Notes on Data Encryption

Riccardo Lo Iacono

January 15, 2026

Document is WIP, typos could be found

Contents.

1 Introduction to Data Encryption **1**
1.1 Notions of abstract algebra 1

2 Public key encryption **5**
2.1 Diffie-Hellman 5
2.2 ElGamal 6
2.3 RSA 7

3 Integers factorization **8**
3.1 Extended Euclidean algorithm . . . 8
3.2 Integers factorization 8

4 Elliptic curves based algorithm **10**
4.1 Elliptic curves 10

5 Digital signature **13**
5.1 Signing with RSA 13
5.2 Signing with ElGamal 13

6 Code theory **15**
6.1 Code representations 15
6.2 Theorems and inequalities 16

Acronyms **19**

- 1 - Introduction to Data Encryption.

Data encryption and codes focuses on two aspects of cybersecurity, that are cryptography and linear codes. For this reason, the present notes will be structured as follow: in *Sections 2-5* we discuss some algorithms used in cryptography, mainly focusing on public key encryption and hyperelliptic curves based algorithms; in *Section 6* we focus on linear codes.

The remainder of this section will provide the reader some notions of abstract algebra, central to many of the topics described in the following sections.

- 1.1 - Notions of abstract algebra.

In the remainder of this section we recall some notions of abstract algebra needed later in the document; we begin by defining what divisibility, greatest common divisor (GCD) and least common multiple (lcm) are in $\mathbb{Z}$, then describing what congruence modulo n is, we continue with defining groups and rings and conclude with an overview on field extension.

- 1.1.1 - Divisibility.

Definition. Let $a, b \in \mathbb{Z}$ and let $b \neq 0$. We say that b divides a, and write $b|a$, if and only if

$$\exists q \in \mathbb{Z} \text{ s.t. } a = q \cdot b.$$

Viceversa, we say that b does not divide a, and we write $b \nmid a$.

Definition (GCD) Let $a, b \in \mathbb{Z}$, we say that the integer $d \in \mathbb{Z}, d > 0$ is the greatest common divisor of a and b if the following holds:

1. $d|a$ and $d|b$, and
2. if $\exists d' : d'|a$ and $d'|b$ then $d'|d$.

Definition (lcm) Let $a, b \in \mathbb{Z}$, we say that the integer $m \in \mathbb{Z}, m \neq 0$ is the least common multiple of a and b if the following holds:

1. $a|m$ and $b|m$, and
2. if $\exists m' : a|m'$ and $b|m'$ then $m|m'$.

Recall that given $a, b \in \mathbb{Z}, b > 0$, it's always possible to write

$$\text{GCD}(a, b) = \alpha a + \beta b$$

for some $\alpha, \beta \in \mathbb{Z}$. We now describe the *Euclidean algorithm* to compute the GCD; in *Section 3.1* we descride a more efficient implementation of the algorithm.

Given $a, b \in \mathbb{Z}$ we proceed as follow[1]:

1. Compute q_i, r_i such that $a = q_i b + r_i$.
2. At step i, let $a = r_{i-1}$.
3. Repeat until $r_i = 0$.
4. Compute α and β by replacing r_i at the step $i-1$.

Most of what we will discuss later on is based on the concept of congruence modulo n, for this reason we briefly overview what this represents.

Let $n \in \mathbb{Z}$ be a fixed integer. Then, for any $a, b \in \mathbb{Z}$

$$a \equiv b \pmod{n} \iff n|(a-b)$$

is an equivalence relation. In fact is:

- *reflexive:* $\forall a \in \mathbb{Z}, a \equiv a \pmod{n} \implies n|(a-a) \implies n|0$.
- *simmetric:* $\forall a, b \in \mathbb{Z}$, if $a \equiv b \pmod{n} \implies n|(a-b)$, but then $n|-(b-a) \implies b \equiv a \pmod{n}$.
- *transitive:* by the same reasoning done for the simmetric property.

Since congruence modulo n is a equivalence relation, this means we can consider the equivalence classes it defines, denote these as $[a] = \{b \in \mathbb{Z} \mid a \equiv b \pmod{n}\}$. One can prove that congruence modulo n defines exactly n distinct classes, whose representatives are the integers $0 \leq k < n$.

Remark. One can think of congruence modulo n, as the remainder of the division by n of same $a \in \mathbb{Z}$.

- 1.1.2 - Groups.

Definition. A *group* is a pair $(G, \cdot)$ where G is a set and

$$\cdot : G \times G \to G$$

is a binary operation satisfying:

1. (Associativity) For all $a, b, c \in G$ we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
2. (Identity) There exists an element $e \in G$ such that for all $a \in G$ we have $e \cdot a = a \cdot e = a$.
3. (Inverses) For every $a \in G$ there exists $a^{-1} \in G$ with $a \cdot a^{-1} = a^{-1} \cdot a = e$.

[1] The procedure described assumes $a \geq b$.

If, in addition, $a \cdot b = b \cdot a$ for all $a, b \in G$, the group is called *abelian* (or commutative).

Definition. Let $H \subseteq G$. We say that H is a subgroup of G, if and only if H is non-empty and is a group under the operation inherited from G.

Definition. A group $(G, \cdot)$ is *cyclic* if there exists an element $g \in G$ such that

$$G = \{g^n \mid n \in \mathbb{Z}\}.$$

Such an element g is called a *generator* of G.

Definition. Let $(G, \cdot)$ be a cyclic group, and let $g \in G$. We call order of g, and denote it as $\mathrm{Ord}(g)$ the smallest integer $k \in \mathbb{Z}$ such that $g^k = e$.

Groups are at the core of abstract algebra as they provide the ground of more complex structures. We now recall on of the most fundamental theorems in group theory: Lagrange's theorem.

Theorem (Lagrange) *Let $(G, \cdot)$ be a finite cyclic group, let $g \in G$. Then,*

$$\mathrm{Ord}(g) | \mathrm{Ord}(G).$$

- 1.1.3 - Rings.

Definition (Ring) A (associative) *ring* is a triple $(R, +, \cdot)$ where $(R, +)$ is an abelian group with identity 0, and $\cdot$ is a binary operation on R satisfying:

1. (Associativity of multiplication) $a(bc) = (ab)c$ for all $a, b, c \in R$.
2. (Distributivity) $a(b + c) = ab + ac$ and $(a + b)c = ac + bc$ for all $a, b, c \in R$.

A ring is called *commutative* if $ab = ba$ for all $a, b \in R$. A ring has *unity* (or 1) if there exists $1 \in R$ with $1 \cdot a = a \cdot 1 = a$ for all $a \in R$.

Definition (Field) A *field* is a commutative ring $(F, +, \cdot)$ with unity $1 \neq 0$ such that every nonzero element has a multiplicative inverse. Equivalently,

$$(F \setminus \{0\}, \cdot)$$

is an abelian group.

We denote fields by $\mathbb{F}$.

Definition (Finite field / Galois field) A *finite field*, or *Galois field*, is a field with finitely many elements.

We will consider multiplicative Galois fields of order a prime, which we denote as $\mathrm{GF}(p)^*$.

Consider the field $\mathrm{GF}(p)[x]$, that is, the fields of polynomials with coefficients in $\mathrm{GF}(p)^*$. Take $g \in \mathrm{GF}(p)[x]$, we say that g is primitive (or irreducible) if it cannot be factored as product of other polynomials in $\mathrm{GF}(p)^*$.

In our discussion the use of irreducible polynomials is due to some codes explained in *Section 6*; what matter for the moment is that these allow to extend the field we are considering, to one with more elements. Precisely, let $g \in \mathrm{GF}(p)[x]$ be a primitive polynomial. For the sake of our discussion assume ∂g to be at most 3, then since it's irreducible, g also admits no roots. Impose i to a root for g.

In doing so, we've defined a new field $\mathrm{GF}(q)[x]$, whose elements are the set

$$\{a + ib \mid a, b \in \mathrm{GF}(p), i \text{ is a root for } g\}.$$

We call such process, symbolic extension of $\mathrm{GF}(p)[x]$.

- 2 - Public key encryption.

The algorithms we are about to discuss are based on some properties of groups we did not discuss in the previous section. We did so because these are somewhat obvious and can be understood to be true without a formal proof.

In addition, when we analyze the complexity of these algorithms, n is the number of digits needed to represents a given integer in a choosen notation. This means that for each $k \in \mathbb{Z}^+, n = \lfloor \log k \rfloor + 1$. With this notation, addition has a complexity of $\mathcal{O}(n)$ time, while multiplication, as well as division, has a complexity of $\mathcal{O}(n^2)$ time.

- 2.1 - Diffie-Hellman.

Let A and B be two entities who want to communicate. Both require that no one but them can understand what the communication is about. One of the many solutions is the Diffie-Hellman algorithm, which is based on the hard problem of computing the discrete logarithm.

The algorithm proceeds as follow:

1. Assume a trusted third party (TTP) exists; that is, assume that there exists a third "person", besides A and B, that acts as mediator between A and B.
2. This TTP chooses a $p \in \mathbb{Z}$ such that p is prime. Once p is fixed, consider $\text{GF}(p)^*$ and fix a $\gamma \in \text{GF}(p)^*$.
3. Tell A and B to choose a value in $\{0, \dots, p-1\}$, call it a and b respectively; this will be their private key.
4. Let A and B compute their public key as $\alpha = \gamma^a, \beta = \gamma^b$ respectively.
5. A and B agree on a $k \in \text{GF}(p)^*$ such that:

$$\text{A obtains } k \text{ as: } \beta^a \pmod{p} = \gamma^{b^a} \pmod{p} = \gamma^{ab} \pmod{p},$$
$$\text{B obtains } k \text{ as: } \alpha^b \pmod{p} = \gamma^{a^b} \pmod{p} = \gamma^{ab} \pmod{p}.$$

In the steps above p, and therefore $\text{GF}(p)^*$, γ, α and β are public.

Consider a malitious individual, call him E, that wants to intrude in the communication; what can he do? Not much actually. In fact, he can only try to compute $a = \log_\gamma \alpha$ (or $b = \log_\gamma \beta$ equivalentely). But, recall that $\alpha, \beta \in \text{GF}(p)^*$, which makes hard to compute the actual value of either a or b.

- 2.1.1 - Complexity of Diffie-Hellman.

In our analisis of Diffie-Hellman complexity we distinguish two case: the complexity for A and B, and the complexity for E.

Consider A, they have to compute $\alpha = \gamma^a$ and $k = \beta^a$ which requires $\mathcal{O}\left(n^3\right)$ time. Similarly, B has to compute $\beta = \gamma^b$ and $k = \alpha^b$, again in $\mathcal{O}\left(n^3\right)$ time.

If we consider E, they have to compute either a or b. Then, they can either try by bruteforce, which requires $\mathcal{O}\left(e^n \cdot n^2\right)$ time; or if they choose a randomised algorithm, it would take $\mathcal{O}\left(\sqrt{e^n \cdot n^2}\right)$ time.

Remark. Recall that with our notation, $n = \lfloor \log p \rfloor + 1$. In addition, we can assume that $k \simeq p$.

- 2.2 - ElGamal.

ElGamal algorithm is somewhat an extention of the Diffie-Hellman protocol; in fact, apart from the last additional step, both work the same. More precisely, a TTP chooses a 128 bits, sometimes even 256 or 512 bits, prime p and fixes a $\gamma \in \mathrm{GF}(p)^*$. As for Diffie-Hellman, both A and B choose a private key, say a and b, and compute their public key $\alpha = \gamma^a \pmod{p}$ and $\beta = \gamma^b \pmod{p}$ respectively.

Once k has been choosed, assuming B wants to send a text t to A, they compute the cypher $c = kt \pmod{p}$, and sends it to A. On the recieving side, A decodes t by computing $t = k^{-1}c \pmod{p}$, where k^{-1} is the multiplicative inverse of $k \pmod{p}$.

Example

Let $p = 47$ and $\gamma = 2$. We have that

$$\mathrm{GF}(47)^* = \{0, 1, 2, \ldots, 46\}.$$

Say A chooses $a = 17$, thus computing their public key, they get $\alpha = 2^{17}$. Say B chooses $b = 31$, therefore computing the public key, they get $\beta = 2^{31}$.

It's easy to see that $k = 12 \pmod{47}$. Hence, if $t = 40$ is the text to encrypt, B cyphers it as $c = km = 10$ and sends it to A. A recieves c and decodes $t = 40$ by computing $k^{-1} = 33$.

- 2.2.1 - Complexity of ElGamal.

Since Elgamal is essentially Diffie-Hellman with the additional step of cyphering the text t, it's easy to observe that the time needed by the algorithm is still $\mathcal{O}\left(n^3\right)$ for A and B; while it's still $\mathcal{O}\left(e^n \cdot n^2\right)$ for E.

- 2.3 - RSA.

The RSA algorithm, named after its inventors *R. Rivest, A. Shamir* and *L. Adleman*, is a encryption algorithm based on the hard problem of integer factorization. In fact, is well known that for $n \in \mathbb{Z}$ sufficiently large, computing its prime factors is not an easy task.

As for the Diffie-Hellman algorithm, assume A and B to be the entities involved into the communication. The algorithm proceeds as follow: one of the two entities, say A, chooses two prime numbers p and $q, p \neq q$ with approximately the same length, and computes $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. Additionally they choose an integer e such that $\text{GCD}(e, \lambda) = 1$; that is, e, λ are coprime. The pair (n, e) is A's public key. Since e, λ are comprime Bezout's identity holds, therefore $1 = ed + \lambda\mu$. Let m be a message B wants to send to A, they cypher it as $c \equiv m^d \pmod{n}$ and transmit c to A. On the receiving side, A takes c and decyphers it as

$$m = c^d \pmod{n} \tag{1}$$

Understanding why *Equation 1* holds is not immediate, for this reason the remainder of this section will be used to show that $m \equiv c^d \pmod{n}$ is indeed correct. Begin by observing that $c \equiv m^e \pmod{n}$, meaning that $c^d \equiv (m^e)^d \pmod{n}$. Recall that Bezout's identity holds, thus $c^d \equiv m^{1-\lambda\mu} \pmod{n}$. But $\lambda = \text{lcm}(p-1, q-1)$, thus $m^{1-\lambda\mu} = m^{1-(p-1)s\mu}$. At this point there are two possibilities:

$$c^d \equiv \begin{cases} 0 \pmod{p}, & \text{if } m = 0, \text{ or} \\ m \pmod{p}, & \text{if } m \neq 0. \end{cases}$$

In fact, if $m \neq 0$, we have that

$$\begin{aligned} \left(m^{p-1}\right)^s &\equiv m^p \pmod{p} \\ &\implies m^{p-1} \equiv 1 \pmod{p} \\ &\implies m^\lambda \equiv 1 \pmod{p} \\ &\implies m^{\lambda\mu} \equiv 1^\mu \pmod{p} \end{aligned}$$

Since a similar reasoning can be done for $\lambda = (q-1)t$, we can conclude that $c^d \equiv m \pmod{p} \wedge c^d \equiv m \pmod{q}$. Furthermore $p \neq q$, therefore $c^d \equiv m \pmod{n}$.

- 2.3.1 - Complexity of RSA.

It's easy to see that as for ElGamal, the complexity of RSA is $\mathcal{O}\left(n^3\right)$ for A and B, and $\mathcal{O}\left(e^n n^2\right)$ for E. In fact, if we consider B and A, their most expensive computation regards the cyphering/decyphering of c; while E has to factorize n to get p and/or q, requiring $\mathcal{O}\left(e^n n^2\right)$ computations by brute force.

- 3 - Integers factorization and fuzzy algorithms.

From what discussed in *Section 2*, it's easy to understand that integer factorization is a difficult task; additionally, many of the algorithms that allows us to actually factorize integer make an extensive use of Euclidean algorithm.

This section will focus on presenting an enhanced Euclidean algorithm, the so called *extended Euclidean algorithm*, and some algorithms for integers factorization.

- 3.1 - Extended Euclidean algorithm.

Recall that the classic Euclidean algorithm for some $a, b \in \mathbb{Z}$, at any step i, computes q_i, r_i as the integer solution to the equation

$$a = q_i b + r_i \quad 0 \leq r_i < b,$$

where $a = r_{i-1}, i > 0$; that is, it defines to sequences $r = \{r_i\}_{i \in I}$ and $q = \{q_i\}_{i \in I}$.

The extedend Euclidean algorithm works in a similar manner; in addition to the sequences $r = \{r_i\}_{i \in I}, q = \{q_i\}_{i \in I}$, it defines the sequences $s = \{s_0\}_{i \in I}$ and $t = \{t_i\}_{i \in I}$ with the initial condition $s_0 = t_0 = 1, s_1 = t_1 = 1$ and $r_0 = \max(a, b), r_1 = \min(a, b)$. Then, for $i = 2, \ldots, n$, compute

$$\begin{cases} q_i = \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor, \\ r_i = r_{i-2} q_{i-1} + r_{i-1}, \\ s_i = s_{i-2} q_{i-1} + s_{i-1}, \\ t_i = t_{i-2} q_{i-1} + t_{i-1}. \end{cases}$$

As for the classic Euclidean algorithm, the computation stops when $r_i = 0$ and $r_{i-1} = \mathrm{GCD}(a, b)$.

- 3.2 - Integers factorization.

The algorithm discussed in this section are called *Pollard's factorization algorithms*, which is a set of fuzzy algorithms for integer factorization; that is, these algorithm may not always work, but if they do, they factorization is very fast.

- 3.2.1 - Pollard's $p-1$ factorization.

Definition (B-smoothness) Let B be a positive integer. We say that $n \in \mathbb{Z}$ is B-smooth if all its prime factor are less or equal to B. That is, if

$$p_i | n \implies p_i \leq B.$$

Pollard's $p-1$ factorization is based on the following: fix a bound B. Define Q as the LCM of all primes powers less or equal B that are less or equal to n. Observe that if $q^l \leq n$, then $l \log q \leq \log n$, thus $l = \lfloor \frac{\log n}{\log q} \rfloor$. Therefore

$$Q = \prod_{q \leq B} q^{\frac{\log n}{\log q}}.$$

Note that $q \leq p-1 < p \leq n$, hence if p is a prime factor of n and $p-1$ is B-smooth, then $p-1|Q$; consequently for any a such that $\mathrm{GCD}(a,p) = 1$, by Fermat's little theorem we have $a^Q \equiv 1 \pmod{p}$. For this reason, if $d = \mathrm{GCD}\big(a^Q - 1, n\big)$, then $p|d$.

Below we provide the pseudo-code to implement Pollard's $p-1$ factorization.

1. Select a smoothness bound B.
2. Randomly pick $a \in \{2, \ldots, n-1\}$ and compute $d = \mathrm{GCD}(a, n)$. If $d \geq 2$ **return** it.
3. For each prime $q \leq B$:
   - 3.1 Compute $l = \left\lfloor \frac{\log n}{\log q} \right\rfloor$.
   - 3.2 Compute $a = a^{q^l} \pmod{n}$.
4. Compute $d = \mathrm{GCD}(a-1, n)$.
5. For $d = 1, n$ error, **else return** d.

Figure 1: Pseudocode for Pollard's $p-1$ factorization.

- 3.2.2 - Pollard's ρ factorization.

Pollard's ρ algorithm, proceeds to define a sequence $\{x_i\}_{i \in I}, x_i \in G = \mathbb{Z}_p$ and it stops when we find x_i, x_j such that $x_i \equiv x_j \pmod{p}$. The question then is: how do we define this sequence? We first begin by partioning G into three sets S_0, S_1 and S_2 of roughly equal size. For instance, let

$$S_i = \{x \in G \mid x \equiv i \pmod{3}\}, i = 0, 1, 2.$$

Then, define the sequence as follow:

$$x_{i+1} = \begin{cases} \alpha x_i, & \text{if } x_i \in S_0 \\ x_i^2, & \text{if } x_i \in S_2 \\ \gamma x_i, & \text{if } x_i \in S_1 \end{cases}$$

for $i \geq 0$, with $x_0 = 1$. Observe that any x_i can be expressed as the product $\alpha^{a_i}\gamma^{b_i}$, for some a_i, b_i. Thus, it holds

$$(b_i - b_j) \cdot \log_\alpha \gamma \equiv (a_i - a_j) \pmod{n}.$$

Provided that $b_i \neq b_j$ (the case $b_i = b_j$ happens with a negligible probability), the discrete logarithm can be easily computed. It can be proved that Pollard's ρ algorithm takes $\mathcal{O}(\sqrt{n})$ time.

- 4 - Projective geometry and elliptic curves based algorithms.

Consider the following: can two parallel line meet? Any reader with little to no background in geometry will provide a negative answer to the question. Which, to be clear, is the most obvious answer.

This said, let us consider a real life example: take for instance the rails of a railway, though this are indeed parallel if we look further enougth these merge at a point. More precisely, their projection meets at a point. This is the idea behind projective geometry.

Formally speaking we have the following: let $\mathbb{F}$ be a field, we call projective plane of dimension n the set

$$\mathbb{P}_n(\mathbb{F}) = \left\{ P = \langle \vec{v} \rangle \;\middle|\; \vec{0} \neq \vec{v} \in \mathbb{F}^{n+1} \right\},$$

where $\vec{v} = [x_0, \ldots, x_n]$ and $\langle \vec{v} \rangle = \{\lambda \vec{v} \mid \lambda \in \mathbb{F}\}$.

Throughout this section we will consider $\mathbb{P}_2$. Observe that

$$\mathbb{P}_2 = \{p = \langle \vec{v} \rangle \mid \vec{v} = [x_0, y_0, z_0], z_0 \neq 0\} \cup \{q = \langle \vec{v} \rangle \mid \vec{v} = [x_0, y_0, z_0], z_0 = 0\}.$$

Given a point $p \in \mathbb{P}_n$, and in particular in $\mathbb{P}_2$, we say that p is a proper point (or affine point) if $p = \langle [x_0, y_0, z_0] \rangle, z_0 \neq 0$; we say that p is an improper point (or point at infinity) if $p = \langle [x_0, y_0, z_0] \rangle, z_0 = 0$. Here by 0 we refer to the zero element of the field.

- 4.1 - Elliptic curves.

Definition (Hyper elliptic curves) Let $n > 0$ be a positive integer. We call hyper elliptic curve on a field $\mathbb{F}$ the set of points in $\mathbb{P}^2(\mathbb{F})$ satisfying the equation

$$y^2 = x^{2n+1} + ax + b.$$

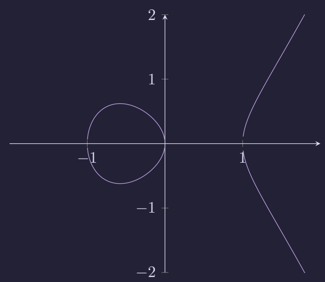
In the case of $n = 1$, we talk about elliptic curves.

Example

Let us consider the curve $y^2 = x^3 - x$, and let $\mathbb{F} = \mathbb{R}$. Then, the curve we get is the following.

We now show, given two point P_1 and P_2 on curve $\mathcal{C}$, how to compute the sum $P_1 + P_2$. Observe that given any two distinct point, we can consider the line passing through them; that is, given $P_1 = (X_1, Y_1)$ and $P_2 = (X_2, Y_2)$, the line passing both has equation

$$Y - Y_1 = \frac{Y_2 - Y_1}{X_2 - X_1}(X - X_1). \tag{2}$$



Hence, combining the equation of the curve and the one above, we can easily compute the points in which these meet. We then consider the point that differ from the initial ones and consider the equation of the vertical line that passes through it. Once again, we combine the latter and the curve equation and get the point $R = P_1 + P_2$.

- 4.1.1 - ElGamal on elliptic curves.

ElGamal algorithm on elliptic curves, apart from some minor differences, works exacly the same as the one described in *Section 2.2*. In fact, a TTP chooses an elliptic curve $\mathcal{C}$ over the field $\mathrm{GF}(p)^*, p$ a prime, and selects γ such that $\gamma \in \mathcal{C}$. Let A and B be the two entities that want to communicate. As for the algorithm described previously, A and B computes $\alpha = \gamma^a, \beta = \gamma^b$ for some private value a and b. Let us note that, in the case of elliptic curves, the group is additive; that is, we compute $a\gamma$ ($b\gamma$ respectively), i.e., the sum of γ with itself a-times (b-times respectively).

Since *Equation 2* suffices the sum of two distinct point, we can't use it. for this reason, in general *Equation 2* is replaced by the equation for the tangen line.

The remainder of the algorithm is the same as the one described in *Section 2.2*.

- 4.1.2 - Lenstra factorization.

Lenstra factorization algorithm, also knows as Elliptic Curves Method (ECM), is a factorization algorithm for integers by means of elliptic curves. In it's essence, the algorithm is very simple, as we are about to describe.

Start by considering an elliptic curve $\mathcal{C}$, in Weierstrass form (eg: $y^2 = x^3 + 2x + 3$ over $\mathrm{GF}(7)^*$), and a point $P \in \mathcal{C}$. Since, in general we consider $\mathbb{Z}_n$, we will be talking about a ring.

Fix a upperbound B, as in the case of Pollard factorization, and compute $2P, 3P, \ldots, kP, \ldots$ up to B. At this point two things may happen:

1. The computation of all $B - 1$ points proceeds flawlessly; in which case we can either attempt with a different curve, a different point or

both, or assume that the number we are trying to factor is actually a prime number[2].

2. At some point during the computation, we fail. That is, we have found some element $v = (X_2 - X_2)$ (*Equation 2*) which has no inverse. Hence, a non trivial divisor is given by $\mathrm{GCD}(n, v)$.

About the complexity: ECM is sub-exponential, that is, it's at the edge between polynomial and exponential algorithms. More rigorously, ECM has a complexity of $L_{1/2,1}(p)$, where

$$L_{\alpha,c}(x) = \exp^{((c+\mathcal{O}(1))(\ln x)^{\alpha}(\ln \ln x)^{1-\alpha})}$$

with $0 \leq \alpha \leq 1$.

[2] More correctly, we should call these pseudoprime numbers, since no factorization has been found nor we are sure it's prime.

- 5 - Digital signature.

Consider the following: an entity A produce some message and it wants to make sure that, if asked to, A can be prove that such message was indeed produced by it; how can this be done? In other words, how can we guarantee that a message was produced by A and not some other entity B? A simple solution is that of signing the message, that is, let A compute some value x such that no other entity besides it can compute it correctly.

In some cases is enough to sign a compressed version of the document, e.g. it's hash value.

In the following we show how to sign a document, assumed as an integer m, by means of ElGamal and RSA.

- 5.1 - Signing with RSA.

Assume that A has already computed its public key (e, n) and the private key d. These, of course, must satisfy $1 = ed + \lambda\mu$.

To sign the document A has to:

1. Compute the hash of its message m, call it $h = H(m)$, with H some hash function.
2. Sign h by computing $f = h^d \pmod{n}$.
3. Source the signed document (i.e. f) to a TTP.

Let us consider the case in which one wants to check whether A produced h. Such entity has to compute $f^e = (h^d)^e \pmod{n} = h$. Hance, proving that A indeed produced h, and therefore m. It's somewhat obvious that if someone tries to falsy claim the ownership of m, once asked to compute h this will fail.

- 5.2 - Signing with ElGamal.

As for the case of RSA, assume that A is an entity that wants to sign the messages these produce. Again, assume that it has already choosed its private key a. Additionally assume that A, or a TTP, chooses a hash function $h : M \to \mathbb{Z}_{p-1}$, where M is the set of all messages.

To sign a message $m \in M, A$ computes $\overline{m} = h(m) \in \mathbb{Z}_{p-1}$. It then chooses a value k such that there exist $k^{-1} \in \mathbb{Z}_{p-1}$. Lastly, computes

$$s = k^{-1}(\overline{m} - ar) \pmod{p-1},$$

where $r = \gamma^k \pmod{p}$. The pair (r, s) is the signature of m.

Say now A is asked to prove the ownership of m, what it has to do is compute

$$v = \gamma^{\overline{m}} - r^s\alpha^r \pmod{p}$$

with α being A's public key. Note that if A is the owner of the message, the above will give as a result zero. Following the same reasoning, if A doesn't own the message the decryption wont succed.

- 6 - Code theory.

Code theory, alongside Information theory, is at the core of modern computer science; the former, due to Richard Hamming, deals with how informations are encoded and corrected when errors occur during trasmission, the latter, due to Claude E. Shannon, deals with communication channels, protocols, and limits of computability.

In what follows, we assume the reader has a basic knowledge of linear algebra, as many reference to it will be made. In addition to that, some definitions are needed to properly follow what is here explained.

Definition. Let A be a finite set of symbols; A is called an alphabet.

Definition. Let A be an alphabet, and n a positive integer. The set

$$A^n = \{(a_1, a_2, \ldots, a_n) \mid a_1, a_2, \ldots a_n \in A\}$$

represents the set of words over A. For $\overline{w} \in A^n$ we define the weight $\|\overline{w}\|$ of $\overline{w}$ as the number of non-zero coeffient of a_i in $\overline{w}$. In addition, $\|\overline{w}\| = 0 \iff \overline{w} = \overline{0}$.

Definition (Hamming distance) Consider $\overline{w_1}, \overline{w_2} \in A^n$. We define their (Hamming) distance as the number of coeffients for which these differ. That is,

$$d(\overline{w_1}, \overline{w_2}) = \|\overline{w_1} - \overline{w_2}\|.$$

It's easily shown that Hamming distance is a metric.

Definition. We call $\mathcal{C} \subseteq A^n$ a code. In addition, if $\mathcal{C}$ is a vector subspace of A^n, we say that $\mathcal{C}$ is a linear code.

In the remaining part of this section, we often use the notation of $[n, k, d]_q$ code, in which n is the length of the strings, k the dimesion of the code, d the minimal distance and q the number of elements in A.

- 6.1 - Code representations.

Consider any linear code $\mathcal{C}$. How do we represent it? Since we are considering vector subspaces, two good solutions are given by linear algebra. In fact, we can represent $\mathcal{C}$ either by means of parametric equations or as set of solutions to a given set of Cartesian equations.

- 6.1.1 - Generating matrix.

Assume $\mathcal{C}$ to be represented by means of parametric equations. This means that $\mathcal{C}$ can be written as linear combination of k linearly indipendent vectors of A^n, that is,

$$\mathcal{C} = \{\overline{c} \in A^n \mid \overline{c} = \alpha_1 \overline{w_1} + \cdots + \alpha_k \overline{w_k}\}$$

with $\alpha_1, \ldots, a_k \in A$ and $\overline{w_1}, \ldots, \overline{w_k}$ vectors generating A^n.

If we denote by G the matrix of the coefficients of the k linearly indipendent vectors, any word $\overline{c} \in \mathcal{C}$ can be written as $\overline{c} = \overline{\alpha} G$ with $\overline{\alpha} \in A^k$.

- 6.1.2 - Parity matrix.

If $\mathcal{C}$ is represented by means of Cartesian equations, this means that any $\overline{c} \in \mathcal{C}$ satisfies a system of equations of the form

$$\begin{cases} \alpha_{11}a_1 + \alpha_{12}a_2 + \ldots + \alpha_{1n}a_n = 0 \\ \alpha_{21}a_1 + \alpha_{22}a_2 + \ldots + \alpha_{2n}a_n = 0 \\ \quad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\ \alpha_{(n-k)1}a_1 + \alpha_{(n-k)2}a_2 + \ldots + \alpha_{(n-k)n}a_n = 0 \end{cases}$$

with $\alpha_{ij} \in A$.

By elementary linear algebra, we can symply consider the matrix H of coefficient of the linear equations. We call H parity matrix. Hence, any $\overline{c} \in \mathcal{C}$ is such that $\overline{c}H^T = \overline{0}$.

- 6.2 - Theorems and inequalities.

Theorem 6.1 (on minimal distance) *Let $\mathcal{C}$ be a linear code, defined by means of a parity matrix H. Let d be the minimal distance of the code, then:*

1. *There exist d linearly dependent columns of H, and*
2. *Any $d-1$ columns of H are linearly independent.*

Proof For (1.): recall that the minimal distance is the minimal weight between words. Hence, if

$$\exists \mathbf{w} \in \mathcal{C} \text{ s.t. } \mathbf{w} = (0, \ldots, 0, \alpha_1, 0, \ldots, 0, \ldots, \alpha_d, 0, \ldots, 0)$$

with $\alpha_i \neq 0$, then there must exist a linear combination of d columns of H which results into the zero vector. But if this happens, then they are linearly dependent.

(2.) is trivial: if $d-1$ columns would be linearly independent, then there would exist $\mathbf{w} \in \mathcal{C}$ with minimal weight $d-1$. But, by hypotesis, d is the minimal weight of the code. □

Theorem 6.2 (error detection and error correction) *Let $\mathcal{C}$ be a linear code, let d be the minimal distance of the code. Then, $\mathcal{C}$ can*

1. *Detect up to $d-1$ errors, and*
2. *Correct up to $(d-1)/2$ errors.*

Proof For (1.): let $\mathbf{w} \in \mathcal{C}$ be the word transmitted and let $\mathbf{z}$ be the one recived. Recall that both part know the code. If e is the number of errors occured during transmission, we have that

- if $e \leq d-1$, then we know by fact that $\mathbf{z} \notin \mathcal{C}$;
- if $e \geq d$, nothing can be said and $\mathbf{z}$ is recognized as a valid code.

For (2.): with the same notation of above, if $e \leq {}^{(d-1)}/_{2}$ then to correct $\mathbf{z}$, we would need a word $\mathbf{w}$ with a distance e from $\mathbf{z}$. Suppose by contradiction that $\mathbf{w_1} \neq \mathbf{w_2}$ are two codewords with distance e from $\mathbf{z}$. That is

$$d(\mathbf{w_1}, \mathbf{z}) \leq \frac{d-1}{2} \qquad d(\mathbf{w_2}, \mathbf{z}) \leq \frac{d-1}{2}.$$

By the triangular inequality, we would have

$$d(\mathbf{w_1}, \mathbf{w_2}) \leq 2 \cdot \frac{d-1}{2} = d-1$$

which is a contradiction. Therefore, $\mathbf{w_1} = \mathbf{w_2}$. $\square$

Theorem 6.3 (Singleton inequality) *Let $\mathcal{C}$ be a $[n,k,d]_q$ code. Then it holds that*

$$d \leq n-k+1.$$

Proof By the dimensional theorem we have have that $\text{Rank}(H) = r = n-k$. Hence, at most r columns are linearly independent. On the other hand, by *Theorem 6.1*, there are at most $d-1$ linearly independent vectors. Therefore, combining the two equations we get

$$d-1 \leq r \iff d-1 \leq n-k \iff d \leq n-k+1.$$

$\square$

If a code is such that its minimal distance d is exactly $n-k+1$, we call such a code *optimal*.

Theorem 6.4 (Hamming's inequality) *Let $\mathcal{C}$ be a $[n,k,d]_q$ code. Then it holds*

$$q^e \left(\sum_{i=0}^{e} (q-i)^i \binom{n}{i} \right) \leq q^n$$

where $e \leq {}^{(d-1)}/_{2}$.

Proof Trivial. Let $\mathbf{w}$ be the codeword to transmit. If no error occured, we get a codeword with distance 0 from $\mathbf{w}$. If one error occured, the we have n possible codewords with distance one from $\mathbf{w}$. By the same reasoning, summing all the terms for a generic number of erros e, we get

$$\sum_{i=0}^{e} \left((q-i)^i \binom{n}{i} \right).$$

Considering all the codewords in the code, we get

$$q^e \left(\sum_{i=0}^{e} (q-i)^i \binom{n}{i} \right) \leq q^n$$

$\square$

If a $[n, k, d]_q$ code satisfies Hamming's inequality, it is said to be a perfect code.

Acronyms.

ECM Elliptic Curves Method. 11

GCD greatest common divisor. 1

lcm least common multiple. 1

TTP trusted third party. 5