Mcrypt : A library for Rapid Prototyping of Algebraic Attacks

Students : Florian Archambaud, Harimbola Ranaivo

Project supervisor : Ludovic Perret

A report presented for the Master
SFPN

Department of Informatics
Sorbonne University
France
May the 24^{th}, 2023

Contents

1 Introduction

1.1 General context

The subject of our project, algebraic cryptanalysis [**?**, **?**], is a general cryptanalysis technique that reduces a cryptographic primitive's security analysis into computing the complexity of solving an algebraic system.

When talking about algebraic cryptanalysis, one can find two parts : modelization that involves creating a set of algebraic equations that represent the cryptographic system being analysed, and solving that allows to find the common zeroes of the algebraic equations. In this project, we will mainly focus on the modelization part. The purpose is to develop a new library that allows fast prototyping of algebraic attacks, called Mcrypt.

1.2 State of the art

The goal of Mcrypt is to offer multiple functions serving as tools in prototyping algebraic attacks. It uses msolve [**?**], a C library helping its users to solve multivariate polynomial systems. It relies on Gröbner bases [**?**, **?**] algorithms to compute algebraic representations of the solution set from which many, if not all, informations can be extracted. Using msolve can be done independently by executing a C library[1], or via a Julia Package called AlgebraicSolving[2].

Julia[3] is a high performance language (comparable to C) with a dynamic syntax (comparable to Python). From Julia, Nemo[4] is a computer algebra package aiming to manipulate in particular mutlivariate polynomials. For those reasons, we developed Mcrypt in Julia using Nemo.

We also considered the MQEstimator[5], a SageMath package furnishing tools to determine the hardness of Multivariate Quadratic polynomial problems, from which we studied the docker structure as well. The Multivariate Quadratic (MQ) problem consists in finding the solutions of a given system of m quadratic equations in n unknowns over a finite field, and it is an NP-complete problem of fundamental importance in computer science.

1.3 Organization of the report and main results

After the introduction, the report is organised as follows. In Section 2, we introduce the **Mcrypt library**, present how to install the package and explain the use of a docker with makefile. We also expose the structure of the Mcrypt package and how to use it. Showing all the functions we were asked to develop, the mathematical representation of some functions, an example of execution alongside the output. Those functions are mainly used to generate a random non-linear system of quadratic equations (**GenRand()**, Section 2.3.1.4), a system

[1] `https://github.com/algebraic-solving/msolve`
[2] `https://github.com/algebraic-solving/AlgebraicSolving.jl`
[3] `https://julialang.org/`
[4] `https://nemocas.github.io/Nemo.jl/latest/`
[5] `https://github.com/Crypto-TII/multivariate_quadratic_estimator`

of equations defined by the product of linear forms (**GenSysPower()**, Section 2.3.1.5) , and an instance of the MinRank problem which is described in Section 2.3.2

Then, Section 3 presents the results of some tests we carried on to measure the performances of Mcrypt's functions. Exposing the performance comparison between two functions which generate a system of quadratic equations and showing the detailed timing for both of them.

All in all, it is about 250 lines of code. We also developed a manual including software tips that is not part of the report. The manual contains installation tips on the software we used during this project, and useful features of Julia.

2 Mcrypt library

2.1 Installation

First, install Julia, and download the files for Mcrypt.

If this is your first time using Mcrypt, then the dependencies (Julia packages used by Mcrypt and not in the default version of Julia) might not be added to your machine. In that case, follow the steps below :

```
Open julia interactive environment(REPL) in the package folder
Start "package mode" by typing ]
execute the following command : instantiate
then execute : activate .
Press backspace to leave the package mode
```

If this is not your first time, you should not have to re-install the dependencies, just use this instead :

```
Open julia interactive environment(REPL) in the package folder
Start "package mode" by typing ]
execute the following command : activate .
Press backspace to leave the package mode
```

Now to gain access to all functions provided by the package, execute the following command:

```
julia> using Mcrypt
```

2.2 Docker

We provide a docker for Mcrypt. The goal of a docker is to simplify the process of software development and deployment. It enables users to use the application on different operating system by packaging all dependencies using containerization. [6]

[6] `https://www.ibm.com/topics/docker`

2.2.1 Elements for the docker

We created two files : the Dockerfile and the makefile (this one is optional but makes the process easier). We will explain the use of each of them for the docker in Mcrypt.

Dockerfile

```
FROM julia:latest
ADD "." "/Mcrypt"
WORKDIR "/Mcrypt"
RUN julia -e 'using Pkg; Pkg.add(["Nemo", "Random", "
   AlgebraicSolving"]);';
CMD ["julia"]
EXPOSE 3000
```

FROM julia:latest → We import the latest Julia image (a virtual machine where Julia is installed) as a base for our docker.

ADD "." "/Mcrypt" → We copy the content of the current repository (Mcrypt's root repository) inside our machine into a new repository named Mcrypt in the root repository of our virtual machine.

WORKDIR "/Mcrypt" → We locate ourselves inside /Mcrypt in our virtual machine for the next operations.

RUN julia -e 'using Pkg; Pkg.add(["Nemo", "Random", "AlgebraicSolving"]);'; → We install the supplementary packages we need for Mcrypt inside the virtual machine.

CMD ["julia"] → We launch the Julia line of command inside the virtual machine when we use the docker.

EXPOSE 3000 → Just for reference purpose (the suppression of this line shouldn't provoke an error), we use the port n3000 on our computer to connect to the docker.

makefile

```
PACKAGE=mcrypt
DOCKER_IMG_NAME=$(PACKAGE)

all: rundocker

builddocker:
        sudo docker build -t $(DOCKER_IMG_NAME) .

rundocker: builddocker
        sudo docker run -it $(DOCKER_IMG_NAME)
```

The command **buildocker** is used to create the docker (option -t to name it for the following command), while **rundocker** executes it (option -t to identify the docker and -i to interact with the virtual machine).

2.2.2 How to launch the docker for Mcrypt

After installing Docker Desktop and downloaded Mcrypt, in Mcrypt root repository, we create and run the docker. Execute the following command in a prompt

```
$make rundocker
```

This should launch the docker, displaying a Julia line of command, and operations using Mcrypt can then be computed by following some familiar steps :

Go to package mode by typing] and enter the following command :

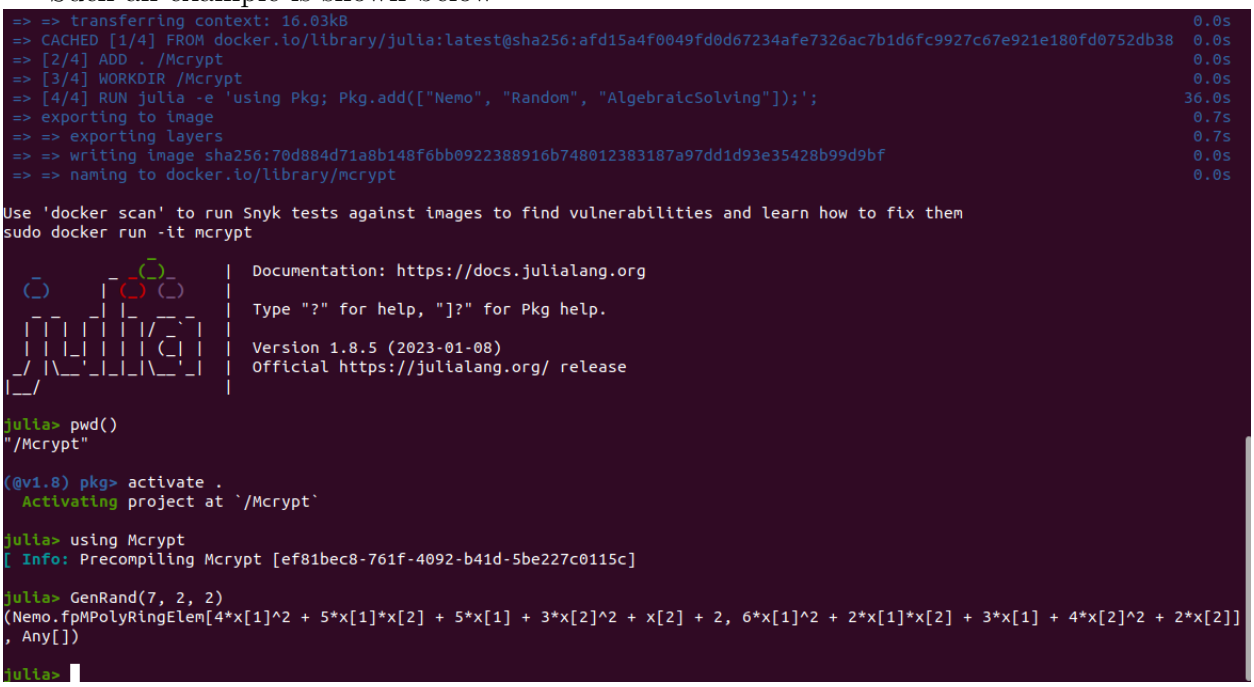
```
activate .
```

Press backspace to leave the **package mode**

Now you can use the package :

```
julia> using Mcrypt
```

Such an example is shown below



2.3 The main functions in the Mcrypt library

This section will serve to show the functions currently available in Mcrypt.

2.3.1 Polynomial generation

In this part, we will present the functions that generate polynomials $p_1, \dots p_m \in \mathbb{K}[x_1, \dots, x_n]$.

GenRand is a function that generates a system of random equations with or without solution according to its parameters. It is aimed to facilitate the calls to GenRandMqSum, GenRandMqMat and GenSysSol by using only integers as parameters.

GenRandMqSum and GenRandMqMat output a system of random quadratic polynomial equations.

GenSysSol gives a system of quadratic polynomial equations with its solution.

GenSysPower outputs a system of polynomial equations with a particular structure, with or without a preemptive solution.

2.3.1.1 GenRandMqSum

Algorithm 1 Quadratic polynomials with sum

Require: $\mathbb{K}$ (field), x (vector of n variables), m (integer)
  $p :=$ vector of size m
  $random(\mathbb{K}) :=$ function that extracts a random element from $\mathbb{K}$
  for i from 1 to m **do**
    $p_i(x) := \sum_{j=1,k=1}^{n}(random(\mathbb{K}) \times x_j \times x_k) + \sum_{j=1}^{n}(random(\mathbb{K}) \times x_j) + random(\mathbb{K})$
  end for
  return p

Description : Generates a vector of quadratic equations as

$$p(x_1, \ldots, x_n) = \sum_{1 \leq i \leq j \leq n} a_i \times x_i \times x_j + \sum_{1 \leq i \leq n} b_i \times x_i + c$$

with $a_i, b_i, c \in \mathbb{K}$

Function : GenRandMqSum$(\mathbb{K}, var, m)$

- $\mathbb{K}$: Field
- var : Array of variables
- m : Number of equations

Command :

```
julia> var = [string("x[", i, "]") for i in 1:2]
julia> GenRandMqSum(GF(7), var, 2)
```

The previous commands set an array of 2 variables and generate a system of 2 equations by sum in a finite field of characteristic 7 for those variables.

Output :

```
2-element Vector{gfp_mpoly}:
 4*x[1]^2 + 3*x[1]*x[2] + 3*x[1] + 5*x[2]^2 + 2*x[2] + 2
 6*x[1]^2 + x[1]*x[2] + 4*x[1] + 2*x[2]^2 + x[2] + 5
```

Algorithm 2 Quadratic polynomials with matrix product

Require: $\mathbb{K}$ (field), x (vector of n variables), m (integer)
  $p :=$ vector of size m
  for i from 1 to m **do**
    $Q :=$ symmetric matrix of size $n \times n$ with random elements from $\mathbb{K}$
    $p_i(x) := x^T \cdot Q \cdot x$
  end for
  return p

2.3.1.2 GenRandMqMat

Description : Generates a vector of quadratic equations using matrix product

$$p(x_1, \ldots, x_n) = x^T \cdot Q \cdot x$$

$x \in \mathbb{K}^n$, Q $\in \mathbb{K}^{n*n}$

Function : GenRandMqMat($\mathbb{K}$ var, m)

- $\mathbb{K}$: Field
- *var* : Array of variables
- m : Number of equations

Command :

```
julia> var = [string("x[", i, "]") for i in 1:2]
julia> GenRandMqMat(GF(7), var, 2)
```

The previous commands set an array of 2 variables and generate a system of 2 equations by matrix product in a finite field of characteristic 7 for those variables.

2.3.1.3 GenSysSol

Algorithm 3 Quadratic polynomials with matrix product and solution

Require: $\mathbb{K}$ (field), x (vector of n variables), m (integer)
  $p :=$ vector of m quadratic polynomials generated randomly
  $s :=$ vector of size n with random elements from $\mathbb{K}$
  for i from 1 to m **do**
    $p_i(x) := p_i - p_i(s)$
  end for
  return p, s

Description : Generates a vector of quadratic equations with its solution using matrix product

Function : GenSysSol(𝕂, var, m)

- 𝕂 : Field
- *var* : Array of variables
- *m* : Number of equations

Command :

```
julia> var = [string("x[", i, "]") for i in 1:2]
julia> GenSysSol(GF(7), var, 2)
```

The previous commands set an array of 2 variables and generate a system of 2 equations by matrix product in a finite field of characteristic 7 for those variables with its vector solution.

Output :

```
(gfp_mpoly[4*x[1]^2 + 6*x[1]*x[2] + 3*x[1] + x[2] + 3, 6*x[1]^2 +
x[1]*x[2] + 6*x[2]^2 + 4*x[2] + 1], gfp_elem[4, 3])
```

2.3.1.4 GenRand

Description : Generates a vector of quadratic equations with or without solution using a sum or matrix product

Function : GenRand(q, n, m, method = 1)

- *q* : Field's characteristic
- *n* : Number of variables
- *m* : Number of equations
- *method* : Choice of method (0 = sum, 1 = matrix product, 2 = matrix product with solution)

Command :

```
julia> GenRand(7, 2, 2, 2)
```

The previous command generates a system of 2 equations by matrix product in a finite field of characteristic 7 for 2 variables with its vector solution.

Output :

```
(gfp_mpoly[x[1]^2 + 6*x[1]*x[2] + 3*x[1] + 4*x[2]^2 + 2*x[2] +
3, 5*x[1]^2 + 6*x[1]*x[2] + 4*x[1] + 2*x[2]^2 + x[2] + 1], gfp_elem[6, 4])
```

2.3.1.5 GenSysPower

Algorithm 4 Polynomials with power method

Require: $\mathbb{K}$ (field), x (vector of n variables), d (integer)
  $p :=$ vector of size n
  $random(\mathbb{K}) :=$ function that extracts a random element from $\mathbb{K}$
  for i from 1 to n **do**
    $p_i := 1$
    for j from 1 to d **do**
      $p_i := p_i \times (\sum_{k=1}^{n}(random(\mathbb{K}) \times x_k) + random(\mathbb{K}))$
    end for
  end for
  return p

Description : Generates a vector of polynomials with or without its solution using the product below:

$$p_i(x_1, \ldots, x_n) = \prod_{i=1}^{d} A(x_1, \ldots, x_n),$$
$$A = \sum_{j=1}^{n} a_{ij} \times x_j + c_i$$

a_{ij}, $c_i \in \mathbb{K}$

Function : GenSysPower(n, d, q, solution = true)

- n : The number of variables = The number of equations
- d : The degree
- q : The characteristic of the field
- *solution* : With solution if true

Command :

```
julia> GenSysPower(2, 2, 7)
```

The previous command generates a system of 2 equations of degree 2 for 4 variables in a finite field of characteristic 7 with its vector solution.

Output :

```
(gfp_mpoly[4*x[1]^2 + 5*x[1]*x[2] + 6*x[1] + 5*x[2]^2 + 6*x[2],
5*x[1]^2 + 6*x[1]*x[2] + 2*x[1] + 5*x[2] + 1],
gfp_elem[2, 1])
```

2.3.2 MinRank instance generation

The MinRank problem belongs to NP-complete problems and consists, for a (k+1)-tuple of matrices (M_0, M_1 ... M_k) $\in \mathbb{M}^{m*n}(\mathbb{F}_q)$ and an integer R, in finding a k-tuple of coefficients (λ_1, λ_2 ... λ_k) $\in \mathbb{F}_q$ such that $\sum_{i=1}^{k}(\lambda_i \times M_i) - M_0 \leq R$ [**?**].

The function GenInstMinRank takes a set of parameters (one of which is an integer R) as input to generate a matrix vector M and a coefficient vector λ in a finite field such that $\sum_{i=1}^{k}(\lambda_i \times M_i) - M_0 \leq R$.

The function MinRank generates a vector of equations in a finite field defined by v * M = 0 with v a vector of variables and M a matrix with the same number of variables as v but distinct from those of v and one row from M having the same variable in each of its columns multiplied by a random element of the field.

2.3.2.1 GenInstMinRank

Algorithm 5 MinRank instance generation with solution

Require: $\mathbb{K}$ (field), k (integer), m (integer), n (integer), R (integer)
$\lambda :=$ vector of size k with random elements from K
$M :=$ vector of size $(k+1)$ with matrices of size $m \times n$ filled with random elements from $\mathbb{K}$
$Aux :=$ matrix of size $m \times n$ and rank R obtained randomly with elements from $\mathbb{K}$
$M_0(x) := \sum_{i=1}^{k}(\lambda_i \times M_i) - Aux$
return M, λ

Description : Generates an instance for the MinRank problem with its solution

Function : GenInstMinRank(d, k, m, n, R)

- d : Field's characteristic
- k : Number of instances
- m : Number of rows
- n : Number of columns
- R : Rank

Command :

```
julia> GenInstMinRank(7, 3, 5, 5, 3)
```

```
julia> GenInstMinRank(7, 3, 5, 5, 3)
(Matrix{Nemo.gfp_elem}[[2 4 … 6 3; 6 3 … 4 2; … ; 6 3 … 5 3; 1 1 … 1 5], [2 4 …
 3 1; 5 4 … 5 2; … ; 5 0 … 1 6; 3 0 … 0 0], [4 5 … 3 5; 4 2 … 0 5; … ; 1 5 … 6
6; 4 5 … 5 5], [1 4 … 3 3; 2 4 … 0 5; … ; 5 2 … 6 2; 2 2 … 6 4]], Nemo.gfp_elem
[3, 1, 4])
```

2.3.2.2 MinRank

Listing 1: MinRank generation

```
function MinRank(q, n)
    lambda_i = [string("lamb[", i, "]") for i in 1:n]
    v_i = [string("v[", i, "]") for i in 1:n]
    K = GF(q)
    S, vars = PolynomialRing(K, [lambda_i; v_i])
    lambda = vars[1:n]
    v = transpose(vars[n+1:n*2])
    M = sum([Matrix(diagonal_matrix(lambda[i], n)) * rand(K, n,
        n) for i in 1:n])
    eqs = transpose(v * M)
    return eqs
end
```

Description : Generates a set of equations with

$$v \cdot M = 0$$

$$M = \sum_{i=1}^{n} \lambda_i \times M_i$$

where $M_i \in \mathbb{M}_{n*n}(\mathbb{F}_q)$ generated randomly
v a vector of size n
$(\lambda_1, \ldots, \lambda_i) \in \mathbb{F}_q^n$

Function : MinRank(d, n)

- d : Characteristic of the field
- n : Number of variables, equations, rows and columns

Command :

```
julia> MinRank(7, 3)
```

The previous command generates a system of 3 equations in a finite field of characteristic 7

```
julia> MinRank(7, 3)
3-element Vector{Nemo.gfp_mpoly}:
 lamb[1]*v[1] + 4*lamb[1]*v[2] + 5*lamb[1]*v[3] + lamb[2]*v[1] + 4*lamb[2]*v[3] + 6*lamb[3]*v[1] + 4*lamb[3]*v[2]
 4*lamb[1]*v[1] + 4*lamb[1]*v[2] + lamb[1]*v[3] + 2*lamb[2]*v[2] + 3*lamb[2]*v[3] + lamb[3]*v[1] + 6*lamb[3]*v[2] + lamb[3]*v[3]
 3*lamb[1]*v[1] + 2*lamb[1]*v[2] + lamb[1]*v[3] + lamb[2]*v[1] + 3*lamb[2]*v[3] + 4*lamb[3]*v[1] + 6*lamb[3]*v[2] + 4*lamb[3]*v[3]
```

2.3.3 Verification : Issol

To verify the veracity of most of our functions, we had to code a way to evaluate whether the solution for a system would resolve this system or not. With this goal, the function Issol evaluates a system of equations by replacing its variables by the solution we want to test, and outputs a boolean reflecting the result of the verification.

Description : This function verifies if a vector is a solution of a system of equation

Algorithm 6 Verification of a solution

Require: *syst* (vector of equation), *vect* (vector of solution)

```
for eq in syst do
    if evaluate(eq, vect) != 0 then
        return false
    end if
end for
return true
```

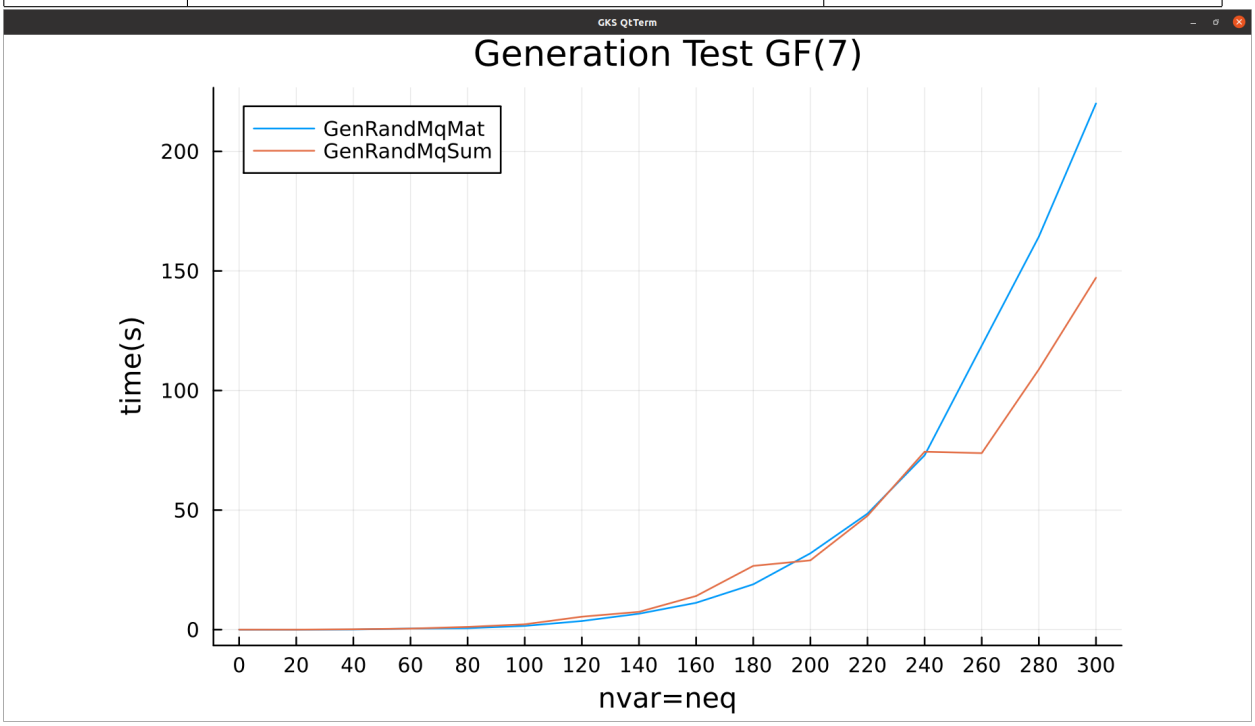
3 Tests

3.1 Performance comparison

3.1.1 Timing quadratic equations generation functions

This comparison is performed on GF(7), measuring if it would be preferable to use one of the methods we coded for generating quadratic equations over the other in term of time cost.

nvar = neq	using $p = \sum_{1 \leq i \leq j \leq n} a_i x_i x_j + \sum_{1 \leq i \leq n} b_i x_i + c$	using $p = x_i^T Q x_i \ 1 \leq i \leq n$
40	293ms	96.5ms
80	1.40s	809ms
120	6.46s	3.74s
160	15.66s	11.9s
200	31.9s	33.3s



What we can observe is that, as we coded them, the two methods seem to take a similar

length of time. Generating quadratic equations using matrices seems to take more time at some point.

Also, as we see will see below, it verifies that the time used for generating equations is negligible compared to the time spent during their resolution.

3.1.2 Timing functions calling msolve

For all the timing done, we use GF(7) and nvar = neq

	Random Generation		Power Equation deg=2	
nvar=neq	without solution	with solution	without solution	with solution
10	0.93	1.07	0.88	0.99
11	6.55	6.61	6.09	6.80
12	53.54	53.91	59.40	66.07
13	373.09	374.23	409.92	573.33
14	2804.62(46mn)	2806.1(46mn)	>1h	>1h
15	>1h	>1h	>1h	>1h

	Power Equation deg=3	
nvar=neq	without solution	with solution
6	0.14	0.13
7	2.32	2.93
8	69.34	79.19
9	2164.71(35mn)	2393.95(38mn)
10	>1h	>1h

The execution time in those tests are mainly computing the Gröbner basis as we saw that the generation of equations are negligible. The Gröbner basis of random equations are computed faster than power equations as the number of equations and variables increases. The difference between the execution time on a system of equations with and without solution is significant on the power equations, while on random equations it is quite the same.

3.2 Function test

We made a test folder to put a set of test for each function in Mcrypt. Here is an example :

```
51  #test GenInstMinRank
52  M, lamb = GenInstMinRank(7, 3, 5, 5, 3)
53  MSpace = MatrixSpace(GF(7), 5, 5)
54  MSum = sum(Int(lamb[i].data) * M[i+1] for i in 1:3) - M[1]
55  @test rank(MSpace(MSum)) == 3
```

We generate an instance of MinRank with $R = 3$ and test if the rank of the matrix is indeed 3.

4 Conclusion

In conclusion, this project was intended to focus on the modelization part of algebraic cryptanalysis. Thus, we developed the first steps of the library, furnishing basic tools to test algebraic attacks by generating systems of equations to solve, as well as a docker so that users can operate on the library on different operating systems without worrying about dependencies. All the functions provided in the library are tested separately to ensure their correctness.

References

[1] Bard, G. V. *Algebraic Cryptanalysis*, 1 ed. Springer New York, NY, 2009.

[2] Berthomieu, J., Eder, C., and Safey El Din, M. msolve: A Library for Solving Polynomial Systems. In *2021 International Symposium on Symbolic and Algebraic Computation* (Saint Petersburg, Russia, July 2021), 46th International Symposium on Symbolic and Algebraic Computation.

[3] Buchberger, B. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation 41*, 3-4 (2006), 475–511.

[4] Buchberger, B., Collins, G. E., Loos, R. G. K., and Albrecht, R. Computer algebra symbolic and algebraic computation. *SIGSAM Bull. 16*, 4 (1982), 5–5.

[5] Faugère, J., Levy-dit-Vehel, F., and Perret, L. Cryptanalysis of minrank. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings* (2008), D. A. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, pp. 280–296.

[6] Perret, L. *Bases de Gröbner en Cryptographie Post-Quantique. (Gröbner bases techniques in Quantum-Safe Cryptography).* 2016.