrow 81^2 = (-10)^2 = 9,$$

so $82^2 = 81$, $82^4 = 9$, $82^8 = 81$, $82^{16} = 9$. Finally:

$$82^{16+8+4+1} = 9 \cdot 81 \cdot 9 \cdot 82 = (-10)(-10)(-9) = (-10) \cdot 90 = (-10) \cdot (-1) = 10 \bmod 91.$$

□

Example: $N = 91 = 7 \cdot 13$, $\varphi(91) = 6 \cdot 12 = 72$. Choose $e = 13$. To compute $d = e^{-1} \bmod 72$, apply Euclid's extended algorithm:

$$\begin{aligned} 72 &= 5 \cdot \underline{13} + \underline{7}, \\ \underline{13} &= \underline{7} + \underline{6}, \\ \underline{7} &= \underline{6} + 1. \end{aligned}$$

$$1 = \underline{7} - \underline{6} = \underline{7} - (\underline{13} - \underline{7}) = 2 \cdot \underline{7} - \underline{13} = 2 \cdot (-5 \cdot \underline{13}) - \underline{13} = (-11) \cdot 13 = 61 \cdot 13 \bmod 72.$$

So $d = 61$. But now, there is an unexpected difficulty. Let $0 \leq m < 91$ be some message. Recall the representetion given by the Chinese Remainder Theorem:

$$m \bmod 91 = (m \bmod 7, m \bmod 13).$$

So:

$$m^{13} \bmod 91 = (m^{6 \cdot 2+1} \bmod 7, m^{12+1} \bmod 13) = (m \bmod 7, m \bmod 13) = m \bmod 91.$$

With other words, the public key $e = 13$ does not transform any plain message by encryption, and the same does its corresponding secret key, $e = 61$. □

What did it happen in the last example? As it comes out, the just presented theory of RSA is not deep enough, although is formulated so in almost all cryptography books and online resources. In order to avoid the use of some *dead* keys and to find only the true keys, we start now a theoretic development that will end with a reformulation of the RSA protocol.

Definition: An integer is called *square-free* if no square integer different from 1 divides it. In the prime number decomposition of a square-free number, all primes arize with exponent 1.

Definition: The least common multiple of two integers $\operatorname{lcm}(a,b)$ is defined as:

$$\operatorname{lcm}(a,b) = \frac{ab}{\gcd(a,b)}.$$

Definition: Let N be a square-free integer and $N = p_1 p_2 \dots p_k$ be its prime factor decomposition. Define:

$$\lambda(N) = \text{lcm}(p_1 - 1, p_2 - 1, \dots, p_k - 1).$$

Generalized Little Theorem of Fermat: *Let N be square-free and let $e = a\lambda(N) + 1$. Then for all $x \in \mathbb{Z}$,*

$$x^e = x \bmod N.$$

Proof: For all $i = 1, \dots, k$, $e = (p_i - 1)f_i + 1$, so:

$$n^e = (n^{p_i - 1})^{f_i} \cdot n = n \bmod p_i,$$

because if $n = 0 \bmod p_i$ then $n^e = 0 \bmod p_i$, and in case $n \neq 0 \bmod p_i$ then $n^{p_i - 1} = 1 \bmod p_i$. As the primes p_i are all different, we apply the Chinese Remainder Theorem. □

If p is an odd prime, and $N = 2p$, it follows $n = n^p \bmod 2p$. This result is already stronger then Fermat's Little Theorem, which states only $n = n^p \bmod p$.

This result explains in full generality the dead keys observed in the last example. What we have to do now, is to replace $\varphi(N)$ with $\lambda(N)$ and to reformulate the RSA cryptographic protocol:

Setup.

Alice:

1. Chooses big prime numbers $p \neq q$.
2. Computes $N = pq$ and $\lambda(N) = \text{lcm}(p - 1, q - 1)$.
3. Chooses e such that $\gcd(e, \lambda(N)) = 1$.
4. Announces the public key (N, e).
5. Finds d such that $d = e^{-1} \bmod \lambda(N)$.
6. Keeps the secret key d.

Protocol.

Bob: Has to encrypt a message $m \in \mathbb{Z}_N$. He sends $c = m^e \bmod N$.

Alice: Computes $m = c^d \bmod N$.

Theorem: *If $p \neq q$ are prime numbers, $N = pq$, $\lambda(N) = \text{lcm}(p - 1, q - 1)$, $\gcd(e, \lambda(N)) = 1$, $d = e^{-1} \bmod \lambda(N)$ and $0 \leq m < N$, then:*

$$(m^e \bmod N)^d \bmod N = m.$$

Proof: As $ed = 1 \bmod \lambda(N)$, we have both:

$$\begin{aligned} ed &= 1 \bmod (p - 1), \\ ed &= 1 \bmod (q - 1), \end{aligned}$$

so for some integers $k, h \in \mathbb{N}$,

$$\begin{aligned} ed &= k(p - 1) + 1, \\ ed &= h(q - 1) + 1. \end{aligned}$$

For this point on, the proof works exactly like the preceding proof. □

At the first sight we loose a lot of possible encryption keys, so we could make less encryptions. This is not true, as proven below:

Theorem: *Let* $3 \leq e_1 < \varphi(N)$ *be such that* $\gcd(e_1, \varphi(N)) = 1$ *and let* $e_2 = e_1 \bmod \lambda(N)$. *Then for all* $m \in \mathbb{Z}$*:*

$$m^{e_1} = m^{e_2} \bmod N.$$

Proof:

$$m^{e_1} = m^{k\lambda(N)+e_2} = (m^{\lambda(N)})^k m^{e_2} = u^k m^{e_2} = m^{e_2} \bmod p.$$
$$m^{e_1} = m^{k\lambda(N)+e_2} = (m^{\lambda(N)})^k m^{e_2} = v^k m^{e_2} = m^{e_2} \bmod q.$$

Here $u = 0$ if $p|m$, $u = 1$ if $p \not| m$, $v = 0$ if $q|m$ and $v = 1$ if $q \not| m$. By the Chinese Remainder Theorem again,

$$m^{e_1} = m^{e_2} \bmod N.$$

□

So only the keys between 3 and $\lambda(N)$ lead to different encryptions, what follows above $\lambda(N)$ are only repetitions. If someone uses a key e between 3 and $\varphi(N)$ for encryption, the decryptor may use the smaller key $(e \bmod \lambda(N))^{-1} \bmod \lambda(N)$ for decryption. Moreover, in order to avoid dead keys, one may choose a true encryption key $e' \in [3, \lambda(N) - 1]$ but publish $e = k\lambda(N) + e'$.

The National Institute for Standards in Technology (NIST) presents RSA in the variant with $\lambda(N)$. It is a mystery why this development is not presented in books and online resources.

We repeat the first example with $\lambda(N)$.

Example: $N = 91 = 7 \cdot 13$, $\lambda(91) = \mathrm{lcm}(6, 12) = 12$. We choose $e = 5$. For computing $d = e^{-1} \bmod 12$, we apply again Euclid extended:

$$\begin{aligned} 12 &= 2 \cdot \underline{5} + \underline{2}, \\ \underline{5} &= 2 \cdot \underline{2} + 1. \end{aligned}$$

$$1 = \underline{5} - 2 \cdot \underline{2} = \underline{5} - 2 \cdot (-2 \cdot \underline{5}) = 5 \cdot 5 \bmod 12,$$

so $d = 5$. If $m = 10$, $c = 10^5 = 10^4 \cdot 10 \bmod 91$. Successive squaring yields:

$$10 \rightsquigarrow 100 = 9 \rightsquigarrow 81,$$

so $c = 81 \cdot 10 \bmod 91 = -100 \bmod 91 = 82$. For decryption, compute $m = 82^5 \bmod 91 = 82^{4+1} \bmod 91$. Again by successive squaring:

$$82^2 = (-9)^2 = 81 \rightsquigarrow 81^2 = (-10)^2 = 9,$$

so $82^2 = 81$, $82^4 = 9$. Finally:

$$82^{4+1} = 9 \cdot 82 = 9 \cdot (-9) = -81 = 10 \bmod 91.$$

□

The practical advantage of RSA mod $\lambda(N)$ is an evidence.

19 Elgamal

Another important public key cryptosystem is Elgamal. Unlike RSA, the security of Elgamal is based on the alledged difficulty of the discrete logarithm, DLG.

Let M be the set of plain texts and $(G, \cdot, 1)$ a cyclic group. We suppose that there is an action of the group G onto M:

$$\alpha : G \times M \to M,$$

denoted $\alpha(g, m) = gm$. This action has the following properties:

$$\begin{aligned} g_1(g_2 m) &= (g_1 g_2)m, \\ 1m &= m, \\ h \neq 1 &\rightarrow \exists m \; hm \neq m. \end{aligned}$$

These properties easily imply that the group has at most the number of elements of M. Now let $g \in G$ be a generator of the cyclic group. In an abstract form, the algorithm works as follows:

Setup.

Alice:

1. Chooses randomly a number x. This number is the secret key.
2. Computes the public key $h = g^x \in G$.
3. Announces the public key (G, g, h).

Protocol.

Bob:

1. Must encrypt the message $m \in M$.
2. Chooses a random number y. This is its temporary key and shall be used only for this encryption.
3. Computes $c_1 = g^y \in G$ and $c_2 = h^y m \in M$.
4. Sends the pair (c_1, c_2) to Alice.

Alice:

1. Finds out $m = (c_1^x)^{-1} c_2$.

Theorem: *The cryptosystem Elgamal correctly decrypts the messages.*

Proof:

$$(c_1^x)^{-1} c_2 = (g^{xy})^{-1}(g^{xy} m) = (g^{-xy} g^{xy})m = 1m = m.$$

□

There are many possibilities to realize Elgamal.

Example: Let p a big prime number of about 1024 bits. Suppose that $p - 1$ is divisible with a relatively big prime q, of about 160 bits. Choose an element $g \in \mathbb{F}_p^\times$ of order q in the following way. Find $r \in \mathbb{F}_p^\times$ such that:

$$g = r^{\frac{p-1}{q}} \neq 1.$$

Then $|\langle g \rangle| = q$. Take $M = \mathbb{F}_p$ and $G = \langle g \rangle$.

Example: Take $M = G = \mathbb{F}_{p^k}^\times$ using a representation of the field such that a generator of $\mathbb{F}_{p^k}^\times$ is known. A primitive polynomial of degree k yields such a representation.

Example: Take $M = \mathbb{Z}_{dp^k}$ with $d \in \{1, 2\}$ and p odd prime, $G = \mathbb{Z}_{dp^k}^\times$ and g a generator of G.

Example: Take G a cyclic subgroup of an elliptic curve.

Now we introduce a key exchange protocol, which is very similar with Elgamal.

Protocolul Deffie-Hellman. Two distant parts, Alice and Bob, posses a symmetric cryptosystem (Enc, Dec). For starting their communication, they must establish a common symmetric key k. To this goal, they establish a cyclic group G and a generator g, with $\langle g \rangle = G$.

1. Alice chooses a random number a, while Bob chooses a random number b.
2. Alice sends g^a to Bob. Bob sends g^b to Alice.
3. Alice computes $k = (g^b)^a = g^{ab}$. Bob gets the same $k = (g^a)^b = g^{ab}$.

Observation: The security depends not so much of the algebraic structure of the group, but of its representation. The most natural representation of a cyclic group is $(\mathbb{Z}/n\mathbb{Z}, +, 0)$ but this does not garantees any kind of security. Take for example $n = 100$. Bob and Alice choose the generator $g = 47$. Suppose that they choose $a = 30$ and $b = 40$. In the first step Alice sends $g^a = ga \bmod 100 = 10$ to Bob and Bob sends $g^b = gb \mod 100 = 80$ to Alice. In the second step Alice computes $80^{30} = 80 \cdot 30 \bmod 100 = 0$ and Bob computes $10^{40} = 10 \times 40 \bmod 100 = 0$.

How can someone break this protocol? Agent Eve computes $g^{-1} \bmod n = 47^{-1} \bmod 100$.

$$\begin{aligned} 100 &= 2 \cdot \underline{47} + \underline{6}, \\ \underline{47} &= 7 \cdot \underline{6} + \underline{5}, \\ \underline{6} &= \underline{5} + 1 \end{aligned}$$

$$1 = \underline{6} - \underline{5} = \underline{6} - (\underline{47} - 7 \cdot 6) = 8 \cdot \underline{6} - \underline{47} = -8 \cdot 2 \cdot \underline{47} - \underline{47} = (-17) \cdot 47 = 83 \cdot 47.$$

So $47^{-1} \bmod 100 = 83$. Now Eve computes $a = 83 \cdot 10 \bmod 100 = 30$ and finds $k = 30 \cdot 80 = 0$.

In a second try, Bob and Alice choose the group $(\mathbb{Z}_{101}^{\times}, \cdot, 1)$. This multiplicative group of a finite field is also cyclic and has 100 elements, so is isomorphic with $(\mathbb{Z}_{100}, +, 0)$. We show that $g = 2$ is a multiplicative generator. As $100 = 4 \cdot 25$, $100/2 = 50 = 32 + 16 + 2$ and $100/5 = 20 = 16 + 4$. We prepare those powers of 2, whose exponents are itself powers of 2:

$$2 \rightsquigarrow 2^2 = 4 \rightsquigarrow 2^4 = 16 \rightsquigarrow 2^8 = 54 \rightsquigarrow 2^{16} = 88 \rightsquigarrow 2^{32} = 68.$$

$$2^{50} \bmod 101 = 68 \cdot 88 \cdot 4 = 100 \neq 1,$$

$$2^{20} \bmod 101 = 88 \cdot 16 = 95 \neq 1.$$

The order of 2 modulo 101 is 100, so 2 is indeed a generator $(\mathbb{Z}_{101}^{\times}, \cdot, 1)$.

This time Alice sends $2^a = 2^{30} = 2^{16}2^8 2^4 2^2 = 88 \cdot 54 \cdot 16 \cdot 4 = 17 \bmod 101$ to Bob and Bob sends $2^b = 2^{40} = 2^{32}2^8 = 68 \cdot 54 = 36 \bmod 101$ to Alice. Now, Alice computes $36^{30} = 1 \bmod 101$ and Bob computes $17^{40} = 1 \bmod 101$. This time Eve does not win enough information by only intercepting this communication, because the discrete logarithm is a difficult problem for her.

The next sections deal with more exotic cryptosystems.

20 Rabin

The security of Rabin's cryptosystem is based as well on the difficulty of factoring, via computing a square root modulo pq. This system is much more fast than other cryptosystems, but introduces an indeterminacy which makes it not suitable for the automatic encryption of streams. Every block should be desambiguated, and this does not work while sending pictures, movies or sound.

Setup.

Alice:

1. Chooses two prime numbers $p = 4k + 3 \neq q = 4m + 3$.
2. The secret key is the pair (p, q).
3. The public key is the pair (N, B) where $N = pq$ and $B \in \{0, 1, \ldots, N-1\}$ is randomly choosen.

Protocol.

Bob:

1. Encrypts a message $m \in M$.
2. Computes and sends $c = m(m + B) \bmod N$.

Alice:

1. Computes $m = 2^{-1}(\sqrt{B^2 + 4c} - B) \bmod N$.

Theorem: *Alice computes four possible messages, but only one of them is m.*

Proof: From

$$m^2 + mB - c = 0 \bmod N,$$

it follows:

$$m = 2^{-1}(\sqrt{B^2 + 4c} - B) \bmod N.$$

The difficulty is to compute the square root. If $\Delta = B^2 + 4c$, first compute the square root from $\Delta \bmod p$ and another one from $\Delta \bmod q$. For every root $a \bmod r$, the element $r - a$ is a root as well. Explicitly the computation works as follows:

- Compute square roots modulo $\bmod p$ and $\bmod q$. For primes of the form $4k + 3$,

$$m_p = \Delta^{\frac{1}{4}(p+1)} \bmod p,$$

$$m_q = \Delta^{\frac{1}{4}(q+1)} \bmod q.$$

- Apply Euclid's extended algorithm to find $y_p, y_q \in \mathbb{Z}$ such that:

$$y_p p + y_q q = 1.$$

So $y_p = p^{-1} \bmod q$ and $y_q = q^{-1} \bmod p$.

- Apply the Chinese Remainder Theorem to find the square roots modulo N:

$$\begin{aligned} r_1 &= (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod N, \\ r_2 &= N - r_1, \\ r_3 &= (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod N, \\ r_4 &= N - r_3. \end{aligned}$$

□

Example: Let $p = 127$ and $q = 131$, so $N = 16637$. Let $B = 12345$. If $m = 4410$, then:

$$c = m(m + B) \bmod N = 4663.$$

For decryption, we can work with the so called reduced discriminant.

$$\Delta' = \left(\frac{B^2}{4} + c\right) \bmod N = 1500.$$

$$\begin{aligned} \sqrt{\Delta'} &= \pm 22 \bmod 127, \\ \sqrt{\Delta'} &= \pm 37 \bmod 131. \end{aligned}$$

Using CRT we get ± 3705 and ± 14373. After subtracting $B/2 \bmod N$, we get: 4410, 5851, 15078, 16519. The first message is m. □

21 Paillier

We start with a particular form of the cryptosystem Pailler.

Setup.

Alice:

1. Chooses two prime numbers $p \neq q$ such that $\gcd(pq, (p-1)(q-1)) = 1$. If p and q have equal bit-lengths, the last condition is automatically fulfilled.
2. The public key is $N = pq$.
3. The secret key is a d such that:

$$\begin{aligned} d &= 1 \bmod N, \\ d &= 0 \bmod (p-1)(q-1). \end{aligned}$$

   The number d is easily computed using the Chinese Remainder Theorem.

Protocol.

Bob:

1. Must encrypt a message m with $0 \leq m < N$.
2. Chooses an $r \in \mathbb{Z}_{N^2}^{\times}$.
3. Computes $c = (1+N)^m r^N \bmod N^2$.

Alice:

1. Computes $t = c^d \bmod N^2$.
2. Finds $m = (t-1)/N$ by division with remainder in $\mathbb{Z}$.

Theorem: *The Paillier cryptosystem decrypts messages unambiguously.*

Proof:

$$t = c^d \bmod N^2 = (1+N)^{md} r^{dN} \bmod N^2.$$

As $(p-1)(q-1) | d$, $r^{dN} = 1$ because r is an element in a group with:

$$\varphi(N^2) = \varphi(p^2q^2) = p^2q^2(1 - \frac{1}{p})(1 - \frac{1}{q}) = pq(p-1)(q-1) = N(p-1)(q-1)$$

elements. So:

$$t = (1+N)^{md} \bmod N^2 = 1 + mdN \bmod N^2.$$

The last equality takes place because the following terms from the development of Newton's binomial are divisible with N^2. Recall that $d = 1 \bmod N$,

$$t = 1 + mN \bmod N^2.$$

As $m < N$, $t = 1 + mN$, so

$$m = \frac{t-1}{N}.$$

□

For a more general form of this algorithm, we need the following definition:

Definition: L is the function defined as:

$$L(x) = \frac{x-1}{N}.$$

Here we mean the division with remainder in $\mathbb{Z}$, and we consider only the quotient. L acts like a discrete logarithm in $\mathbb{Z}_{N^2}^{\times}$.

Setup.

Alice:

1. Chooses two prime numbers $p \neq q$ such that $\gcd(pq, (p-1)(q-1)) = 1$. Let $N = pq$ and $\lambda = \operatorname{lcm}(p-1, q-1)$.
2. Chooses $g \in \mathbb{Z}_{N^2}^{\times}$ such that $N | \operatorname{ord}(g)$.
3. Let $\mu = (L(g^{\lambda} \bmod N^2))^{-1} \bmod N$.
4. The public key is the pair (N, g).
5. The secret key is the pair (λ, μ).

Protocol.

Bob:

1. Must encrypt a message m with $0 \leq m < N$.
2. Chooses an $r \in \mathbb{Z}_{N^2}^{\times}$.
3. Computes and sends $c = g^m r^N \bmod N^2$.

Alice:

1. Computes $m = L(c^{\lambda} \bmod N^2)\mu \bmod N$.

Theorem: *The Paillier cryptosystem unambiguously decrypts the message.*

Proof:

$$L(c^{\lambda} \bmod N^2)\mu \bmod N = L(g^{\lambda m} r^{\lambda N} \bmod N^2)\mu \bmod N =$$
$$= L(g^{\lambda m} \bmod N^2)(L(g^{\lambda} \bmod N^2))^{-1} \bmod N = m\mu^{-1}\mu = m.$$

□

In the case that p and q have the same binary length, $g = N+1$, $\lambda = \varphi(N)$ and $\mu = \varphi(N)^{-1} \bmod N$, we get back the first variant of the algorithm.

Why does the function $L(x)$ play the role of a discrete logarithm? As we see:

$$(1+N)^x = 1 + Nx \bmod N^2,$$

so for $y = (1+N)^x \bmod N^2$ one gets:

$$x = \frac{y-1}{N} \bmod N,$$

$$L((1+N)^x \bmod N^2) = x,$$

for every $x \in \mathbb{Z}_N$. The security of Paillier is assured by the difficulty of factorisation. Its practical value is assured by a special case, in which the discrete logarithm is easy to compute.

The Paillier cryptosystem has some homomorphic properties:

$$D(E(m_1 r_1)E(m_2 r_2)) \bmod N^2) = (m_1 + m_2) \bmod N,$$

$$D(E(m_1 r_1)^k \bmod N^2) = k m_1 \bmod N.$$

Those properties make this primitive suitable for applications like the electronic vote or the shared secret computation.

22 Goldwasser-Micali

Goldwasser-Micali is a method of bit encryption whose security is based on the difficulty of QUADRES, respectively of the factorization.

Setup.

Alice:

1. Chooses two prime numbers $p \neq q$. Let $N = pq$.
2. Chooses $k \in \mathbb{Z}_N$ such that:
$$\left(\frac{k}{p}\right) = -1 \ \wedge \ \left(\frac{k}{q}\right) = -1.$$
This can be done choosing non-quadratic residues modulo p and q and applying the Chinese remainder Theorem. A faster idea is to choose $p = 3 \bmod 4$, $q = 3 \bmod 4$ and $k = N - 1$. Such a number k is called a pseudo-square. In $\mathbb{Z}_N$ it has the Legendre symbol of a square, but it is not a square.
3. The public key is the pair (N, k).
4. The secret key is the pair (p, q).

Protocol.

Bob:

1. Must encrypt the message $m \in \{0, 1\}$.
2. Chooses $r \in \mathbb{Z}_N^\times$ randomly.
3. Computes and sends $c = k^m r^2 \bmod N$.

Alice:

1. Computes $\left(\frac{c}{p}\right) = c^{\frac{p-1}{2}} \bmod p$. If c is a square, $m = 0$; otherwise, $m = 1$.

Part IV
Attack methods

23 Easy attacks against RSA

Theorem: *If we know N, and one pair e, d, then we can factorize N, and consequently we can compute the secret key for every public key.*

Proof: We know that $ed-1 = s(p-1)(q-1)$. Choose an $x \in \mathbb{Z}_N^\times$, then:

$$x^{ed-1} = 1 \bmod N.$$

In fact $ed-1$ is a multiple of $\lambda(N)$, and $ed-1$ is even. We write $ed-1 = 2^t h$, where h is odd.

Let $b = x^h \bmod N$. Its order is 1 or a power of 2. We also know that its order in $\mathbb{Z}_N^\times$ is the least common multiple of its orders in $\mathbb{Z}_p^\times$ and in $\mathbb{Z}_q^\times$ respectively. We hope that these orders are different, and this happens with a significant probability.

We compute $g = \gcd(b-1, N)$. If $g \neq 1$, we have already got a multiple of p or of q. If $g = 1$, we substitute b with $b^2 \mod N$, and we compute again $\gcd(b-1, N)$.

If at some point $g = N$, then the orders of $x^h \mod p$ and $x^h \mod q$ were equal, and we choose another x.

□

Example: Let $N = 21$, and we know $e = d = 5$. Then $ed - 1 = 24 = 3 \cdot 8$. Choose $x = 2$. As $\gcd(2, 21) = 1$, this choice does not help us to find directly a divisor of N. Now $b = 2^3 = 8$, $b - 1 = 7$, $\gcd(7, 21) = 7$ and $21 = 7 \cdot 3$. □

Theorem: *If we know N and $\varphi(N)$, we can factorize N.*

Proof:

$$\varphi(N) = (p-1)(q-1) = pq - p - q + 1 = N - (p+q) + 1,$$
$$p + q = N + 1 - \varphi(N).$$

So, p and q are solutions of the degree two equation:

$$X^2 - (N + 1 - \varphi(N))X + N = 0.$$

□

Theorem: *If a message m is sent using the same module N but different public keys e_1 and e_2 such that $\gcd(e_1, e_2) = 1$, then one can decrypt the message without knowing any secret key.*

Proof: Consider $\gcd(e_1, e_2) = 1$ and the encrypted messages:

$$\begin{aligned} c_1 &= m^{e_1} \bmod N, \\ c_2 &= m^{e_2} \bmod N. \end{aligned}$$

The attacker computes:

$$\begin{aligned} t_1 &= e_1^{-1} \bmod e_2, \\ t_2 &= \frac{e_1 t_1 - 1}{e_2}. \end{aligned}$$

So $e_2 t_2 = e_1 t_1 - 1$. Finally:

$$c_1^{t_1} c_2^{-t_2} = m^{e_1 t_1} m^{-e_2 t_2} = m^{1 + e_2 t_2 - e_2 t_2} = m \mod N.$$

□

Theorem: *Let m a message with $0 \leq m < \min(N_1, \ldots, N_e)$, where the numbers N_i are pairwise relatively prime. Using the same key e one computes the RSA cyphers:*

$$c_j = m^e \bmod N_j.$$

Then one can fast compute m from $c_1, \ldots, c_e$ without using any secret key.

Proof: Using the Chinese Remainder Theorem and the fact that the moduli N_j are pairwise relatively prime, we find a c such that:

$$c = c_j \bmod N_j,$$

for all $j = 1, \ldots, e$ and with $0 \leq c < N_1 N_2 \ldots N_e$. As $0 \leq m^e < N_1 N_2 \ldots N_e$ and $c = m^e \bmod N_1 N_2 \ldots N_e$ it follows that $c = m^e$, hence $m = \sqrt[e]{c}$. □

Observation: This attack is the simplest form of the so called Hastad attack. In a later section we will see more complicated variants that use the Coppersmith method.

Observation: This attack works also with only two cyphers, if the value of m is sufficiently small. We need that $m^e < N_1 N_2$ which means $m < \sqrt[e]{N_1 N_2}$. □

Conclusion: These examples show that the primitive method RSA has some native weaknesses. So it is necessary to improve the method in a way that makes it less deterministic. Such cautions are:

1. To randomly modify the modulus N and the public key e, at best with every new transmission.
2. To use relatively big keys like $e = 65537$ and relatively big prime numbers p and q.
3. To complete the message m with a block of random bits, at best with a block of a length equal with the length of m. This way, even if one repeats the message, it will not produce the same cypher. It produces every time a different cypher.

24 The Wiener attack

The Wiener attack is another method of attack against RSA. This method works against relatively small decryption exponents. What does this really mean, will follow from the theory.

Continuous fraction decomposition: Let $\alpha \in \mathbb{R}$ be a real number. We construct the sequences $(\alpha_n) \subset \mathbb{R}$ and $(a_n), (p_n), (q_n) \subset \mathbb{Z}$ recursively, as follows:

$$\alpha_0 = \alpha,\ p_0 = a_0,\ q_0 = 1,$$

$$p_1 = a_0 a_1 + 1,\ q_1 = a_1,$$

$$a_i = [\alpha_i],$$

$$\alpha_{i+1} = \frac{1}{\alpha_i - a_i},$$

$$p_i = a_i p_{i-1} + p_{i-2},\ i \geq 2,$$

$$q_i = a_i q_{i-1} + q_{i-2},\ i \geq 2.$$

The integers (a_i) build the continuous fraction of α. The first one might be negative, all the remaining members being natural numbers. This sequence is periodic if and only if α is rational or is a real solution of some degree 2 equation with coefficients in $\mathbb{Z}$. Fractions

$$\frac{p_i}{q_i}$$

are called convergents of α. The sequence $p_i/q_i \to \alpha$ and for all i, $\gcd(p_i, q_i) = 1$.

Fact: If $p, q \in \mathbb{Z}$, and if:

$$\left|\alpha - \frac{p}{q}\right| \leq \frac{1}{2q^2},$$

then there exists some i such that $p = p_i$ and $q = q_i$, in other words, p/q is a convergent of α.

Now we make the following suppositions: $N = pq$, the prime numbers p and q are relatively close to each other, say $q < p < 2q$, and d is relatively small, say:

$$d < \frac{1}{3}\sqrt[4]{N}.$$

Also the public key e is smaller then $\Phi = (p-1)(q-1)$. As we know, there is some k such that $ed - k\Phi = 1$, which means:

$$\Big|\frac{e}{\Phi} - \frac{k}{d}\Big| = \frac{1}{d\Phi}.$$

The numbers Φ and N can be considered close to each other, because:

$$|N - \Phi| = |p + q - 1| < 3\sqrt{N}.$$

We observe that $\frac{e}{N}$ is a good approximation of $\frac{k}{d}$:

$$\Big|\frac{e}{N} - \frac{k}{d}\Big| = \Big|\frac{ed - kN}{dN}\Big| = \Big|\frac{ed - k\Phi - Nk + k\Phi}{dN}\Big| = \Big|\frac{1 - k(N - \phi)}{dN}\Big| \leq$$

$$\leq \Big|\frac{3k\sqrt{N}}{dN}\Big| = \frac{3k}{d\sqrt{N}} < \frac{1}{3d^2} < \frac{1}{2d^2}.$$

This is true, because from $e < \Phi$ results:

$$k < d < \frac{1}{3}\sqrt[4]{N}, \quad \frac{k}{d} < \frac{\sqrt[4]{N}}{3d}.$$

With the last inequality one gets:

$$\frac{3k}{d\sqrt{N}} < \frac{3}{3d}\frac{\sqrt[4]{N}}{\sqrt{N}} = \frac{1}{d\sqrt[4]{N}} < \frac{1}{3d^2}.$$

But $\gcd(k, d) = 1$, so k/d is a reduced fraction.

Conclusion: *$\frac{k}{d}$ is a convergent of $\frac{e}{N}$. This way, one can find out the decryption key d checking out some convergents of $\frac{e}{N}$.*

Example:

$N = 9\ 449\ 868\ 410\ 449$,

$e = 6\ 792\ 605\ 526\ 025$.

We don't know for shure that the method will work, but we know that we must look for those convergent denominators d which are:

$$d < \frac{1}{3}\sqrt[4]{N} < 584.$$

We compute the convergents of $\alpha = e/N$. We get:

$$1, \frac{2}{3}, \frac{3}{4}, \frac{5}{7}, \frac{18}{25}, \frac{23}{32}, \frac{409}{569}, \frac{1659}{2308}.$$

We find that $d = 569$ checking that:

$$(m^e)^d = m \bmod N,$$

for some randomly chosen values of m.

25 The Coppersmith method

The attacks using the Coppersmith method work against RSA.

First Theorem of Coppersmith: *Let N be some integer and let $f \in \mathbb{Z}[x]$ be a monic polynomial of degree d. Let $X = N^{\frac{1}{d}-\varepsilon}$ for some $\frac{1}{d} > \varepsilon > 0$. Given the pair $\langle N, f\rangle$ the attacker Eve can effectively find all integers $x_0 < X$ satisfying $f(x_0) \equiv 0 \pmod N$. The time is bounded by the time necessary for the LLL algorithm to run for a lattice of dimension $O(w)$ with $w = \min\left\{\frac{1}{\varepsilon}, \log_2 N\right\}$.*

The Hastad attack. A caution against the classical Hastad attack was to pad the original message with random bits. We will see that in some situations, even this caution is not sufficient. Suppose that the attacker catches cyphers $c_i = f_i(m)^e$ with $1 \leq i \leq k$, where the functions f_i are linear. With other words, the message m has been completed with some address or with some request number before it has been encoded - like $m_i = i \cdot 2^m + m$ is encrypted with the key e, and then sent to user i. If the number of users to receive such messages is big enough, the message can be easily decrypted.

Hastad's Theorem: *Let $N_1, \ldots, N_k$ relatively prime integers and $N_{\min} = \min(N_i)$. Let $g_i \in \mathbb{Z}_{N_i}[x]$ be k polynomials of maximal degree q. Suppose that there is a unique $m < N_{\min}$ which satisfies $g_i(m) = 0 \bmod N_i$ for all $1 \leq i \leq k$. Moreover, suppose $k > q$. Then there exists an efficient algorithm that computes m.*

Proof: As the numbers N_i are pairwise relatively prime, one can apply the Chinese Remainder Theorem to compute the coefficients T_i such that $T_i = 1 \bmod N_i$ and $T_i = 0 \bmod N_j$ for $j \neq i$. Let $g(x) = \sum T_i g_i(x)$. We know that $g(m) = 0 \bmod \prod N_i$. The degree of g is at most q. Using the Coppersmith Method we can compute all integer roots x_0 of the congruence $g(x_0) = 0 \bmod \prod N_i$ such that:

$$|x_0| < \Big(\prod N_i\Big)^{\frac{1}{q}}.$$

But we know that:

$$m < N_{\min} < \Big(\prod N_i\Big)^{\frac{1}{k}} < \Big(\prod N_i\Big)^{\frac{1}{q}}.$$

□

The Franklin-Reiter attack. Let (N, e) be the public key of Alice. Let $m_1, m_2 \in \mathbb{Z}_N$ two messages that satisfy $m_1 = f(m_2) \bmod N$, where $f(x) = ax + b \in \mathbb{Z}_N[x]$ is a known linear polynomial. For example, Bob gets his message back together with an error message, and resends the whole message to the same address. Let c_1 and c_2 the encrypted messages sent by Bob.

The Franklin-Reiter Theorem: *If (N, e) is an RSA public key and if $m_1 \neq m_2 \in \mathbb{Z}_N$ are two messages such that $m_1 = f(m_2) \bmod N$ for some linear polynomial $f = ax + b \in \mathbb{Z}_N[x]$ with $b \neq 0$, then m_1 and m_2 can be decrypted in a time which is quadratic in $\log_2 N$ and e.*

Proof: As $c_1 = m_1^e \bmod N$, we know that m_2 is a common root of the polynomials $g_1(x) = f(x)^e - c_1 \in \mathbb{Z}_N[x]$ and $g_2(x) = (ax + b)^e - c_2 \in \mathbb{Z}_N[x]$. The linear factor $x - m_1$ divides both polynomials. So the attacker computes $\gcd(g_1(x), g_2(x))$ using Euclid's algorithm for polynomials. If this polynomial is not linear, it must be decomposed and there are more solutions to check. □

Coppersmith Attack. This attack works when the padding with random bits has been done with insufficiently many bits. Eve intercepts the message and keeps it. Bob sees that Alice does not answer and resends the encrypted message, this time with another padding.

Second Theorem of Coppersmith: *Let (N, e) be some RSA public key, where N has length of n bits. Let $m = [n/e^2]$. Let $M \in \mathbb{Z}_N$ a message of at most $n - m$ bits. Let $m_1 = 2^m M + r_1$ and $m_2 = 2^m M + r_2$ where $r_1 \neq r_2$ and $0 \leq r_1, r_2 \leq 2^m$. If some attacker has the cyphers c_1 and c_2, but does not know r_1 and r_2, she can however decrypt M efficiently.*

Proof: Let $g_1(x, y) = x^e - c_1$ and $g_2(x, y) = (x + y)^e - c_2$. We know that for $y = r_2 - r_1$ those polynomials have $x = m_1$ as a common root. With other words, $\Delta = r_2 - r_1$ is a root of the

resultant $\mathrm{res}_x(g_1, g_2) \in \mathbb{Z}_N[y]$. Moreover,

$$|\Delta| < 2^m < N^{\frac{1}{e^2}}.$$

So Δ can be found with the Coppersmith method. Once Δ known, one can find m_2 using the Franklin-Reiter method, and from m_2 we find M. □

Explanation: The resultant is the determinant of the Sylvester Matrix. If $A(x, y)$ has degree $d = 3$ in x and $B(x, y)$ has degree $e = 2$ in x, then the resultant $\mathrm{res}_x(A, B)$ is the determinant of the matrix:

$$\begin{pmatrix} a_0 & 0 & b_0 & 0 & 0 \\ a_1 & a_0 & b_1 & b_0 & 0 \\ a_2 & a_1 & b_2 & b_1 & b_0 \\ a_3 & a_2 & 0 & b_2 & b_1 \\ 0 & a_3 & 0 & 0 & b_2 \end{pmatrix}$$

where a_i and b_j are polynomials in y. There are e columns of a_i and d columns of b_j. If P_k is the vector space of polynomials of degree strictly less than k, then the Sylvester Matrix represents the linear mapping:

$$\varphi : P_e \times P_d \to P_{d+e},$$
$$\varphi(P, Q) = AP + BQ,$$

written relatively to the powers of x in decreasing order, so such that $A = a_0 x^d + \cdots + a_d$ and $B = b_0 x^e + \cdots + b_e$. □

26 Factoring algorithms

Many factoring algorithms work by random walks over remainder classes, to the goal of a collision. The corresponding probability computation is called generically *Birthday paradox*. The original problem is the following: if all school-children in a class compare their birth-dates (only day and month), the probability of a collision is surprisingly big. For 24 children, the probability is already bigger then 1/2.

The following result is also important for finding collisions in hash functions:

Birthday paradox. *If n independent random variables can take m possible values, the probability of no collision occurring is:*

$$\Big(1 - \frac{1}{m}\Big)\Big(1 - \frac{2}{m}\Big) \ldots \Big(1 - \frac{n-1}{m}\Big) = \prod_{i=1}^{n-1} \Big(1 - \frac{i}{m}\Big) \simeq$$

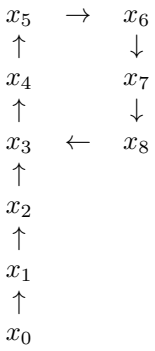
$$\simeq \prod_{i=1}^{n-1} e^{\frac{-i}{m}} = e^{-\sum_{i=1}^{n-1} \frac{i}{m}} = e^{-\frac{n(n-1)}{2m}} \simeq e^{-\frac{1}{2m}(n-\frac{1}{2})^2}.$$

If $n > 1.2\sqrt{m}$ then the probability of a collision is $> 1/2$.

Naive algorithm. Check the numbers d with $2 \leq d \leq \sqrt{n}$ if $d|n$. If a divisor is found, then stop. You can restart the algorithm for the numbers d and n/d. For the RSA moduli, which have the form $N = pq$, if we found p, we have already solved the factorisation. Let m be the length of n in bits. The naive algorithm needs in worst case $O(\sqrt{n}) = O(2^{\frac{m}{2}})$ steps.

Pollard's Rho. This algorithm applies the birthday paradox as follows: Let p be an unknown prime, that divides n. Consider the elements $x_1, \ldots, x_k \in \mathbb{Z}_n$ with $k > 1.2\sqrt{p}$. Then, with probability $> 1/2$, there exist $i \neq j$ with $x_i = x_j \bmod p$. If $x_i \neq x_j$, $d = \gcd(n, x_i - x_j)$ is a divisor of n.

If we construct a pseudo-random sequence $x_{i+1} = f(x_i) \bmod n$ with f polynomial, it has the supplementary property that if $a = b \bmod p$ then $f(a) = f(b) \bmod p$. A collision looks like in the following diagram:



The name Pollard's Rho comes from the similarity of this diagram with the greek letter ρ. Aparently, after computing some x_i, we must compute all $\gcd(n, x_i - x_j)$ with $j < i$, looking for a result which is both different from 1 and from n. (If we get n, we restart with another x_1.) This is a big volume of computation. So it is better to apply the following result, that Donald Knuth attributed to Robert W. Floyd:

Floyd's Lemma: *Let p be some prime. Consider the sequence $(x_k) \subset \mathbb{Z}_n$ with $n > p$. If for $i < j$ one has $x_i = x_j \bmod p$, then there is a $k \in \{i, i+1, \ldots, j-1\}$ with $x_k = x_{2k} \bmod p$.*

Proof: According to the hypothesis there is an initial segment of length i and a cycle of length $j - i$. So we may get a collision if $k > i$ and the difference $2k - k = k$ is a multiple of $j - i$. Now consider $k = i$. If $2k = j$, this is ok. Otherwise, $2k - k = k \neq 0 \bmod (j - i)$. We start adding units to k. After at most $j - i - 1$ steps, we get $2k - k = 0 \bmod (j - i)$. So $k \leq i + (j - i - 1) = j - 1$.

□

The algorithm looks as follows:

input n;

choose $x < n$;

$y = x$;

repeat

$x = f(x) \bmod n$;

$y = f(y) \bmod n$;

$y = f(y) \bmod n$;

$g = \gcd(n, x - y)$;

untill $((x \neq y)$ **and** $(g > 1))$;

return g;

Frequently one uses $f(x) = x^2 + 1$ ore more generally $f(x) = x^2 + a$.

Theorem: *Pollard's Rho finds a divisor of n in time $O(\sqrt[4]{n}) = O(2^{\frac{m}{4}})$.*

Proof: The Birthday Paradox occurs in $O(\sqrt{p})$ steps. But $p \leq \sqrt{n}$. □

Pollard's $p-1$. Let p be a prime number that divides n. There exists a B such that $B! = (p-1)k$. So:

$$2^{B!} = 2^{(p-1)k} = 1 \bmod p,$$

in the field $\mathbb{Z}_p$. Hence $\gcd(2^{B!} - 1, n)$ is a factor of n, if it is smaller than n. The algorithm has the following shape:

$i = 1$;

choose $a \in \{2, 3, \ldots, p-1\}$;

loop

$i = i + 1$;

$a = a^i \bmod n$;

$b = \gcd(a - 1, n)$;

if $(1 < b < n)$ **return** b;

Obsevation: In the worst case $n = pq$ and the primes $x = p$ and $x = q$ are close to each other with $x - 1 = 2y$, where y is prime. In this situation Pollard's $p - 1$ is not better than the naive algorithm. In all situations with $x - 1 = 2p'q'k$, Pollard's $p - 1$ is at least as good as Pollard's Rho.

Fermat. Let $n = pq$ with $p < q$ are odd and different from 1. Then the following numbers are integers:

$$x = \frac{p+q}{2} \quad , \quad y = \frac{p-q}{2},$$

and, moreover,

$$n = (x + y)(x - y) = x^2 - y^2.$$

As $y^2 \geq 0$ we have $x \geq \sqrt{n}$. According to these observations, we construct the following program:

input n;

$x = [\sqrt{n}] + 1$; $z = x^2 - n$;

repeat

if $y = \sqrt{z} \in \mathbb{N}$ **return** $x - y$;

$x = x + 1$;

$z = z + 2x - 1$;

untill $x = n$;

This algorithm is efficient for RSA moduli $n = pq$ only if both primes are close to $\sqrt{n}$.

The quadratic sieve. The idea is similar with Fermat's algorithm above. We must find x and y with $x^2 - y^2 = kn$, meaning:

$$x^2 = y^2 \bmod n.$$

Then if $x \neq \pm y \bmod n$, $\gcd(x \mp y, n)$ is a proper divisor of n. As a square has modulo $n = pq$ at least 4 different square roots, we find a good root with the probability of at least $1/2$.

To this end we look for a basis $\{p_1, p_2, \ldots, p_k\}$, i. e. a set of relatively small prime numbers, such that for some numbers a_i with $i = 1, \ldots, k$, the numbers $a_i^2 - n$ admit only those numbers in their decomposition. There are heuristics to find such a representation.

$$\begin{aligned} a_1^2 - n &= p_1^{j_{1,1}} p_2^{j_{1,2}} \ldots p_k^{j_{1,k}} \\ a_2^2 - n &= p_1^{j_{2,1}} p_2^{j_{2,2}} \ldots p_k^{j_{2,k}} \\ \vdots & \qquad\qquad \vdots \\ a_k^2 - n &= p_1^{j_{k,1}} p_2^{j_{k,2}} \ldots p_k^{j_{k,k}} \end{aligned}$$

We look for a subset $I \subseteq \{1, 2, \ldots, k\}$ such that:

$$\prod_{i \in I} (a_i^2 - n) = \prod_{i \in I} p_1^{j_{i,1}} p_2^{j_{i,2}} \ldots p_k^{j_{i,k}} = p_1^{\sum\limits_{i \in I} j_{i,1}} p_2^{\sum\limits_{i \in I} j_{i,2}} \ldots p_k^{\sum\limits_{i \in I} j_{i,k}}$$

is a square. If we found I, we define:

$$x = \prod_{i \in I} a_i.$$

Then:

$$\prod_{i\in I}(a_i^2 - n) = y^2,$$

$$\prod_{i\in I} a_i^2 = y^2 \bmod n,$$

$$x^2 = y^2 \bmod n.$$

To find I, we look for a vector $(z_1, z_2, \dots z_k) \in \{0,1\}^k$ to solve the following system of linear equations over $\mathbb{F}_2$:

$$\begin{aligned} b_{1,1}z_1 + \cdots + b_{k,1}z_k &= 0 \\ \vdots & \quad \vdots \\ b_{1,k}z_1 + \cdots + b_{k,k}z_k &= 0 \end{aligned}$$

where every $b_{u,v} = j_{u,v} \bmod 2$. This system is solved like every system of linear equations over the field $\mathbb{F}_2$.

27 Algorithms for the discrete logarithm

Let p be a prime number and let g be a primitive root modulo p, i.e. a generator of the multiplicative group $\mathbb{F}_p^\times$. Given some $y \in \mathbb{F}_p^\times$, it is asked a number x such that $y = g^x$. This number is called *discrete logarithm* of y, denoted $x = \log_g y \bmod p$.

Baby Step - Giant Step. This algorithm uses the fact that every $x < p$ can be represented as:

$$x = \lceil\sqrt{p}\rceil x_1 + x_2, \quad \text{cu} \quad 0 \le x_1, x_2 \le \lfloor\sqrt{p}\rfloor.$$

Hence:

$$y = g^x = g^{\lceil\sqrt{p}\rceil x_1 + x_2} = \left(g^{\lceil\sqrt{p}\rceil}\right)^{x_1} \cdot g^{x_2},$$

$$y \cdot \left(g^{-1}\right)^{x_2} = \left(g^{\lceil\sqrt{p}\rceil}\right)^{x_1}.$$

So we compute $z = g^{-1} \bmod p$ using the extended Euclid algorithm, and we compute $h = g^{\lceil\sqrt{p}\rceil} \bmod p$ using the fast exponentiation algorithm. Then we write down two lists as follows:

$$L_1 = \{(x_1, h^{x_1}) \,|\, x_1 = 0, 1, \dots, \lfloor\sqrt{p}\rfloor\},$$

$$L_2 = \{(x_2, yz^{x_2}) \,|\, x_2 = 0, 1, \dots, \lfloor\sqrt{p}\rfloor\}.$$

We look for a coincidence (collision) like $(x_1, w) \in L_1$, $(x_2, w) \in L_2$. Then $x = \lceil\sqrt{p}\rceil x_1 + x_2$ is the discrete logarithm we were looking for.

Theorem: *The Baby Step - Giant Step algorithm needs $O(\sqrt{p}) = O(2^{\frac{m}{2}})$ steps. Here m is the bit-length of p.*

Pollard's Rho for the discrete logarithm. Consider the following discrete intervals:

$$\begin{aligned} M_1 &= \left\{1, \dots, \left\lfloor\frac{p}{3}\right\rfloor\right\}, \\ M_2 &= \left\{\left\lfloor\frac{p}{3}\right\rfloor + 1, \dots, \left\lceil\frac{2p}{3}\right\rceil\right\}, \\ M_3 &= \left\{\left\lceil\frac{2p}{3}\right\rceil + 1, \dots, p-1\right\}. \end{aligned}$$

They build a partition of the interval between 1 and $p-1$ in segments of close lengths $M_1 \cup M_2 \cup M_3 = \{1, \ldots, p-1\}$. We randomly choose a number a and we consider the element $z_0 = g^a y^0 \bmod p$. A pseudo-random sequence z_i is generated by the following rule:

$$z_{i+1} = \begin{cases} z_i^2 \bmod p, & z_i \in M_1, \\ z_i \cdot g \bmod p, & z_i \in M_2, \\ z_i \cdot y \bmod p, & z_i \in M_3. \end{cases}$$

This means that $z_i = g^{a_i} y^{b_i}$ where $a_0 = a$, $b_0 = 0$ and in general:

$$a_{i+1} = \begin{cases} 2a_i \bmod (p-1), & z_i \in M_1, \\ a_i + 1 \bmod (p-1), & z_i \in M_2, \\ a_i \bmod (p-1), & z_i \in M_3, \end{cases} \quad b_{i+1} = \begin{cases} 2b_i \bmod (p-1), & z_i \in M_1, \\ b_i \bmod (p-1), & z_i \in M_2, \\ b_i + 1 \bmod (p-1), & z_i \in M_3. \end{cases}$$

Using Floyd's Lemma, we look for a pair $i = k$ and $j = 2k$ such that $z_i = z_j \bmod p$. This happens with big probability for some $k > 1.2\sqrt{p}$. So:

$$g^{a_i + xb_i} = g^{a_j + xb_j} \bmod p,$$

which means:

$$a_i + xb_i = a_j + xb_j \bmod (p-1),$$
$$x(b_i - b_j) = a_j - a_i \bmod (p-1).$$

As $p-1$ is not prime, one finds more solutions x, which has to be verified. The time is $O(\sqrt{p})$.

28 Abstract definitions of attacks

Chosen plain-text attack. This attack is usually called CPA or passive attack. We suppose that the attacker dispose of an encryption mashine $Enc_k(x)$ but of no decryption mashine.

Chosen cypher-text attack. The CCA1 variant. The attacker has a very limited access to a decryption mashine. The model is motivated by the so called *lunch-time attack*: an officer quit his office for his lunch-break, and a spy comes in covered as a janitor. So the attacker can decrypt only a constant or at most a polynomially bounded number of messages, or even worth, some messages that she cannot choose.

Chosen cypher-text attack. The CCA2 variant, or adaptive chosen cypher-text attack. The attacker can decrypt as many messages as she wants, excepting the message intercepted by the enemy, which is the only one she is really interested in. This hypothesis is justified by the hypothesis that the enemy penetrated your organisation at some low level. She can use the decryption machine as much as she wishes, but has no access to important decisions.

Observation: *A determinist system is not secure against CPA-attacks.*

Example: Suppose that the attacker knows that one of two messages m_1 or m_2 has been encrypted. (These can be "yes" or "no", "sail" or "buy", "girl" or "boy"). The attacker intercepts c and encrypts $c' = Enc_k(m_1)$. If $c' = c$ then $m = m_1$, else $m = m_2$. □

Observation: *Pure RSA is not CPA-secure.*

Proof: The system is determinist □

Observation: *Pure RSA is not CCA-secure.*

Proof: This follows from the homomorphic property of RSA:

$$(m_1 m_2)^e \bmod N = (m_1^e \bmod N)(m_2^e \bmod N) \bmod N.$$

Suppose that the attacker wants to decrypt $c = m^e \bmod N$. The attacker creates the message $c' = 2^e c \bmod N$ and decrypts c', getting a message m'. Now he immediately gets m because:

$$m' \cdot 2^{-1} \bmod N = (2^e c)^d 2^{-1} \bmod N = m.$$

Simply making a system non-determinist is not enough to make a system CCA2-secure, as we see in the following example:

Observation: *Pure Elgamal is not CCA2-secure.*

Proof: The attacker has to decrypt $c = (c_1, c_2) = (g^k, mh^k)$. She creates the message $c' = (c_1, 2c_2)$, she decrypts it, and gets a message m'. Again:

$$m' \cdot 2^{-1} = 2c_2 c_1^{-x} 2^{-1} = 2m2^{-1} = m.$$

□

Observation: *Pure Goldwasser-Micali is not CCA2-secure.*

Proof: The attacker intercepts $c = y^b x^2 \bmod N$. She produces and decrypts $c' = c \cdot z^2 \bmod N$, and finds out directly the same $m \in \{0, 1\}$. □

29 Difficult predicates

Definition: Let S and T be two sets of words. Let $B : S \to \{0, 1\}$ be a predicate and $f : S \to T$ be a one-way function. One says that the predicate B is difficult for f, if $B(x)$ is easy to compute given x, but $B(x)$ is difficult to compute given $f(x)$.

Theorem: *Let G be a cyclic group of order q prime and g a generator of G. Let $B_2(x)$ be the parity predicate, defined by $B_1(x) = x \bmod 2$. Then B_2 is difficult for $f(x) = g^x$. In words, parity is difficult for the discrete logarithm.*

In the proof, let $B_1(x)$ means the last binary digit of x.

Proof: Let $O(h, g)$ be an oracle that computes $B_1(\log_g h)$, the last bit of the discrete logarithm of h. Using this oracle, we could easily compute the discrete logarithm. In the following program $h = g^x$, but x is unknown.

input h;
$t = 2^{-1} \bmod q$;
$y = 0$; $z = 1$;
while $(h \neq 1)$
    $b = O(h, g)$;
    if $b = 1$ **then** $y = y + z$; $h = h/g$; **endif**
    $h = h^t \bmod q$; $z = 2z$;
return y

□

Example: In $\mathbb{F}_{607}$ the element $g = 64$ generates the group G of order 101. One can compute $\log_{64} 56 \bmod 101$, which is $x = 86$. The computations are done modulo 607, but the multiplicative inverse of 2 is computed modulo 101. □

The RSA function $c = m^e \bmod N$ has the following difficult predicates:

$$\begin{aligned} B_1(m) &= m \bmod 2. \\ B_h(m) &= \begin{cases} 0, & m < \frac{N}{2}, \\ 1, & m \geq \frac{N}{2}. \end{cases} \\ B_k(m) &= m \bmod 2^k, \quad k = \log\log N. \end{aligned}$$

Denote the corresponding oracles $O_1(c, N)$, $O_h(c, N)$ and $O_k(c, N)$. Between O_1 and O_h one has the following connection:

$$\begin{aligned} O_h(c, N) &= O_1(c \cdot 2^e \bmod N, N), \\ O_1(c, N) &= O_h(c \cdot 2^{-e} \bmod N, N). \end{aligned}$$

Those predicates are difficult from RSA. For example O_h: if this predicate would be easy, we could invert RSA as follows:

Start with $y = c$, $l = 0$ and $h = N$. As long as $h - l \geq 1$ one repeats the following loop:

$b = O_h(y, n)$;

$y = y \cdot 2^e \bmod N$;

$m = (h + l)/2$;

if $b = 1$ **then** $l = m$ **else** $h = m$

At the end one computes $m = [h]$.

Part V
Signatures and key-exchange protocols

30 Deffie-Hellman

One of the first byproducts of the public key cryptography was the key-exchange protocol Diffie-Hellman. Alice and Bob communicate through an unsecure public channel. They dispose of a very good symmetric cryptosystem (Enc, Dec) but at the beginning of a communication session they want to establish a common key which has to remain a secret for all third person. Alice and Bob know a cyclic group G and a generator g. Possible, at the beginning of the communication one of them propose the pair (G, g). Even if the group and the generator are known, this does not diminish the security of the protocol, because of the difficulty of the discrete logarithm.

Recall the protocol:

Alice	Bob
Chooses a, sends g^a, computes $k = (g^b)^a$.	Chooses b, sends g^b, computes $k = (g^a)^b$.

As they both compute $k = g^{ab}$, they keep the same key for their symmetric communication, without having it communicated explicitely.

If $G = (\mathbb{Z}_n, +, 0)$, the protocol is absolutely unsecure if g is public. Indeed, the attacker can compute $h = g^{-1} \bmod n$ by the extended Euclidian algorithm. The element h exists because g is invertible modulo n if and only if $\gcd(g, n) = 1$ if and only if g is a generator of G. As g^a in G is in fact $ga \bmod n$, the attacker computes $hga \bmod n = a \bmod n$, and then computes the key $k = (gb)a \bmod N$.

If G is a multiplicative subgroup in the multiplicative group of a ring of remainders $(\mathbb{Z}_n^\times, \cdot, 1)$ or of a finite field $(\mathbb{F}_q^\times, \cdot, 1)$, the protocol is secure if the group is sufficiently big. In the case $(\mathbb{Z}_p^\times, \cdot, 1)$ one can take p a prime of ~ 1024 bits.

If G of p elements is the cyclic subgroup $G =< P >$ of an elliptic curve $E(\mathbb{F}_p)$, for the same level of security a p of 160 bits is enough. This is because for $\mathbb{F}_p^\times$ there are subexponential algorithms to compute the discrete logarithm, but no such algorithms are known for elliptic curves.

The pure Diffie-Hellman protocol can be corrupted by the *man in the middle* attack. In this attack, the agent Eve replaces Bob in his communication with Alice and replaces Alice in her communication with Bob, staying metaphorically between them, like a translator. This works as follows:

Alice		Eve	Eve		Bob
a	$\longrightarrow$	g^a	n	$\longrightarrow$	g^n
g^m	$\longleftarrow$	m	g^b	$\longleftarrow$	b
g^{am}		g^{am}	g^{bn}		g^{bn}

So Alice computes $k = g^{am}$, and believes that this is the symmetric key to Bob, and Bob computes $k = g^{bn}$ and believes that this is his symmetric communication key to Alice, but both have symmetric communications with Eve. Eve cand decrypt and recrypt the messages in order to let them continue this information exchange, but this communication has not anymore secrets for her. Also, Eve can modify the content of messages to her advantage.

It is clear now how important for security is the use of signatures for the authentication of messages. We touched this subject in the context of symmetric cryptography in the chapter about hash functions and MAC (message authentication code). The subject deserve to be resumed in the context of public key cryptography. There are some different possible schemes of authentication - here are the most simple:

Using appendix. The signature is an appendix to an encrypted message. The computation process works as follows:

$$\begin{aligned} \text{message} + \text{secret key of Alice} &= \text{signature}, \\ \text{message} + \text{signature} + \text{public key of Alice} &= \text{yes / no}. \end{aligned}$$

Signature with message recovering. If Alice wants to address to the public and wants to stress the autheticity of her message, she can apply a scheme with message recovering:

$$\begin{aligned} \text{message} + \text{secret key of Alice} &= \text{signature}, \\ \text{signature} + \text{public key of Alice} &= \text{yes / no} + \text{message}. \end{aligned}$$

We can use RSA as system of signature with message recovering. Let d be the secret key of Alice. Using the decryption procedure, Alice generates the signature $s = m^d \bmod N$. Every user can use the public key of Alice to verify the authenticity and to recover the message. They can compute $m = s^e \bmod N$. As you see, the secret and the public key change their roles. Now we use the secret key for the encryption and the public key for the decryption.

This idea can be enriched with hash functions, as follows:

We want to sign the message m using a public hash function h and a secret RSA key d. All other participants know the public key e. We compute the signature:

$$s = h(m)^d \bmod N,$$

and we publish the signed message (m, s). To verify the signature compute $h' = s^e \bmod N$ and $h = h(m)$. If $h = h'$ then the message is authentic, if $h \neq h'$, then it isn't.

31 DSA

Digital Signature Algorithm (DSA) is a signature scheme with appendix. The scheme produces a pair (r, s) of around 160 bits. It uses a hash function H, which is public. The component r depends of a temporary key k which changes for every new message. The component s depends of:

- message,

- the secret key of the sender,

- the component r, (so it depends on k as well).

Here are the steps of the protocol:

1. Choose a prime number q of around 160 bits and a prime number p that is between 512 and 2048 bits, such that $q|(p-1)$.

2. Randomly generate $h < p$ and compute:

$$g = h^{\frac{p-1}{q}} \bmod p,$$

   untill the result is a $g \neq 1$. This g has order q in the group $\mathbb{F}_p^\times$.

3. Once we have (p, q, g), every user generates its secret key x with $0 < x < q$. The associated public key is y:

$$y = g^x \bmod p.$$

   The secret of x is protected by the difficulty of the discrete logarithm.

4. To **sign** the message m:
   - Compute $h = H(m)$.
   - Choose a temporary key k, $0 < k < q$.
   - Compute $r = (g^k \bmod p) \bmod q$.
   - Compute $s = (h + xr)k^{-1} \bmod q$.
   - Send (r, s) as a signature of m.
5. To **verify** the signature of $[m, (r, s)]$:
   - Compute $h = H(m)$.
   - Compute $a = hs^{-1} \bmod q$.
   - Compute $b = rs^{-1} \bmod q$.
   - Compute $v = (g^a y^b \bmod p) \bmod q$, where y is the public key of the sender.
   - Accept the signature only if $v = r$.

Theorem: *Authentic messages are correctly recognized by DSA.*

Proof: Indeed the following equalities are true modulo p:

$$g^a y^b = g^{hs^{-1}} y^{rs^{-1}} = g^{hs^{-1}} g^{rxs^{-1}} = g^{(h+rx)s^{-1}} = g^k.$$

Then:

$$v = (g^a y^b \bmod p) \bmod q = (g^k \bmod p) \bmod q = r.$$

□

Example: Let $q = 13$, $p = 4q + 1 = 53$, $g = 16$. The signer chooses the secret key $x = 3$, so the public key is $y = 16^3 \bmod 53 = 256 \cdot 16 \bmod 53 = 15$. If $H(m) = 5$ and the temporary key is $k = 2$,

$$r = (g^k \bmod p) \bmod q = (16^2 \bmod 53) \bmod 13 = 44 \bmod 13 = 5,$$

$$s = (h + xr)k^{-1} \bmod q = (5 + 3 \cdot 5) \cdot 7 \bmod 13 = 10.$$

For the verification compute:

$$a = hs^{-1} \bmod q = 5 \cdot 4 \bmod 13 = 7,$$

$$b = rs^{-1} \bmod q = 5 \cdot 4 \bmod 13 = 7,$$

$$v = (g^a y^b \bmod p) \bmod q = (16^7 15^7 \bmod 53) \bmod 13 =$$

$$= (240^7 \bmod 53) \bmod 13 = (28^7 \bmod 53) \bmod 13 = 44 \bmod 13 = 5.$$

□

Example: Here is an example of EC-DSA, the DSA variant using elliptic curves. Consider the elliptic curve $y^2 = x^3 + x + 3$ over $\mathbb{F}_{199}$. The curve has $q = 197$ elements, so is a cyclic group because 197 is prime. One generator is the point $P = (1, 76)$.

Consider the secret key $x = 29$. The public key is the point $Y = [x]P = [29](1, 76) = (113, 191)$.

Suppose that the user wants to sign a message with the hash value $H(m) = 68$. He chooses the temporary key $k = 153$ and computes:

$$r = \pi_x([k]P) = \pi_x([153](1, 76)) = \pi_x(185, 35) = 185,$$

then computes:

$$s = (H(m) + xr)k^{-1} \bmod q = (68 + 29 \cdot 185)153^{-1} \bmod 197 = 78.$$

The signature is $(r, s) = (185, 78)$. Verification works as follows:

$$a = H(m)s^{-1} \bmod q = 68 \cdot 78^{-1} \bmod 197 = 112,$$

$$b = rs^{-1} \bmod q = 185 \cdot 78^{-1} \bmod 197 = 15,$$

$$Z = [a]P + [b]Y = [112](1, 76) + [15](113, 191) = (111, 60) + (122, 140) = (185, 35),$$

$$r = \pi_x(Z) = 185.$$

□

32 MQV

The key exchange protocol MQV (Menezes–Qu–Vanstone) is a generalization of Diffie-Hellman. Let G be a cyclic group generated by the public element g. This group might be a subgroup of an elliptic curve, as recommended by NIST. Alice and Bob have long-lasting pairs (public key, secret key), like:

$$(A = g^a, a) \;\; ; \;\; (B = g^b, b).$$

For a new key-exchange, they generate some temporary keys:

$$(C = g^c, c) \;\; ; \;\; (D = g^d, d).$$

Alice sends C to Bob, and Bob sends D to Alice. Let L be defined as:

$$L = \left\lceil \frac{[\log_2 |G|] + 1}{2} \right\rceil.$$

Alice knows A, B, C, D, a, c and computes:

$$\begin{aligned} s_A &= 2^L + (C \bmod 2^L), \\ t_A &= 2^L + (D \bmod 2^L), \\ h_A &= c + s_A a, \\ P_A &= (DB^{t_A})^{h_A}. \end{aligned}$$

Bob knows A, B, C, D, b, d and computes:

$$\begin{aligned} s_B &= 2^L + (D \bmod 2^L), \\ t_B &= 2^L + (C \bmod 2^L), \\ h_B &= d + s_B b, \\ P_B &= (CA^{t_B})^{h_B}. \end{aligned}$$

Theorem: *According to the definitions above, $P_A = P_B$ is the shared secret.*

Proof: We observe that by definition, $s_A = t_B = x$ and $s_B = t_A = y$. Below $\log_g$ is the discrete logarithm according to the generator g.

$$\log_g P_A = \log_g((DB^y)^{h_A}) = (d + yb)(c + xa).$$

$$\log_g P_B = \log_g((CA^x)^{h_b}) = (c + xa)(d + yb).$$

□

33 OAEP-RSA

OAEP-RSA (Optymized asymmetric encryption padding for RSA) is a mechanical protocol that contains RSA and has as a goal the CCA2 security.

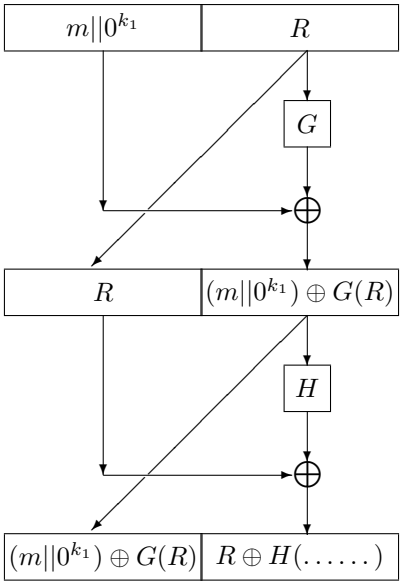
Let $f : \{0,1\}^k \to \{0,1\}^k$ be a bijective function (trap door permutation, one-way function), for example RSA given as $f(m) = m^e \bmod N$. Let k_0, k_1 be natural numbers such that 2^{k_0} and 2^{k_1} are too big to be treated by brute force, so $k_0, k_1 \geq 128$. Let $n = k - k_0 - k_1$. We encrypt blocks of length n. In particular $n + k_1 = k - k_0$.

We build two hash functions G and H as follows:

$$G : \{0,1\}^{k_0} \to \{0,1\}^{n+k_1},$$

$$H : \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}.$$

The protocol consists of two unsymmetric Feistel rounds. Here R is a binary word of length k_0.



The function f (respectively RSA) is applied on the result of the Feistel net. The encryption reads:

$$Enc(m) = f([(m||0^{k_1}) \oplus G(R))]||[R \oplus H((m||0^{k_1}) \oplus G(R))]).$$

Decryption works as follows. Apply f^{-1} over the cypher c, and get:

$$f^{-1}(c) = T||(R \oplus H(T)),$$

where $T = (m||0^{k_1}) \oplus G(R)$. Compute $H(T)$ and deduce $R = (R \oplus H(T)) \oplus H(T)$. Now we can compute $G(R)$ and deduce m from $m||0^{k_1} = [(m||0^{k_1}) \oplus G(R)] \oplus G(R)$. □

Theorem: *OAEP-RSA is semantically secure against CCA2 attacks.*

Part VI
Advanced protocols

34 INS secret sharing

Secret sharing means to distribute some cryptographic elements to some agents (users, computers, organizations) with the goal that the shared secret can be decrypted only together and only in a collaborative way - meaning that all agents agree with the decryption.

Example 1: Let s be the secret password enabling the use of a nuclear weapon. Suppose that the secret s is shared by 4 agents with different power of decision. Parts of the secret can be possessed by the President P, the Vicepresident V, the Secretary of Defence S and the General G. We want that the following minimal sets are necessary present and agree, in order to use the nuclear weapon:

$$\{P, G\} \ ; \ \{V, S, G\},$$

but no other minimal subset. Let s_P, s_V, s_S and s_G the partial secret passwords possessed by every agent. □

By **acces structure** we mean a collection $\Gamma \subset \mathcal{P}(P)$ such that $P \in \Gamma$ and for every $A \in \Gamma$, if $A \subset B$, then $B \in \Gamma$. **A sharing secret scheme** consists of two algorithms called **Share** and **Recombine**:

Share(s, Γ) sends to every agent X his own secret s_X.

Recombine considers $H = \{s_X \mid X \in \mathcal{O}\}$. If $\mathcal{O} \in \Gamma$ then it computes s, otherwise produces an error message.

In the first example,

$$\Gamma = \{PG, VSG, PGV, PGS, PVGS\},$$

while PG and VSG are the minimal sets.

Example 2: Four directors or associates A, B, C and D have equal rights. They establish together that at least two of them must be present in order to open the safe with contracts. In this case:

$$\Gamma = \{AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD\},$$

and $\{AB, AC, AD, BC, BD, CD\}$ are the minimal sets.

Scheme Ito-Niszizeki-Saito. Let $m(\Gamma)$ be the set of minimal subsets. For every $\mathcal{O} \in m(\Gamma)$, if $k = |\mathcal{O}|$, we generate the secrets $s_1, \ldots, s_k$ such that:

$$s_1 \oplus s_2 \oplus \cdots \oplus s_k = s.$$

We can generate $k-1$ such secrets randomly, and s_k is computed such that:

$$s_k = s_1 \oplus \cdots \oplus s_{k-1} \oplus s.$$

Example 1: We make the decompositions:

$$s = s_1 \oplus s_2 = s_3 \oplus s_4 \oplus s_5.$$

The parts get the following shares:

$$\begin{aligned} s_P &= s_1, \\ s_V &= s_3, \\ s_S &= s_4, \\ s_G &= s_2 || s_5. \end{aligned}$$

The algorithm Recombine must first find if a minimal access set is available, and then must compute and check the secret using the found access set.

Example 2: In this case one computes 6 decompositions:

$$s = s_{AB} \oplus t_{AB} = s_{AC} \oplus t_{AC} = s_{AD} \oplus t_{AD} = s_{BC} \oplus t_{BC} = s_{BD} \oplus t_{BD} = s_{CD} \oplus t_{CD},$$

and the shares are:

$$\begin{aligned} s_A &= s_{AB}||s_{AC}||s_{AD}, \\ s_B &= t_{AB}||s_{BC}||s_{BD}, \\ s_C &= t_{AC}||t_{BC}||s_{CD}, \\ s_D &= t_{AD}||t_{BD}||t_{CD}. \end{aligned}$$

35 RSS secret sharing

The RSS (Replicated Secret Sharing) scheme works as follows:

1. Let A_1, A_2, ..., A_t maximal sets which are *not* in Γ. This implies that for every set B, if $B \neq A_i$ and $A_i \subset B$ then $B \in \Gamma$.
2. Let B_1, ..., B_t the complements of A_1, ..., A_t.
3. One chooses a secret s and decomposes s as binary sum of t words:
$$s = s_1 \oplus s_2 \oplus \cdots \oplus s_t.$$
4. Person P gets s_i if and only if $P \in B_i$.

Example 1: In this case the maximal sets not in Γ are:

$$A_1 = \{P, V, S\} \quad , \quad A_2 = \{V, G\} \quad , \quad A_3 = \{S, G\}.$$

Their complements are:

$$B_1 = \{G\} \quad , \quad B_2 = \{S, P\} \quad , \quad B_3 = \{P, V\}.$$

The decomposition will be:

$$s = s_1 \oplus s_2 \oplus s_3,$$

and the distribution will be:

$$\begin{aligned} s_P &= s_2||s_3, \\ s_V &= s_3, \\ s_S &= s_2, \\ s_G &= s_1. \end{aligned}$$

Example 2: In this case the maximal sets not in Γ are:

$$A_1 = \{A\} \quad , \quad A_2 = \{B\} \quad , \quad A_3 = \{C\} \quad , \quad A_4 = \{D\},$$

and the complements are:

$$B_1 = \{B, C, D\} \quad , \quad B_2 = \{A, C, D\} \quad , \quad B_3 = \{A, B, D\} \quad , \quad B_4 = \{A, B, C\}.$$

The decomposition is:

$$s = s_1 \oplus s_2 \oplus s_3 \oplus s_4,$$

and the distribution:

$$\begin{aligned} s_A &= s_2||s_3||s_4, \\ s_B &= s_1||s_3||s_4, \\ s_C &= s_1||s_2||s_4, \\ s_D &= s_1||s_2||s_3. \end{aligned}$$

36 Shamir secret sharing

The methods presented in the last two sections are quite inefficient, specially when it comes to recombine the partial secrets. Shamir's method below is an algebraic scheme and is very efficient. Its disadvantage is that it works only for an extreme case. However, this extreme case is one for which the other two schemes prove to be very inefficient: *every subset of t persons of n cannot rebuild the secret, but every subset of $t+1$ persons of n can rebuild the secret and gets access.*

Here is Shamir's Method:

1. Choose a field $\mathbb{F}_q$ with $q > n$.
2. The shared secret is an element $s \in \mathbb{F}_q$ randomly chosen. One chooses randomly other t elements, not necessarily different, $f_1, \ldots, f_t \in \mathbb{F}_q$, and considers the polynomial:
$$f(X) = s + f_1 X + \cdots + f_t X^t \in \mathbb{F}_q[X].$$
3. Every person gets a unique label $x_i \in \mathbb{F}_q$. The person i gets the pair $s_i = (x_i, f(x_i))$.

Theorem: *Given the construction above, every subset of $t+1$ persons can rebuild the secret $s = f(0)$, while every subset of t persons cannot rebuild it.*

Proof 1: Take $t+1$ persons. Everyone contributes with an equation, more precisely the person with label x_i brings the equation:
$$s + f_1 x_i + f_2 x_i^2 + \cdots + f_t x_i^t = f(x_i).$$
These $t+1$ equations build together a system of linear equations with $t+1$ unknowns. The determinant of the matrix:
$$(x_i^j)_{i=1,\ldots,t+1}^{j=0,\ldots,t},$$
is the so called Vandermonde determinant, and equals:
$$\prod_{i>k}(x_i - x_k) \neq 0.$$
Henceforth the system is soluble, and allows the unique solution $(s, f_1, \ldots, f_t)$. The value s is the secret looked for. If the number of equations is smaller then $t+1$, the solution is no more unique, and we cannot find the secret s. □

Proof 2: This problem is also an interpolation problem. We must find a polynomial of degree t whose graph contains the points $(x_i, f(x_i))$. Those points are different, as they have different abscissa.
$$f(X) = \sum_{i=0}^{t} f(x_i) \prod_{k=0, k\neq i}^{t} \frac{X - x_k}{x_i - x_k} = \sum_{i=0}^{t} f(x_i)\delta_i(X).$$
In conclusion, the secret s is got directly as:
$$s = f(0) = \sum_{i=0}^{t} f(x_i)\delta_i(0).$$
□

37 Commitments

To make a commitment means in this context to make an encrypted bet, that cannot be either changed nor otherweise interpreted after the announcement of the final result. For example:

Bob and Alice play *stone-paper-scissors* by phone. They use a public hash function H. A round looks as follows. Let R_A be a random string produced in the moment of the transmission.

$$\begin{array}{ll} A \to B & h_A = H(R_A||PAPER), \\ B \to A & SCISSORS, \\ A \to B & R_A||PAPER. \end{array}$$

Using the hash function, Bob can verify that Allice played fair. If Alice wants to cheat, she should find a string R'_A such that:

$$H(R_A||PAPER) = H(R'_A||STONE).$$

But this is very difficult if the function H is resistant to the second pre-image.

Definition: A commitment scheme is called **binding** if no opponent can win the following game (in polynomial time):

- The opponent produces c, x, r such that $c = C(x, r)$.

- The opponent produces $x' \neq x$ and r' such that $C(x, r) = C(x', r')$.

Definition: A commitment scheme is called **concealing** (secure) if no opponent can win the following game with probability bigger than $1/2$:

- The opponent produces words $x_0 \neq x_1$ with $|x_0| = |x_1|$.

- The authority generates r and a $b \in \{0, 1\}$ randomly.

- The authority computes $c = C(x_b, r)$ and sends it to the opponent.

- The opponent guesses b.

Theorem: *No commitment scheme can be absolutely binding and absolutely concealing.*

Proof: If the scheme is absolutely binding, the mapping from commitments to commited values would be injective. It follows that the scheme must be determinist, so it cannot be absolutely concealing. □

The scheme $H(R||m)$ introduced above is computationally binding and information-theoretically concealing.

There are some commitment schemes based on the difficulty of the discrete logarithm. Let G be a cyclic group of prime order q, $G = \langle g \rangle$ with $g \in \mathbb{F}_p^\times$, and let $h \in G$ be such that $\log_g h$ is not known by the user making a commitment. The engaged value is some $x \bmod q$.

The exponential commitment scheme is $B(x) = g^x$.

Pedersen's commitment scheme is $B_a(x) = h^x g^a$, where a is randomly chosen.

Elgamal's commitment scheme is $E_a(x) = (g^a, xh^a)$.

Observation: $B(x)$ is information-theoretically binding, but injective.

Observation: $B_a(x)$ is information-theoretically binding and computationally concealing.

Observation: $E_a(x)$ is information-theoretically binding and computationally concealing.

Observation: The schemes B and B_a have homomorphic properties:

$$B(x_1)B(x_2) = B(x_1 + x_2),$$

$$B_{a_1}(x_1)B_{a_2}(x_2) = B_{a_1+a_2}(x_1 + x_2).$$

This is an advantage in some context, and it will be exploited in the design of an electronic vote system.

38 Oblivious transfer

Like the encrypted commitments, the oblivious transfer is used to make a connection between two parts that do not trust each other. The protocol must respect the following conditions:

A Sender sends two messages m_0 and m_1 over the system. The Receiver communicates to the system a randomly chosen $b \in \{0,1\}$, and gets the message m_b. The Receiver should never guess m_{1-b}, while the Sender should never guess b. In this way the two parts have together a relation of collaboration, despite their mutual lack of trust.

First let us recapitulate the Elgamal cryptosystem with a hash function. Let $G = \langle g \rangle$ be a cyclic group of order q, consider a pair (public key, secret key) $= (h = g^x, x)$, and $H : G \to \{0,1\}^n$ a hash function, where $n = \log q$. For messages m of length n, Bob chooses a temporary key k and computes the pair of messages:

$$(c_1, c_2) = (g^k, m \oplus H(h^k)).$$

For the decryption, Alice computes:

$$c_2 \oplus H(c_1^x) = m \oplus H(h^k) \oplus H(g^{kx}) = m \oplus H(h^k) \oplus H(h^k) = m.$$

The oblivious transfer works as follows: The Receiver has two public keys, but knows only one of the corresponding secret keys. The Sender encrypts the messages, using for every message one of the public keys, and sends them. The Receiver can decrypt a message, and the Sender does not know which of them has been decrypted.

Concretely, here are the steps of this method:

- The Sender chooses a random $c \in G$ and sends it to the Receiver. Nobody knows the discrete logarithm $\log_g c$.

- The Receiver chooses randomly a bit $b \in \{0,1\}$. He also randomly chooses a secret key $x \in \mathbb{Z}_q$. He computes two public keys:

$$h_b = g^x,$$

$$h_{1-b} = c/h_b.$$

One public key is the good one for the Receiver, the other one is the bad one for the Receiver. The Sender does not know which is good and which is bad. The Receiver sends h_0, which might be good or bad, to the Sender.

- The Sender computes:

$$h_1 = c/h_0,$$

The Sender moreover randomly chooses a temporary key $k \in \mathbb{Z}_q$, computes

$$c_1 = g^k,$$

$$e_0 = m_0 \oplus H(h_0^k),$$

$$e_1 = m_1 \oplus H(h_1^k),$$

and sends c_1, e_0 and e_1 to the Receiver.

- The Receiver computes:

$$e_b \oplus H(c_1^x) = e_b \oplus H(g^{kx}) = m_b \oplus H(g^{kx}) \oplus H(g^{kx}) = m_b.$$

The Receiver cannot compute m_{1-b} because he has not the key. On his turn, the Sender does not know which message has been decrypted.

39 Zero Knowledge Proofs

A zero-knowledge protocol is a method to share a secret between two persons: a prover called Peggy and a verifier called Victor. Peggy wants to convince Victor that she knows a certain information, without making Victor to know it as well. She is allowed to communicate only some very small parts of her secret. Victor must verify in a convincing way that Peggy knows the secret, but without becoming himself aware of the secret, neither during the protocol nor afterwards. The name *zero-knowledge* comes from this ignorance of the verifier, which must be guaranteed by the method.

The graph-isomorphism problem leads to the first example of zero-knowledge proof. A graph $G = (V, E)$ is a set of vertexes V and a set of edges $E \subset V \times V$. Consider the graphs $G_i = (V_i, E_i)$, $i = 0, 1$. A function $\varphi : V_0 \to V_1$ is called a *graph-isomorphism* if and only if:

1. φ is a bijection.
2. $xy \in E_0$ if and only if $\varphi(x)\varphi(y) \in E_1$.

In the classic example of zero-knowledge proof we have the following situation:

- The isomorphic graphs $G_0 \simeq G_1$ are public, but not their isomorphism. The graphs are big, they have at least 10^4 vertexes.

- Peggy knows a secret isomorphism $\varphi : G_0 \to G_1$. She wants to convince Victor that she knows an isomorphism, without giving him the isomorphism.

- Peggy chooses randomly $i \in \{0, 1\}$ and applies a secret random permutation ψ on G_i. This way she produces a third graph H. So Peggy knows now the following isomorphisms:

$$\begin{aligned} \varphi : &\quad G_0 &&\longrightarrow G_1, \\ \psi : &\quad G_i &&\longrightarrow H, \\ \alpha : &\quad G_{1-i} &&\longrightarrow H. \end{aligned}$$

In the case that $i = 1$, $\alpha = \psi \circ \varphi$, else if $i = 0$, $\alpha = \psi \circ \varphi^{-1}$. The isomorphisms are easy to evaluate by binary search, because they are given as lists or dictionaries. Peggy publishes H.

- Victor states a challenge: chooses $b \in \{0, 1\}$ and asks about the isomorphism of the graphs G_b and H. Peggy sends ψ or α.

If Peggy would not know φ, she could correctly answer only with probability $1/2$. By repeating the procedure n times, the probability to successfully cheat becomes $(1/2)^n$. □

Schnorr's identification protocol. Consider a cyclic group $G = \langle g \rangle$ of order q and a public element $y \in G$. Peggy's secret is the discrete logarithm $x = \log_g y$. The protocol has the following steps:

1. Peggy randomly chooses k and sends Victor $r = g^k$.
2. Victor sends Peggy a challenge e.
3. Peggy sends Victor $s = k + xe$. She can do so as she knows both x and k.
4. Victor computes $g^s y^{-e}$ and verifies if he has got r.

Theorem: *If Peggy knows $x = \log_g y$, the protocol does work. Victor finds back r without knowing x. If Peggy tries to cheat, she can successfully cheat only with a probability of $1/q$.*

Proof: Indeed,

$$g^s y^{-e} = g^{k+xe-xe} = g^k = r.$$

□

Observation: One must avoid to choose two times the same temporary key k, because in this case Victor could find x. From:

$$r = g^s y^{-e} = g^{s'} y^{-e'},$$

it results:

$$s + x(-e) = s' + x(-e') \bmod q,$$
$$x = (s - s')(e - e')^{-1} \bmod q.$$

□

Zero-knowledge using Pedersen commitments. Let $G = \langle g \rangle$ be a cyclic group of prime order q and $h \in G$ with unknown $\log_g h$. For committing on a value x, one randomly chooses a value a and publishes the commitment:

$$B_a(x) = h^x g^a.$$

An important case is $x \in \{-1, 1\}$. The person doing such a commitment must prove that

$$x \in \{-1, 1\}$$

without telling the value of x. The protocol works as follows:

1. Peggy randomly chooses d, r, $w \bmod q$.
2. Peggy publishes $B_a(x)$, α_1, α_2, where:

$$\alpha_1 = \begin{cases} g^r (B_a(x) h^{+1})^{-d}, & x = 1, \\ g^w, & x = -1. \end{cases}$$

$$\alpha_2 = \begin{cases} g^w, & x = 1, \\ g^r (B_a(x) h^{-1})^{-d}, & x = -1. \end{cases}$$

3. Victor sends a random challenge c to Peggy.
4. Peggy computes:

$$\begin{aligned} d' &= c - d, \\ r' &= w + a d'. \end{aligned}$$

   and sends Victor:

$$(d_1, d_2, r_1, r_2) = \begin{cases} (d, d', r, r'), & x = 1, \\ (d', d, r', r), & x = -1. \end{cases}$$

5. Victor verifies the following equalities:

$$\begin{aligned} c &= d_1 + d_2, \\ g^{r_1} &= \alpha_1 (B_a(x) h^{+1})^{d_1}, \\ g^{r_2} &= \alpha_2 (B_a(x) h^{-1})^{d_2}. \end{aligned}$$

6. Victor accepts Peggy's commitment only if all three equalities are verified.

Theorem: *If Peggy commits for a value $x \in \{-1, 1\}$, the protocol runs successfully, and Victor does not become aware of the committed value.*

Proof: The equality $c = d_1 + d_2$ is trivially true.

If $x = 1$, then:

$$\alpha_1 (B_a(x) h^{+1})^{d_1} = g^r (B_a(x) h^{+1})^{-d} (B_a(x) h^{+1})^d = g^r = g^{r_1},$$

$$\alpha_2(B_a(x)h^{-1})^{d_2} = g^w(B_a(x)h^{-1})^{d'} = g^{w+ad'} = g^{r'} = g^{r_2}.$$

If $x = -1$ then:

$$\alpha_1(B_a(x)h^{+1})^{d_1} = g^w(B_a(x)h^{+1})^{d'} = g^{w+ad'} = g^{r'} = g^{r_1},$$
$$\alpha_2(B_a(x)h^{-1})^{d_2} = g^r(B_a(x)h^{-1})^{-d}(B_a(x)h^{+1})^d = g^r = g^{r_2}.$$

□

40 Electronic vote

We present here an electronic vote system for a referendum. The vote expressed by an elector is a secret commitment for a value $v \in \{-1, 1\}$. We suppose that there are m electors indexed with j. The fairness of the process and the result are observed in n independent counting centers, indexed with i. These counting centers monitorise the whole process, without any teritorial dependence.

Some features to implement are the followings:

- Only authorized electors may vote.

- Every authorized elector votes at most once.

- Nobody knows the value voted by a given elector, excepting the elector himself.

- No vote can be duplicated.

- The result is corrected computed.

- Every participant - elector or counting center - can verify the correctness of the process.

- The protocol works even if someone tries to cheat.

Setup. Let $G = \langle g \rangle$ be a cyclic group of prime order q, $h \in G$, $h = g^x$, and nobody knows x. Every counting center i has a public key E_i. Every elector j has a public key k_j for his signature. A public hash function H is given.

Voting. Every elector j:

1. Chooses his vote option $v_j \in \{-1, 1\}$.
2. Randomly chooses an $a_j \in \mathbb{Z}_q$.
3. Publishes his encrypted vote as a Pedersen commitment, $B_j = B_{a_j}(v_j)$ and (α_1, α_2).
4. Publishes a certificate $c_j = H(\alpha_1||\alpha_2||B_{a_j}(v_j))$ that can be used as an automatic challenge, to verify that $v_j \in \{-1, 1\}$ without giving the value of v_j.
5. The pair (B_j, c_j) is signed with the public signature algorithm using the signature key k_j.

Distribution to counting centers. Use Shamir's secret sharing to distribute the values a_j and v_j to the independent counting centers.

1. Every elector j chooses randomly following polynomials modulo q. The degree t of these polynomials satisfies $t < n$, where n is the number of the counting centers.

$$\begin{aligned} R_j(x) &= v_j + r_{1,j}x + \cdots + r_{t,j}x^t, \\ S_j(x) &= a_j + s_{1,j}x + \cdots + s_{t,j}x^t. \end{aligned}$$

2. For every center i, $1 \leq i \leq n$, the elector j computes the pair:

$$(u_{i,j}, w_{i,j}) = (R_j(i), S_j(i)) \bmod q.$$

3. Elector j encrypts the pair $(u_{i,j}, w_{i,j})$ using the public key E_i of the counting center, and sends the resulting message to i.

4. Elector j publishes his commitments relatively to the polynomials R_j and S_j as Pedersen commitments:
$$B_{l,j} = B_{s_{l,j}}(r_{l,j})$$
for all l, $1 \leq l \leq t$.

Consistency check. Every counting center i verifies that the pair $(u_{i,j}, w_{i,j})$ received from the elector j is consistent with the conditions so far. We apply the homomorphic properties of the Pedersen commitments. One computes and verifies:

$$B_j \prod_{l=1}^{t} {B_{l,j}}^{i^l} = B_{a_j}(v_j) \prod_{l=1}^{t} B_{s_{l,j}}(r_{l,j})^{i^l} = h^{v_j} g^{a_j} \prod_{l=1}^{t} \left(h^{r_{l,j}} g^{s_{l,j}}\right)^{i^l} =$$

$$= h^{\left(v_j + \sum_{l=1}^{t} r_{l,j} i^l\right)} g^{\left(a_j + \sum_{l=1}^{t} s_{l,j} i^l\right)} = h^{u_{i,j}} g^{w_{i,j}}.$$

Vote counting. Everyone of the n public counting centers publics the sum of the votes:

$$T_i = \sum_{j=1}^{m} u_{i,j},$$

and the sum of the encryptions of the randomly chosen values:

$$A_i = \sum_{j=1}^{m} w_{i,j}.$$

Every counting center and every elector can check again the consistence of these sums using the properties of the Pedersen commitment:

$$\prod_{j=1}^{m} \left(B_j \prod_{l=1}^{t} {B_{l,j}}^{j^l}\right) = \prod_{j=1}^{m} h^{u_{i,j}} g^{w_{i,j}} = h^{T_i} g^{A_i}.$$

Every elector and every center can compute the result of the referendum by considering $t+1$ values T_i and solving a system of linear equations or an interpolation problem:

$$T_i = \sum_{j=1}^{m} u_{i,j} = \sum_{j=1}^{m} R_j(i) =$$

$$= \left(\sum_{j=1}^{m} v_j\right) + \left(\sum_{j=1}^{m} r_{1,j}\right) i + \cdots + \left(\sum_{j=1}^{m} r_{t,j}\right) i^t$$

The system consists of $t+1$ equations:

$$V + W_1 i + \cdots + W_t i^t = T_i.$$

After solving the system, we get $V = \sum_{j=1}^{m} v_j$, and we deduce how many $+1$ and how many -1 have been opted for, because the number of participants is known.

□

41 The millionaire problem

Alice, Bob and Charles want to find out who is earning more money, but keeping secret the amount of money everybody actually earns. This problem can be solved as follows:

Round I. Alice chooses a random number r, which is very big and contains only randomly chosen digits. She adds her income a to this number. She gives Bob this number $r + a$, and Bob adds his income b. Bob gives to Charles $r + a + b$, Charles adds his income c, and sends the total sum $r + a + b + c$ back to Alice. Alice computes and publishes the mean of the incomes:

$$m = \frac{a + b + c}{3}.$$

Everyone with an income $\leq m$ announces this now. If all of them make this announcement, then they had equal incomes from the beginning, and the protocol is closed. If two of them make this announcement, then the third one has the biggest income, and the protocol is closed again. If only one of them makes this announcement, then the other two must continue the decision process as in Round II.

By the way, one can start with an arbitrary number of participants. In this case they repeat Round I untill the problem is solved, or there are exactly two left. The two finalists proceed as in Round II.

Round II. Now Alice and Bob want to compare their income without communicating it. They find an appropriated unit such that their incomes can be rounded expressed as numbers a and b with $0 \leq a, b \leq 100$. Let B be Bob's public key and B' his secret key. Alice chooses some arbitrary x and computes $c = Enc_B(x)$. Alice sends $d = c - a$ to Bob.

Bob computes $y_i = Dec_{B'}(d + i)$ with $i = 0, 1, \ldots, 101$.

Bob computes $z_i = f(y_i)$ with $i = 0, 1, \ldots, 101$. Here f is a hash-function.

If there are $i > j$ with $|z_i - z_j| \leq 1$, we repeat the protocol. It is not difficult to find a good choice of x, such that this does not happen.

Bob sends to Alice the finite sequence:

$$z_0, z_1, \ldots, z_b, z_{b+1} + 1, z_{b+2} + 1 \ldots, z_{101} + 1,$$

permuted randomly.

Alice computes $f(x)$ and $f(x) + 1$ and looks for them in the sequence. If she finds $f(x)$, Bob earns more than Alice. If she finds $f(x) + 1$, Alice earns more than Bob.

42 The secret of the agency

Look an hypothetical situation. From the head-quarters SRI Bucharest n 3-agent teams leave to the head-quarters SRI in Cluj. They bring to Cluj a highly classified information. Only one agent of the $3n$ knows the secret information. Every agent got the order to act with his 2 team mates as if he was the one who has the secret. In general, every one knows that he does not know the secret, but suppose that one of his two mates knows the secret. They protect their team mates accordingly.

SRI Cluj wants to find out quickly which 3-team contains the agent who really knows the secret. All teams are invited to sit down and every team gets a round table. While they are drinking their well-come coffee, every agent establishes a secret bit with everyone of his team-mates, who sits right and left. So, at the table of the team composed by the agents A, B and C the secret bits b_{AB}, b_{BC} and b_{AC} are defined by the respective pairs of agents, and the bit b_{AB} is known only by this pair, and so on. Every agent writes on his napkin the boolean sum of the bits he knows:

$$A: \quad b_{AB} \oplus b_{AC} = x,$$

$$B: \quad b_{AB} \oplus b_{BC} = y,$$
$$C: \quad b_{AC} \oplus b_{BC} = z.$$

The servant, while collecting the napkins, computes:

$$x \oplus y \oplus z = 0,$$

and knows that the secret is not known at this table. The unique agent who knows the secret computes:

$$A: \quad b_{AB} \oplus b_{AC} \oplus 1.$$

At his table the sum will be:

$$x \oplus y \oplus z = 1,$$

and the servant knows that one of these three agents has the secret. □

43 Secure circuit evaluation I

Alice and Bob want to correctly evaluate a function $f(a, b)$ for an input a given by Alice and an input b given by Bob. However, the value a must remain secret for Bob and the input b must remain secret for Alice. Despite this condition, they must trust the computation and the result.

As every input can be transformed in a pair of binary (boolean) strings, they use an encrypted Boolean circuit, following the method called Yao's garbled circuits. The original circuit computing $f(x, y)$ is known at the beginning by both parts.

The general procedure works according to the following steps:

- Alice encrypts the circuit and her input a.

- Bob encrypts his input b helped by Alice using *oblivious transfer*.

- Bob evaluates the circuit and gets an encrypted result.

- Alice and Bob decrypts the result together.

Encryption of the circuit. Suppose that Alice wants to encrypt a conjunctive gate:

$\cdot$	0	1
0	0	0
1	0	1

Suppose that the gate has two input wires w_a and w_b and an outup wire w_c. Alice substitutes the values 0 and 1 for w_a with randomly chosen different binary words X_a^0 and X_a^1. She does the same for the wires w_b and w_c. The table of this gate looks now as follows:

$\cdot$	X_a^0	X_a^1
X_b^0	X_c^0	X_c^0
X_b^1	X_c^0	X_c^1

The table is now encrypted in a group of 4 cyphers:

$$Enc_{X_a^0||X_b^0}(AX_c^0),$$
$$Enc_{X_a^1||X_b^0}(AX_c^0),$$
$$Enc_{X_a^0||X_b^1}(AX_c^0),$$
$$Enc_{X_a^1||X_b^1}(AX_c^1).$$

Here A is a mark meaning correct decryption, for example the text *The correct value is* . When she chooses the random cyphers X_c^i, Alice must verify that only:

$$Dec_{X_a^0||X_b^0}(Enc_{X_a^0||X_b^0}(AX_c^0)) = AW,$$

while the other possible keys $X_a^1||X_b^0$, $X_a^0||X_b^1$ and $X_a^1||X_b^1$ do not lead to a decryption with this shape. As this is possible with a big probability, it usually works from the first try.

Now the 4 cyphers (and other similar cyphers coming from other gates) are randomly permuted, such that their position in the list does not give any information about their meaning.

Data transfer. Alice sends Bob the encrypted circuit and the encryption of her input a. For example, if $a = a_4a_3a_2a_1a_0 = 11100$ then Alice sends to Bob $X_{a_4}^1$, $X_{a_3}^1$, $X_{a_2}^1$, $X_{a_1}^0$ and $X_{a_0}^0$. Bob needs also the encryption of his own input b in a way such that Alice cannot guess its value. This can be done by oblivious transfer. If Bob's input is $b = b_4b_3b_2b_1b_0 = 00101$, Bob initially needs $b_0 = 1$ and needs the correct one cypher from the set $\{X_{b_0}^0, X_{b_0}^1\}$. By oblivious transfer of type *one of two*, he gets $X_{b_0}^1$ but can never decrypt $X_{b_0}^0$, while Alice will never know which of the two strings was the string, Bob really needed.

Evaluation. Now Bob goes through the encoded circuit and can decode just one line

$$AX_c = Dec_{X_a||X_b}(Enc_{X_a||X_b}(AX_c)),$$

from the only one line which produces an output AW. Bob goes on with the evaluation, until he finds the output bits.

End of the protocol. For every output bit x, Alice translates the significance of X_x^0 and X_x^1, while Bob tells which of them was really computed by the output gate. So both will find out the result. □

44 Secure circuit evaluation II

Another problem of this type is the following: Alice has a method to optimise the tax declaration. As she has this advantage, she does not want to make it public. Bob is the Mafia, and wants to optimise his tax declaration. Alice does not want to give Bob the method, because the method is her profesional secret. But Bob does not want to give Alice the datas about his income, because this data is Bob's profesional secret.

So Alice knows the algorithm to compute a function f, while Bob has a value x and wants to compute $f(x)$. But Alice does not want to reveal Bob anything from the algorithm, while Bob does not want to reveal Alice anything from the data. Any computation can be presented such that $x \in \{0,1\}^n$ is a boolean input while f is a boolean circuit. We may suppose that the circuit consists of the gates $\wedge$ and $\neg$ only, because $x \vee y = \neg(\neg x \wedge \neg y)$.

Bit encryption. The encryption of the bits can be realized with Goldwasser-Micali. Bob chooses prime numbers $p = 3 \bmod 4$ and $q = 3 \bmod 4$, $p < q$. His secret key is the pair (p, q) while the public key is $N = pq$. For every bit b he chooses a nonce k and computes:

$$Enc(b) = (-1)^b k^2 \bmod N.$$

Bob sends Alice the tuple $Enc(x) = Enc(b_1) \dots Enc(b_n)$.

Negation. Alice chooses a random number r and computes:

$$Enc(\neg b) = (-1)r^2 Enc(b) \bmod N.$$

Conjunction. Alice needs the help Bob in order to compute a conjunction. But she does not want that Bob knows which are the bits whose conjunction has to be computed. So she chooses random bits $c, c' \in \{0,1\}$ and random numbers $r, r' \in \mathbb{N}$. Then she computes the values:

$$Enc(d) = (-1)^c r^2 Enc(b) \bmod N,$$

$$Enc(d') = (-1)^{c'} r'^2 Enc(b) \bmod N,$$

and sends them to Bob. We see that d and d' are the following bits:

$$d = \begin{cases} b, & c = 0, \\ \neg b, & c = 1. \end{cases} \qquad d' = \begin{cases} b', & c' = 0, \\ \neg b', & c' = 1. \end{cases}$$

Bob knows the factorisation of N, so he can find d and d'. Bob computes now the following values and sends them to Alice in the given order: $Enc(d \wedge d')$, $Enc(d \wedge \neg d')$, $Enc(\neg d \wedge d')$, $Enc(\neg d \wedge \neg d')$. They correspond to the combinations (c, c') being $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. Alice knows what combination (c, c') she had chosen, and decides which of the four encrypted numbers has to be used.

Final evaluation. The output values $a_1, \ldots a_m \in \mathbb{Z}_N$ are send to Bob, he decrypts them with Goldwasser-Micali and finds out the bits $Dec(a_1), \ldots, Dec(a_m) \in \{0, 1\}^m$.

45 Secure circuit evaluation III

In the last chapter we presented the encryption of boolean circuits for the two-side encrypted computation. Now we present the encryption of arithmetic circuits for the many-side encrypted computation. This protocol is based on Shamir's Secret Sharing.

Suppose that $n \geq 3$ users $A_1, \ldots, A_n$ want to compute together an arithmetic function

$$f(x_1, \ldots, x_n),$$

where every x_i is the value introduced by A_i. This value must remain secret for the other participants at the computation process, but all of them must trust the computation process and the result, because finding the result is their common interest. For example, if $n = 6$, the function:

$$f(x_1, \ldots, x_6) = x_1 x_2 + x_3 x_4 + x_5 x_6$$

can be computed by a circuit with 3 multiplicative gates and 2 additive gates.

As in practical applications the domain of interest is limited, one can choose a very big prime number p, of order, say 2^{1024}, and consider the inputs $x_i = x_i \bmod p$, while finally $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \bmod p$. In this case we substitute the additive and multiplicative gates with $+ \bmod p$, respectively $- \bmod p$, and $\times \bmod p$.

Value sharing. Every participant A_i chooses randomly $f_1, \ldots, f_t \in \mathbb{F}_p$ and constructs the sharing polynomial:

$$h_i(x) = x_i + f_1 x + \cdots + f_t x^t.$$

The user A_j gets from the user A_i a representant of its value x_i, which is:

$$x_i^{(j)} = h_i(j).$$

This happens also for $i = j$. Every user will evaluate the circuit as follows below, using only the values got from the other users. The user j uses the values:

$$x_1^{(j)}, x_2^{(j)}, \ldots, x_n^{(j)}.$$

Additive gates: At additive gates, one just adds modulo p. Indeed, suppose that two shared values $a^{(i)}$ and $b^{(i)}$ reach an additive gate, and they have been shared using polynomials:

$$f(x) = a + f_1 x + \cdots + f_t x^t,$$

$$g(x) = b + g_1 x + \cdots + g_t x^t.$$

If $a + b = c \bmod p$ and $f + g = h$ in $\mathbb{Z}_p[x]$, we observe that:

$$c^{(i)} = h(i) = (f(i) + g(i)) = a^{(i)} + b^{(i)},$$

so every user must add the shared value and gets a value corresponding to the sum. The same happens also for the difference.

Multiplicative gates: First we recall a property of the Lagrange interpolation over fields. If $f(x)$ is a polynomial and we share the values $f(j)$, then there is a vector, $(r_1, \ldots, r_n)$, such that:

$$f(0) = \sum_{i=1}^{n} r_i f(i),$$

and this vector works for all polynomials of degree $\leq n - 1$. This vector is called *recombination vector*.

Suppose that every participant has a shared value for a and b over $a^{(i)} = f(i)$ and $b^{(i)} = g(i)$, where $f(0) = a$ and $g(0) = b$. We want to compute a shared value $c^{(i)} = h(i)$ such that for a certain polynomial $h(x)$, $h(0) = c = ab$. The principle is the following:

1. Every user locally computes $d^{(i)} = a^{(i)} b^{(i)}$.
2. Every user produces a polynomial $\delta_i(x)$ of degree at most t such that $\delta_i(0) = d^{(i)}$.
3. Every user i distributes to every user j (including to himself) the value $d_{i,j} = \delta_i(j)$.
4. Every user i computes now:
$$c^{(i)} = \sum_{j=1}^{n} r_j d_{i,j}.$$

Step (1) computes a polynomial $h'(x)$ of degree $\leq 2t$, with $d^{(i)} = h'(i)$ and $c = h'(0)$. If:

$$2t \leq n - 1,$$

then:

$$c = \sum_{i=1}^{n} r_i d^{(i)}.$$

Now consider the polynomials $\delta_i(x)$ generated at step (2), and the polynomial:

$$h(x) = \sum_{i=1}^{n} r_i \delta_i(x).$$

As $\delta_i(x)$ have all degree $\leq t$, so is also $h(x)$. We have:

$$h(0) = \sum_{i=1}^{n} r_i \delta_i(0) = \sum_{i=1}^{n} r_i d^{(i)} = c.$$

On the other hand, the sharing implicitely done at step (4) corresponds to h:

$$h(j) = \sum_{i=1}^{n} r_i \delta_i(j) = \sum_{i=1}^{n} r_i d_{i,j} = c^{(j)}.$$

So if $t < n/2$, this protocol generates shared values of the final result. These values are finally recombined by interpolation. □

46 Shamir's no-key protocol

Curiously, there is also the possibility of encrypted communication without using keys. The security is based on the difficulty of the discrete logarithms.

Setup. Let p be a very big prime, publicly known. For communication, Alice chooses randomly an $a \in \mathbb{Z}_{p-1}^{\times}$ and computes $a' \in \mathbb{Z}_{p-1}^{\times}$ such that $aa' = 1 \bmod p-1$. Bob also chooses randomly a $b \in \mathbb{Z}_{p-1}^{\times}$ and computes a $b' \in \mathbb{Z}_{p-1}^{\times}$ such that $bb' = 1 \bmod p-1$. Alice and Bob do not change informations about their choices.

Protocol. Alice computes $c = m^a \bmod p$ and sends it to Bob. Bob computes $d = c^b \bmod p$ and sends it to Alice. Alice computes $e = d^{a'} \bmod p$ and sends it to Bob. Bob computes $m = e^{b'} \bmod p$, and gets the clear text.

Indeed,

$$e^{b'} \bmod p = d^{a'b'} \bmod p = c^{ba'b'} \bmod p = m^{aba'b'} \bmod p = m^{k(p-1)+1} \bmod p = m \bmod p.$$

In the less probable case that $d = m$, Bob sends some random word back to Alice, because he does not want the enemy to know that he already decrypted the message. For this case, the message must contain a correctness key, which is known only by Alice and Bob. □

The encryption $c = m^a \bmod p$ with decryption $m = c^{a'} \bmod p$ is called Pohlig - Hellman encryption.