Cryptography: notes

Niccolò Simonato

December 7, 2022

Contents

Chapter 1

Elements of Number Theory

1.1 Definitions of Number Theory

1.1.1 The cyclic group $\mathbb{Z}_n^*$

The cyclic group $\mathbb{Z}_n^*$ is defined as

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : (a, n) = 1\}$$

The **generator** of $\mathbb{Z}_n^*$ is a number in $\mathbb{Z}_n^*$ such that:

$$\forall\, a \in \mathbb{Z}_n^* : \exists i : a \equiv_m g^i$$

For this reason, $\mathbb{Z}_n^*$ is also referred as $< g >$.
An interesting property is that only for $[1, 2, 4, \phi, \phi^\alpha, 2\phi^\alpha]$, $\mathbb{Z}_n^*$ is cyclic, where ϕ is a prime number and ϕ^α is a power of a prime number.

1.1.2 Pseudoprime number

For an integer $a > 1$, if a composite integer x divides $a^{x-1} - 1$, then x is called a **Fermat pseudoprime** to base a.

$$x \text{ is spsp(a)} \iff x | a^{x-1} - 1$$

In other words, a composite integer is a Fermat pseudoprime to base a if it successfully passes the Fermat primality test for the base a.
x is $spsp(a)$ is a predicate, that means that some x is a *strong pseudoprime number in base a*.

1.1.3 Carmichael numbers

Carmichael numbers are defined as follows:

- Let n be a composite number;
- Then:
$$b^n \equiv_n b \implies b \text{ is a } \textbf{Carmichael number}$$

1.2 Reminders of Modular Arithmetic

1.2.1 Little Fermat's Theorem

Theorem 1 (Little Fermat's Theorem)**.** *Consider what follows:*

- *Let p be a prime number and $p \nmid a$.*
- *Then*
$$a^{p-1} \equiv_p 1$$

Proof. Consider what follows:

- Let's consider:
$$A = \{na \bmod p : n \in [1, p-1]\}$$
- A has all distinct elements due to the Bijection Lemma (Lemma 1).
- Then
$$A = \mathbb{Z}_n^*$$
- Then,
$$\prod_{n \in A} n \equiv_p \prod_{n=1}^{p-1} na \implies (p-1)! \equiv_p (p-1)! a^{p-1}$$
- Therefore, $a^{p-1} \equiv_p 1$ for the Wilson's Theorem 1.2.2.

□

1.2.2 Wilson's Theorem

Theorem 2 (Wilson's Theorem)**.** *Consider what follows:*

- *Let* $n \in \mathbb{N} \setminus \{0,1\}$
- *Then,*

$$(n-1)! \equiv_n -1$$

Proof. The proof is **by induction** on n:

- **Base case**: $n = 2$

$$1! \equiv_2 -1$$

- **Inductive step**: The theorem is assumed to be true up until $n-1$. Let's consider the case of n:
  - Consider the polynomial

$$g(x) = (x-1)(x-2)\dots(x-(n-1))$$

  - g has degree $p-1$ and costant term $(p-1)!$. It's roots are in $[1, p-1]$.
  - Consider

$$h(x) = x^{p-1} - 1$$

    h has also degree $p-1$ and leading term x^{p-1}.
  - Let $f(x) = g(x) - h(x)$.
  - Then, f has degree at most $p-2$ (since the leading terms cancel), and modulo p also has the $n-1$ roots $1, 2, ..., n-1$.
  - But Lagrange's theorem says it cannot have more than $p-2$ roots.
  -

$$\therefore f \equiv_n 0$$

  - Its costant term,

$$(n-1)! + 1 \equiv_n 0 \iff (n-1)! \equiv_n -1$$

□

1.2.3 Euler-Fermat's Theorem

Theorem 3 (Euler-Fermat's Theorem)**.** *Consider what follows:*

- *Let* $n \in \mathbb{Z}$ *be a number.*
- *Then,*

$$a^{\varphi(n)} \equiv_n 1$$

1.2.4 Bézout's identity

Theorem 4 (Bezout's identity)**.** *Consider what follows:*

- *Let a and b be integers with greatest common divisor d.*
- *Then there exist integers x and y such that $ax + by = d$.*
- *Moreover, the integers of the form $az + bt$ are exactly the multiples of d.*

1.2.5 Chinese Reminder's Theorem

Theorem 5 (Chinese Reminder's Theorem)**.** *Given a system of congruences:*

$$\begin{aligned} x &\equiv_{m_1} a_1 \\ x &\equiv_{m_2} a_2 \\ &\dots \\ x &\equiv_{m_r} a_r \end{aligned}$$

In which each module is prime with each others, that is:

$$\forall i \neq j : (m_i, m_j) = 1$$

then ***there exists a simultaneous solution*** x ***to all of the congruences****, and* ***any two solutions are congruent to one another modulo****.*

$$M = m_1 m_2 \dots m_r$$

Proof. Consider what follows:

- Suppose that x' and x" are two solutions.
- Let $x = x' - x''$.
- Then x must be congruent to 0 modulo each m_i and hence modulo M.
- Let $M_i = \frac{M}{m_i}$, to be the product of all of the moduli except for the i-th.
- Then $GCD(m_i, M_i) = 1$ and therefore $\exists N_i : M_i N_i \equiv_{m_i} 1$.
- Set $x = \sum_i a_i M_i N_i$;
- Then, for each i we see that the terms in the sum other than the i-th term are all divisible by m_i, because $m_i | M_j$ when $i \neq j$.
- Thus, for each i we have: $x \equiv_{m_i} a_i M_i N_i \equiv_{m_i} a_i$,

□

1.2.6 Bijection of a modular function Lemma

Lemma 1 (Bijection of a modular function Lemma)**.** *Consider:*

- $$a \in \mathbb{Z}_n^*$$
- $$f : \mathbb{Z}_n \to \mathbb{Z}_n$$

Then, f is a bijection.

Proof. Part 1: f is injective.

- Assume $f(x_1) = f(x_2) \in \mathbb{Z}_n \iff ax_1 \equiv_n ax_2$.
- Then $\exists k \in \mathbb{Z} : ax_1 - ax_2 = kn$.
- Therefore,

$$a(x_1 - x_2) = kn \iff a^{-1}a(x_1 - x_2) = a^{-1}kn$$
$$\iff (x_1 - x_2) = a^{-1}kn \iff x_1 - x_2 \equiv_m 0.$$

- $\therefore x_1 \equiv_m x_2$, so f is an injection.

Part 2: f is surjective.

- Let $b \in \mathbb{Z}_n$.
- Let $\overline{x} = a^{-1}b \in \mathbb{Z}_n$.
- Then, $f(\overline{x}) \equiv_m a \cdot \overline{x}$
  $\equiv_m a \cdot a^{-1} \cdot b \equiv_m b$.
- Therefore, f is surjective.

Since f is injective $\wedge$ f is surjective, then f is bijective. □

1.2.7 Euler's φ function Lemmas

Lemma 2 (Sum of the prime divisors' φ-function)**.**

$$\forall n \in \mathbb{N}\setminus\{0\} : \sum_{\frac{n}{d}} \varphi(d) = n$$

Where $\frac{n}{d}$ is the set of prime divisors of n.

Proof. The proof is by **contradiction**:

- Let B be
$$\{\frac{h}{n} : h \in \mathbb{Z}_n \wedge n \in \mathbb{N}\}$$
- Therefore,
$$B = \cup_{\frac{n}{d}} \{a \in \{1,\dots,n\} \wedge (a,d) = 1\}$$
- Then,
$$n = |B| = \sum_{\frac{d}{n}} \varphi(d)$$
- Consider
$$(a_1,d_1) = 1 \wedge (a_2,d_2) = 1, \text{where} d_1|n \wedge d_2|n$$
- Then,
$$\frac{a_1}{d_1} = \frac{a_2}{d_2} \iff a_1 d_2 = a_2 d_1$$
- Then,
$$d_1|a_1 d_2 \implies d_1|d_2 \wedge d_2|a_2 d_1 \implies d_2|d_1$$
- Therefore, $d_1 = d_2$. This is clearly a *contradiction*, because in B each divisor is counted once.

□

Lemma 3 (Number of divisors of a prime number's power)**.** *Let $p \in \mathbb{N}$ be a prime number, and $\alpha \in \mathbb{N}$.*
Then,

$$\varphi(p^\alpha) = p^{\alpha-1}(p-1)$$

Proof. The proof is by **induction** on α.
Case base: $\alpha = 1$

$$p = \sum_{\frac{d}{p}} \varphi(d) = \varphi(1) + \varphi(p) = 1 + \varphi(p) \implies \varphi(p) = p - 1$$

Case base: $\alpha = 2$

$$\begin{aligned} p^2 &= \sum_{\frac{d}{p^2}} \varphi(d) = \varphi(1) + \varphi(p) + \varphi(p^2) \\ &= 1 + p - 1 + \varphi(p^2) \implies \varphi(p^2) = p^2 - p - 1 + 1 \\ &= p \cdot (p-1) = p^{\alpha-1}(p-1) \end{aligned}$$

Inductive Step: we can now assume that $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$ up until $\alpha - 1$. We'll proceed now to demonstrate that this is also valid for each α.

$$\begin{aligned} p^\alpha &= \sum_{\frac{d}{p^\alpha}} \varphi(d) \\ &= \sum_{i=0}^{\alpha-1} \varphi(p^i) + \varphi(p^\alpha) \\ &\implies \varphi(p^\alpha) = p^\alpha - p^{\alpha-1} = p^{\alpha-1}(p-1). \end{aligned}$$

□

1.2.8 Multiplicativity of the Euler's φ-function

Theorem 6 (Multiplicativity of the Euler's φ-function)**.** *Let's consider the Euler's function $\varphi(n) = |\mathbb{Z}_n^*$.*
Let's consider that $n = \prod_{i=1}^{r} p_i^{\alpha_i}$, where p_i is a prime number, and $\alpha_i \in \mathbb{N}$.
Then, =

1.3 Theorems for Cryptography purposes

1.3.1 The Miller-Rabin Theorem

Theorem 7 (The Miller-Rabin Theorem)**.** *The **Miller-Rabin Theorem states that**:*

- *Let p be a prime number, such that $p \geq s$*
- *Let $a \in \mathbb{N} : p \nmid a$*

- *Let $p-1 = 2^s d$, where d is odd.*
- *Then,*
$$a^d \equiv_p 1 \vee a^{2^r d} \equiv_p -1, \textit{for some } r \in \{0,1,\ldots,s-1\}$$

Proof. Let's consider the first case:

- $x^2 \equiv_p \pm 1$, for some $x \in \mathbb{N}$.
- $a \in \mathbb{N} \wedge p \nmid a \implies a^{p-1} \equiv_p 1$, due to the Little Fermat's Theorem 1.2.1.
- Since
$$p-1 = 2^s \cdot d \wedge 2 \nmid d \implies a^{p-1} \equiv_p (a^{\frac{p-1}{2}})^2 \equiv_p 1$$
- Then, is proved that
$$a^{\frac{p-1}{2}} \equiv_p \pm 1$$
- Therefore:
$$a^{2^r d} \equiv_p -1 \text{for} r = s-1$$

Consider now the case when $a^{2^r d} \equiv_p 1$:

- If $s = 1$, then $\frac{p-1}{2} = d \implies a^d \equiv_p 1$
- If $s = 2$, then $\frac{p-1}{4} = d \implies a^d \equiv_p 1$
- In general, for $s \geq 3$ we can consider successive square roots, until $r = 0$: then, $a^{\frac{p-1}{2^s}} \equiv_p 1$

□

1.3.2 Miller's Theorem

Theorem 8 (Miller's Theorem)**.** *The Miller's Theorem states what follow:*

- *Let n be a composite and odd number.*
- *Then, n is $spsp(a)$ for at most $\frac{1}{4}$ of the $a_i \in \mathbb{Z}_n^*$*

The following lemma is a consequence of this theorem.

Lemma 4 (Corollary of the Miller's Theorem)**.** *If n is composite and odd, then*

$$\exists a \in \mathbb{Z}_n^* : a \leq b, \textit{such that } n \textit{ is not } spsp(a)$$

1.3.3 Ankey-Montgomery-Bach Theorem

Theorem 9 (Ankey-Montgomery-Bach Theorem)**.** *The Ankey-Montgomery-Bach Theorem states that:*

- *If the GRH*[1] *holds;*
- *If n is composite and odd;*

Then,

$$\exists \in [2, 2log^2(n)] \text{ such that } n \text{ is not } spsp(a)$$

1.4 Euler's Product Theorem

Theorem 10 (Euler's Product Theorem)**.** *If $\mathbb{R}e(s) > 1$, then*

$$\zeta(s) = \sum_{n=1}^{+\infty} \frac{1}{n^2} = \prod_{p} (1 - \frac{1}{p^2})^{-1}$$

[1] Generalized Riemann Hypothesis

Chapter 2

Efficient implementations of elementary operations

2.1 Notation

- Let b be a numeric base.
- Let n be a number in N.
- Length of a number: $l_b(n)$, k. It's equal to $\log(n)$.
- (a,b) is the Maximum Common Divisor of a,b.
- Let $n \in N$: $n = (d_{k-1}, d_{k-2}, \ldots, d_1, d_0)$[1].
- $\varphi(n)$: the number of elements a in $[1,n]$ such that $(a,n) = 1$.
- $\equiv_p$ is the equivalence in base p. Ex.: $5 \equiv_3 = 5 \bmod 3 = 2$.
- Let $\mathbb{Z}_n[X]$ be the set of polynomials in X with coefficients in $\mathbb{Z}_n$.

2.2 Classification of the algorithms' complexity

In order to better identify the classes of complexity of the algorithms, the following 3 classes are defined:

- Polynomial time: $O(\log^{\alpha}(n))$ bit operations, where $\alpha > 0$.
- Exponential time: $O(exp(c \cdot \log(n)))$ bit operations, where $c > 0$.
- Sub-exponential time: $O(exp(c \cdot \log(n))^{\alpha})$ bit operations, where $c > 0, \alpha \in]0,1[$.

[1] $d_{k-1} \neq 0$

2.3 Basic bit operations

2.3.1 Sum of 3 bits - 3-bit-sum

Given n_1, n_2 their sum produces $n_1 + n_2$ and their carry.
Since $n_1, n_2 \in [0,1]$, then this operation can be done in $O(1)$.

2.3.2 Summation of 2 numbers

Given n_1, n_2 their sum produces $n_1 + n_2$.
Since the sum is computed bit by bit, the 3-bit-sum is performed

$$\max\{\text{lenght}(n_1), \text{length}(n_2)\}$$

times.
Each time the carry-on of the previous sum is added to the two digits.
This operation has then complexity:

$$O(\max\{lenght(n_1), \text{length}(n_2)\}) = O(\max\{\log(n_1), \log(n_2)\})$$

2.3.3 Summation of n numbers

The summation of n numbers is simply the sum of two numbers, but performed $n-1$ times.
Let's assume that:

$$\forall i \in [1,n] : M \geq a_i$$

The complexity of this operation is then:

$$O((n-1) \cdot \log(M)) = O(n)$$

2.3.4 Product of 2 numbers

If we consider the classic implementation of the binary multiplication, that is just a sequence of summations.

- The number of summations to execute is equal to the length of the smallest number, $O(\log(n))$.
- The maximum cost of a single summation is $O(\log(m))$.

- Then, $T(m \cdot n) = O(\log(m) \cdot \log(n))$, but, if we consider the worst case[2], that becomes $O(\log^2(m))$.

2.3.5 Division of 2 numbers

Let's consider the division of two numbers m, n. This operation consists in finding two numbers q, r such that $m = q \cdot n + r$.
This is achieved by performing a succession of subtractions, until the ending condition $0 \leq r < n$ is reached.

- Let's consider that the number of steps of this algorithm is $O(\log(q))$.
- Moreover:
$$q \leq m \therefore \#steps = O(\log(m))$$
- It's assumed that the cost of the single subtraction is $O(\log(n))$.
- Then:
$$T(\frac{m}{n}) = O(\log(n) \cdot \log(m))$$

2.3.6 Production of n numbers

Let's assume that:
$$j \in [1, s+1] \text{ and } M = \max(m_j)$$
The cost of the operation $\prod_{j=1}^{s+1} m_j$ is then $O(s^2 \cdot \log^2(M))$. This will now be considered our inductive hypothesis.
Proof by induction, on s:

- (1) **Base case**:
$$T(m_1 \cdot m_2) = O(\log(m_1) \cdot \log(m_2)) = O(k_1 \cdot k_2) \leq c \cdot k_M^2$$
- (2) **Base case**:
$$\begin{aligned} T(m_1 \cdot m_2 \cdot m_3) &= T(m_1 \cdot m_2) + T((m_1 \cdot m_2) \cdot m_3) \\ &\leq c \cdot k_M^2 + c \cdot k_{m_1 \cdot m_2} + k_{m_3} \\ &\leq c \cdot k_M^2 + c \cdot k_{M^2} + k_M \end{aligned}$$

[2] two numbers that are equally large

- Inductive step: we assume the inductive hypothesis to be true up to s. Then:

$$\begin{aligned} T(\prod_{j=1}^{s+1} m_j) &= T([\prod_{j=1}^{s} m_j] \cdot m_{s+1}) \\ &\leq c \cdot \sum_{j=1}^{s} (j \cdot k_M^2) \\ &= c \cdot k_M^2 \cdot \frac{s \cdot (s-1)}{2} \\ &= O(k_M^2 \cdot s^2) \\ &= O(s^2 \cdot \log^2(M)) \end{aligned}$$

Applications

- An analogous dimonstration can be used to prove that

$$T(\prod_{j=1}^{s+1} m_j \bmod n) = O(s \cdot \log^2(M))$$

- This proof can be used to show that

$$T(m!) = O(m \cdot \log^2(m))$$

2.4 Optimizations of more complex operations

2.4.1 Powers & Modular Powers

Let's consider what follows:

$$a^n = a \cdot a \cdot a \cdot \cdots \cdot a$$

Where a is repeated n times.

Trivial implementation

The most trivial implementation would consists in computing the product $\prod_{j=1}^{n} a$.
This would imply a cost of $O(n^2 \cdot \log^2(a)))$.
What follows is a suggestion that could improve the cost of this operation.

Square & Multiply method for scalars, modular powers

Each number in $\mathbb{Z}$ can be represented in a binary notation.
Let's consider

$$n = (b_{k-1}, b_{k-2}, \ldots, b_0) = \sum_{i=0}^{k-1} b_i \cdot 2^i$$

It is clear that we can spare a lot of computational resources by just calculating the powers of 2 and summing the ones that have $b_i = 1$. The following algorithm explains the procedure in detail. Let's compute the complexity of this algorithm:

Algorithm 1: The Square & Multiply Method

```
1  P ← 1;
2  M ← m;
3  A ← a mod n;
4  while M > 0 do
5  |  q ← ⌊M/2⌋;
6  |  r ← M − s · q;
7  |  if r = 1 then
8  |  |  P ← P · A mod n;
9  |  end
10 |  A ← A^2 mod n;
11 |  M ← q;
12 end
13 return P
```

- All of the assignments $X \leftarrow Y$ are implemented in $O(\log(Y))$.
- The cost of 3 is $O(\log(a) \cdot \log(n))$, because it ensures that $A \leq n$.
- Instructions 5 and 6 can be executed in $O(\log(m))$.
- Instrucion 8 can be executed in $O(\log^2(n))$.
- The cost of 10 is $O(\log^2(n))$, because it ensures that $A \leq n$.
- The loop is executed $\log(m)$ times.
- The total cost of this algorithm is then $O(\log(n) \cdot \log(a) + \log(m) \cdot (\log^2(n) + \log(m)))$
  $= O(\log^2(m) + \log(m) \cdot \log(n))$.

This algorithm can be easily converted for the computation of non-modular powers by applying the following changes:

1 $A \leftarrow a \bmod n \Longrightarrow A \leftarrow a$;
2 $P \leftarrow P \cdot A \bmod n \Longrightarrow P \leftarrow P \cdot A$;
3 $A \leftarrow A^2 \bmod n \Longrightarrow A \leftarrow A^2$;

Square & Multiply method for polynomials

Let's consider $\Re = \frac{\mathbb{Z}_n[x]}{x^r-1}$. The modular powers of the elements in this set can be computed by using a variation of the Square & Multiply method.

- Assume that $f, g \in \Re$.
- Let
$$h(x) = f(x) \cdot g(x) = \sum_{j=0}^{2r-2} h_j \cdot x^j$$
- Where
$$h(j) = (\sum_{i=0}^{j} f_i \cdot g_{j-i} \bmod n) \bmod n$$
- Then:
$$T(h_j) = O(j \cdot \log^2(n))$$
- Then:
$$T(h(x)) = O(\sum_{j=0}^{\log^2(n)}) = O(r^2 \cdot \log^2(n))$$

This result will be useful in the following computations.
Let's now consider $\frac{h(x)}{x^r-1}$.

- When $j > r-1$, $h_j \cdot x^j$ does not take any part in the computations.
- When $j = r$, then:
$$\frac{h_r \cdot x^r}{x^r - 1} = h_r + \frac{h_r}{x^r - 1}$$
Or, in other words: $h_r \cdot x^r \equiv_{x^r-1} h_r$.
- In the other cases:
$$h_{r+i} \cdot x^{r+i} \equiv_{x^r-1} h_{r-i} \cdot x^{r-i} \text{ for } 1 \leq i \leq r-2$$

Then:

$$h(x) \equiv_{(n,x^r-1)} f(x) \cdot g(x) \equiv [\sum_{j=0}^{r-2} ((h_j + h_{r+j}) \bmod n) \cdot x^j] + h_{r-1} \cdot x^{r-1}$$

Therefore:

$$T(h(x) \bmod (n, x^r - 1)) = O(r^2 \cdot \log^2(n))$$

Finally, we can analyze the complexity of the computation of the modular power $h(x)$ elevated to n.
In order to optimize the use of the computational resources, we can use a variation of the Square & Multiply method (See 1); although, this time, the computation of the partial products will be conducted by using the previously explained procedure (See 3).
The cost of this method would then be

$$\begin{aligned} T(\#Loops \cdot (h(x) \bmod (n, x^r - 1))) &= O(\log(n) \cdot r^2 \cdot \log(n)) \\ &= O(r^2 \cdot \log^3(n)) \end{aligned}$$

2.4.2 Finding the b-expansion of n (n_b)

Let's consider the cost in bit operations of the conversion of a number n to a new base b.
The algorithm used will be the classical: a succession of divisions by b.

- Let's consider
$$r_i \in \{0, 1, \ldots, b-1\}$$
- Let
$$n_b = (r_{k+1}, r_k, \ldots, r_1, r_0)$$
- Then:
$$\begin{aligned} n &= q_0 \cdot b + r_0 \\ q_0 &= q_1 \cdot b + r_1 \\ &\ldots \\ q_k &= 0 \cdot b + q_k \end{aligned}$$
- Consider then that
$$q_k = r_{k+1}$$

- And that

$$b^{k+2} > n > b^{k+1} \iff (k+2)\cdot\log(b) \leq \log(n) \leq (k+1)\cdot\log(b)$$

- Therefore,

$$k = O(\frac{\log(n)}{\log(b)})$$

We can now proceed with the computation of the cost of this operation:

$$\begin{aligned} T(n_b) &= T(\#Divisions \cdot q_i \bmod b) \\ &= O(\frac{\log(n)}{\log(b)} \cdot \log(n) \cdot \log(b)) \\ &= O(\log^2(n)) \end{aligned}$$

2.4.3 How to use Bezout formula to compute modular inverses

An efficient way of computing the modular inverse of a given number a with in the group $\mathbb{Z}_m^*$ uses the corollary of the *Bezout identity* and the *Extended Euclidean Algorithm*.
That is, given $a \cdot x \equiv_m 1$, we want to compute x.
Extended Euclidean Algorithm, given a, m computes the $gcd(a,m)$ and also returns the coefficients x, y for which $ax + my = 1$.
At this point, the modular inverse of a in $\mathbb{Z}_m^*$ is x:

- Let's consider that $ax + my \equiv_m 1$;
- Since $my \equiv_m 0$, then $ax \equiv_m 1$;
- $\therefore x \bmod n$ is the modular inverse of a in $\mathbb{Z}_m^*$ for its definition.

The complexity of this operation is then $O(\log(x)\log(n))$, because we have to compute the remainder of the division between x and n (this does not take into account the execution of the *Extended Euclidean Algorithm*).

2.4.4 Computing the order of an element in a cyclic group

The order of an element a in $\mathbb{Z}_p^*$ ($m = order(a)$) is the minimum m such that $a^m \equiv_p 1$.
This problem is computationally hard, because the most efficient way to compute order(a) is to brute force its value.
The only optimization available is that we don't have to compute the modulars powers of a from scratch each time, but we can save the results at each iteration.
Therefore, at each step we can only compute the modular product $(a^{p-1} \cdot a) \bmod p$, that has a cost $O(\log^2(p))$. The cost of this algorithm is then $O(order(a) \cdot \log^2(p))$, because we have to compute $a^i \bmod p$ for each attempt to find $order(a)$.

Algorithm 2: The Extended Euclidean Algorithm

Data: a, p
Result: $order_p(a)$

```
x ← a;
ord ← 1;
while x ≢_p 1 do
    x ← x · a;
    ord ← ord + 1;
end
return ord
```

2.4.5 Extended Euclidean Algorithm

The Extended Euclidean Algorithm is a variation of the classic Euclidean Algorithm, that computes the GCD between two numbers a, b.
It also provides the coefficients λ, μ such that $\lambda \cdot a + \mu \cdot b = \text{GCD}(a, b)$.
This algorithm has a cost $O(\log^3(max\{a, b\}))$.

2.4.6 Computation of square and m-th root of n

The following algorithm can be used to compute efficiently $\lfloor \sqrt[m]{n} \rfloor$.
It is assumed that the length of the result is known and is l.
Let's consider the cost of this algorithm:

- Computing x_i^m has cost $O(\log^2(n))$
- Comparing x_i^m and n has cost $O(\log(n))$.
- The length of the loop is $O(\log(n))$ iterations.
- The total cost is therefore $O(\log^3(n))$.

2.4.7 Compute n, m given n^m

We can extract the base and the exponent of an integer by making different attempts.
Let's consider the cost of this operation:

- We have to make at most m attempts by brute force;
- At each attempt we have to compute $\lfloor sqrt[m_i]n^m \rfloor$;

Algorithm 3: The Extended Euclidean Algorithm

Data: a, b
Result: $(\lambda, \mu, GCD(a,b))$

$old\_r \leftarrow a$;
$r \leftarrow b$;
$old\_s \leftarrow 1$;
$s \leftarrow 0$;
$old\_t \leftarrow 0$;
$t \leftarrow 1$;
while $r \neq 0$ **do**
    $quotient \leftarrow floor(old\_r/r)$;
    $old\_r \leftarrow r$;
    $old\_s \leftarrow s$;
    $old\_t \leftarrow t$;
    $r \leftarrow old\_r - quotient \cdot r$;
    $s \leftarrow old\_s - quotient \cdot s$;
    $t \leftarrow old\_t - quotient \cdot t$;
end
return *(s,t,old_r)*

Algorithm 4: The Efficient m-th root of n

Data: n, m
Result: $\lfloor \sqrt[m]{n} \rfloor$

$x_0 \leftarrow 2^{l-1}$;
for $i \leftarrow 1$ **to** $l-1$ **do**
    $x_i \leftarrow x_{i-1} + 2^{l-i-1}$;
    if $x_i^m > n$ **then**
        $x_i \leftarrow x_{i-1}$;
    end
end
return x_{l-1}

- This operation has total cost of

$$\sum_{m=3}^{\log(n)} O(\frac{\log(n)}{m} \cdot \log^2(m) \cdot \log(m)) + O(\log^3(n))$$

[3];

- That is equal to

$$O(\log^3(n)) \sum_{m=3}^{\log(n)} O(\frac{\log^3(m)}{m}) + O(\log^3(n))$$

- $$\sum_{m=3}^{\log(n)} O(\frac{\log^3(m)}{m})$$

  can be approximated by calculating the correspondant integral, to $O(\log\log(n))$.

- The final cost is therefore

$$O(\log^3(n) \cdot (\log\log(n))^2) = O(\log^{3+\epsilon}), \text{ with } \epsilon \in (0,1)$$

2.4.8 Efficient computation of the $\operatorname{lcm} 1, \ldots, B+1$

Consider a number $B \in \mathbb{N}$. Assume that you know $M(B) := \operatorname{lcm} 1, \ldots, B+1$. Then, you can easily obtain $M(B+1)$:

$$M(B+1) = \begin{cases} M(B) \text{ otherwise} \\ qM(B) \text{ if } q^\alpha | (B+1) \wedge q \nmid B \end{cases}$$

Assuming that p is prime.

[3] $\frac{\log(n)}{m}$ is the length of the loop

Chapter 3

Algorithms for primality test

3.1 Miller-Rabin probabilistic primality algorithm

Algorithm 5: Miller-Rabin primality test

Data: $n \in \mathbb{N}$, an *odd* number
Compute s, d such that: $n - 1 = 2^s \cdot d$;
Randomly choose $a \in \mathbb{Z}_n^*$;
if $(a, n) > 1$ **then**
    return *n is composite*;
end
$b \leftarrow a^d \bmod n$;
if $b \equiv_n \pm 1$ **then**
    `/*Due to the Little Fermat's Theorem */`
    return *n is prime or spsp*;
end
$e \leftarrow 0$;
while $b \not\equiv_n \pm 1 \wedge e \leq s - 2$ **do**
    $b \leftarrow b^2 \bmod n$
end
if $b \not\equiv_n 1$ **then**
    return *n is composite*;
end
return *n is prime or spsp*;

3.1.1 Computational complexity of the Miller-Rabin test

Testing the primality for a single value of a has cost of $O(\log^3(n))$ b.o.. Although, we could test for each value in $\mathbb{Z}_n^*$, and that would cost $O(\varphi(n)\cdot\log^3(n))$ b.o..

3.2 Primality Test in $\mathbb{P}$, AKS algorithm

3.2.1 Useful Lemmas

Lemma 5 (Newton's formula lemma)**.** *This lemma states as follows:*
n is prime $\iff (x+b)^n \equiv_n x^n + b$.

Proof. The proof is by identity:

- $(x+b)^n \equiv_n x^n + b \implies n$ is prime.
  - By *contradiction*:
    - Assume that n is composite.
    - Then $\exists p|n$, where $p < n$ is a prime number.
    - Consider
    $$\binom{n}{p} = \frac{n\cdot n-1\cdot\dots\cdot n-p+1}{p\cdot p-1\cdot\dots\cdot 1} > 1$$
    - Assume that:
    $$p^\alpha || n$$
    - Then
    $$p \nmid n$$
    - Let
    $$N = \prod_{i=n-p+1}^{n-1} i$$
    - Let
    $$M = \prod_{i=1}^{p-1} i$$
    - Then,
    $$\binom{n}{p} = \frac{p^\alpha\cdot N}{p\cdot M} = p^{\alpha-1}\cdot\frac{N}{M}$$
    -
    $$(N,p) = (M,p) = 1 \implies p^{\alpha-1} | \binom{n}{p} \wedge p^\alpha \nmid \binom{n}{p}$$

* Therefore:

$$p^{\alpha-1} || \binom{n}{p} \text{ and } \binom{n}{p} \equiv_p 0$$

* But this is a **contradiction**, since $\binom{n}{p} \not\equiv_p 0$, therefore n is prime.

- n is prime $\implies (x+b)^n \equiv_n x^n + b$
  - Consider that $p | \binom{p}{k}$, for $k < n$.
  - Since $\binom{p}{k} \equiv_p 0$, then

$$(x+b)^p = \sum_{k=0}^{p} \binom{p}{k} x^{p-k} \cdot b^k \equiv_p x^p + b^p \equiv_p b$$

□

Lemma 6 (Nair's Lemma)**.** *This lemma states as follows:*

- *Let* $m \geq 7 \in \mathbb{Z}$
- *Let* $\text{lcm}(x, y)$ *be the Least Common Multiplier of* x *and* y*.*
- *Let* $n \leq m \in \mathbb{Z}$*.*
- *Then:*

$$\text{lcm}(m, n) \geq 2^m$$

Lemma 7 (AKS Lemma)**.** *This lemma states as follows:*

- *Let* $n \geq 4$
- *Then:*

$$\exists r \leq \lceil \log_2^5(n) \rceil \text{ such that } d = \wr \nabla \lceil (n)_{\mathbb{Z}_n^*} > \log_2^2(n)$$

Proof. The proof is by *contradiction*:

- Let $n \geq 4 \Rightarrow \lceil \log_2^5(n) \rceil \geq 32$.
- Let V be $\lceil \log_2^5(n) \rceil$.
- Let

$$\Pi = n^{\lfloor \log_2(V) \rfloor} \cdot \prod_{i=1}^{\lceil \log_2^2(n) \rceil} (n^i - 1)$$

  .
- Let v be $\{s \in \{\ldots, v\} : s \nmid \Pi\}$

- Assume by contradiction that$\nu = 0$:
  - Then, by definition: $\forall s \in \nu : s \nmid \Pi$.
  - Consider that $\text{lcm}\{1,\dots,V\}|\Pi$
  - Consider that:

$$\Pi \leq n^{\lfloor \log_2(V) \rfloor} \cdot \prod_{i=1}^{\lceil \log_2^2(n) \rceil} n^i = \tag{3.1}$$

$$n^{\lfloor \log_2(V) \rfloor + \sum_{i=1}^{\lfloor \log_2^2(n) \rfloor} i} = \tag{3.2}$$

$$n^{\lfloor \log_2(V) \rfloor + \frac{1}{2} \lfloor \log_2^2(n) \rfloor \cdot (\lfloor \log_2^2(n) \rfloor + 1)} < \lfloor (\log_2(n))^4 \rfloor \tag{3.3}$$

$$= 2^{\log_2(n) \cdot \lfloor (\log_2(n))^4 \rfloor} \tag{3.4}$$

$$< 2^{\log_2^5(n)} \tag{3.5}$$

$$< 2^V \tag{3.6}$$

  - So, $\text{lcm}\{1,\dots,V\}|\Pi \implies \text{lcm}\{1,\dots,V\} \leq \Pi < 2^V$
  - Since $V \geq 32$ and $\text{lcm}\{1,\dots,V\} \geq 2^V$ due to Lemma6, we have a **contradiction**.
  - Therefore, $\nu \neq 0$
- Let then r be $\min(\nu) \geq 2$
- Assume that q is a prime and $q|r$
  - Consider also that $r|V$, since $r \leq V \implies q^\alpha|V \implies \alpha \leq \lfloor \log_2(V) \rfloor$
- Assume also that every $q|r$, also $q|n$.
- Then, $r = \prod_{q|r} q^\alpha | \prod_{q|r} q^{\lfloor \log_2(V) \rfloor} | \prod_{q|n} q^{\lfloor \log_2(V) \rfloor}$, where p is a prime number.
- Let n be $\prod_{q|n} q^\beta$, where $\beta \geq 1$
- Then, we have $n^{\lfloor \log_2(V) \rfloor} = \prod_{q|n} q^{\beta \cdot \lfloor \log_2(V) \rfloor}$
- Before, we proved:

$$r | \prod_{q|n} q^{\lfloor \log_2(V) \rfloor} | n^{\lfloor \log_2(V) \rfloor} | \Pi$$

- Therefore, $r|\Pi$, but $r \in \nu$, so $r \nmid \Pi$, that is a **Contradiction**.
- Then not every prime divisor of n is a prime divisor of r.

- Consider that $\frac{r}{(r,n)} \in \nu \implies \frac{r}{(r,n)} \leq r = \min(\nu) \implies \frac{r}{(r,n)} = r \implies \frac{r}{(r,n)} = 1$
  - By *contradiction*:
  - Assume that $\frac{r}{(r,n)} \notin \nu$
  - Then, $\frac{r}{(r,n)} | \Pi$
  - Let $r = \prod_{p|r} p^{\alpha}$
  - If $p|r \wedge p \nmid n \implies p|\frac{r}{(r,n)} \implies p^{\alpha}|\frac{r}{(r,n)}$
  - But, if $\frac{r}{(r,n)}|\Pi \implies p^{\alpha}|\prod_{i=1}^{\dots}(n^i - 1)|\Pi \implies r|\Pi$, that is a **contradiction**. Therefore, $\frac{r}{(r,n)} \in \nu$
- So, $\text{ord}(n)_{\mathbb{Z}_r^*} > \lfloor \log_2^2(n) \rfloor$
  - By *contradiction*:
  - $\exists i \leq \lfloor \log_2^2(n) \rfloor$ such that $n^i \equiv_r 1$
  - $\implies r|\prod_{i=1}^{\lfloor \log_2^2(n) \rfloor}(n^i - 1)|\Pi$, but this is a **contradiction**.

□

3.2.2 Agrawal - Kayal - Saxema Theorem

Theorem 11 (Agrawal - Kayal - Saxema Theorem)**.** *Let $n \geq 4 \in \mathbb{N}$, and let $0 < r < n$ such that $(n,r) = 1$ and $order(n) > (\log_2(n))^2$. Then:*

$$n \text{ is prime} \iff \begin{cases} n \text{ is not a perfect power} \\ \nexists p \leq r \\ (x+b)^n \equiv_{(n,x^r-1)} x^n + b, \text{ for every } b \in \mathbb{N} \text{ s.t. } 1 \leq b \leq \sqrt{n} \cdot \log_2(n) \end{cases}$$

3.2.3 AKS algorithm

Correctness

Theorem 12 (Correctness of the AKS algorithm)**.** *The AKS algorithm for the primality test of a given number n is correct.*
n is prime $\iff$ the algorithm returns $TRUE$.

Proof. The proof examines the execution case by case.

- Let's assume that n is prime:
  - Then, the algorithm cannot stop at step 3 or at step 8.

Algorithm 6: AKS primality test pseudocode

Data: $n \in \mathbb{N}$
Result: $TRUE$ if n is prime
1 **if** $n = \alpha^{\beta}$, *where* $\alpha, \beta > 1 \in \mathbb{N}$ **then**
2 | **return** $FALSE$
3 **end**
4 $r \leftarrow \arg\min_x(x, n) = 1$;
5 $d \leftarrow \wr \nabla \lceil (n)_{\mathbb{Z}_r^*} > \lceil \log_2^2(n) \rceil$;
6 **if** $\exists b \leq r : 1 < (b, n) < n$ **then**
7 | **return** $FALSE$
8 **end**
9 **if** $n \leq r$ **then**
10 | **return** $TRUE$
11 **end**
12 **if** $\exists b \in \mathbb{N} : 1 \leq b \leq \sqrt{r} \cdot \log_2(n) \wedge (x+b)^n \not\equiv_{(x^r-1,n)} x^n + b$ **then**
13 | **return** $FALSE$
14 **end**
15 **return** $TRUE$;

  - By Lemma 5, $(x+b)^n \equiv_n x^n + b \forall b \in \mathbb{Z}$ ∴ the algorithm cannot terminate at step 14.
  - Therefore, the algorithm can only terminate at step 11 or 15, so: n is prime $\implies$ the algorithm returns $TRU£$.

- Let's assume that the algorithm returns $TRUE$:
  - Then, the algorithm has terminated at step 11 or 15.
  - If the algorithm has terminated at step 11, then $n \leq r$. Since we checked at 8 that (b, n) is trivial $\forall b \leq r$, then n has no trivial divisors, hence it's prime.
  - If the algorithm has terminated at step 15, let's consider that at 3 and at 8 we verified that condition 1 and 3 of the Theorem 11 hold, respectively.
  - Then, it's verified that: the algorithm returns $TRUE \Rightarrow n$ is prime.
- Therefore, is verified that n is prime $\iff$ the algorithm returns $TRUE$.

□

Complexity

Theorem 13 (Complexity of the AKS algorithm)**.** *The AKS algorithm sustains the following costs for each step:*

- *Step 3 (checking if $n = a^b$) costs $O((\log(n))^{3+\epsilon})$ b.o..*
- *Step 5 (picking r) has cost of $O((\log(n))^{7+\epsilon})$ b.o., in the worst case.*
- *Step 8 (computing (b, n) multiple times) has cost of $O((\log(n))^{7+\epsilon})$ b.o., in the worst case.*
- *Step 14 (verifying that $(x+b)^n \not\equiv_n x^n + b$) has cost of $O(\log(n) \cdot \log(r))$ b.o.*
- *The total cost of the AKS algorithm is then $O(r^{5/2}\log^4(n))$ b.o.*

Chapter 4

Factoring Problem

4.1 Factoring Problem

This section explores the state of the art on the factoring problem, that is at the base of the attacks of numerous crypto algorithms.

4.1.1 Eratostene's sieve

In ancient times, a Greek mathematician named Eratostene came up with an intuitive factoring method, based on the extraction of the prime numbers up to a given N. Let's consider the cost of this algorithm:

- For each $p \leq N^{1/2}$, one squaring operation is executed, and

$$l_p = \lfloor \frac{N}{p} \rfloor - p \leq \frac{N}{p}$$

  sums are executed.

- Since $p^2 + kp \leq N$ for $k \leq l_p$, then:

$$\sum_{p \leq N^{1/2}} (\log^2(p) + \sum_{k=1}^{l_p} \log(p^2 + kp)) \leq$$

$$\sum_{p \leq N^{1/2}} (\log^2(p) + \log(N) \sum_{p \leq N^{1/2}} \frac{N}{p})$$

$$= N\log(N) \sum_{p \leq N^{1/2}} \frac{1}{p} + O(N^{1/2}\log(N))$$

$$= O(N\log(N)\log(\log(N)))$$

Algorithm 7: Eratostene's sieve

Data: $N \in \mathbb{N}$
Result: c, the list of booleans that represents the prime integers up to N

```
1  c[N] ← {True * N};
2  p ← 2; while p^2 ≤ N do
3  |  n ← p^2;
4  |  while n ≤ N do
5  |  |  c[N] ← False;
6  |  |  n ← p + n;
7  |  end
8  |  repeat
9  |  |  p ← p + 1;
10 |  until c[p] = True;
11 end
12 return c;
```

4.1.2 Trial division method

The simplest algorithm to factorize a given number is to proceed by attempts.
This algorithm tries to do it in the most efficient way possible, but it is still less efficient than the methods that will be proposed later. If the list of prime numbers up to

Algorithm 8: Trial-division method

Data: $N \in \mathbb{N}$, L the list of prime numbers up to $\sqrt{N}$
Result: c, the list of booleans that represents the prime integers up to N

```
1  c ← List(empty);
2  for m ∈ L do
3  |  if m|N then
4  |  |  c ←append(c, m);
   |  |  /*Appends the element m to the list c                    */
5  |  end
6  end
7  return c;
```

$\sqrt{N}$ is provided in input, it is necessary to compute, in the worst case, $\sqrt{N}$ reminders, each one at the cost of $\log^2(N)$.
Therefore, the cost of this method is $O(\sqrt{N}\log^2(N))$.

4.1.3 Fermat factoring method

This method aims to factorize a number N in two numbers p, q, such that $N = p \cdot q$. Also, there must be some x, y such that:

$$N = x^2 - y^2 = (x + y)(x - y)$$

If N is odd it can be shown that $y = \frac{N-1}{2}, x = \frac{N+1}{2}$. Let's now consider the cost of this

Algorithm 9: Fermat's factoring method

Data: $N \in \mathbb{N}$
Result: p, q
1 $y \leftarrow 1$;
2 **repeat**
3 $\quad x \leftarrow N + y^2$;
4 $\quad$ **if** $(\lfloor\sqrt{N + y^2}\rfloor)^2 = N + y^2$ **then**
5 $\quad$ **end**
$\quad$ `/*Checks if` $N + y^2$ `is a perfect square */`
6 $\quad$ **return** x, y $y \leftarrow y + 1$;
7 **until** $y = \frac{N-1}{2}$;
8 **return** $p = x + y, q = x - y$

algorithm:

- Each iteration has a cost of

$$O(\log^2(y) + \log^2(N + y^2) + \log^3(N + y^2)) = O(\log^3(N + y^2))$$

- The loop is repeated $O(\log^A(N))$ times.
- Therefore, the complexity of this algorithm is $O(\log^{A+3}(N))$ b.o..

4.1.4 Pollard's rho-method

The Pollard's ρ-method tries to factorize a number N, by attempting to find a collision when applying the same function multiple times. That can be summarized as follows:

- Let $F : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$.
- Let $x_0 \in \mathbb{Z}_N^*$ be a randomly chosen seed.

- Let $F^{(i)}(x_0) = F \circ F \circ \cdots \circ F(x_0)$.
- We want to find i, j such that $F^{(i)}(x_0) \equiv_N F^{(j)}(x_0)$, and compute $(N, F^{(i)}(x_0) - F^{(i)}(x_0))$

Let's now investigate why this algorithm is correct:

- Assume that p is prime and that $p|N$.
- Build a sequence of T numbers: $\{F_{x_0}\}_{k \leq T \in \mathbb{N}}$
- Assume that exists a collision, so:

$$\begin{aligned} \exists F^{(i)}(x_0) \equiv_p F^{(j)}(x_0) & \\ & \iff p|F^{(i)}(x_0) - F^{(j)}(x_0) \\ & \iff (p, F^{(i)}(x_0) - F^{(j)}(x_0)) > 1 \end{aligned}$$

- Due to the Birthday Paradox, we know that:

$$\begin{aligned} \mathbb{P}[\exists F^{(i)}(x_0) \equiv_p F^{(j)}(x_0)] = \frac{1}{2}, \text{ when } T \leq \sqrt{p} \\ \text{Since } p|N \implies p \leq \sqrt{N} \implies T = O(\sqrt[4]{N}) \end{aligned}$$

- Therefore, probably we will find the collision.

The cost of the algorithm is easy to compute:

- We have approximately $O(T^2) = O(\sqrt{N})$ steps, that is the quantity necessary to make the Birthday Paraodx hypothesis hold;
- Each one has a cost of $O(log^2(N))$ b.o.
- The expected cost is then $\mathrm{O}(\sqrt{(}N) log^2(N))$ b.o.

4.1.5 Pollard's $p-1$ method

This method is an alternative to the ρ-method. Consider what follows:

- Let N be an odd number.
- Choose $a \in \mathbb{Z}_N^*$.

Algorithm 10: Pollard's ρ-method

Data: $N, T \in \mathbb{N}, F : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$
Result: List of d, non-trivial factors of N

```
1  x_0 is randomly chosen in Z*_N;
2  m ← 1;
3  y_1 ← F(x_0);
4  y_2 ← F(y_1);
5  while m ≤ T do
6  |  d ← (N, y_1 − y_2);
7  |  if d > 1 ∧ d < N then
8  |  |  return d
9  |  end
10 |  m ← m +1;
11 |  y_1 ← F(y_1);
12 |  y_2 ← F(F(y_2));
13 end
```

- Recall that due to the Little Fermat Theorem (**??**):

$$p \text{ is prime } \iff a^k \equiv_p 1$$

  When k is a multiple of $p-1$.

- Therefore:

$$p|(a^k - 1, N)$$

- So, we can assume that if $p|N$, then:

$$p - 1 = \prod_q q^\alpha$$

  Where q is prime, and α is a generic exponent.

- Also, we assume that:

$$\exists B \in \mathbb{N} : p|N \wedge (q^\alpha | p-1 \implies q^\alpha \leq B)$$

The algorithm 11 resumes the method proposed by Pollard.
The cost of this algorithm can be estimated as follows:

- Step 2 costs $O(B^2 \log^2(B))$;
- Step 3 costs $O(\log^2(N))$;

- Step 4 costs $O(\log^2(N)\log(k))$;
- The total cost is then $O(B^2\log^2(B))$ per attempt.

Algorithm 11: Pollard's $p-1$ method

Data: $N \in \mathbb{N}$
Result: List of d, non-trivial factors of N

```
1 B is randomly chosen;
2 k ← B!;
3 a is randomly chosen in Z*_N;
4 d ← (N, a^k − 1);
5 if d = 1 ∨ d = N then
6 |   GoTo 3 OR;
7 |   GoTo 1
8 end
```

4.1.6 Pomerance's quadratic sieve

This algorithm is subexponential, and performs the factorization of a given number in $O(\exp(c_1\sqrt{\log(N)\log(\log(N))}))$ b.o.

Kraithcik's idea

This algorithm follows the Kraithcik's idea, that can be seen as a generalization of the Fermat's Factoring method. It's summerized as follows:

- Find a sequence of congruences:

  $$A_i \equiv_N B_i$$

  Where $A_i \neq B_i$.
- Find the complete factorization for a large enough number of A_i, B_i.
- Find a set S of congruences built starting from the ones found previously, such that:

  $$\prod_{i\in S'} A_i \equiv_N X^2$$
  $$\prod_{i\in S} B_i \equiv_N Y^2$$

  And therefore $X^2 \equiv_N Y^2$

- Compute:
$$d = (X - Y, N)$$
Since we know the factorization of A_i (and B_i), we have that:
$$A_i = \prod_{p_j \leq B} p^{\alpha_{ij}}, \alpha_{ij} \in \mathbb{N}$$
$$\therefore A_1 \cdot A_2 = \prod_{p_j \in B} p^{\alpha_{1j}+\alpha_{2j}}$$
$$A_1 \cdot A_2 \text{ is a square } \iff \alpha_{1j} + \alpha_{2j} \text{ is even}$$
We can know map this problem to a linear system, where each A_i can be represented as a vector, in which each element is the exponent $\alpha_{i,j} \bmod 2$.
We now have a matrix of $k = \pi(B)$ columns and $T = |S|$ rows, and we want to solve the system $\lambda \cdot v_2(A) = 0$.

The algorithm

In order to better understand the algorithm, the following clarification are needed:

- Let $A_j \in \mathbb{N}$ s.t.:
$$Q(A_j) = \prod_{p \in \mathbb{B}} p^{\alpha_{p,j}}$$
Where $\alpha_{p,j} \in \mathbb{N} \wedge k = |\mathbb{B}|$
- Let $(A_j) = (\alpha_{2,j}, \alpha_{3,j}, \ldots, \alpha_{p,j})$
- Let $_2(A_j) = (\alpha_{2,j} \bmod 2, \alpha_{3,j} \bmod 2, \ldots, \alpha_{p,j} \bmod 2)$

This algorithm has a cost of $O(\exp(c\sqrt{\log(N)\log(\log(N))}))$

Algorithm 12: Pomerance's quadratic sieve

Data: $N, B \in \mathbb{N}$
Result: List of d, non-trivial factors of N

1 $s \leftarrow \lfloor \sqrt{N} \rfloor$;
2 Pick randomly $A \in \mathbb{N}$;
3 $Q(A) \leftarrow (A + s)^2 + N$;
4 $\mathbb{B} \leftarrow \{2\} \cup \{p : p \text{ is prime } \wedge\, p \leq B \wedge p | Q(A)\}$;
5 $X \leftarrow \sqrt{\prod_{i=1}^{m} (A_i + s)^2 \bmod N}$;
6 $Y \leftarrow \sqrt{\prod_{i=1}^{m} Q(A_i) \bmod N}$;
7 $d \leftarrow (X - Y, N)$;

Chapter 5

Attacks

5.1 Attacks' principles

5.1.1 Birthday Paradox

This Paradox is not an actual one, but is the result of a cognitive bias.
The idea is that in a room full of people, the probability of having no one with matching birthdays is actually lower than expected.
Consider what follows:

- Let n be the number of possible birthdays
- Let N be the number of people.
- Let S be a space $\{1,\dots,n\}^N \ni (c_1,\dots,c_N)$.
- The probability of a given combination of birthdays to happen is then:

$$\mathbb{P}[(c_1,\dots,c_N)] = n^{-N} = \frac{1}{|S|}$$

- Let's now compute the probability of having **no collisions**:

$$\begin{aligned} p(N) &= \mathbb{P}[\forall\, i \neq j : c_i \neq c_j] \\ &= \frac{\prod_{i=0}^{N-1} n-i}{n^N} \\ &= \frac{n\cdot(n-1)\cdot\dots\cdot(n-N+1)}{n\cdot n\cdot\dots\cdot n} \\ &= \prod_{i=0}^{N-1} 1-\frac{i}{n} \end{aligned}$$

- Let'now compute the probability of having **at least one collision**:

$$q(N) = 1 - p(N) \geq 1 - \epsilon$$
$$\iff p(N) \leq \epsilon$$

- Let's now consider that:

$$\begin{aligned} p(N) = \prod_{i=0}^{N-1} 1 - \frac{i}{n} &\leq \prod_{i=0}^{N-1} e^{-\frac{i}{n}} \\ &= e^{\sum_{i=0}^{N-1} -\frac{i}{n}} \\ &= e^{-\frac{1}{n}\sum_{i=0}^{N-1} i} \\ &= e^{-\frac{(N)(N-1)}{2n}} \leq \epsilon \end{aligned}$$

  Therefore,

$$N \geq \frac{1 + \sqrt{1 - 8n\log(\epsilon)}}{2} \simeq \sqrt{n|\log(\epsilon)|}$$

- So, when $N \simeq n$, then $\epsilon \simeq \frac{1}{2}$

5.2 Attacks to RSA

5.2.1 Chosen-ciphertext/Known-plaintext attack

This attacks can happen when the attacker knows a portion or the complete plaintext m associated to the ciphertext c.
The attacks proceeds as follows:

- The attacker E generates a random number $r \in \mathbb{Z}_{n_A}^*$, so $(r, n_A) = 1$. This quantity is called the "blinding factor".
- E asks A to sign the quantity

$$r^{c_A} \cdot c \bmod n_A$$

  In this way, $r^{c_A} \cdot c$ acts like a random message.

- If A signs $r^{c_A} \cdot c$, it gets:

$$\begin{aligned} f_A^{-1}(r^{c_A} \cdot c) &= (r^{c_A} \cdot c)^{d_A} \\ &= r^{c_A d_A} \cdot c^{d_A} \bmod n_A \\ &= r \cdot m \bmod n_A \end{aligned}$$

  Since $r \cdot m$ looks random, it's hard to identify the attack here.

- Now that E has the quantity $r \cdot m \bmod n_A$, and since he produced r, he knows $r^{-1} \bmod n_A$.
  Therefore, he can compute:
$$r^{-1} \cdot r \cdot m \bmod n_A = m$$

5.2.2 Random fault attack

This attack on RSA can be done when a signature is computed incorrectly.
Recall that the digital signature with RSA works as follows.
Alice computes:
$$s = m^d (\bmod n)$$
Bob, to verify the signature, computes:
$$s^e = m (\bmod n)$$
Where:

- s is the signature;
- m is the cleartext message;
- d is the private key (exponent);
- n is the public key (modulus);
- e is the public key (exponent).

This attack exploits an optimization that is commonly used to compute the signature. Since n is the product of two big prime numbers p, q, the CRT (Chinese Reminder Theorem) can be used to compute two smaller modulus (p, q respectivly), instead of a bigger one.
$$\begin{cases} s_1 \equiv m^{d_p} (\bmod p) \\ s_2 \equiv m^{d_q} (\bmod q) \end{cases} \implies s (\bmod pq)$$
Consider now the case when s_2 is uncorrectly computed:
$$\begin{cases} s_1 \equiv m^{d_p} (\bmod p) \\ \hat{s_2} \equiv m^{d_q} (\bmod q) \end{cases} \implies \hat{s} (\bmod pq)$$
Then we have that:
$$\begin{cases} s^e \equiv m (\bmod p) \\ \hat{s^e} \not\equiv m (\bmod q) \end{cases} \implies \begin{cases} s^e - m \equiv 0 (\bmod p) \\ \hat{s^e} - m \not\equiv 0 (\bmod q) \end{cases} \implies \begin{cases} p | s^e - m \\ q \nmid \hat{s^e} - m \end{cases}$$
Therefore, exists some k such that:
$$p \cdot k = \hat{s^e} - m$$
Therefore, we can obtain easily p by computing $gcd(s^e - m, N)$.

5.2.3 RSA basic attack

This attack is conducted when Alice sends to Bob1 and Bob2 the same message, by using the same public key (modulus). Assume what follows:

- $$n_X = n_Y$$
- $$(e_X, e_Y) = 1$$

Therefore, an intruder can compute r, s such that:

$$r \cdot e_X + s \cdot e_Y = 1$$

By using the *Extended Euclidean Algorithm* (recall that e_X, e_Y are public). Since e_X, e_Y are positive, then one between r, s must be negative. Then:

- The intruder computes $c_1^{-1} \bmod n_X$, using the *E.E.A.*.
- The intruder then computes:

$$(c_1^{-1})^{-r} \cdot c_2^s = (M^{-e_X})^{-r} \cdot M^{e_Y \cdot s} \qquad = M^{r \cdot e_X + s \cdot e_Y} \bmod n_X = M \bmod n_X$$

Therefore, if different public keys aren't used, an intruder can easily intercept a broadcasted message.

5.2.4 Broadcast attack

In this attack, the user broadcasts a message by using different values for each receiver ($\forall i \neq j : n_i \neq n_j$), but uses the same value of e for everyone. We'll assume that $r \geq e$.
The attack is conducted as follows:

- The intruder collects $C_1, C_2, \ldots, C_r$, the cyhpered texts.
- The intruder uses a subset of the cyphered texts $D \subset \{C_1, C_2, \ldots, C_r\}$. Consider $|D| = e$.
- The intruder solves the system of equations

$$C \equiv_n C_i, \forall i = 1, 2, \ldots, e$$

- C is the unique solution to the system, due to the Chinese Reminder Theorem.

- Consider that:

$$\begin{aligned} M \in \mathbb{Z}_{n_i}^* \forall i = 1,2,\ldots,e \implies \\ M < n_i \forall i = 1,2,\ldots,e \implies \\ M^2 < n_1 \cdot n_2 \implies \\ M^e < \prod_{i=1}^{e} n_i = N \implies \\ M^e = C \iff M = C^{\frac{1}{e}} \end{aligned}$$

  Therefore, finding the solution in modulus N is equivalent to find the solution to the system.

In order to prevent this kind of attack, when using RSA in broadcasts is important to use different values for e, n for each user.

5.3 Attacks to Block Ciphers

5.3.1 Known plaintext attack to an Affine block cipher

Consider the following cryptosystem:

$$\begin{aligned} \Sigma = \mathbb{Z}_N \mathscr{M} = \mathscr{C} = \mathbb{Z}_N^{\updownarrow} \\ K_E = (A,b), K_D = (A^{-1},b) \\ \{ : \mathbb{Z}_N^{\updownarrow} \to \mathbb{Z}_N^{\updownarrow} \\ m \to (A \cdot m + b) \bmod N \\ \{^{-1} : \mathbb{Z}_N^{\updownarrow} \to \mathbb{Z}_N^{\updownarrow} \\ c \to A^{-1}(c-b) \bmod N \end{aligned}$$

Where:

- $N > 0 \in \mathbb{N}$;
- $\updownarrow \geq 1 \in \mathbb{Z}$;
- A is an invertible $\updownarrow \times \updownarrow$ matrix, and $\forall 1 \leq i,j \leq \updownarrow : A_{i,j} \in \mathbb{Z}$.
- $b \in \mathbb{Z}_N^{\updownarrow}$

Note that:

- When $\updownarrow = 1$, you get the Caesar's method;
- When $A = I^1$ and $b \in \mathbb{Z}_N^{\updownarrow}$ you get the Vigenére's method.

The attacks is conducted as follows:

1. The intruder generates $\updownarrow + 1$ plaintexts $(p_0, \ldots, p_{\updownarrow})$;
2. The intruder collects the corrispondent ciphertexts $(c_0, \ldots, c_{\updownarrow})$; Consider that:

$$c_i \equiv A \cdot p_i + b \bmod N$$

3. Now, consider that:

$$c_i - c_0 \equiv A \cdot p_i + b - A \cdot p_0 - b \bmod N$$

4. Then we have to matrixes:

$$C = (c_1 - c_0, \ldots, c_{\updownarrow} - c_0), P = (p_1 - p_0, \ldots, p_{\updownarrow} - p_0)$$

5. So, we have that:

$$C \equiv A \cdot P \bmod N$$

6. If $(detP, N) = 1 \implies P$ is invertible, therefore:

$$C \cdot P^{-1} \equiv A \cdot P \cdot P^{-1} \bmod N$$
$$A \equiv C \cdot P^{-1} \bmod N \wedge b \equiv c_0 - A \cdot p_0 \bmod N$$

7. Then, fixed $p_0 \in \mathbb{Z}_N^{\updownarrow}$, let v_i be a vector such that $v_i = (0, 0, \ldots, 1, \ldots, 0)$, where the 1 is in the i-th position.
8. Now it's clear that $p_i = p_0 + v_i \bmod N$.
9. Therefore, $P = I_{\updownarrow} = P^{-1}$.
10. $\implies C = A \cdot P \bmod N = A$

[1] The identity matrix.

Chapter 6

Digital Signature

6.1 Problem definition

Essentially, the digital signature is an identity problem.
The public signature is a cryptosystem (M, C, K, f, f^{-1}), where:

- M is the set of possible cleartext messages;
- C is the set of possible cyphered texts;
- K is the set of possible keys;
- f is the encyphering function;
- f^{-1} is the decyphering function.

6.1.1 Main concept

The functioning of the digital signature is the following:

- Let's assume that the comunication is between two users, A and B.
- A computes $f_B(m)$, where $m \in M$;
- A computes $f_A^{-1}(A^n)$, where A^n is A's nickname;
- A sends $[f_B(m), f_B(f_A^{-1}(A^n))]$;
- B verifies that $f_B^{-1}(f_B(m)) = m$;
- B verifies that $f_B^{-1}(f_B(f_A^{-1}(A^n))) = f_A^{-1}(A^n)$.

Since f_A is the unique function that can invert f_A^{-1}, B can verify that the message m is signed from A (actually its nickname).
It's important that f is chosen in such a way that computing f^{-1} is computationally hard without the key.
In this case, f_A **is public** and f_B **is private**.

6.1.2 Checking the message's integrity

An alternative way is to substitute the message at step 6.1.1 with the following:

$$m_1, m_2 \leftarrow [f_B(m), f_B(f_A^{-1}(m))]$$

Doing that, B can verify the following identity:

$$f_B^{-1}(f_B(m)) = m_1 \wedge f_A(f_B^{-1}(f_B(f_A^{-1}(m)))) = m_2$$

In this case, **both** f_A **and** f_B **are public**.

6.2 Hash functions

Definition 1 (Ideal Hash Function)**.** *An **Ideal Hash Function** is a function*

$$h : M \to \{0,1\}^k$$

That verifies the following properties:

- $$m \neq m' \implies h(m) \neq h(m')$$
- $k \in \mathbb{N}$ *and is large enough.*

The goal is to produce $h(m)$ in a such way that a unique input produces a unique output. Since length(m) is not fixed,this is clearly unfeasible, due to the Pigeon-boxes's principles. Although, it is possible to relax this constraint in the following way:

Definition 2 (Hash Function)**.** *An **Hash Function** is a function*

$$h : M \to \{0,1\}^k$$

That verifies the following properties:

- $$\mathbb{P}[h(m) = h(m') : m \neq m'] < 2^{-60}$$
- $k \in \mathbb{N}$ *and is large enough.*

6.2.1 Digital signature with "partial" message integrity check

An alternative way is to substitute the message at step 6.1.1 with the following:

$$m_1, m_2 \leftarrow [f_B(m), f_B(f_A^{-1}(h(m)))]$$

Doing that, B can verify the following identity:

$$f_B^{-1}(f_B(m)) = m_1 \wedge f_A(f_B^{-1}(f_B(f_A^{-1}(h(m))))) = f_A(f_A^{-1}(h(m)) = h(m)$$

In this case, f_A **is private** and f_B **is public**.
Since the length of $h(m)$ is fixed, the message is generally smaller, and this can improve the speed of the algorithm.

Digital signature based on RSA

This kind of digital signature is based on the following cryptosystem:

- For the user A:
$$M = C = Z_{n_A}^*$$
Where $n_A = p_A \cdot q_A$, and n_A is **public**. Also,
$$f_A(m) = m^{c_A} \bmod n_A, f_A^{-1}(c) = c^{d_A} \bmod n_A$$

- For the user B:
$$M = C = Z_{n_B}^*$$
Where $n_B = p_B \cdot q_B$, and n_B is **public**. Also,
$$f_B(m) = m^{c_B} \bmod n_B, f_B^{-1}(c) = c^{d_B} \bmod n_B$$

- It's important that $n_A \neq n_B$.

The functioning of this system is the following:

- Let s_A be the "nickname" of A.
- A computes $f_A^{-1}(f_B(s_A \bmod n_B))$. The module is applied because it can happen that if $n_A \geq n_B$ the identity could not be verified.
- B verifies that
$$f_B^{-1}(f_A(f_A^{-1}(f_B(s_A \bmod n_B)))) = s_A$$

Chapter 7

Cryptography Principles

7.1 Cryptosystems

7.1.1 Definitions

Definition 3 (Cryptographic transformation)**.** *A cryptographic transformation is defined as* ***any injective function*** *such that:*

$$: \mathscr{M} \rightarrow \mathscr{C}$$

Definition 4 (Enciphering and Deciphering functions)**.** *An* ***Enciphering function*** *is a cryptographic transformation. Its inverse* $^{-1}$ *is called* ***Deciphering function*** *and is defined as:*

$$^{-1} : \mathscr{C} \rightarrow \mathscr{M}$$

Definition 5 (Cryptosystem)**.** *A* ***Cryptosystem*** *is a tuple* $(\mathscr{M}, \mathscr{C}, \mathscr{K}, {}_{k_e}, {}^{-1}_{k_d})$*, where:*

- $\mathscr{M}$ *is the space of clear-text messages;*
- $\mathscr{C}$ *is the space of ciphered messages;*
- $\mathscr{K}$ *is the space of the keys;*
- ${}_{k_e}$ *is the enciphering function with key* k_e*;*
- ${}^{-1}_{k_d}$ *is the deciphering function with key* k_d*.*

Chapter 8

Asymmetric Algorithms

8.1 RSA

This section is dedicate to the entire RSA cryptosystem.

8.1.1 RSA cryptosystem

Definition 6 (RSA cryptosystem)**.** *The RSA cryptosystem is defined as* $(\mathcal{M}, \mathcal{C}, \mathcal{K}, {}_{k_e}, {}^{-1}_{k_d})$*, where:*

- $\mathcal{M} = \mathcal{C} = \mathbb{Z}_n^*$;
- $\mathcal{K} = \mathbb{Z}^*_{\varphi(n)}$
- ${}_A(x) = x^e \bmod n$
- ${}^{-1}_A(y) = y^d \bmod n$

The meaning of the values A, e, n will be explained in the following paragraph.

8.1.2 How RSA works

Initialization

In order to communicate with the RSA cryptosystem, each user A must follow these steps:

1. Pick $p, q \in \mathbb{Z} : p \neq q$, two large prime numbers.
2. Compute $n = p \cdot q$;
3. Compute $\varphi(n) = (p-1)(q-1)$;

4. Pick $e \in \mathbb{N} : (e, \varphi(n)) = 1$
5. Compute $d \in \mathbb{Z}^*_{\varphi(n)} : e \cdot d \equiv_{\varphi(n)} 1$;
6. Then, (e, n) will be A's public key;
7. (p, q, d) will then be his private key.

1 on 1 Communication

If the user A wants to communicate to B, it must encrypt its message with B's public key. Assume that M is the clear-text message and that C is the enciphered message. Then, A computes as follows:

$$C = M^{e_B} \bmod n_B$$

In order to read the ciphered message, B has to decipher C by using the deciphering function with his private key. B computes as follows:

$$\begin{aligned} C^{d_B} \bmod n_B &= (M^{e_B})^{d_B} \bmod n_B \\ &= M^{e_B \cdot d_B} \bmod n_B \\ &= M \bmod n_B \\ &= M \end{aligned}$$

This works because e_B and d_B are picked in such a way that they're each other's inverse in $\mathbb{Z}^*_{\varphi(n)}$, and therefore:

$$e_B \cdot d_B \equiv_{n_B} 1$$

Also, since n_B is supposed to be a really large number, is safe to assume that $M \leq n_b$, therefore:

$$M \bmod n_B = M$$

8.1.3 Broadcast communications with RSA

In order to communicate with multiple subjects, each user has just to repeat the Initialization and Communication procedure described before, one time for each other node in the network.
It's important that the user does not use the same public keys multiple times, in order to prevent Broadcast attacks.

Chapter 9

Block Ciphers

9.1 Definition

Given:

- $\updownarrow$, the length of the block;
- Σ, the alphabet;

A block cipher is a cryptosystem such that:

$$M = \mathscr{C} = \Sigma^{\updownarrow}$$

Proposition 1. *Consider waht follows:*
The enciphering function of a block cypher is a permutation of $\Sigma^{\updownarrow}$.

Proof. The proof proceeds as follows:

- Let $\{ : \Sigma^{\updownarrow} \to \Sigma^{\updownarrow}$.
- Let $\{(m) = \{([p_1, p_2, \ldots, p_e]) = \Pi(p_1, p_2, \ldots, p_e)$, where $p_i \in \Sigma$.
- Then, let $\{^{-1}(c) = \{([c_1, c_2, \ldots, c_e]) = \Pi(c_1, c_2, \ldots, c_e)$, where $c_i \in \Sigma$.
- Let then $K_E = \Pi$ be the set of encyphering keys;
- Let then $K_D = \Pi^{-1}$ be the set of decyphering keys;
- Then, $|K| = (|\Sigma|^{\updownarrow})!$
- Due to the Pigeonhole principle, if $\{$ is injective, then it's also surjective, and therefore a bijection.

□

9.2 Transimission methods

Since block cyphers encrypt just message of length $\updownarrow$, it's necessary two think of a way to send longer messages through a communication device.

9.2.1 Electronic Code Book mode (ECB)

Consider what follows:

- Let $M \in \Sigma^t$ be the message, and $t = |M|$.
- Let $\updownarrow$ be the length of the block, such that $\updownarrow \nmid t \wedge t > \updownarrow$.
- M is splited in blocks of length $\updownarrow$, and the remaining part is filled with garbage. This concatenation is called M'.
- M' is sent to the receiver.
- If M' is received in the correct order, the message is received correctly.

9.2.2 Cipher Block Chaining mode (CBC)

This method introduces a XOR encryption to the communication.
Consider what follows:

- Let $\Sigma = \{0,1\}, M, C \in \{0,1\}^{\updownarrow}$
- Let $\oplus$ be the XOR operator (as known as the sum in $\mathbb{Z}_2$).
- Let $P \in \{0,1\}^{\updownarrow}$ the initial fixed plaintext, pre-agreed.
- Assume that $|M| = k \cdot \updownarrow$.
- Alice sends the ciphered message $C = [c_0, c_1, \ldots, c_k]$ to Bob. That is:

$$\begin{cases} c_0 & = \{(P) \\ c_j & = \{(c_{j-1} \oplus m_j), \text{for } 1 \leq j \leq k \end{cases}$$

- In order to receive correctly the message, it's important that Bob receives correctly c_0 and computes $\{^{-1}(c_0)$. If that matches the original P, then he can proceed by computing the other blocks:

$$m_j = c_{j-1} \oplus \{^{-1}(c_j), 1 \leq j \leq k$$

This works because:

$$\begin{aligned} c_1 = \{(c_0 \oplus m_i) \wedge c_0 = \{(c_0) \\ c_0 \oplus \{^{-1}(c_1) = c_0 \oplus \{^{-1}(\{(c_{0 \oplus m_1})) \\ = c_0 \oplus c_0 \oplus m_1 = m_1 \end{aligned}$$

9.2.3 Cipher Feedback mode (CFB)

Let the common knowledge of the communication be:

- P an initial value;
- $\updownarrow$, the length of the block.
- $1 \leq r \leq \updownarrow$ be a number.

Then, let $M = [m_1, \ldots, m_t]$ be the cleartext message, where:

$$|m_i| = r \implies r||M|$$

The communication is executed as follows:

Algorithm 13: Cipher FeedBack Mode communication (CFB) [Sender]

$I_1 \leftarrow P \in \{0,1\}^{\updownarrow}$;
for $j \in 1, \ldots, t$ **do**
  $O_j \leftarrow \{(I_j)$;
  $u_j \leftarrow O_j \bmod 2^r$;
  $c_j \leftarrow m_j \oplus u_j$;
  $I_{j+1} \leftarrow 2^r I_j + c_j \bmod 2^{\updownarrow}$;
end
return C

Algorithm 14: Cipher FeedBack Mode communication (CFB) [Receiver]

$I_1 \leftarrow P$;
for $j \in 1, \ldots, t$ **do**
    $O_j \leftarrow \{(I_j)$;
    $u_j \leftarrow O_j \bmod 2^r$;
    $m_j \leftarrow c_j \oplus u_j$;
    $I_{j+1} \leftarrow 2^r I_j + c_j \bmod 2^{\updownarrow}$;
end
return C

9.2.4 Output Feedback mode (OFB)

9.3 Feistel's Ciphers

The Feistel's cipher cryptosystem is the predecessor of the AES. It is defined as follows:

$$\begin{aligned} \Sigma &= \{0,1\} \\ \mathscr{M} = \mathscr{C} = \mathscr{K} &= \Sigma^{\updownarrow} \\ \{_k &: \Sigma^{\updownarrow} \to \Sigma^{\updownarrow} \\ \mathscr{F}_k &: \Sigma^{2\updownarrow} \to \Sigma^{2\updownarrow} \end{aligned}$$

This cipher loops the enciphering function F r times, in which the blocks have a lenght of $2 \cdot \updownarrow$. f is a sort of internal enciphering function.
There's also a key generating function, that has the following signature:

$$\mathscr{K} \to \mathscr{K}^r$$

The algorithm enciphers as follows:

1. The cleartext message M is composed of two parts: L_0, R_0, where $|L_0| = |R_0| = \updownarrow$.
2. Consider that K_i is the i-th key produced.
3. For every $i : 1 \leq i \leq r$, C_i is computed:

$$C_i = [L_i, R_i] = [R_{i-1}, L_{i-1} \oplus f_{K_i}(R_{i-1})$$

4. The message that is sent is $F_{K_r} = [R_r, L_r]$

The deciphering method proceeds as follows:

1. Note that the keys must be used in reverse order.
2. At each step, it is computed:

$$R_i \leftarrow L_{i-1} \oplus f_{K_i}(R_{i-1}), L_i \leftarrow R_{i-1}$$

3. Also, note that $f_{K_i} = f_{K_i}^{-1}$, since it's used with the XOR operator.

An important remark is that the complexity of this algorithm depends on f_K

Useful Facts

- The GMP library is a free library for arbitrary precision arithmetic. It implements all the basic arithmetic operations with the maximum efficency possible.

List of Algorithms

List of Theorems

List of Lemmas