Introduction to Cryptography and Coding – M1 MOSIG

Vanessa Vitse

2015-2016

Contents

1 Modular arithmetic (15/02/16—17/02/16)

1.1 Prerequisites

Basic properties of the integers

Definition 1.1 (Divisibility)**.** *Let a and b two integers. Then a* divides *b (or b is a* multiple *of a) if there exists an integer c such that $b = a \cdot c$. This is denoted $a|b$.*

Property 1.2. *1. For all $a \in \mathbb{Z}$, $1|a$ and $a|a$ (reflexivity).*

2. If $a|b$ and $b|c$ then $a|c$ (transitivity).

3. If $a|b$ and $a|c$ then $a|(b+c)$.

4. For all integer $c \neq 0$, $a|b \Leftrightarrow ac|bc$.

Definition 1.3 (Prime numbers)**.** *A* prime number *is a positive integer $p \neq 1$ that is only divisible by ± 1 and $\pm p$. The set of prime numbers is denoted $\mathcal{P}$; $\mathcal{P} = \{2, 3, 5, 7, 11, 13, 17, \ldots\}$. A positive integer that is not a prime is called* composite*.*

Theorem 1.4. *There are infinitely many prime numbers. Let $\pi(n)$ be the number of primes smaller than n, then $\pi(n) \sim n/\log n$.*

Remark. *So informally, the probability that a random integer n is prime is about $1/\log n$.*

Theorem 1.5 (Fundamental theorem of arithmetic)**.**
Every nonzero integer n can be written as a product of primes:

$$n = \pm 1 \, . \, p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}, \quad p_i \in \mathcal{P}, \quad \alpha_i \in \mathbb{N}.$$

This decomposition is unique if $p_1 < p_2 < \cdots < p_k$ and $\alpha_i > 0$ for all i.

Lemma 1.6 (Euclid's lemma)**.** *Let p be a prime number and a, b two integers. Then $p|ab \Rightarrow p|a$ or $p|b$.*

Asymptotic notations and complexity basics

f, g real functions, g is positive

- $f = O(g)$ if there exists a constant c s.t. $|f(x)| \leq cg(x)$ for all sufficiently large x.
- $f = o(g)$ if $f/g \to 0$ as $x \to \infty$.
- $f \sim g$ if $f/g \to 1$ as $x \to \infty$.
- $f = \Theta(g)$ if there exist constants c_1, c_2 s.t. $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for all sufficiently large x.
- $f = \Omega(g)$ if there exists a constant c s.t. $f(x) \geq cg(x)$ for all sufficiently large x.

Some properties:

Property 1.7. *1.* $f = o(g) \Rightarrow f = O(g)$ *and* $g \neq O(f)$

2. $f \sim g \Leftrightarrow f = (1 + o(1))g$

- The size of an integer a is the number of bits in the binary representation of $|a|$, that is $\lfloor \log_2 |a| \rfloor + 1$
- *Polynomial-time algorithm*: algorithm whose running time is bounded by a polynomial in the length of the input, otherwise said the complexity is in $n^{O(1)}$ where n is the size of the input.
- *Exponential-time algorithm*: algorithm whose running time is exponential in the binary length of the input, that is in $\exp(O(1)n)$.
- *Subexponential-time algorithm*: the complexity is "in between" polynomial and exponential complexities, more precisely the complexity is in

  $$L_n(\alpha) = \exp(O(1)(n^\alpha (\log n)^{1-\alpha})$$

  where $0 < \alpha < 1$ and n is the size of the input. Note that if $\alpha = 1$, then the complexity is exponential and when $\alpha = 0$ it is exponential.

Example. *Addition, multiplication, Euclidean division of integers are polynomial algorithms.*

1.2 Congruences

Theorem 1.8 (Euclidean division)**.** *For* $a, b \in \mathbb{Z}$, $b \neq 0$, *there exist unique* $q, r \in \mathbb{Z}$ *s.t.* $a = bq + r$ *and* $0 \leq r < |b|$. *The integer* r *is the* remainder *in the division of* a *by* b, *and* q *is the* quotient.

Definition 1.9 (Congruence)**.** *Let* $x, y, n \in \mathbb{Z}$. *Then* x *is* congruent *to* y modulo n *if their remainders in the division by* n *are the same.*

In particular

$$\begin{aligned} x = y \bmod n &\Leftrightarrow n|(x-y) \\ &\Leftrightarrow \exists k \in \mathbb{Z}, x = kn + y \end{aligned}$$

Property 1.10. *1. This is an equivalence relation (reflexive, transitive and symmetric)*

2. Compatibility with addition and multiplication mod n*: for all integers* a, b, a', b' *s.t.* $a = a' \bmod n$ *and* $b = b' \bmod n$, *then* $a + b = a' + b' \bmod n$ *and* $ab = a'b' \bmod n$.

The congruence equivalence relation partitions the set $\mathbb{Z}$ into equivalence classes:

Definition 1.11 (Residue classes modulo n)**.** $\mathbb{Z}/n\mathbb{Z}$ *is the set of equivalence classes or* residue classes *modulo* n *for the congruence relation. For any integer* m *in a residue class, we call* m *a* representative *of that class.*

Note that there are precisely n distinct residue classes modulo n, given for example by $0, \ldots, n-1$ (corresponding to the remainders in the Euclidean division by n).

Property 1.12. $(\mathbb{Z}/n\mathbb{Z}, +, \cdot)$ *is a (commutative and unit) ring. See next chapter.*

1.3 Modular exponentiation

Question: given $x \in \mathbb{Z}/n\mathbb{Z}$ and $e \in \mathbb{N}^*$, how to compute $x^e \bmod n$?

An obvious way is to iteratively multiply by x a total of e times. The complexity is then in $O(e\log(n)^2)$. Another (much faster) way is to apply the "square-and-multiply" algorithm; the idea is based on the following mathematical property:

Property 1.13. *Let $e = (e_{\ell-1}\dots e_0)_2$ be the binary expansion of e, that is $e = \sum_{i=0}^{\ell-1} e_i 2^i$. Then*

$$x^e = \prod_{i=0}^{\ell-1} (x^{2^i} \bmod n)^{e_i} = \prod_{i=0, e_i\neq 0}^{\ell-1} (x^{2^i} \bmod n).$$

This yields the following algorithm:

Algorithm 1: "Right-to-left" algorithm for modular exponentiation

Input : $x \in \mathbb{Z}/n\mathbb{Z}$, $e, n \in \mathbb{N}^*$
Output: $y = x^e \bmod n$
$y \leftarrow 1$
$t \leftarrow x$
while $e \neq 0$ **do**
  if $e\%2 = 1$ **then**
    $y \leftarrow y \cdot t \bmod n$
  $e \leftarrow e \gg 1$
  $t \leftarrow t^2 \bmod n$
return y

Exercise 1. Propose another algorithm which reads the bits of e from "left-to-right". What is the complexity of these algorithms ?

1.4 Extended Euclid algorithm

Definition 1.14 (gcd, lcm, coprimality)**.** *For $a, b \in \mathbb{Z}$, we call* $\gcd(a,b)$ *or $a \wedge b$ the greatest common divisor of a and b and lcm(a,b) or $a \vee b$ their least common multiple. We say that a and b are* coprime *if* $\gcd(a,b) = 1$.

In particular,

$$x|a \text{ and } x|b \Leftrightarrow x|(a \wedge b),$$
$$a|m \text{ and } b|m \Leftrightarrow (a \vee b)|m.$$

Property 1.15. *If $a = p_1^{\alpha_1}\dots p_n^{\alpha_n}$ and $b = p_1^{\beta_1}\dots p_n^{\beta_n}$, then*

$$\begin{cases} a \wedge b = p_1^{\min(\alpha_1,\beta_1)} \dots p_n^{\min(\alpha_n,\beta_n)} \\ a \vee b = p_1^{\max(\alpha_1,\beta_1)} \dots p_n^{\max(\alpha_n,\beta_n)} \end{cases}$$

In particular,

$$(a \wedge b) \times (a \vee b) = ab$$

Property 1.16 (Gauss lemma)**.** *If p, q are coprime and x is an integer s.t. $p|qx$, then $p|x$.*

Lemma 1.17 (Bézout lemma)**.** *For $a, b \in \mathbb{Z}$, there exist $u, v \in \mathbb{Z}$ such that $au + bv = \gcd(a, b)$.*

Proof. Constructive proof. We use the fact that if r is the remainder in the Euclidean division of a by b, then

$$a \wedge b = b \wedge r.$$

Now let $r_0 := a$ and $r_1 := b$. We compute iteratively

$$\begin{array}{rcll} r_0 & = & r_1\, q_1 + r_2 & \text{with} \quad 0 \leq r_2 < |r_1| \quad \rightarrow \quad a \wedge b = r_0 \wedge r_1 = r_1 \wedge r_2 \\ r_1 & = & r_2\, q_2 + r_3 & \text{with} \quad 0 \leq r_3 < r_2 \quad \rightarrow \quad r_1 \wedge r_2 = r_2 \wedge r_3 \\ & \vdots & & \vdots \\ r_{n-2} & = & r_{n-1}\, q_{n-1} + r_n & \text{with} \quad 0 \leq r_n < r_{n-1} \quad \rightarrow \quad r_{n-2} \wedge r_{n-1} = r_{n-1} \wedge r_n \\ r_{n-1} & = & r_n\, q_n + r_{n+1} & \text{with} \quad r_{n+1} = 0 \quad \rightarrow \quad r_{n-1} \wedge r_n = r_n \end{array}$$

In particular, $a \wedge b$ is equal to the last non-zero remainder r_n. To get u and v we explicitly introduce the sequences $(s_i), (t_i)$ such that $s_i a + t_i b = r_i$.

- Initialisation: $\begin{cases} s_0 = 1 & t_0 = 0 \\ s_1 = 0 & t_1 = 1 \end{cases}$
- Induction hypothesis: $\begin{cases} s_{i-1}a + t_{i-1}b = r_{i-1} \\ s_i a + t_i b = r_i \end{cases}$

Writing $r_{i+1} = r_i q_i - r_{i-1} = (s_i a + t_i b)q_i - (s_{i-1}a + t_{i-1}) = (s_{i-1} - q_i s_i)a + (t_{i-1} - q_i t_i)b$, you get

$$\begin{cases} s_{i+1} = s_{i-1} - q_i s_i \\ t_{i+1} = t_{i-1} - q_i t_i \end{cases}$$

□

Exercise 2. Write algorithms that compute gcd's and Bézout coefficients.

Theorem 1.18 (Chinese Remainder Theorem – CRT)**.** *Let n, m be two coprime integers and a, b two integers. Then the system*

$$\begin{cases} x = a \bmod n \\ x = b \bmod n \end{cases}$$

admits a unique solution $x \bmod mn$.

Proof. From Bézout, there exist u, v s.t. $un + vm = 1$ and $x_0 = bun + avm$ is a particular solution. If x_1 is another solution of the previous system then $\begin{cases} x_1 - x_0 = 0 \bmod n \\ x_1 - x_0 = 0 \bmod m \end{cases}$. From Gauss lemma, we deduce that $x_1 = x_2 \bmod mn$. □

Exercise 3.

1. Let $a, b, c \in \mathbb{Z}$ such that $(a, b) \neq (0, 0)$. Show that the equation
$$ax + by = c \tag{1}$$
has a solution iff $a \wedge b$ divides c.
2. Find all the integer solutions of the following equations: $7x - 9y = 6$, $11x + 17y = 5$.

Exercise 4. In a country named ASU where the currency is the rallod, the national bank issues banknotes of 95 rallods and coins of 14 rallods.

1. Show that it is possible to pay any integer amount (provided that each participant has access to as many coins and banknotes as needed).
2. Suppose that you need to pay an amount S and that you have access to as many coins and banknotes as needed, but that your creditor cannot give the change. Thus it is possible for example to pay $S = 14$ rallods but impossible to pay 13 or 15 rallods. Show that it is always possible to pay any large enough amount.

Exercise 5. A rooster costs 5 silver coins, a hen 3 coins and a set of 4 chicks 1 coin. Someone bought 100 chickens for 100 coins. How many pieces of each kind has he bought?

1.5 Modular inverse

Definition 1.19. *Let $x, n > 0$ two integers. We say that x admits a* multiplicative inverse *modulo n if there exists $y \in \mathbb{Z}$ such that $x \cdot y = 1 \bmod n$; this is denoted by $y = x^{-1} \bmod n$.*
Similarly, $x \in \mathbb{Z}/n\mathbb{Z}$ is invertible *if there exists $y \in \mathbb{Z}/n\mathbb{Z}$ such that $xy = 1 \bmod n$.*

Remark. *If $a \in \mathbb{Z}$ is invertible and if $a' = a \bmod n$ then a' is also invertible modulo n.*

Theorem 1.20. *An integer a is invertible modulo n iff a and n are coprime.*

Proof. Direct application from Bézout: $ua + vn = 1 \Rightarrow u = a^{-1} \bmod n$. □

Remark. *If p is prime, then every element of $(\mathbb{Z}/p\mathbb{Z})^*$ is invertible.*

Algorithm 2: Computation of inverse modulo n

Input : $a \in \mathbb{Z}$, $n \in \mathbb{N}^*$
Output: $a^{-1} \bmod n$
$s_0 \leftarrow 1$ $s_1 \leftarrow 0$; **while** $b \neq 0$ **do**
  $tmp \leftarrow a$
  $a \leftarrow b$
  $b \leftarrow tmp\%a$
  $q \leftarrow tmp/a$
  $tmp \leftarrow s_0 - qs_1$ $s_0 \leftarrow s_1$ $s_1 \leftarrow tmp$
return s_0

Exercise 6. Solve the following system
$$\begin{cases} 35x = 7 \ [4] \\ 22x = 33 \ [5] \end{cases}$$

Remark. *Given $n \in \mathbb{N}^*$, $g \in \mathbb{Z}/n\mathbb{Z}$ and $x \in \mathbb{Z}$, it is easy to compute $g^x \bmod n$ (there exist algorithms with polynomial complexity). However, there is no efficient algorithm which computes x given $n, g, g^x \bmod n$: this problem is called* discrete logarithm problem *and is useful for many asymmetric cryptographic protocols.*

Definition 1.21 (Euler's totient function)**.** *Euler's totient function (or Euler's phi function) is defined by*

$$\forall n \in \mathbb{N}^*,\ \varphi(n) = |(\mathbb{Z}/n\mathbb{Z})^*|.$$

This equivalent to say that $\varphi(n)$ is the number of integers between 0 and $n-1$ that are coprime with n.

Examples. $\varphi(1) = 1; \varphi(2) = 1; \varphi(3) = 2; \varphi(4) = 2...$

Computation of Euler's totient function

Property 1.22.

- *$\varphi(mn) = \varphi(m)\varphi(n)$ for all coprime positive integers n, m.*
- *$\varphi(p^e) = p^e - p^{e-1} = p^e(1 - 1/p)$ for all prime p and positive integer e.*
- *$\varphi(n) = n\prod_{i=1}^{r}(1 - 1/p_i)$ where $n = p_1^{e_1} \dots p_k^{e_k}$ is the factorisation of n into primes.*

Proof.

- Consider the map $a \in \mathbb{Z}/nm\mathbb{Z} \mapsto (a \bmod n, a \bmod m) \in \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}$ which is well-defined and a bijection according to CRT. Moreover $a \wedge mn = 1$ iff ($a \wedge m = 1$ and $a \wedge n = 1$), so that the previous application gives a bijection between $(\mathbb{Z}/mn\mathbb{Z})^*$ and $(\mathbb{Z}/n\mathbb{Z})^* \times (\mathbb{Z}/m\mathbb{Z})^*$.
- Among the p^e elements between 0 and $p^e - 1$, the only elements which are multiples of p are not invertible; these are $0 \cdot p, 1 \cdot p, \dots, (p^{e-1} - 1) \cdot p$ and there are precisely p^{e-1}.
- Direct from the previous items.

□

2 Algebraic structures for cryptography (29/02/16—30/03/16)

2.1 Groups

Let G be a set. A *binary operation* (or *composition law*) is a map $f : G \times G \to G$. Binary operations are usually written in infix notations, i.e. $a + b$, $a \times b$, $a \cdot b$, ...or simply by juxtaposition, i.e. ab, instead of $f(a, b)$.

Example. *On the set $\mathbb{N}$ of natural integers $+$ and $\times$ are binary operations, but $-$ is not.*

Definition 2.1. *Let G be a set and $\cdot$ a binary operation on G. Then $(G, \cdot)$ is a* group *if*

1. *the binary operation is* associative*: for all $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$*
2. *there exists a (necessarily unique) element $e \in G$, called the* neutral element *or the identity, such that for all $a \in G$, $a \cdot e = e \cdot a = a$*

3. *for each $a \in G$, there exists a (necessarily unique) element $b \in G$, called the* group inverse *of a, such that $a \cdot b = b \cdot a = e$*

A group is called abelian *or commutative if its group law is commutative, i.e. $a \cdot b = b \cdot a$ for all $a, b \in G$.*

Examples. *$(\mathbb{Z}/n\mathbb{Z}, +)$, $(\mathbb{Z}/n\mathbb{Z}, \times)$ are finite groups, but $(\mathbb{Z}^*, \times)$ is not (no inverse).*

The inverse of an element a is often denoted by a^{-1}; similarly, the element $a \cdot a \cdot a \cdot \ldots \cdot a$ (n times) is denoted by a^n. This notation can be extended to $\mathbb{Z}$ by setting $a^{-n} = (a^{-1})^n$ and $a^0 = e$.

In particular, we have that for any $g \in G$ and $a, b \in \mathbb{Z}$,

$$g^{a+b} = g^a g^b$$
$$g^{ab} = (g^a)^b$$

These properties will be extensively used in asymetric cryptographic protocols (El Gamal, RSA...).

Let $(G, \dot{})$ be a group with neutral element e.

Definition 2.2. *The order of an element $g \in G$ is defined by*

$$ord(g) = \min\{n \in \mathbb{N}^* : g^n = e\}$$

Remarks:

1. If G is finite, then the order of any element of G is finite. Indeed, the sequence (g^n) has value in a finite set and is ultimately periodic, in particular there exists $a > b$ integers such that $g^a = g^b$ and $\mathrm{ord}(g) \leq a - b$.
2. We have the following equivalence

$$g^n = e \Leftrightarrow \mathrm{ord}(g) | n.$$

(sufficiency is clear, for the necessary condition make the euclidean division of n by $\mathrm{ord}(g)$).

Theorem 2.3 (Lagrange)**.**

$$\forall g \in G, ord(g) | \#G$$

In particular, we have the following corollary which is at the heart of RSA:

Corollary 2.4 (Euler-Fermat)**.** *Let $a \in \mathbb{Z}/n\mathbb{Z}^\times$ be an invertible element modulo n. Then*

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

In particulier if p is prime, for any $a \in \mathbb{Z}$ we have

$$a^p = a \bmod p.$$

We denote by $\langle g \rangle$ the sub-group

$$\{g^n : n \in \mathbb{Z}\} \subset G.$$

If $\langle g \rangle$ is finite, then $\#\langle g \rangle = \mathrm{ord}(g)$.

2.2 Applications to cryptography: DLP, DH and ElGamal

1. **Discrete logarithm problem**
   Let G be a group and $g \in G$. If $h = g^x \in \langle g \rangle$, the integer x (defined modulo the order of g) is called the discrete logarithm of h in basis g.

   The discrete logarithm problem consists in finding x given g and h. The difficulty of this problem depends highly on the order of g and the group G considered. For example in $G = (\mathbb{Z}/nZ, +)$, the computation of x (which is in this case obtained as the product of h and the multiplicative inverse of g modulo n) can be obtained in polynomial time using the Extended Euclidean algorithm. But if $\langle g \rangle = (\mathbb{Z}/p\mathbb{Z}^*, \times)$, there is no efficient (i.e. polynomial time) algorithm to solve the corresponding DLP. In this case, we say that the function $x \in \mathbb{Z}/(p-1)\mathbb{Z} \mapsto g^x \in \mathbb{Z}/p\mathbb{Z}^*$ is a "one-way-function" in the sense that there exist polynomial time algorithm (fast exponentiation) to compute its values but there is no efficient algorithm to compute its inverse. Such one-way functions are essential in asymetric cryptography.

2. **Diffie-Hellman key exchange protocol**

   Alice and Bob want to share a secret using a public channel.

   To do so, they first consider $G = \langle g \rangle$ where DLP is assumed to be hard (typically $\mathbb{Z}/p\mathbb{Z}^*$[1]). Then, Alice chooses a secret integer a and sends $K_a = g^a$ to Bob. Bob chooses a secret integer b and sends $K_b = g^b$ to Alice. Alice computes $K = K_b^a$ and Bob computes $K = K_a^b$. Both Alice and Bob have arrived at the same value K, because $K_b^a = (g^b)^a = g^{ab} = (g^a)^b = K_a^b$.

   An eavesdropper has access to g, K_a, K_b but can neither deduce a or b (because DLP is difficult), nor K (this is called the Diffie-Hellman problem, which is supposed to be as hard as DLP).

3. **ElGamal encryption scheme**

   We can easily transform the previous key exchange protocol into an encryption scheme. Assume that the setting is the same as in Diffie-Hellman and that Bob wants to send a message to Alice. The three classical subroutines are

   - Key-generation: Alice chooses a random integer a in between 2 and $\text{ord}(g)-1$ and publishes her public key $K_{A,pub} = (g, h)$ where $h = g^a$.
   - Encryption: Bob chooses b in between 2 and $\text{ord}(g) - 1$ and computes $(c_1, c_2) = (g^b, h^b m)$ which is the encryption of m.
   - Decryption: Alice decrypts (c_1, c_2) by computing $c_1^{-a} c_2$.

   It is easy to check the correctness of this scheme: $c_1^{-a} c_2 = g^{-ab} g^{ab} m = m$. The security is similar as in Diffie-Hellman.

2.3 Fields

Definition 2.5. *A field is a set K together with two binary operations $+$ and $\cdot$ such that*

1. *$(K, +)$ is a commutative group with neutral element denoted by 0,*
2. *$\cdot$ is associative and commutative with neutral element denoted by 1 (which is assumed to be different from 0),*

[1] This group is cyclic, i.e. there always exists an integer g such that $\mathbb{Z}/p\mathbb{Z}^* = \{g^n : n \in \mathbb{N}^*\}$.

3. $\cdot$ *is distributive over* $+$,

4. *every non zero element has an inverse for* $\cdot$.

In particular if $(K, +, \cdot)$ is a field, the $(K^*, \times)$ is a group.

Example. $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ *are fields. If* p *is prime, then* $\mathbb{Z}/p\mathbb{Z}$ *is a field but if* n *is composite, then* $\mathbb{Z}/n\mathbb{Z}$ *is not a field! For example,* 2 *has no multiplicative inverse in* $\mathbb{Z}/4\mathbb{Z}$.

Definition 2.6. *A finite field is a field that has finite cardinality.*

So far the only examples of finite field that we have seen is $\mathbb{Z}/p\mathbb{Z}$ with p prime, but these are not the only ones! The background given in the next section will allow us to construct other finite fields.

2.4 Polynomial arithmetic

Let K be a field. We define $K[X]$ as the set of polynomials with coefficients in K. Addition and multiplication of polynomials are supposed to be known; as a reminder, the euclidean division is also available on polynomials:
for all $P_1, P_2 \in K[X]$, $P_2 \neq 0$, there exist unique $Q, R \in K[X]$ s.t. $P_1 = P_2Q + R$ with $\deg R < \deg P_2$.

Definition 2.7 (Gcd and lcm)**.** *Let* P_1, P_2 *be two polynomials in* $K[X]$. *The gcd of* P_1 *and* P_2 *is the monic polynomial with highest degree dividing both* P_1 *and* P_2. *The lcm of* P_1 *and* P_2 *is the monic polynomial with smallest degree that is a multiple of both* P_1 *and* P_2. *The polynomials* P_1 *and* P_2 *are coprime if* $P_1 \wedge P_2 = 1$.

Property 2.8.

- $Q|P_1$ *and* $Q|P_2 \Leftrightarrow Q|(P_1 \wedge P_2)$.
- $P_1|Q$ *and* $P_2|Q \Leftrightarrow (P_1 \vee P_2)|Q$.
- *Gauss: if* P *and* Q *are coprime and* $P|QR$, *then* $P|R$.
- *Bézout: there exist* $U, V \in K[X]$ *such that* $UP_1 + VP_2 = (P_1 \wedge P_2)$

Gcd's and Bézout coefficients can be computed with the extended Euclid algorithm, as in the integer case.

Exercise 7.

1. Compute the gcd of $X^5+2X^4+2X^3+3X^2+4X+4 \in \mathbb{Z}/7\mathbb{Z}[X]$ and $X^4+3X^3+5X^2+3X+1 \in \mathbb{Z}/7\mathbb{Z}[X]$. (Answer: $X^2 + 4X + 1$).
2. Compute the Bézout coefficients of $P_1 = X^3+2X^2+2X+1 \in \mathbb{Z}/3\mathbb{Z}[X]$ and $P_2 = X^3+X^2+2 \in \mathbb{Z}/3\mathbb{Z}[X]$. (Answer: $(2X + 1)P_1 + XP_2 = 1$).

We can also define a congruence relation for polynomials in an obvious fashion, so that working modulo a polynomial $P \in K[X]$ is equivalent to working in $K[X]/(P)$. In particular, using the Euclidean division we see that the elements of $K[X]/(P)$ (i.e. the residue classes modulo P) are in one-to-one correspondence with the set of polynomials of $K[X]$ of degree strictly smaller than $\deg P$. As a consequence, if K is a finite field then $\#K[X]/(P) = \mathbb{K}^{\deg(P)}$.

Property 2.9.
- *Chinese remainder theorem: let P_1 and P_2 two coprime polynomials in $K[X]$, then for any polynomials Q_1, Q_2, the equations* $\begin{cases} P = Q_1 \bmod P_1 \\ P = Q_2 \bmod P_2 \end{cases}$ *have a solution, unique modulo P_1P_2.*
- *Modular inverse: a polynomial $Q \in K[X]$ is invertible modulo P (i.e. there exists R s.t. $QR = 1 \bmod P$) iff Q and P are coprime.*

Exercise 8.

1. Find a polynomial P in $\mathbb{Z}/3\mathbb{Z}[X]$ such that $\begin{cases} P = X^2 + X \bmod X^3 + 2X^2 + 2X + 1 \\ P = 2X + 1 \bmod X^3 + X^2 + 2 \end{cases}$
2. In $\mathbb{Z}/2\mathbb{Z}[X]$, is $X^3 + X + 1$ invertible modulo $X^4 + X^2 + 1$? If so, compute its modular inverse. (Answer: yes, $X^3 + X^2 + 1$).

Definition 2.10. *A polynomial $P \in K[X_1, \ldots, X_n]$ is* irreducible *if $\deg P > 0$ and P is not a product of two non-invertible polynomials, i.e.*

$$P = P_1P_2 \quad \Rightarrow \quad P_1 \in K^* \text{ or } P_2 \in K^*.$$

Example. *The irreducible polynomials of $\mathbb{C}[X]$ (or more generally $K[X]$ where K is algebraically closed) are exactly the degree one polynomials. In $\mathbb{R}[X]$, the irreducible polynomials are the degree one polynomials and the degree 2 polynomials of negative discriminant.*

Theorem 2.11 (Unique factorization)**.** *Any non-zero polynomial $P \in K[X]$ can be written as*

$$P = c\, P_1^{\alpha_1} \ldots P_k^{\alpha_k},$$

where $c = LC(P) \in K^*$, $\alpha_i \in \mathbb{N}$, and the polynomials P_i are monic irreducible. This decomposition is unique up to permutation and terms with exponent zero.

Exercise 9.

1. List all the irreducible polynomials of $\mathbb{Z}/2\mathbb{Z}[X]$ of degree up to 4.
2. Factorize $X^7 + 1 \in \mathbb{Z}/2\mathbb{Z}[X]$.

An important remark is that if $P \in K[X]$ is an irreducible polynomial then $K[X]/(P)$ is a field. Moreover,

Theorem 2.12 (admitted)**.** *If p is prime and $n \in \mathbb{N}^*$, then there exists a irreducible polynomial $Q \in \mathbb{Z}/p\mathbb{Z}[X]$ of degree n.*

In particular, there exists finite fields with p^n elements for any prime p and any $n \in \mathbb{N}^*$ and all finite fields are obtained with this construction.

3 Factorisation and RSA (04/04/16—06/04/16)

3.1 Factorisation as a hard problem

Multiplication is the simplest example of a **one-way function**. Indeed, multiplying two integers is easy, even for very large numbers. In terms of complexity, multiplying two n-bit integers costs $O(n^2)$ basic operations with the standard algorithm, or down to $O(n\log(n)\log\log(n))$ with modern algorithms for very large numbers. But the converse operation, factoring an integer as a product of two non-trivial numbers, is often much harder. This observation is the basis of several cryptosystems, and in particular of the widespread Rivest-Shamir-Adleman (RSA) scheme.

The simplest factorisation algorithm is trial division : to factor N, divide N by 2, 3, 4, 5, 6 ... until a zero remainder is obtained. If no divisor is found before $\sqrt{N}$ then N is prime. This works fine if N has a small divisor, but in the worst case this requires $O(\sqrt{N})$ divisions, which is exponential in the size of N.
An obvious improvement is to only test divisibility of N by prime numbers smaller than $\sqrt{N}$, but this supposes the knowledge of those primes... They can be found using the sieve of Erathostenes.

factorisation records: $L(1/3)$ complexity, 728-bit semi-prime factorisation in 2009 (2000 CPU-years)

Difficulty relies partially on the fact that there is a lot of prime numbers (recall density)

To construct factorisation challenges, need to be able to produce efficiently large prime numbers

Primality testing much easier that factorisation: in theory there exists a deterministic polynomial algorithm for primality testing but not really practicable

Simplest test of primality: Fermat (non-)primality test gives a certificate of non-primality but no information about the factorisation.

3.2 RSA in confidentiality

In cryptosystems based on RSA, we use the multiplicative group of $\mathbb{Z}/N\mathbb{Z}$ where $N = pq$ is a product of two large primes. After choosing p, q, we also choose an encryption exponent e such that $e \wedge \varphi(N) = 1$ and compute its inverse $d = e^{-1} \bmod \varphi(N)$. In particular, there exists an integer k such that $ed = 1 + k\varphi(N)$.

Theorem 3.1. *Let p, q, N, e, d as previously defined. Then the maps*

$$x \in \mathbb{Z}/N\mathbb{Z} \mapsto x^e \in \mathbb{Z}/N\mathbb{Z}$$

$$x \in \mathbb{Z}/N\mathbb{Z} \mapsto x^d \in \mathbb{Z}/N\mathbb{Z}$$

are inverses of each other.

Proof. ...

The first map is the encryption function in RSA and the second is obviously the decryption one.

Efficiency of RSA:

- modulus N: efficient primality tests allow to choose randomly and quickly large primes (in the 1024–4096 bit range), multiplying them is easy

- computation of d: use Extended Euclid algorithm
- encryption and decryption: use fast modular exponentiation algorithms
- to speed up encryption, e is often chosen as a relatively small integer with low Hamming weight, i.e. few 1's in is binary expansion; typically $e = 65537 = 2^{16} + 1$ (which is prime)
- decryption can be sped up using CRT

Exercise 10. Alice wants to send the message $m = 100$ to Bob with a RSA encryption. The public key of Bob is $(N, e) = (319, 11)$. What do Alice and Bob need to compute ?

Security of RSA:

- It relies on the difficulty of computing e-th root modulo a non-prime integer N. So far, the only known algorithm for this is to factorize N, compute the inverse d of e modulo $\varphi(N)$ and proceed as the private key owner.
- Note that the knowledge of e and d allows to recover a multiple of $\varphi(N)$ and thus recover the factorisation of N (see next exercise).
- RSA encryption is deterministic : it is easy to decrypt a value x chosen from a small subset (e.g. the alphabet) simply by enumerating possible encryptions of elements of this subset.
  For a secure implantation of RSA use the optimal asymmetric encryption scheme (OAEP) of Bellare and Rogaway.

Exercise 11. RSA and factorisation
Let $N = pq$ be a RSA modulus.

1. We know that $\varphi(N) = 792$, recover the factorisation of $N = 851$.
2. Show that the difficulty of computating $\varphi(N)$ from N is polynomially equivalent to the difficulty of factoring N.

Exercise 12. Common modulus
Alice and Bob choose a common modulus N but different keys (e_A, d_A) and (e_B, d_B) respectively. Suppose that $e_A \wedge e_B = 1$.
Show that if Alice and Bob encrypt a same message M then anybody who listens the encryptions $C_A = M^{e_A} \bmod N$ and $C_B = M^{e_B} \bmod N$, can recover M.

Exercise 13. Common exponent
William, Jack and Averell public keys are $(N_W, 3)$, $(N_J, 3)$ and $(N_A, 3)$ respectively. Joe sends a confidential message M encrypted with their respective public keys. Show how Lucky Luke is able to recover easily the message sent by Joe to his brothers.

Exercise 14. Let $N = pq$ be a RSA modulus, and let d such that $q - p = 2d > 0$.

1. Show that $N + d^2$ is a perfect square.
2. Show that if d is small, then it is easy to factorise N.

Exercise 15. Ciphertext attack
Given access to a RSA decryption box, show that it is possible to decrypt efficiently messages not submitted to this box (hint: use multiplicative property of RSA encryption).

3.3 RSA in signature

It is easy to convert the previous encryption scheme into a signature one.

4 Attacks on factorization and DLP 27/04/2016-02/05/2016

The difficulty of factorizing (more precisely finding a factor) of a given integer depends mostly on the size of its smallest prime factor. It is generally admitted that the hardest case is when the integer is a product of two large prime integers of similar size, typically RSA modulus.

On the other hand, the difficulty of solving the DLP highly depends on the considered group. Let us recall the definition of the general *discrete logarithm problem* (DLP) on a group G:

Problem. *Let G be a group and $g \in G$ be an element of finite order n. The discrete logarithm of $h \in \langle g \rangle$ is the integer $x \in \mathbb{Z}/n\mathbb{Z}$ such that $h = g^x$.*
Given g and $h \in \langle g \rangle$, the discrete logarithm problem consists in computing the discrete logarithm of h in base g.

Exercise 16. Let $g = 2$ and $h = 9$. Find the discret log of h in base g for $G = (\mathbb{Z}/13\mathbb{Z}, +)$ first and then $G = (\mathbb{Z}/13\mathbb{Z}^*, \times)$.

Exercise 17. Explain how to solve DLP when $G = (\mathbb{Z}/n\mathbb{Z}, +)$. What is the complexity of the proposed method?

We can distinguish essentially two kinds of DLP algorithms: the generic ones, available for any group, and algorithms that take into account the group representation, thus specific to a family of group.

Definition 4.1. *A DLP algorithm is* generic *when the only authorized operations are*

- *addition of two elements,*
- *opposite of an element,*
- *equality test of two elements.*

In particular, generic attacks can be applied indifferently to any group, which is represented as a black box.

Example. *Brute force search: $\forall x \in \{0; \cdots : n-1\}$, test if $g^x = h$. This algorithm has an exponential complexity in the size of the group.*

There is a lower bound on the complexity of a generic algorithm:

Theorem 4.2 (Shoup)**.** *The complexity of generic attacks on the DLP defined over G is in $\Omega(\sqrt{p})$, where p is the largest prime factor of $\#G$.*

We give below examples of algorithms with this optimal complexity. Consequently, to improve the complexity, one has to use additional information on the given group.

4.1 Pohlig-Hellman reduction

We assume that the order n of the element $g \in G$ is known together with its prime factorization $n = \prod_{i=1}^{N} p_i^{\alpha_i}$. Since the discrete log in base g is defined modulo n, the idea is to use the Chinese Remainder Theorem (CRT) and compute first the discrete log modulo $p_i^{\alpha_i}$ for each i. The basic outline of Pohlig-Hellman is:

1. Work with the subgroup generated by $g^{n/p_i^{\alpha_i}}$, of order $p_i^{\alpha_i}$, to find the discrete logarithms modulo $p_i^{\alpha_i}$ and use CRT to deduce the DL in G.
2. Further simplification: to obtain DL modulo $p_i^{\alpha_i}$, compute iteratively its expression in base p_i by solving α_i DLPs in the subgroup generated by $g_i = g^{n/p_i}$, of order p_i.

We illustrate this algorithm on the following example:

Let $G = (\mathbb{Z}/163\mathbb{Z})^*$. We want to compute the DL of $h = 129$ in base $g = 42$. Note that 163 is prime, so by Fermat's theorem $g^{162} = 1 \bmod 163$ and the order of g is a divisor of $162 = 2 \cdot 3^4$. Since $g_2 = g^{162/2} = -1 \bmod 163$ and $g_3 = g^{162/3} = 104 \bmod 163$, we see that g has order exactly 162.

1. Solving the DLP mod 2: we want to find x such that $h^{3^4} = (g^{3^4})^x = g_2^x$; as $h^{3^4} = -1 = g_2$, we deduce that $x = 1 \bmod 2$.
2. Solving DLP mod 3^4: we want to find x such that $h^2 = (g^2)^x$, i.e. $15 = 134^x$. We compute iteratively $x_0, \ldots, x_3$ such that $x = x_0 + 3x_1 + \cdots + 3^3 x_3$. We first compute x_0 by exponentiating both elements by 3^3: $15^{3^3} = (134^x)^{3^3} = (134^{3^3})^x \Rightarrow 104 = 104^{x_0} \Rightarrow x_0 = 1$. Then x_1 satisfies $15 = 134^{1+3x_1+3^2x_2+3^3x_3}$, so $(15 \cdot 134^{-1})^{3^2} = (134^{3x_1+3^2x_2+3^3x_3})^{3^2} = (134^{3^3})^{x_1} \Rightarrow 1 = 104^{x_1} \Rightarrow x_1 = 0$. After similar iterative computations, we get $x_2 = 2$ and $x_3 = 2$, so that $x = 73 \bmod 3^4$ and $x = 73 \bmod 162$.

As illustrated on this example, we see that solving DLP in a group of size n is approximately as hard as solving DLP in a group of size the largest prime factor of n.

4.2 Baby-step Giant-step

The basic idea behind BSGS is to use birthday paradox and space time trade-off to speed up exhaustive search.

Let $d = \lfloor \sqrt{\#G} \rfloor$. The outline of the algorithm is composed of three steps:

1. Store the list $L = \{(g^j, j) : 0 \leq j \leq d\}$;
2. for $0 \leq k \leq \#G/d$, compute $h(g^{-d})^k$ and check if it appears in L;
3. search for a collision $h(g^{-d})^k = g^j$ and deduce that the DL of h is $j + dk$.

The complexity of this algorithm is clearly in $O(\sqrt{\#G})$ in memory and time (considering that the cost of membership test is in $O(1)$ using an hash table).

Exercise 18. Use BSGS to compute in $(\mathbb{Z}/47\mathbb{Z})^*$ the discrete log of 33 in base 45.

Exercise 19. Combine BSGS and Pohlig-Hellman to compute in $(\mathbb{Z}/137\mathbb{Z})^*$ the discrete log of 55 in base 3.

4.3 Pollard-Rho

This generic algorithm is an improvement of BSGS based on an iteration of pseudo-random functions, that has the same time complexity but a memory cost in $O(1)$. It is also interesting as a first example of algorithm whose main idea can be applied both to factoring and to computation of discrete logs.

4.3.1 Pollard-Rho for DLP

Suppose we want to compute the discrete log of an element h in a group G in base $g \in G$. The key ingredient of Pollard-Rho is the iteration of a map $f : G \to G$ satisfying the following properties:

- the function f has to be easy to compute
- for any $z \in G$, if we know two integers a and b such that $z = g^a h^b$, we can easily compute two integers a', b' such that $f(z) = g^{a'} h^{b'}$;
- f behaves approximately like a random function.

A simple example is to divide G in three subsets G_1, G_2 and G_3, for instance according to the values of the least significant bits in a binary representation of the elements of G, and setting

$$f(z) = \begin{cases} gz & \text{if } z \in G_1 \\ hz & \text{if } z \in G_2 \\ z^2 & \text{if } z \in G_3 \end{cases}$$

More "randomness" can be achieved by dividing G in more subsets and having more different expressions for $f(z)$.

In order to compute the DL of h in base g, we start with an element $z_0 = g^{a_0} h^{a_0} \in G$ where a_0 and b_0 are random integers. We then compute the sequence $z_1, z_2, \ldots$ where $z_{i+1} = f(z_i)$; we compute in parallel the sequences $(a_i), (b_i)$ such that $z_i = g^{a_i} h^{b_i}$. Since G is finite, after some time we will obtain a collision, i.e. find two different indices i and j such that $z_i = z_j$ (note that this implies $z_{i+k} = z_{j+k}$ $\forall k \geq 0$). Then $g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$, so $g^{a_i - a_j} = h^{b_j - b_i}$. From this we can recover the discrete log of h if $b_j - b_i$ is invertible modulo the order of g, or at least partial information if $b_j \neq b_i$; otherwise we start again with a different z_0.

In pratice, the sequence (z_i) should not be stored. Different methods can be used in order to find a collision with only $O(1)$ memory requirement. Floyd's cycle detection uses two sequences (z_i) and (w_i) with $w_i = z_{2i}$ and looks for equalities $z_i = w_i = z_{2i}$ for $i > 0$. Indeed, let i_0 and j_0 be the smallest integers such that $i_0 < j_0$ and $z_{i_0} = z_{j_0}$, and let $\ell = j_0 - i_0$. Then for all $i \geq i_0$ and all $k \in \mathbb{N}^*$ one has $z_i = z_{i+k\ell}$. In particular, for $k = \lceil i_0/\ell \rceil$ and $i = k\ell$ (so that $i_0 \leq i \leq i_0 + \ell$), we obtain $z_{2i} = z_{i+k\ell} = z_i$, and a collision will be found after at most $i_0 + \ell = j_0$ iterations.

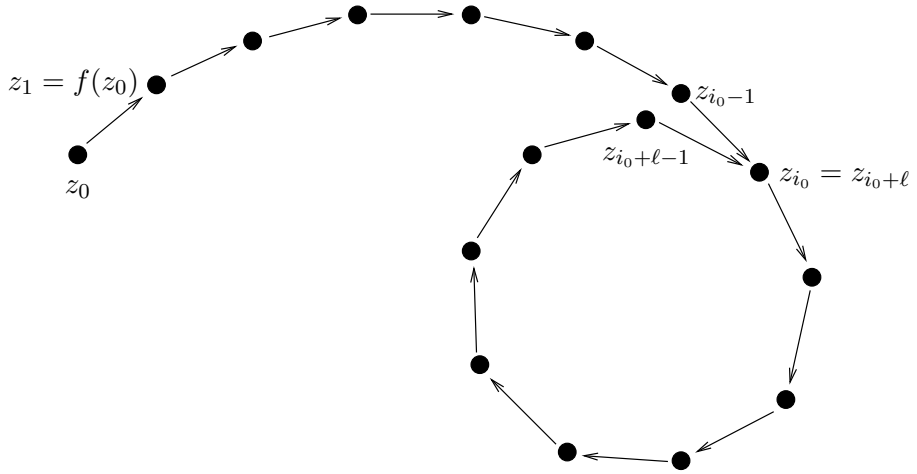

Figure 1: The sequence iterated by f from z_0 in Pollard-Rho.

In order to analyze the complexity of this algorithm, we have to understand when appears the first collision, i.e. the smallest integer j_0 such that there exists $i_0 < j_0$ with $z_{i_0} = z_{j_0}$. For a random function f, a birthday-paradox based argument shows j_0 is in $O(\sqrt{\#G})$ with overwhelming probability. Of course f is not truly random, but the above estimate holds heuristically and is observed in practice.

4.3.2 Pollard-Rho for factoring

For factoring an integer n, we can also iterate a function $f : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$. This time f must satisfy the following properties:

- the function f has to be easy to compute
- for any divisor d of n, if $x = y \bmod d$ then $f(x) = f(y) \bmod d$;
- f behaves approximately like a random function.

For the first condition, obviously we do not know the divisors of n... Usually f is given by a simple polynomial expression, typically $f(x) = x^2 + 1 \bmod n$. This is not very random, but when iterated it is usually chaotic enough for our purpose.

In order to factorize n, we start with a random element $x_0 \in \mathbb{Z}/n\mathbb{Z}$ and compute the sequence $x_1, x_2, \ldots$ where $x_{i+1} = f(x_i)$. After some time (on average in $O(\sqrt{n})$) the sequence will start to repeat itself modulo n. But we can also look at the sequence modulo p, where p is a factor of n. The sequence $(x_i \bmod p)$ will also become periodic, on average after $O(\sqrt{p})$ iterations, hence much sooner than modulo n. Now if $x_i = x_j \bmod p$ but $x_i \neq x_j \bmod p$ then $gcd(x_i - x_j, n)$ gives a non-trivial divisor of n. Combining with Floyd's cycle detection, we obtain the following outline:

1. Pick a random integer $x = y$ in $\mathbb{Z}/n\mathbb{Z}$.
2. $x \leftarrow f(x) \bmod n$, $y \leftarrow f(f(y)) \bmod n$ (so that after i steps, $x = x_i$ and $y_i = x_{2i}$).
3. Compute $g = \gcd(x - y, n)$.
   (a) If $g = 1$ then go to step 2.

(b) If $g = n$ then start again with a different x or a different function f.

(c) If $g \neq 1, n$, then return g, a non-trivial factor of n

Exercise 20. Use this algorithm with $f(x) = x^2 + 1$ and $x_0 = 0$ to compute a factor of $n = 5293$.