Final Project of the course Discrete Mathematics
Rabin Cryptosystem Implementation

Genalti Gianmarco, 952905

Politecnico di Milano

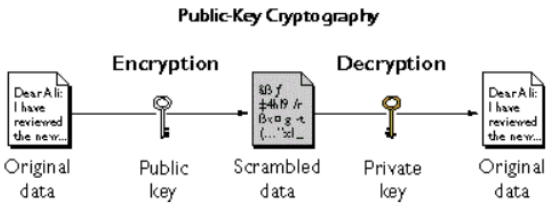
Academic Year 2019-2020

Abstract

The aim of this project is to perform an in-depth analysis of Rabin Cryptosystem , then implement through MATLAB© its fundamental features and a program which takes an image as an input and can perform both encryption and decryption with such system.

1 Public Key Cryptography

Public Key Cryptography is a cryptosystem which relies on the existence of two different keys: the Public Key, which only permits encryption and can be widely disseminated; the Private Key, which permits the decryption of an information encrypted with the Public one. Any person can encrypt a message using the Public Key, but that encrypted message can only be decrypted with the receiver's Private Key.



2 Rabin Cryptosystem Overview

Rabin Cryptosystem is a Public Key Cryptosystem which relies on a homonymous one-way function: for one-way function, we intend a function which is easy to compute given an input but computationally hard to invert from a random output; we'll prove that reversing such function is equivalent to factorize prime numbers. Indeed, given two prime numbers p and q, being b the total number of bits of our numbers, a complexity of $O(b^2)$ is required to compute $N = pq$. But the best factoring algorithm known run in $O(exp((\frac{64b}{9})^{\frac{1}{3}} log(b)^{\frac{2}{3}}))$ time, a non-polynomial time. In particular we'll use as Public Key N, while the Private Key will be (p, q).

3 Mathematical Context

To run the Rabin algorithm we'll heavily rely on modular arithmetic. Let's put some theoretical fundamentals:

Definition 1. *Let $n \in \mathbb{N}$ be a natural number. We say that a,$b \in \mathbb{Z}$ are equal modulo n and we write $a \equiv b \mod n$ if n divides $b - a$.*

Proposition 1. *Equality modulo n is an equivalence relation.*

Let's try to characterize this relation in a more practical way:

Theorem 1. *$a, b \in \mathbb{Z}$ are equal modulo n if and only if a, b have the same remainder when divided by n.*

This result allows us to identify the equivalence classes modulo n in a quick way, given a possible remainder r, we can write:

$$[r]_n = \{r + hn \mid h \in \mathbb{Z}\}$$

Finally:

Definition 2. *The quotient set of $\mathbb{Z}$ modulo equality modulo n is the set of integers modulo n and it is written as $\mathbb{Z}_n$.*

We can define two inner operations analogous to sum and product over the integers:

Definition 3. *In $\mathbb{Z}_n$, the sum is defined as $[r]_n + [r']_n = [r + r']_n$ while the product is defined as $[r]_n[r']_n = [r + r']_n$*

And follows:

Definition 4. *$x \in \mathbb{Z}_n$ is invertible if there exists $y \in \mathbb{Z}_n$ such that $xy \equiv 1$ mod n. In such a case y is the inverse of x and it is denoted as x^{-1}. The set of invertible elements in $\mathbb{Z}_n$ is named $\mathbb{Z}_n^*$.*

As it often happens in mathematics, a particular role is played by prime numbers:

Theorem 2. *If p is a prime number, then every non-zero element of $\mathbb{Z}_p$ is invertible.*

3.1 Main Results

Let a, b be positive integers, let's call $GCD(a, b)$ the Greatest Common Divisor between a and b.
In order to perform the Rabin algorithm, we'll be required to solve a particular type of linear equation, a crucial result comes from Integers Theory:

Theorem 3 (Euclidean Algorithm)**.** *Let a, b be positive integers. Set $r_0 = a, r_1 = b$, and iteratively set r_n to be the remainder and q_{n-1} to be the quotient of $r_{n-2} : r_{n-1}$, if $r_{n-1} \neq 0$. Then, there exists j such that $r_j = 0$ and $r_{j-1} = GCD(a, b)$.*

The so called Bézout coefficients (r_{j-1}, s_{j-1}) are integers such that $ar_{j-1} + bs_{j-1} = GCD(a, b)$. They can be calculated iteratively by extending the prevoius algorithm:

Theorem 4 (Extended Euclidean Algorithm)**.** *Initialize $s_0 = 1, t_0 = 0$ and $s_1 = 0, t_1 = 1$, then at each step of the classical Euclidean Algorithm calculate $r_n = r_{n-2} - q_{n-1}r_{n-1}$ and $s_n = s_{n-2} - q_{n-1}s_{n-1}$, at the end of the algorithm you'll get the Bézout coefficients.*

```
function [ya,yb,GCD] = euclid2(a,b)
%EUCLID
%Extended Euclid's algorithm in dimension 2. Given integers p ...
    and q,
%returns the greatest common divisor and the coefficients yp and ...
    yq s.t
%yp*p + yq*q = GCD(p,q)
r(1) = max(a,b);
r(2) = min(a,b);
s(1) = 1;
s(2) = 0;
t(1) = 0;
t(2) = 1;
```

```
12  j = 2;
13  while r(j) ≠ 0
14      r(j+1) = mod(r(j−1),r(j));
15      q(j) = (r(j−1)−r(j+1))/r(j);
16      s(j+1) = s(j−1)−q(j)*s(j);
17      t(j+1) = t(j−1)−q(j)*t(j);
18
19      j = j+1;
20  end
21  % disp(r)
22  % disp(s)
23  % disp(t)
24  GCD = r(j−1);
25  if max(a,b)== a
26      ya = s(j−1);
27      yb = t(j−1);
28  else
29      ya = t(j−1);
30      yb = s(j−1);
31  end
32
33  end
```

We'll work in particular over the fields $\mathbb{Z}_p$ with p prime number. Such fields possess unique features:

Theorem 5 (Euler's Theorem)**.** *For every $x \in \mathbb{Z}_n^*$, it holds $x^{\phi(n)} \equiv 1$ mod n. ϕ is Euler's function.*

We'll exploit the fact the for a prime number p we have $\phi(p) = p - 1$.

Theorem 6 (Chinese Remainder Theorem)**.** *Let $n_1, ..., n_k$ be pairwise coprime positive integers, and let $b_1, ..., b_k$ be integers. Then, the linear system:*

$$\begin{cases} x \equiv b_1, & mod\ n_1 \\ : & \\ x \equiv b_k, & mod\ n_k \end{cases}$$

has only one solution in $\mathbb{Z}_N$, where $N = n_1 \cdots n_k$

```
1   function xm = crt(c1,c2,a,b)
2   %Chinese Remainder Theorem
3   %Applies the chinese remainder theorem over the system
4   %x = c1 (mod a)
5   %x = c2 (mod b)
6   %given c1, c2, a, b and returns the solution xm
7   [ya,yb,GCD]=euclid2(a,b);
8   if(GCD ≠ 1)
9       error("a and b are not coprimes!")
10  end
```

```
11  n = a*b;
12  xm = mod(a*ya*c2 + b*yb*c1,n);
13  end
```

4 Rabin Algorithm's Scheme

4.1 Keys Generation

1. We need to choose a private key (p, q). These two numbers are required to be prime numbers such that $p \equiv 3 \mod 4$ and $p \equiv 3 \mod 4$ (Gaussian Primes).

2. We generate the public key $n = pq$ and spread it.

4.2 Encryption

3. We convert our message M to a number $m < n$ in a reversible way, and we create the cyphertext $c \equiv m^2 \mod n$

```
1  function c = encrypter(n,m)
2  %ENCRYPTER
3  %   Given two primes p and q s.t. p=3 (mod 4) q=3 (mod 4) and a ...
       message m
4  %   integer returns the cypher message c encrypted with Rabin ...
       algorithm
5  c = mod(m^2,n);
6  end
```

4.3 Decryption

4. We calculate the square roots of c modulo p and q, respectively:

$$m_p = c^{\frac{1}{4}(p+1)} \mod p$$
$$m_q = c^{\frac{1}{4}(q+1)} \mod q$$

Note that $\frac{1}{4}(p+1)$ is an integer number, due to the fact that 4 divides $(p+1)$ since p is a Gaussian prime. I'll prove that m_p (analogously m_q) is actually a square root of c modulo p (q respectively):

$$c \equiv m^2 \bmod pq \implies \mathrm{c} \equiv m^2 \bmod p \pmod q$$
$$m_p^2 \equiv c^{\frac{1}{2}(p+1)} \equiv c \cdot c^{\frac{1}{2}(p-1)} \equiv c \bmod p \text{ (analogous for } q)$$

The last step follows from Thm.5 in the special case of p prime.

5. Using Euclidean Extended Algorithm (Thm.4) we compute y_p and y_q such that $y_p p + y_q q = 1$ (obviously $GCD(p, q) = 1$ since they are both primes).

6. Through Chinese Remainder Theorem (Thm.6) we find the **four** square roots of c modulo n, solving the following systems:

$$\begin{cases} \mathrm{x} \equiv m_p, & \bmod p \\ \mathrm{x} \equiv m_q, & \bmod q \end{cases} \qquad \begin{cases} \mathrm{x} \equiv -m_p, & \bmod p \\ \mathrm{x} \equiv m_q, & \bmod q \end{cases}$$
$$\begin{cases} \mathrm{x} \equiv m_p, & \bmod p \\ \mathrm{x} \equiv -m_q, & \bmod q \end{cases} \qquad \begin{cases} \mathrm{x} \equiv -m_p, & \bmod p \\ \mathrm{x} \equiv -m_q, & \bmod q \end{cases}$$

Note that, given m_1 and m_3 solutions of the upper two systems, from Thm.6 proof follows their calculation as:

$$\begin{aligned} m_1 &= p y_p m_q + q y_q m_p \bmod n \\ m_3 &= p y_p m_q - q y_q m_p \bmod n \end{aligned}$$

But then we can find a correspondence with the lower two systems' solution:

$$\begin{aligned} n - m_1 &\equiv -p y_p m_q - q y_q m_p \bmod n \\ n - m_3 &\equiv -p y_p m_q + q y_q m_p \bmod n \end{aligned}$$

So the lower two systems' solutions can be easily computed as:

$$\begin{aligned} m_4 &= n - m_1 \\ m_2 &= n - m_3 \end{aligned}$$

Moreover we show that this values are actually square roots of c modulo n, let's take m_1 but with the procedure being analogous also for the other ones:

$$m_1^2 = (p y_p m_q)^2 + (q y_q m_p)^2 + 2(pq m_p m_q y_q y_p) \bmod n$$

But we know that $m_p^2 \equiv c$ and $m_q^2 \equiv c \bmod n$ since we defined them as so. Moreover y_p and y_q were computed to be such that $p y_p + q y_q = 1$. The last term of the expression above is clearly divided by n, so we can rewrite:

$$m_1^2 = c[(p y_p)^2 + (q y_q)^2] = c[(p y_p + q y_q)^2 - 2pq y_p y_q] \equiv c(p y_p + q y_q)^2 \equiv c \bmod n$$

7. The last thing to do is to identify and collect the right message between these four candidates. But how?

```
function m = decrypter(c,p,q,n)
%DECRYPTER
%Given a cypher message c and the private key (p,q) returns the four
%roots of m^2
if nargin == 3
    n = p*q;
end
ep = (p+1)/4;
eq = (q+1)/4;
mp = 1;
mq = 1;
for i = 1:ep
    mp = mod(mp*c, p);
end
for i = 1:eq
    mq = mod(mq*c, q);
end
m(1) = crt(mp,mq,p,q);
m(2) = n - m(1);
m(3) = crt(-mp,mq,p,q);
m(4) = n - m(3);
end
```

5 Rabin's in-depth technicalities

5.1 Security

To analyse this cryptosystem we'll start from its effective **security**. The Rabin's Cryptosystem enjoys a very nice property, which is formalized by the following theorem:

Theorem 7 (Rabin, January 1979)**.** *Let* **AL** *be an algorithm for finding one of the solutions of*

$$y^2 \equiv m \ mod \ n$$

whenever a solution exists, and requiring $F(n)$ *steps. Then there exists an algorithm for factoring* n *requiring* $2F(n) + log_2(n)$ *steps.*

Proof. [In the case of $n = pq$ with p,q primes] Let k be any integer $0 < k < n$ such that $GCD(k, n) = 1$, this implies that $GCD(k, p) = GCD(k, q) = 1$. Then $y^2 \equiv k^2 \bmod n$ has exactly **four** solutions. Indeed let r,s be the remainders of

respectively $k : p$ and $k : q$:

$$y^2 \equiv k^2 \equiv (hp + r)^2 \equiv r^2 \bmod p$$
$$y^2 \equiv s^2 \bmod q$$

And so the four roots are given by $\pm r$ and $\pm s$.
Now, given any m different from p or q, we denote $\sqrt{m}$ as one of the solutions of $y^2 \equiv m \bmod n$. We choose at random a k different from p, q and $0 < k < n$ and we compute $k^2 \equiv m \bmod n$. Through **AL** we compute a solution $k_1 = \sqrt{m}$ (in $F(n)$ operations) and we define an equivalence relation $\mathcal{R}$ such that $k_1 \mathcal{R} k$ if k_1 is solution of $k^2 \equiv m \bmod n$, this allows us to define an equivalence class for the four roots, given k, defined as $[k]_{\mathcal{R}}$. In a **random** choice of k, the probabilities of choosing a particular element within its equivalence class are uniformly distributed. Hence with probability $\frac{1}{2}$:

$$k \equiv k_1 \bmod p,\ k \equiv -k_1 \bmod q \quad \textbf{OR} \quad k \equiv -k_1 \bmod p,\ k \equiv k_1 \bmod q$$

Then we have a probability of $\frac{1}{2}$ that $GCD(k - k_1, n) = p$ or to q. Computing the GCD with Euclidean Algorithm requires at most $log_2(n)$ subtractions and divisions. So we estimate a worst time complexity of $2F(n) + 2log_2(n)$. For big n we have asymptotically the same complexities amidst the factorization of n (since we retrieve p or q) and the search for roots of $y^2 \equiv m \bmod n$.

□

This powerful result states that if an adversary can invert the so-called Rabin Function, which is the one we use for the encryption and consisting in squaring a number modulo n, then with the same algorithm he can factorize n with a negligible number of additional steps for a big n. In our particular case we'll use n product of two prime numbers. Note that RSA Cryptosystem, which is probably the most widely used public key cryptosystem, does not enjoy such property, or at least nobody proved such a result for that. It is important to emphasize that here we referred to a "random" attack, like most of the Public-Key Systems, Rabin's is vulnerable to a "targeted" attack coming from an adversary which has obtained some cyphertexts of arbitrary plaintexts or decryptions of some chosen cyphertexts (**CPA** and **CCA**). This is due to the **deterministic** behaviour of the encryption.

5.2 Main Drawback of Rabin Cryptosystem

The main drawback of Rabin Cryptosystem is a natural consequence of the mathematics on which the algorithm is built. We proved that the output of the decryption process consists in four different roots, one is the correct message, the

other three **are not**. In general, without additional informations is impossible to distinguish which output is the correct one, this implies that a truly working Rabin algorithm is in practice weaker than the theoretical one. Indeed, adding informations about the correct message in the cyphertext corresponds to a loss of security from enemy attacks, moreover this kind of processes negatively affect the speed of both the encryption and decryption process.

5.3 Extra Bits Strategies

What are suitable strategies to recognize the correct root? Most of these are based on a probabilistic interpretation, for example some are based on redundancies of the bits which form the message. Trying to exploit this regularities, a common practice is to repeat the l least significant bits of the message at the end of it and then try to find among the four roots the one which present a similar pattern. This family of strategies, known as **padding**, is heuristic and has not mathematical guarantees.
Here I'll analyze a particular strategy which, instead, **guarantees through mathematics the correct choice of the root**, at the price of a sensitive loss of speed (especially in the encryption process) and the security of the process (involves adding informations to the cyphertext).
In particular, we'll add **two bits**, b_1 and b_2, to the cyphertext. First of all, some necessary mathematics:

Definition 5. *Let p be an odd prime number. An integer m is a quadratic residue modulo p if $\exists k$ integer such that $m \equiv k^2 \mod p$. The Legendre symbol is a function of m and p defined as*

$$(\tfrac{m}{p}) = \begin{cases} 1, & \text{if } m \text{ is a quadratic residue modulo } p \text{ and } m \not\equiv 0 \mod p \\ 0, & \text{if } m \equiv 0 \mod p \\ -1, & \text{otherwise} \end{cases}$$

We can get the Legendre symbol through a modular exponentiation thanks to the following theorem:

Theorem 8 (Euler's Criterion, Legendre symbol formulation)**.** *Let p be an odd prime number and m an integer coprime to p, then*

$$(\tfrac{m}{p}) = m^{\frac{p-1}{2}} \mod p$$

Note that in such case $(\frac{m}{p}) \neq 0 \; \forall m, p$, and that $\frac{p-1}{2} \in \mathbb{Z}$ due to p being odd.

Proof. **Claim:** $kx \equiv l \mod p$ has a unique solution x modulo p given that p does not divide k. Indeed assume $kx_1 \equiv kx_2 \mod p$, then p divides $k(x_1 - x_2)$,

hence p must divide $(x_1 - x_2)$ and so they are equal modulo p.
We take all the nonzero remainders modulo p which haven't their square congruent to m modulo p and group them into pairs (x, y) according to the rule that $xy \equiv m \bmod p$. Note that thanks to the claim we made before, we can find for any x a unique y, and vice versa, moreover since their square isn't congruent to m modulo p, x and y will differ for sure. Two case follows:
• m is **not** a quadratic residue, $(\frac{m}{p}) = -1$
Then we can simply take all the $p-1$ remainders and group them into $\frac{p-1}{2}$ pairs (no one will have the square congruent to m) and we conclude that $1 \cdot 2 \cdot ... \cdot (p-1) \equiv m^{\frac{p-1}{2}} \bmod p$.
• m is a quadratic residue, $(\frac{m}{p}) = 1$
Then exactly two remainders, r and $-r$, are such that $r^2 \equiv m \bmod p$. Pairing them together and the others as before, we get $(-r) \cdot r \equiv -m \bmod p$ and so $1 \cdot 2 \cdot ... \cdot (p-1) \equiv -m^{\frac{p-1}{2}} \bmod p$.

$$1 \cdot 2 \cdot ... \cdot (p-1) \equiv -(\tfrac{m}{p}) m^{\frac{p-1}{2}} \bmod p$$

In the second case, thanks to Thm.5, $\exists x \; m^{\frac{p-1}{2}} \equiv (x^2)^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \bmod p$ $\implies$ $(\frac{m}{p}) = m^{\frac{p-1}{2}}$ and that $1 \cdot 2 \cdot ... \cdot (p-1) \equiv -1 \bmod p$ (Wilson's Theorem).
In the first case we exploit Wilson Thm. to get $m^{\frac{p-1}{2}} \equiv -1 \bmod p$, so $(\frac{m}{p}) = m^{\frac{p-1}{2}}$. The case in which $(\frac{m}{p}) = 0$ is not allowed, so $(\frac{m}{p}) = m^{\frac{p-1}{2}}$ under the conditions of the Theorem. □

```
function b = mylegendre(m,p)
%Given an integer m coprime with an odd prime p, this function ...
    returns the
%Legendre Symbol (m/p)
[ym,yp,GCD] = euclid2(m,p);
if GCD ≠ 1
    error('This m is not suitable for this kind of computation ...
        since not coprime with p');
end
b = 0;
if mod(m,p) ≠ 0
    exp = (p-1)/2;
    b = 1;
    for i = 1:exp
        b = mod(b*m,p);
    end
    if b == p - 1
        b = -1;
    end
end
end
```

Then we'll proceed to generalize such symbol:

Definition 6. *For any integer m and any positive odd integer n, the Jacobi symbol $(\frac{m}{n})$ is defined as the product of the Legendre symbols corresponding to the prime factors of n:*

$$(\frac{m}{n}) = (\frac{m}{p_1})^{t_1} \cdot (\frac{m}{p_2})^{t_2} \cdot ... \cdot (\frac{m}{p_k})^{t_k}$$

where $n = p_1^{t_1} \cdot p_2^{t_2} \cdot ... \cdot p_k^{t_k}$ is the prime factorization of n.

Some **properties** of this symbol that we'll need later:

1. It is **completely multiplicative** in each of the arguments, if the other one is fixed.
2. $(\frac{-1}{n}) = (-1)^{\frac{n-1}{2}} = \begin{cases} 1, & n \equiv 1 \bmod 4 \\ \text{-1}, & n \equiv 3 \bmod 4 \end{cases}$
3. If $a \equiv b \bmod n$ then $(\frac{a}{n}) = (\frac{b}{n})$.

```
1 function b = myjacobi(m,p,q)
2 % Given an integer m and prime numbers p,q returns the Jacobi ...
      symbol (m/pq)
3 b = mylegendre(m,p)*mylegendre(m,q);
4 end
```

Note that with this kind of implementation, computing the Jacobi symbol requires the factors of n, something that is not public. In a realistic environment the computation of Jacobi symbol has to be done with a particular algorithm which takes a computation time analogous to the Euclidean Algorithm for the computation of GCD. Here we are only interested in highlight the existence of a safe way to extract the correct root from Rabin Algorithm and to try it.

Finally, the first bit b_1 that we will append to our cyphertext c will be the Jacobi symbol (can be mapped in $\{0,1\}$ with $x \mapsto \frac{x+1}{2}$) of the message m. The second bit b_2 we'll append to c is the least significant bit of m (this one tells if m is odd or even, an information that also an adversary could exploit).

Let's go straightforward to the step, in decryption, where we have to use the Chinese Remainder Theorem: we have four systems, but we know that if the system $CRT(m_p, m_q)$ produces m as solution, then the solution of system $CRT(-m_p, -m_q)$ is simply given by $n - m$. Analogous consideration for the other two systems $CRT(-m_p, m_q)$ and $CRT(m_p, -m_q)$. We consider the first two systems as a pair and the other two as another pair.
First of all we notice that $n = pq$ where $p, q \equiv 3 \bmod 4$ is odd (congruent to 1 modulo 3), this implies that $N - m$ is odd **if and only if** m is even. b_2 can

tell us which is the right solution in the case we can distinguish which pair of systems contains the right solution.
Since $n - m \equiv -m \bmod n$, from the properties we listed after the definition of the Jacobi symbol:

$$\left(\frac{n-m}{n}\right) = \left(\frac{-m}{n}\right) = \left(\frac{-1}{n}\right)\left(\frac{m}{n}\right) = \left(\frac{m}{n}\right)$$

So the two solutions of a pair of systems share the same Jacobi Symbol.
Remember how do we calculated m_p and m_q: they are quadratic residues modulo p and modulo q respectively. So $\left(\frac{m_p}{p}\right) = \left(\frac{m_q}{q}\right) = 1$, from the properties of Jacobi Symbol follows $\left(\frac{-m_p}{p}\right) = \left(\frac{-m_q}{q}\right) = -1$. Then looking at the systems (taking only one system for pair since we proved that Jacobi symbol it's the same within a pair):

$$\begin{cases} \mathrm{x} \equiv m_p, & \bmod p \\ \mathrm{x} \equiv m_q, & \bmod q \end{cases} \longrightarrow \left(\frac{x}{p}\right) = \left(\frac{m_p}{p}\right) \text{ and } \left(\frac{x}{q}\right) = \left(\frac{m_q}{q}\right)$$

$$\begin{cases} \mathrm{x} \equiv -m_p, & \bmod p \\ \mathrm{x} \equiv m_q, & \bmod q \end{cases} \longrightarrow \left(\frac{x}{p}\right) = \left(\frac{-m_p}{p}\right) \text{ and } \left(\frac{x}{q}\right) = \left(\frac{m_q}{q}\right)$$

$$\text{In the first case } \left(\frac{m}{n}\right) = \left(\frac{m}{p}\right)\left(\frac{m}{q}\right) = \left(\frac{m_p}{p}\right)\left(\frac{m_q}{q}\right) = 1$$
$$\text{In the second case } \left(\frac{m}{n}\right) = \left(\frac{m}{p}\right)\left(\frac{m}{q}\right) = \left(\frac{-m_p}{p}\right)\left(\frac{m_q}{q}\right) = -1$$

So, the two solutions provided by the first pair of systems have 1 as Jacobi Symbol modulo n, while the solutions provided by the second pair have -1 as Jacobi Symbol. Thanks to both b_1 and b_2 we are now able to distinguish the correct solution among the four.
We implement modified versions of encrypter and decrypter as follows:

```
function c = encrypter_mod(p,q,m)
%Modified version of the classical decrypter for Rabin's ...
    Cryptosystem. Here
%we add to our cyphertext informations about the parity and the ...
    Jacobi
%symbol
n = p*q;
c = mod(m^2,n);
%% Computation of the additional bits
b1 = myjacobi(m,p,q);
if b1 == 1
    b1 = '1';
else
    b1 = '0';
end
r = dec2bin(m);
b2 = r(end);
cr = dec2bin(c);
```

```
nr = ['1' b1 b2 cr]; %Here we need an additional '1' bit to ...
    prevent cutting the bits if the first is a '0'
c = bin2dec(nr);
end
```

```
function M = decrypter_mod(c,p,q,len)
%Modified version of the classical decrypter for Rabin's ...
    Cryptosystem. Here
%we use additional information to retrieve the correct root ...
    amidst the four
%candidates.
n = p*q;
%% Retrieving b1 and b2 from c
r = dec2bin(c);

b2 = r(2);
nr = r(3:end);
c = bin2dec(nr);
%% Using the additional informations to compute and return the ...
    correct root
ep = (p+1)/4;
eq = (q+1)/4;
mp = 1;
mq = 1;

for i = 1:ep
    mp = mod(mp*c, p);
end
for i = 1:eq
    mq = mod(mq*c, q);
end

m(1) = crt(b1*mp,mq,p,q);
m(2) = n - m(1);
if mod(m(1),2) == 1
    odd = m(1);
    even = m(2);
else
    odd = m(2);
    even = m(1);
end

if b2 == '1'
    M = odd;
else
    M = even;
end

end
```

5.4 Implementation of Rabin Cryptosystem through the Extra Bits Strategy

```
clear all; clc;
p= 2027;
q = 1759;
n = p*q; % n = 3565493

m1 = 2567652; %Necessarily less than n
m2 = 'Walking across the sitting-room, I turned the television off';

fprintf('First message to encrypt is: %d%.', m1);
fprintf('\n')
fprintf('Second message to encrypt is: %s%.', m2);
fprintf('\n')

c1 = encrypter_mod(p,q,m1);

fprintf('The first message was encrypted to: %d%', c1);
fprintf('\n')

dm2 = double(m2);
for i = 1:length(dm2)
    c2(i) = encrypter_mod(p,q,dm2(i));
end

fmt = ['The second message was encrypted to : [', repmat('%g, ', ...
    1, numel(c2)-1), '%g]\n'];
fprintf(fmt, c2);
fprintf('\n');

de1 = decrypter_mod(c1,p,q);

fprintf('The first message was decrypted to: %d%', de1);
fprintf('\n')

for i = 1:length(dm2)
    d2(i) = decrypter_mod(c2(i),p,q);
end
de2 = char(d2);

fprintf('The second message was decrypted to: %s%', de2);
fprintf('\n')
```

```
Command Window
  First message to encrypt is: 2567652
  Second message to encrypt is: Walking across the sitting-room, I turned the television off
  The first message was encrypted to: 13930985
  The second message was encrypted to : [48529, 124097, 77200, 126137, 125713, 77636, 92529, 9216, 1240

  The first message was decrypted to: 2567652
  The second message was decrypted to: Walking across the sitting-room, I turned the television off
fx >>
```

Figure 1: MATLAB output

6 Image encryption and decryption through Rabin Cryptosystem

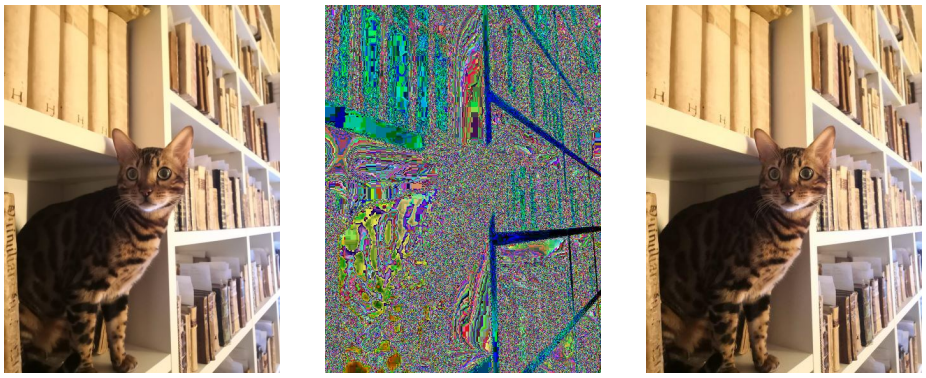
Finally, we'll apply Rabin algorithm to images. The particular case of images leads us to a simplification in the procedure of choosing the correct root. Indeed, even if we will not have certainty of making the correct choice, we'll select the **lowest** root. Take for example $p = 2027$ and $q = 1759$, we calculate $n = 3565493$: in such context the roots are integers less than n, a big number. But images are read by the software as three overlying matrices, where each pixel of the image corresponds to a numerical integer value between 0 and 255, corresponding to its color in a scale of gray. Assume an encryption-decryption process over the values of each pixel, in order to distort the coloration of the image. From the decryption process we'll get four different roots: one will be for sure the correct one, so a valid scale number between 0 and 255, while the other three are different integers between 0 and n. Intuitively, assuming an almost fair distribution of the values from 0 to n, it's very unlikely that there will be two different roots included in the values to 0 to 255! Testing this strategy over some different images always gave me no differences in pixels between decrypted version and original image.

The main MATLAB tools I used in this example implementation, apart from the functions we've previously implemented, are the functions dedicated to image processing through the software: `imread`, `imshow` and `imwrite`. The first function loads an image from the working directory as a 8-bit numerical tensor with the first two dimensions equal to those (in number of pixels) of the image, the third dimension is equal to three because a colour not in the grayscale is computed overlapping three of those; the second displays the image to the user and the last one saves a tensor of 8-bit numbers as an image in the working directory. We can specify file names and extension format. I've also made some conversion between 8-bit numbers and 64-bit numbers, due to a necessity of processing large enough numbers in the Rabin algorithm: the functions I've used to do such conversions are `uint8` and `int64`.

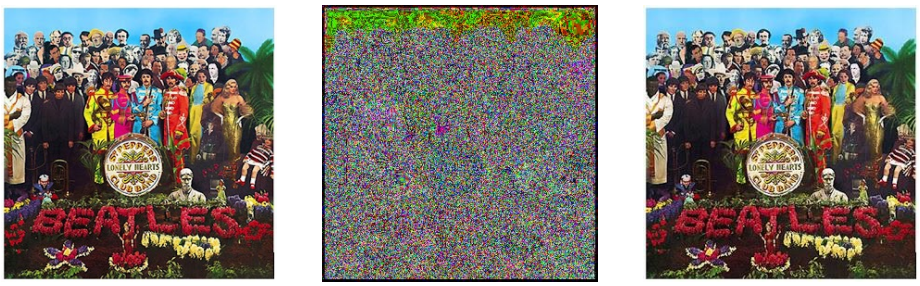
```
1  %% Rabin cryptosystem implementation
2  % Choose p, q prime numbers s.t. p = 3 (mod 4) and q = 3 (mod 4) ...
       and an
3  % image, compute the public key n = p*q then call encrypter.m on ...
       each pixel
4  % of the image to encrypt it using the public key. You can then ...
       decrypt the
5  % image using the private key (p,q) over each pixel. We choose ...
       the lowest
6  % root from the four proposed by the decrypter.
7  clear all; clc;
8  p= 2027;
9  q = 1759;
10 n = p*q;
```

```
img = 'shrink_dali3';
type = 'jpeg';
%% Image reading
A = imread(img,type);
%% Colour sectioning
B1 = int64(A(:,:,1));
B2 = int64(A(:,:,2));
B3 = int64(A(:,:,3));
%% Image encryption
for i = 1:size(B1,2)
    for j = 1:size(B1,1)
        C1(j,i) = encrypter(n,B1(j,i));
        C2(j,i) = encrypter(n,B2(j,i));
        C3(j,i) = encrypter(n,B3(j,i));
    end
end
Cs1 = uint8(mod(C1,255));
Cs2 = uint8(mod(C2,255));
Cs3 = uint8(mod(C3,255));
Cs(:,:,1) = Cs1;
Cs(:,:,2) = Cs2;
Cs(:,:,3) = Cs3;
imshow(Cs);
cname = strcat('encrypted_',img,'.jpg');
imwrite(Cs, cname);
%% Image decryption and decrypted image visualization
for i = 1:size(B1,2)
    for j = 1:size(B1,1)
          D1(j,i) = min(decrypter(C1(j,i),p,q,n));
          D2(j,i) = min(decrypter(C2(j,i),p,q,n));
          D3(j,i) = min(decrypter(C3(j,i),p,q,n));
    end
end
Ds1 = uint8(D1);
Ds2 = uint8(D2);
Ds3 = uint8(D3);
Ds(:,:,1) = Ds1;
Ds(:,:,2) = Ds2;
Ds(:,:,3) = Ds3;
imshow(Ds);
dname = strcat('decrypted_',img,'.jpg');
imwrite(Ds, dname);
```

Here some examples of outputs, original, encrypted and decrypted version:



Dalì, bengal cat.



A famous (and colorful) musical album cover.

Michael O. Rabin

6.1 Computational Time Analysis

Clearly, the process is heavier and takes more time for high resolution images, since they contain more pixels. Indeed, the computational time is directly proportional to the dimension of the image. For the last two examples (298×299 image vs 330×485 image), these are the computational times:

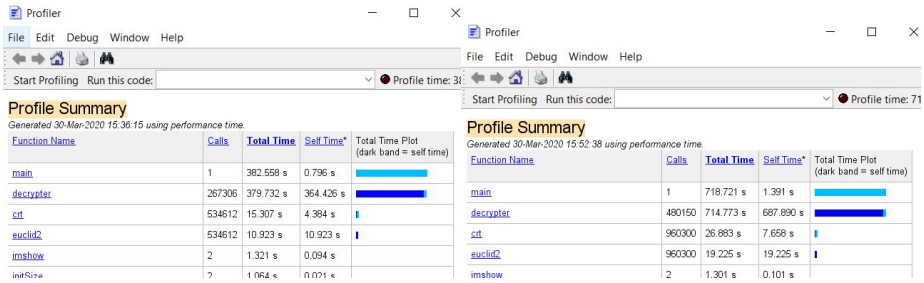


Figure 2: 89.192 pixels take 383 seconds, while 160.050 pixels take 719 second, the time is linearly dependent on total number of pixels

As we expected, the most of the computational time is spent in decryption process, which requires the calculation of two modular exponentiation and the double application of Chinese Remainder Theorem (which include the Extended Euclidean Algorithm). Rabin Algorithm confirms its high speed in the encryption phase, since it only requires a modular squaring (unlike RSA for example, which requires a cube elevation), but here we can "elude" any process of disambiguation among the four roots, something that in general requires additional computational time, as we've seen.

7 Conclusions

Rabin Cryptosystem is a solid Public Key cryptosystem, due to the very good properties that we've proved. Indeed is one of the few cryptosystems for which the equivalence with prime numbers factorization has been proved, moreover its encryption process is one of the fastest and lightest computationally among Public Key Cryptosystems. However it's impossible not to see the problems that the disambiguation of the roots produces, since it imposes a trade-off between speed, accuracy and security. Over the years, many alternative versions of this system have been proposed to overcome this problem, but fundamentally this is what prevented Rabin Cryptosystem from widely spreading in all the applications.

References

[1] Michael O. Rabin *Digitalized Signatures and Public Key Functions as Intractable as Factorization* MIT, Laboratory for computer science, 1979.

[2] Roberto Notari *Notes of Discrete Mathematics* Politecnico di Milano, course notes. 2019

[3] Steven Galbraith. *Mathematics of Public Key Cryptography.* Cambridge University Press, 513-520, 2012.

[4] A. Menezes, P.van Oorschot, S.Vanstone. *Handbook of Applied Crypthography.* [*Chapter 8 - Public-Key Encryption*]. CRC Press, 292-294, 2001. http://cacr.uwaterloo.ca/hac/

[5] Mahad, Z., Asbullah, M. A.1,2, Ariffin, M. R. K. *Efficient Methods to Overcome Rabin Cryptosystem Decryption Failure*
`Malaysian Journal of Mathematical Sciences` April 9-20, 2019 http://einspem.upm.edu.my/journal/fullpaper/vol11sapril/2.%20Zahari.pdf

[6] Wikipedia,
`https://en.wikipedia.org/wiki/Rabin-cryptosystem`
`https://en.wikipedia.org/wiki/Jacobi-symbol`