

# Rapport du Projet de Cryptanalyse

Sid Ali ZITOUNI TERKI

Décembre 2020

# Table des matières

|           |                             |           |
|-----------|-----------------------------|-----------|
| <b>I</b>  | <b>Cryptanalyse de A5/2</b> | <b>2</b>  |
| 1         | .....                       | 3         |
| 2         | .....                       | 6         |
| 3         | .....                       | 7         |
| 4         | .....                       | 9         |
| 5         | .....                       | 11        |
| 6         | .....                       | 13        |
| 7         | .....                       | 14        |
| 8         | .....                       | 15        |
| 9         | .....                       | 17        |
| 10        | .....                       | 20        |
| 11        | .....                       | 22        |
| 12        | .....                       | 24        |
| 13        | .....                       | 26        |
| 14        | .....                       | 31        |
| 15        | .....                       | 32        |
| <b>II</b> | <b>Cryptanalyse de RSA</b>  | <b>33</b> |
| 16        | .....                       | 34        |
| 17        | .....                       | 35        |
| 18        | .....                       | 37        |
| 19        | .....                       | 38        |
| 20        | .....                       | 39        |
| 21        | .....                       | 40        |
| 22        | .....                       | 42        |
| 23        | .....                       | 43        |
| 24        | .....                       | 44        |
| 25        | .....                       | 46        |

# Chapitre I

## Cryptanalyse de A5/2

# 1

```
1 def LFSR_step(state,f):
2     L = len(state)
3     out = state[0]
4     state = state[1:]+[sum(f[j+1]*state[L-1-j] for j in range(L))]
5     return out,state
6
7
8 def A5_2_init(K, IV, f1, f2, f3, f4):
9
10    # On initialise les registres 0.
11    R1 = Sequence([GF(2)(0) for _ in range(19)])
12    R2 = Sequence([GF(2)(0) for _ in range(22)])
13    R3 = Sequence([GF(2)(0) for _ in range(23)])
14    R4 = Sequence([GF(2)(0) for _ in range(17)])
15
16
17    for i in range(64):
18        # On met jour chaque registre avec son polynome
19        # de rtroaction respectif.
20        _, R1 = LFSR_step(R1, f1)
21        _, R2 = LFSR_step(R2, f2)
22        _, R3 = LFSR_step(R3, f3)
23        _, R4 = LFSR_step(R4, f4)
24
25
26
27        # On Xor pour chaque derniers bits de chaque registre
28        # avec l'indice i (qui est le nombre tours faits)
29        # de la clef K.
30        R1[18] = R1[18] + K[i]
31        R2[21] = R2[21] + K[i]
32        R3[22] = R3[22] + K[i]
33        R4[16] = R4[16] + K[i]
34
35
36
37
38    for i in range(22):
39
40        # On met jour chaque registre avec son polynome
41        # de rtroaction respectif.
42        _, R1 = LFSR_step(R1, f1)
43        _, R2 = LFSR_step(R2, f2)
44        _, R3 = LFSR_step(R3, f3)
45        _, R4 = LFSR_step(R4, f4)
46
47
48
49        # On Xor pour chaque derniers bits de chaque registre
50        # avec l'indice i du vecteur initial IV.
51        R1[18] = R1[18] + IV[i]
52        R2[21] = R2[21] + IV[i]
53        R3[22] = R3[22] + IV[i]
54        R4[16] = R4[16] + IV[i]
55
56
57
58        # On fixe certains bits des registres.
59        R1[3] = 1
60        R2[5] = 1
61        R3[4] = 1
62        R4[6] = 1
```

```

63
64
65     return R1, R2, R3, R4
66
67
68 def Maj(a, b, c):
69     # 3 cas possibles pour la fonction majorit.
70     m = 0
71     if a == b:
72         m = a
73
74     if a == c:
75         m = a
76
77     if b == c :
78         m = b
79
80     return m
81
82
83 def A5_2_step(R1, R2, R3, R4, f1, f2, f3, f4):
84
85     # On calcule la fonction majorit
86     # sur les indices 6, 13 et 9 de R4.
87     m = Maj(R4[6], R4[13], R4[9])
88
89
90     # On a 3 choix possibles.
91     # Si l'indice 6 de R4 est gale m
92     # on met jour R1 en ignorant sa sortie.
93
94     if R4[6] == m :
95         _, R1 = LFSR_step(R1, f1)
96
97
98
99     # Si l'indice 13 de R4 est gale m
100    # on met jour R2 en ignorant sa sortie.
101    if R4[13] == m :
102        _, R2 = LFSR_step(R2, f2)
103
104
105
106    # Si l'indice 9 de R4 est gale m
107    # on met jour R3 en ignorant sa sortie.
108    if R4[9] == m :
109        _, R3 = LFSR_step(R3, f3)
110
111
112
113
114
115    # on met jour R4 en ignorant la sortie.
116    _, R4 = LFSR_step(R4, f4)
117
118    # On calcule y1, y2 et y3 avec les formules donnees.
119    y1 = R1[0] + Maj(R1[3], R1[4] + 1, R1[6])
120    y2 = R2[0] + Maj(R2[8], R2[5] + 1, R2[12])
121    y3 = R3[0] + Maj(R3[4], R3[9] + 1, R3[6])
122
123    # On ressort le rsultat de l'additions des 3 variables
124    return (y1 + y2 + y3), (R1, R2, R3, R4)
125
126
127 def A5_2_production(R1, R2, R3, R4, f1, f2, f3, f4):
128
129     # On excute 99 fois la fonction "A5_2_step" en ignorant
130     # son bit de sortie

```

```

131     for i in range(99):
132         _, (R1 ,R2 ,R3 ,R4) = A5_2_step(R1, R2, R3, R4, f1, f2, f3, f4)
133
134     # On prend z une suite iniatilis une liste vide.
135     z = []
136
137     # On excute 228 fois la fonction "A5_2_step" en utilisant son
138     # bit de sortie.
139     for i in range(228):
140         zi, (R1 ,R2 ,R3 ,R4) = A5_2_step(R1, R2, R3, R4, f1, f2, f3, f4)
141         z.append(zi)
142
143
144     return z
145
146 def A5_2(K, IV):
147
148     # On cre les polynomes de rtroaction
149     PR.<X> = PolynomialRing(GF(2))
150     f1 = X**19 + X**18 + X**17 + X**14 + 1
151     f2 = X**22 + X**21 + 1
152     f3 = X**23 + X**22 + X**21 + X**8 + 1
153     f4 = X**17 + X**12 + 1
154
155     # Phase d'initialisation.
156     R1, R2, R3, R4 = A5_2_init(K, IV, f1, f2, f3, f4)
157
158
159     # Phase de productions.
160     return A5_2_production(R1, R2, R3, R4, f1, f2, f3, f4)
161
162
163
164 load('A5_2-test-vector.sage')
165
166 # On va tester si notre fonction nous donne les bonnes rsultats
167 # prvues .
168 zprime = A5_2(K, IV)
169
170 print("z = A5_2(K, IV) est ", zprime == z)

```

Le première fonction sert à mettre à jour les *LFSR*.

La fonction *A5/2 – init* sert à créer les 4 registres à partir d’une clé *K* et d’un vecteur d’initialisation *IV*.

La fonction *Maj* sert à calculer la majorité de 3 bits donnés.

La fonction *A5/2 – step* sert à produire un bit par application.

La fonction *A5/2 – production* sert à produire *N* bits par application avec *N* donné qui est pour l’instant 228.

Ce code est dans "*Exercice1.sage*". Au cas que mes programmes en sage ne fonctionnent pas, je vous ai laissé même la version notebook jupiter, ce code est "*A5\_2.ipynb*"(Toute la partie *A5\_2*).

## 2

Après la phase d'initialisation, on suppose connaître  $R4$  qui est le registre du  $LFSR4$ . Comme on connaît  $R4$ , dans l'algorithme du **A5/2-step**, on trouvera  $m$  et on sait lesquels  $R_i$  (avec  $1 \leq i \leq 3$ ) seront mise à jour.

Au début on pose :

$$R1 = (x_0, \dots, x_{18})$$

$$R2 = (x_{19}, \dots, x_{40})$$

$$R3 = (x_{41}, \dots, x_{63})$$

On suppose qu'il y a eu  $k$  mise à jour pendant la phase de production pour  $R1$  avec  $k \leq 228 + 99$   
 $k \leq 327$ .

Au départ on a que :

$$x_0 = x_0, x_1 = x_1, \dots, x_{18} = x_{18}$$

Pour le premier tour :

$$x_0 = x_1, x_1 = x_2, \dots, x_{18} = x_{18} + x_{17} + x_{16} + x_{13}$$

Après les  $k$  tours :

$$x_0 = x'_0, x_1 = x'_1, \dots, x_{18} = x'_{18}$$

A la fin de l'algorithme **A5/2** , on a pour chaque  $x'_i$  est composée des  $x_i$  initiale (pareil pour  $R2, R3$ ).

Comme on utilise la fonction de rétroaction et donc des additions, on a bien à la fin pour les 3 registres :  
des équations linéaires.

### 3

```

1 load('Exercice1.sage')
2 # On cre les matrices de rtroaction avec cette fonction:
3 def matf(f):# matrice de transition
4     n = f.degree()
5     A = Matrix(GF(2),n,n)
6     for i in range(n-1):
7         A[i,i+1] = 1
8         tmp = f.list()[1:] # c_1,c_2,...,c_L
9         tmp.reverse() # c_L,c_{L-1}, ... , c_1
10        A[n-1] = vector(tmp)
11    return A
12
13 def A5_2_stepret(R1, R2, R3, R4, f1, f2, f3, f4):
14     #nos matrices de retroactions:
15     A1 = matf(f1)
16     A2 = matf(f2)
17     A3 = matf(f3)
18     A4 = matf(f4)
19
20
21     # On calcule la fonction majorit
22     # sur les indices 6, 13 et 9 de R4.
23     m = Maj(R4[6], R4[13], R4[9])
24
25
26     # On a 3 choix possibles.
27     # Si l'indice 6 de R4 est gale m
28     # on met jour R1 en ignorant sa sortie.
29
30     if R4[6] == m :
31         x1 = vector(R1)
32         x1 = A1*x1
33         R1 = Sequence(x1)
34
35
36
37     # Si l'indice 13 de R4 est gale m
38     # on met jour R2 en ignorant sa sortie.
39     if R4[13] == m :
40         x2 = vector(R2)
41         x2 = A2*x2
42         R2 = Sequence(x2)
43
44
45
46     # Si l'indice 9 de R4 est gale m
47     # on met jour R3 en ignorant sa sortie.
48     if R4[9] == m :
49         x3 = vector(R3)
50         x3 = A3*x3
51         R3 = Sequence(x3)
52
53
54
55
56
57     # on met jour R4 en ignorant la sortie.
58     x4 = vector(R4)
59     x4 = A4*x4
60     R4 = Sequence(x4)
61
62     # On calcule y1, y2 et y3 avec les formules donnees.

```



Ce code est dans "*Exercice3.sage*".

## 4

. On essaie de de te donner la forme algébrique normale de la fonction **Maj**. On construit un tableau avec les valeurs possibles et les résultats respectifs, on a donc :

| $a$ | $b$ | $c$ | <b>Maj</b> ( $a, b, c$ ) |
|-----|-----|-----|--------------------------|
| 0   | 0   | 0   | 0                        |
| 0   | 0   | 1   | 0                        |
| 0   | 1   | 0   | 0                        |
| 0   | 1   | 1   | 1                        |
| 1   | 0   | 0   | 0                        |
| 1   | 0   | 1   | 1                        |
| 1   | 1   | 0   | 1                        |
| 1   | 1   | 1   | 1                        |

On voit d'abord que le **poids de Hamming** de  $Maj$  est équilibré :  $w(Maj) = 4$ . On va utiliser la forme algébrique normale avec les supports de  $Maj$ . On a comme support  $(0, 1, 1), (1, 0, 1), (1, 1, 0)$  et  $(1, 1, 1)$ . Les calculs sont :

$$(a + 1) \times b \times c + a \times (b + 1) \times c + a \times b \times (c + 1) + a \times b \times c$$

$\Leftrightarrow$

$$\cancel{a \times b \times c} + b \times c + \cancel{a \times b \times c} + a \times c + \cancel{a \times b \times c} + a \times b + \cancel{a \times b \times c}$$

$\Leftrightarrow$

$$a \times b + a \times b + a \times c$$

D'après le tableau ci-dessous, et sa forme algébrique normale, on voit que la fonction **Maj**( $a, b, c$ ) nous donne cette formule :

$$\mathbf{Maj}(a, b, c) = a \times b + b \times c + a \times c$$

On voit qu'avec la formule ci-dessous, pour calculer le bit sortant de l'algorithme **A5/2-step**. On calcule d'abord  $y_1$  de cette façon :

$$y_1 = R_{1,0} + \mathbf{Maj}(R_{1,3}, R_{1,4} + 1, R_{1,6})$$

et on a que

$$\mathbf{Maj}(R_{1,3}, R_{1,4} + 1, R_{1,6}) = R_{1,3} * (R_{1,4} + 1) + (R_{1,4} + 1) * R_{1,6} + R_{1,3} * R_{1,6}$$

donc :

$$y_1 = R_{1,0} + R_{1,3} * (R_{1,4} + 1) + (R_{1,4} + 1) * R_{1,6} + R_{1,3} * R_{1,6}$$

$\Leftrightarrow$

$$y_1 = R_{1,0} + R_{1,3} * R_{1,4} + R_{1,3} + R_{1,4} * R_{1,6} + R_{1,6} + R_{1,3} * R_{1,6}$$

et c'est pareil pour  $y_2$  et  $y_3$  :  
on a donc :

$$y_2 = R_{2,0} + R_{2,8} * R_{2,5} + R_{2,8} + R_{2,5} * R_{2,12} + R_{2,12} + R_{2,8} * R_{2,12}$$

$$y_3 = R_{3,0} + R_{3,4} * R_{3,9} + R_{3,4} + R_{3,9} * R_{3,6} + R_{3,6} + R_{3,4} * R_{3,9}$$

et on additionne les 3 variables pour avoir un bit sortant :

$$\begin{aligned} y_1 + y_2 + y_3 = \\ R_{1,0} + R_{2,0} + R_{3,0} + R_{1,3} + R_{1,6} + R_{2,8} + R_{2,12} + R_{3,4} + R_{3,6} + R_{1,3} * R_{1,4} + R_{1,4} * R_{1,6} + R_{1,3} * R_{1,6} \\ + R_{2,8} * R_{2,5} + R_{2,5} * R_{2,12} + R_{2,8} * R_{2,12} + R_{3,4} * R_{3,9} + R_{3,9} * R_{3,6} + R_{3,4} * R_{3,9} \end{aligned}$$

On remplace par les variables  $x_i$  donnée dans l'énoncé :

$$\begin{aligned} y_1 + y_2 + y_3 = \\ x_0 + x_{19} + x_{41} + x_3 + x_6 + x_{27} + x_{31} + x_{45} + x_{47} + x_3 * x_4 + x_4 * x_6 + x_3 * x_6 \\ + x_{27} * x_{24} + x_{24} * x_{31} + x_{27} * x_{31} + x_{45} * x_{50} + x_{50} * x_{47} + x_{45} * x_{47} \end{aligned}$$

Et de plus avec la mise à jour des registres. On voit bien des pour chaque bit produit peuvent s'exprimer par des équations quadratiques en les  $x_i$  à cause des monômes qui sont de degrés de 2 de la forme algébrique normale de *Maj*.

## 5

```

1 load('Exercice1.sage')
2
3 def Maj_al_n(a,b,c): # fonction majorit en forme algbrique normale.
4     return a*b +b*c +a*c
5
6
7 def A5_2_stepv_2(R1, R2, R3, R4, f1, f2, f3, f4):
8     # On utilise la fonction majorit en forme algbrique normale.
9
10    # On calcule la fonction majorit
11    # sur les indices 6, 13 et 9 de R4.
12    m = Maj_al_n(R4[6], R4[13], R4[9])
13
14
15    # On a 3 choix possibles.
16    # Si l'indice 6 de R4 est gale m
17    # on met jour R1 en ignorant sa sortie.
18
19    if R4[6] == m :
20        _, R1 = LFSR_step(R1, f1)
21
22
23
24    # Si l'indice 13 de R4 est gale m
25    # on met jour R2 en ignorant sa sortie.
26    if R4[13] == m :
27        _, R2 = LFSR_step(R2, f2)
28
29
30
31    # Si l'indice 9 de R4 est gale m
32    # on met jour R3 en ignorant sa sortie.
33    if R4[9] == m :
34        _, R3 = LFSR_step(R3, f3)
35
36
37
38
39
40    # on met jour R4 en ignorant la sortie.
41    _, R4 = LFSR_step(R4, f4)
42
43    # On calcule y1, y2 et y3 avec les formules donnees.
44    y1 = R1[0] + Maj_al_n(R1[3], R1[4] + 1, R1[6])
45    y2 = R2[0] + Maj_al_n(R2[8], R2[5] + 1, R2[12])
46    y3 = R3[0] + Maj_al_n(R3[4], R3[9] + 1, R3[6])
47
48    # On ressort le rsultat de l'additions des 3 variables
49    return (y1 + y2 + y3),(R1, R2, R3, R4)
50
51
52 def A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4): # On utilise la fonction majorit en forme
53     algbrique normale
54     # Dans cette version on rajoute N pour pouvoir utiliser avec n'importe quel N.
55     # On excute 99 fois la fonction "A5_2_step" en ignorant
56     # son bit de sortie
57     for i in range(99):
58         _, (R1 ,R2 ,R3 ,R4) = A5_2_stepv_2(R1, R2, R3, R4, f1, f2, f3, f4)
59
60     # On prend z une suite iniatis une liste vide.
61     z = []

```

```

62     # On excute N fois la fonction "A5_2_step" en utilisant son
63     # bit de sortie.
64     for i in range(N):
65         zi, (R1 ,R2 ,R3 ,R4) = A5_2_stepv_2(R1, R2, R3, R4, f1, f2, f3, f4)
66         z.append(zi)
67
68
69     return z
70
71
72 def A5_2v_2(N,K, IV):
73
74     # On cre les polynomes de rtraoction
75     PR.<X> = PolynomialRing(GF(2))
76     f1 = X**19 + X**18 + X**17 + X**14 + 1
77     f2 = X**22 + X**21 + 1
78     f3 = X**23 + X**22 + X**21 + X**8 + 1
79     f4 = X**17 + X**12 + 1
80
81     # Phase d'initialisation.
82     R1, R2, R3, R4 = A5_2_init(K, IV, f1, f2, f3, f4)
83
84
85     # Phase de productions.
86     return A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
87
88     # On suppose qu'on connait R4 et donc on va utiliser les K et IV donnees pour l'exemple du (1)
89
90     # On utilise les mmes polynmes de rtraoctions que l'algorithmme "A5_2_init"
91     PR.<X> = PolynomialRing(GF(2))
92     f1 = X**19 + X**18 + X**17 + X**14 + 1
93     f2 = X**22 + X**21 + 1
94     f3 = X**23 + X**22 + X**21 + X**8 + 1
95     f4 = X**17 + X**12 + 1
96
97     # On cre le R4 que l'on connait.
98     _, _, _, R4 = A5_2_init(K, IV, f1, f2, f3, f4)
99
100
101     # Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
102     BPR = BooleanPolynomialRing(64, 'x')
103     v = BPR.gens()
104
105     # On cre les R1, R2 et R3 avec les inconnues associs.
106     R1 = Sequence([v[i] for i in range(19)])
107     R2 = Sequence([v[i] for i in range(19,41)])
108     R3 = Sequence([v[i] for i in range(41,64)])
109
110     # On veut N =228
111     N = 228
112
113     # On calcule la suite chiffri z avec ses 64 inconnues.
114     zprime = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
115
116     # On a cre une liste d'equations quadratiques et on l'affiche les 228 equations.
117     print("Taille de la liste d'equation =", len(zprime))
118
119     # On voit les 228 equations.
120     print("Les %d equations sont : " %(N))
121     for i in range(228):
122         print("zprime[%d] = %s" %(i, zprime[i]))

```

Ce code est dans "*Exercice5.sage*".

## 6

. D'après le (4), on sait que c'est de la forme quadratique en les  $x_i$ . Pour  $R_1$ , on a un bit connu sur les 19 possibles, on a donc 18 inconnues. Après la phase de production, on a : 18 monômes de degré 1 et pour les monômes degré 2 :

$$\binom{18}{2} = \frac{18 * 17}{2}$$

$$\binom{18}{2} = 153$$

on a pour  $R_1$  :  $153 + 18 = 171$  monômes possibles. de même pour  $R_2$  :

$$\binom{21}{2} + 21 = 231$$

et e même pour  $R_3$  :

$$\binom{22}{2} + 22 = 253$$

On a

$$171 + 231 + 253 = 655$$

On trouve bien 655 monômes possibles en sachant qu'il y'a trois  $x_i$  connus.

```

1 # Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
2 BPR = BooleanPolynomialRing(64, 'x')
3 v = BPR.gens()
4
5 # On sait x3 = x24 = x45 = 1, sont les variables connues, on cre M1 la liste des monmes
6 # de degr 1 sans ses 3 variables.
7 M1 = [v[i] for i in range(3)]+[v[i] for i in range(4,24)]+[v[i] for i in range(25,45)]+[v[i] for i in
      range(46,64)]
8 len(M1)
9
10 # On cre M la liste de tous les monmes possibles.
11 M = [M1[i] for i in range(61)]+[M1[i]*M1[j] for i in range(18) for j in range(i+1,18)]+[M1[i]*M1[j]
      for i in range(18,39) for j in range(i+1,39)]+[M1[i]*M1[j] for i in range(39,61) for j in range(
      i+1,61)]
12 print("M =", M)
13 print("La longueur de M est :", len(M))
14 # On voit que sa longueur est bien 655
15 N = 228
16 L = 655

```

Ce code est dans "*Exercice6.sage*".

## 7

```

1 load('Exercice5.sage')
2 load('Exercice6.sage')
3
4 # On suppose qu'on connait R4 et donc on va utiliser les K et IV donnees pour l'exemple du (1)
5
6 # On utilise les mmes polynomes de rtroactions que l'algorithme "A5_2_init"
7 PR.<X> = PolynomialRing(GF(2))
8 f1 = X**19 + X**18 + X**17 + X**14 + 1
9 f2 = X**22 + X**21 + 1
10 f3 = X**23 + X**22 + X**21 + X**8 + 1
11 f4 = X**17 + X**12 + 1
12
13 # On cre le R4 que l'on connait.
14 _, _, _, R4 = A5_2_init(K, IV, f1, f2, f3, f4)
15
16
17 # Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
18 BPR = BooleanPolynomialRing(64, 'x')
19 v = BPR.gens()
20
21 # On cre les R1, R2 et R3 avec les inconnues associs et leurs bits connus.
22 R1 = Sequence([v[i] for i in range(3)]+[1]+[v[i] for i in range(4,19)])
23 R2 = Sequence([v[i] for i in range(19,24)]+[1]+[v[i] for i in range(25,41)])
24 R3 = Sequence([v[i] for i in range(41,45)]+[1]+[v[i] for i in range(46,64)])
25
26 # On calcule la suite chiffr z avec ses 64 inconnues.
27 zprimev2 = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
28
29 # On a cre une liste d'equations quadratiques et on l'affiche les 228 equations.
30 print("Taille de la liste d'equation =", len(zprimev2))
31
32 # On voit les 228 equations.
33 print("Les %d equations sont : " %(N))
34 for i in range(228):
35     print("zprime%d = %s" %(i, zprimev2[i]))
36
37 print("\n")
38 # On cre la matrice Mat de taille N*L dans F2.
39 Mat = matrix(GF(2),N,L)
40
41 # On cre le vecteur de longueur N.
42 Vec = vector(GF(2),N)
43
44 # On utilise le zprime calcul dans le (5).
45 # On utilise les indications de l'nonc.
46 for i in range(N):
47     Vec[i] = zprimev2[i].monomial_coefficient(1)
48     for j in range(L):
49         Mat[i,j] = zprimev2[i].monomial_coefficient(M[j])
50
51 print("Vec =", Vec)
52 print("\n")
53 print("Mat =", Mat)

```

Ce code est dans "*Exercice7.sage*".

## 8

```

1 load('Exercice5.sage')
2 load('A5_2-700.sage')
3
4 N = 700
5 # On considere une excution de A5/2 donnant N = 700 bits de suite chiffrante avec
6 # On a la valeur de R4 aprs la phase d'initialisation
7 # et z la suite chiffrante de 700 bits
8
9 # Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
10 BPR = BooleanPolynomialRing(64, 'x')
11 v = BPR.gens()
12
13 # On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
14 PR.<X> = PolynomialRing(GF(2))
15 f1 = X**19 + X**18 + X**17 + X**14 + 1
16 f2 = X**22 + X**21 + 1
17 f3 = X**23 + X**22 + X**21 + X**8 + 1
18 f4 = X**17 + X**12 + 1
19
20 # On cre les R1, R2 et R3 avec les inconnues associs et leurs bits connus.
21 R1 = Sequence([v[i] for i in range(3)]+[1]+[v[i] for i in range(4,19)])
22 R2 = Sequence([v[i] for i in range(19,24)]+[1]+[v[i] for i in range(25,41)])
23 R3 = Sequence([v[i] for i in range(41,45)]+[1]+[v[i] for i in range(46,64)])
24
25 # On calcule la suite chiffr z avec ses 64 inconnues.
26 zprime = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
27
28 # On a cre une liste d'equations quadratiques et on l'affiche les 700 equations.
29 print("Taille de la liste d'equation =", len(zprime))
30
31 # On voit les 700 equations.
32 print("Les %d equations sont : " %(N))
33 for i in range(N):
34     print("zprime[%d] = %s" %(i, zprime[i]))
35
36 print("\n")
37
38 # On cre la matrice Mat de taille N*L dans F2.
39 Mat = matrix(GF(2),N,L)
40
41 # On cre le vecteur de longueur N.
42 Vec = vector(GF(2),N)
43
44 # On utilise le zprime .
45 # On utilise les indications de l'nonc.
46 for i in range(N):
47     Vec[i] = zprime[i].monomial_coefficient(1)
48     for j in range(L):
49         Mat[i,j] = zprime[i].monomial_coefficient(M[j])
50
51 print("Vec =", Vec)
52 print("\n")
53 print("Mat =", Mat)
54 print("\n")
55
56 # Pour rsoudre l'equation lineaire, on va utiliser la fonction solve_right() donne dans l'nonc.
57 # On additionne vec z.
58 Right = [Vec[i] + z[i] for i in range(N)]
59 #on rsoud l'equation grce la fonction, en mettant tous les inconnues gauche de l'galit
60 # et le reste droite, on pose sol le resultat de la fonction:
61 Sol = Mat.solve_right(Right)
62 print("Sol =",Sol)

```



```

63 print("\n")
64
65 #On a besoin que les 61 premiers bits du sol pour rcuprer
66 # R1, R2 et R3 :
67 R1 = Sequence([Sol[i] for i in range(3)]+[1]+[Sol[i] for i in range(3,18)])
68 R2 = Sequence([Sol[i] for i in range(18,23)]+[1]+[Sol[i] for i in range(23,39)])
69 R3 = Sequence([Sol[i] for i in range(39,43)]+[1]+[Sol[i] for i in range(43,61)])
70 print("R1 =", R1)
71 print("R2 =", R2)
72 print("R3 =", R3)
73 print("R4 =", R4)
74 print("\n")
75
76 # On teste si a nous donne le bon rsultat:
77 ztest = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
78 print(" z = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4) est", ztest == z)

```

Ce code est dans "*Exercice8.sage*".

## 9

. Avant la première boucle **for**, on a

$$R1 = \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 0 \end{pmatrix}$$

Soit  $A_1$  la matrice de rétroaction du  $LF SR1$ . Dans la boucle **for**, à  $i = 0$  :  
On met à jour  $R1$  :

$$R1 = A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 0 \end{pmatrix}$$

On rajoute  $K_0$ , On a que :

$$R1 = A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_0 \end{pmatrix}$$

En temps  $i = 1$  , on multiplie par la matrice  $A_1$  puis on additionne par  $K_1$  :

$$R1 = A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_0 \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_1 \end{pmatrix}$$

De même pour  $i = 2$  :

$$R1 = A_1 \times (A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_0 \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_1 \end{pmatrix}) + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_2 \end{pmatrix}$$

$\Leftrightarrow$

$$R1 = A_1^2 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_0 \end{pmatrix} + A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_1 \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_2 \end{pmatrix}$$

donc on a pour  $i = k$  avec  $0 \leq k \leq 63$  :

$$R1 = A_1^k \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_0 \end{pmatrix} + A_1^{k-1} \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_1 \end{pmatrix} + \dots + A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_{k-1} \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_k \end{pmatrix}$$

et donc à la fin de la première boucle **for** :

$$R1 = A_1^{63} \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_0 \end{pmatrix} + A_1^{62} \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_1 \end{pmatrix} + \dots + A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_{62} \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_{63} \end{pmatrix}$$

$\Leftrightarrow$

$$R1 = \sum_{i=0}^{63} A_1^i \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ K_{63-i} \end{pmatrix}$$

On pose  $X = R1$  avant la deuxième boucle **for**. Dans la deuxième boucle **for**, au temps  $i = 0$  :

$$R1 = A_1 X + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ IV_0 \end{pmatrix}$$

au temps  $i = 1$  :

$$R1 = A_1 \times (A_1 X + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_0 \end{pmatrix}) + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_1 \end{pmatrix}$$

$\Leftrightarrow$

$$R1 = A_1^2 X + A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_0 \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_1 \end{pmatrix}$$

donc on a pour  $i = k$  avec  $0 \leq k \leq 21$  :

$$R1 = A_1^{k+1} X + A_1^k \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_0 \end{pmatrix} + A_1^{k-1} \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_1 \end{pmatrix} + \dots + A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_{k-1} \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_k \end{pmatrix}$$

Et donc à la fin de la deuxième boucle **for** :

$$R1 = A_1^{22} X + A_1^{21} \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_0 \end{pmatrix} + A_1^{20} \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_1 \end{pmatrix} + \dots + A_1 \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_{20} \end{pmatrix} + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_{21} \end{pmatrix}$$

$\Leftrightarrow$

$$R1 = A_1^{22} X + \sum_{i=0}^{21} A_1^i \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_{21-i} \end{pmatrix}$$

On a bien montré les deux égalité de l'énoncé.

```

1 load('Exercice5.sage')
2 load('A5_2-700.sage')
3 # On recupre les Ri de l'exercice 8:
4 R1 = [1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0]
5 R2 = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1]
6 R3 = [1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0]
7 R4 = [1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1]
8
9 # Son IV est nulle
10 IV = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
11
12
13 def matf(f):# matrice de transition
14     n = f.degree()
15     A = Matrix(GF(2),n,n)
16     for i in range(n-1):
17         A[i,i+1] = 1
18         tmp = f.list()[1:] # c_1,c_2,...,c_L
19         tmp.reverse() # c_L,c_{L-1}, ..., c_1
20         A[n-1] = vector(tmp)
21     return A
22
23
24 # On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
25 PR.<X> = PolynomialRing(GF(2))
26 f1 = X**19 + X**18 + X**17 + X**14 + 1
27 f2 = X**22 + X**21 + 1
28 f3 = X**23 + X**22 + X**21 + X**8 + 1
29 f4 = X**17 + X**12 + 1
30
31 # On construit les matrices de retroactions pour chaque fonction de rtroaction
32 A1 = matf(f1)
33 A2 = matf(f2)
34 A3 = matf(f3)
35 A4 = matf(f4)
36
37
38 # Pour dclarer les 64 inconnues que l'on utilisera dans la clef K.
39 BPR = BooleanPolynomialRing(64, 'K')
40 v = BPR.gens()
41
42 # Comme dans l'algorithme de A5/2-init les registres sont vides des 4 LFSR sont nulles.
43 X1 = vector([0 for _ in range(19)]) # registre du LFSR1
44 X2 = vector([0 for _ in range(22)]) # registre du LFSR2
45 X3 = vector([0 for _ in range(23)]) # registre du LFSR3
46 X4 = vector([0 for _ in range(17)]) # registre du LFSR4
47
48 # On calcule les registres comme dans le rsultat du (9).
49 # Dans la premiere boucle for:
50 for i in range(64):
51     X1 = A1*X1+vector([0 for _ in range(18)]+[v[i]])
52     X2 = A2*X2+vector([0 for _ in range(21)]+[v[i]])
53     X3 = A3*X3+vector([0 for _ in range(22)]+[v[i]])
54     X4 = A4*X4+vector([0 for _ in range(16)]+[v[i]])
55
56 # Comme IV est nulle, dans la deuxime boucle for on multiplie pour chaque registre par leurs
    matrices
57 # de rtroaction qui sont mis la puissance 22.
58 X1 = A1**22*X1
59 X2 = A2**22*X2
60 X3 = A3**22*X3
61 X4 = A4**22*X4

```

```

62
63 # A la fin de cet algorithme, on met un seul bit 1 dans chaque registre :
64 X1[3] = 1
65 X2[5] = 1
66 X3[4] = 1
67 X4[6] = 1
68
69 # On fait une liste quation
70 L = list(X1)+list(X2)+list(X3)+list(X4)
71
72 # On concatne les registres trouvs dans le (8)
73 R = list(R1)+list(R2)+list(R3)+list(R4)
74
75 # On va calculer l'idal :
76 I = ideal([L[i] + R[i] for i in range(81)])
77 print("I lidal :",I)
78 print("\n")
79 # On utilise la base groebner.
80 Iq = I.groebner_basis();
81 print("list(IL.groebner_basis() :", list(Iq))
82 print("\n")
83
84 # On cherche les coefficients qui additionnent les variables pour avoir une clef Kprime.
85 Kprime = [el.constant_coefficient() for el in Iq]
86 print("Kprime =", Kprime)
87 print("\n")
88
89 #On calcule avec Kprime pour chercher les Rprime avec 1<=i<=4
90 R1prime, R2prime, R3prime,R4prime =A5_2_init(Kprime, IV, f1, f2, f3, f4)
91
92 # On teste si a nous donne les bons rsultats:
93 print("R1prime = R1 est ", R1prime == R1)
94 print("R2prime = R2 est ", R2prime == R2)
95 print("R3prime = R3 est ", R3prime == R3)
96 print("R4prime = R4 est ", R4prime == R4)
97 print("On a trouv le bon K")

```

Ce code est dans "*Exercice10.sage*".

# 11

. On se place après la phase d'initialisation de l'exécution  $A5/2$ , d'après le (9). On a que pour tout  $R_i$  avec  $1 \leq i \leq 4$  et  $A_i$  leurs matrices de rétroactions respectifs. On a donc à la fin de la boucle du premier **for** :

$$R_i = \sum_{k=0}^{63} A_i^k \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_{63-k} \end{pmatrix}$$

Or pour la première suite chiffrante  $z_0$ , son vecteur d'initialisation  $IV$  est nulle. On a donc à la fin de la boucle du second **for** :

$$R_i = A_i^{22} \times \left( \sum_{k=0}^{63} A_i^k \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_{63-k} \end{pmatrix} \right)$$

mais si  $IV$  n'est pas nulle, on autre résultat qui est :

$$R_i = A_i^{22} \times \left( \sum_{k=0}^{63} A_i^k \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ K_{63-k} \end{pmatrix} \right) + \sum_{i=0}^{21} A_1^i \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ IV_{21-i} \end{pmatrix}$$

Pour pouvoir différencier les  $R_i$  des trois suites chiffrante :  
 pour  $z_0$  ça sera  $R_i$ ,  
 pour  $z_1$  ça sera  $R'_i$ ,  
 pour  $z_2$  ça sera  $\tilde{R}_i$ .

Dans l'énoncé pour la suite chiffrante  $z_1$  a comme  $IV = (0, \dots, 0, 1)$  on a donc d'après les résultats précédents, on a que :

$$R'_i = R_i + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 1 \end{pmatrix}$$

et de même avec  $z_2$  qui a comme  $IV = (0, \dots, 1, 0)$

$$\tilde{R}_i = R_i + A_i \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \\ 0 \end{pmatrix}$$

On connaît  $R_4$  , donc on pourra calculer  $R'_4$  et  $\tilde{R}_4$  sans oublier le bit mis à 1, pour pouvoir commencer la phase **A5/2-step**. Et faire de même avec les autres  $R_i$  avec ses formules : pour  $1 \leq i \leq 4$

$$R'_i = R_i + \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 1 \end{pmatrix}$$

$$\tilde{R}_i = R_i + A_i \times \begin{pmatrix} 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \\ 0 \end{pmatrix}$$

avec

$$R_{1,3} = R'_{1,3} = \tilde{R}_{1,3} = 1$$

$$R_{2,5} = R'_{2,5} = \tilde{R}_{2,5} = 1$$

$$R_{3,4} = R'_{3,4} = \tilde{R}_{3,4} = 1$$

$$R_{4,6} = R'_{4,6} = \tilde{R}_{4,6} = 1$$



## 12

. On considère  $z_0, z_1$  et  $z_3$  des suites chiffrantes produites par l'algorithme **A5/2-production**(phase de productions) par leurs registres respectifs,  $R_i, R'_i$  et  $\tilde{R}_i$  avec  $1 \leq i \leq 4$ .

D'après le (8), (9), (10) et (11), on en déduit deux étapes, la première étape est de chercher la valeur des registres  $R_i$  avec  $1 \leq i \leq 4$  puis la deuxième étape grâce aux valeurs obtenus des  $R_i$ , on cherchera la valeur de la clef  $K$ .

Dans la première étape de l'attaque :

On commence à déclarer les 64 inconnus de  $R_1, R_2$  et  $R_3$  avec

$$R1 = (x_0, \dots, x_{18})$$

$$R2 = (x_{19}, \dots, x_{40})$$

$$R3 = (x_{41}, \dots, x_{63})$$

On crée la matrice de transition avec leurs polynômes de rétroactions respectifs pour chaque registre  $R_i$  avec  $1 \leq i \leq 4$ . On les nomment  $A_i$  avec  $1 \leq i \leq 4$  qui sont les matrices respectifs  $LFSR_i$ .

On calcule les  $R'_i$  avec  $1 \leq i \leq 4$  d'après le (11) puis on impose :

$$R'_{1,3} = R'_{2,5} = R'_{3,4} = R'_{4,6} = 1$$

puis on calcule  $\tilde{R}_i$  avec  $1 \leq i \leq 4$  d'après le (11) puis on impose :

$$\tilde{R}_{1,3} = \tilde{R}_{2,5} = \tilde{R}_{3,4} = \tilde{R}_{4,6} = 1$$

On pose  $N = 228$ , on utilise l'algorithme **A5/2-production**(phase de productions) pour construire  $z, z'$  et  $\tilde{z}$  qui sont respectivement les suites chiffrantes théoriques de  $z_0, z_1$  et  $z_2$ .

On concaténâtes  $z, z'$  et  $\tilde{z}$ , on pose :

$$Eq = [z, z', \tilde{z}]$$

on fait la même opération avec  $z_0, z_1$  et  $z_2$ , on pose :

$$Res = [z_0, z_1, z_2]$$

. On crée une liste de monômes d'après le (6), on sait que la taille de cette liste est 655, on pose  $L = 655$ . Ensuite on crée une matrice  $Mat$  de taille  $N \times L$  dans  $F^2$  et le vecteur  $Vec$  de taille  $3N$  comme d'après le (7). On crée une liste  $Right$  qui est :

$$Right[i] = Res[i] + Vec[i]$$

avec  $0 \leq i < 3N$ . On pose :

$$Y = (Y_0, Y_1, \dots, Y_{654})$$

On calcule cette équation :

$$Mat \times Y = Right$$

A la fin de ce calcul, on prend les 61 premiers valeurs de la solutions, on a donc :

$$R_1 = [Y_0, Y_1, Y_2, 1, Y_3, \dots, Y_{17}]$$

$$R_2 = [Y_{18}, \dots, Y_{22}, 1, Y_{23}, \dots, Y_{38}]$$

$$R_3 = [Y_{39}, \dots, Y_{42}, 1, Y_{43}, \dots, Y_{60}]$$

On teste si on a bien les résultats voulus.

On commence la deuxième étape pour rechercher la clef  $K$ , on utilisera les  $R_i$  avec  $1 \leq i \leq 4$  car on sait que son  $IV$  est nulle. On pose les 64 inconnus de la clef  $K^*$  qui est une clef théorique de la vrai clef  $K$ , on calcule avec l'algorithme  $A5/2 - init$  pour avoir nos  $R_i$

théoriques en sachant que son  $IV$  est nulle. On concatène les  $Rtheorique_i$ ,  $1 \leq i \leq 4$  on a donc :

$$L = [Rtheorique_1, Rtheorique_2, Rtheorique_3, Rtheorique_4]$$

On On concatène aussi les  $R_i$  avec  $1 \leq i \leq 4$  :

$$R = [R_1, R_2, R_3, R_4]$$

On calcul  $I = ideal(L[i] + R[i])$  pour  $i$  entre 0 et 81), puis on calcul la base de **Groebner**. On trouve la clef  $K^*$  puis on teste pour voir si elle est égale à la clef  $K$  qu'on cherche.

## 13

```

1 load('Exercice5.sage')
2 load('A5_2-3frames.sage')
3
4 IV= Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]) # pour z0
5 IVprime = Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]) # pour z1
6 IVtilde = Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]) # pour z2
7
8 # On pose les registres des LFSRi Ri pour z0, Rprime pour z1 et Rtilde pour z2 avec 1 <= i
  <= 4
9 # Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2, R3, R1prime,
  R2prime, R3prime
10 # R1tilde, R2tilde et R3tilde
11 BPR = BooleanPolynomialRing(64, 'x')
12 v = BPR.gens()
13
14 # On cre les R1, R2 et R3 avec les inconnues associs et leurs bits connus.
15 R1 = Sequence([v[i] for i in range(3)]+[1]+[v[i] for i in range(4,19)])
16 R2 = Sequence([v[i] for i in range(19,24)]+[1]+[v[i] for i in range(25,41)])
17 R3 = Sequence([v[i] for i in range(41,45)]+[1]+[v[i] for i in range(46,64)])
18
19 print("R1 =",R1)
20 print("R2 =",R2)
21 print("R3 =",R3)
22 print("R4 =",R4)
23 print("\n")
24
25 def matf(f):# matrice de transition
26     n = f.degree()
27     A = Matrix(GF(2),n,n)
28     for i in range(n-1):
29         A[i,i+1] = 1
30     tmp = f.list()[1:] # c_1,c_2,...,c_L
31     tmp.reverse() # c_L,c_{L-1}, ..., c_1
32     A[n-1] = vector(tmp)
33     return A
34
35
36 # On utilise les mmes polynmes de rtroactions que l'algorithmme "A5_2_init"
37 PR.<X> = PolynomialRing(GF(2))
38 f1 = X**19 + X**18 + X**17 + X**14 + 1
39 f2 = X**22 + X**21 + 1
40 f3 = X**23 + X**22 + X**21 + X**8 + 1
41 f4 = X**17 + X**12 + 1
42
43 # On construit les matrices de retroactions pour chaque fonction de rtroaction
44 A1 = matf(f1)
45 A2 = matf(f2)
46 A3 = matf(f3)
47 A4 = matf(f4)
48
49 # Pour faire les calculs je transforme les registres en vecteur.
50 X1 = vector(R1)
51 X2 = vector(R2)
52 X3 = vector(R3)
53 X4 = vector(R4)
54
55 # On calcul les registres des autres suites chiffrantes.
56 # On commence par celle de z1:
57 X1prime = X1 + vector([0 for _ in range(18)]+[IVprime[21]])
58 X2prime = X2 + vector([0 for _ in range(21)]+[IVprime[21]])
59 X3prime = X3 + vector([0 for _ in range(22)]+[IVprime[21]])
60 X4prime = X4 + vector([0 for _ in range(16)]+[IVprime[21]])

```

```

61
62 # on met un seul bit 1 dans chaque registre :
63 X1prime[3] = 1
64 X2prime[5] = 1
65 X3prime[4] = 1
66 X4prime[6] = 1
67
68 # On le remet en sequence
69 R1prime = Sequence(X1prime)
70 R2prime = Sequence(X2prime)
71 R3prime = Sequence(X3prime)
72 R4prime = Sequence(X4prime)
73
74 print("R1prime =",R1prime)
75 print("R2prime =",R2prime)
76 print("R3prime =",R3prime)
77 print("R4prime =",R4prime)
78 print("\n")
79
80 # et celle de z2:
81 X1tilde = X1 + A1 * vector([0 for _ in range(18)]+[IVtilde[20]])
82 X2tilde = X2 + A2 * vector([0 for _ in range(21)]+[IVtilde[20]])
83 X3tilde = X3 + A3 * vector([0 for _ in range(22)]+[IVtilde[20]])
84 X4tilde = X4 + A4 * vector([0 for _ in range(16)]+[IVtilde[20]])
85
86 # on met un seul bit 1 dans chaque registre :
87 X1tilde[3] = 1
88 X2tilde[5] = 1
89 X3tilde[4] = 1
90 X4tilde[6] = 1
91
92 # On le remet en sequence
93 R1tilde = Sequence(X1tilde)
94 R2tilde = Sequence(X2tilde)
95 R3tilde = Sequence(X3tilde)
96 R4tilde = Sequence(X4tilde)
97
98 print("R1tilde =",R1tilde)
99 print("R2tilde =",R2tilde)
100 print("R3tilde =",R3tilde)
101 print("R4tilde =",R4tilde)
102 print("\n")
103
104 # On pose N = 228
105 N = 228
106 # On calcule les z thoriques
107 z = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
108 zprime = A5_2_productionv_2(N, R1prime, R2prime, R3prime, R4prime, f1, f2, f3, f4)
109 ztilde = A5_2_productionv_2(N, R1tilde, R2tilde, R3tilde, R4tilde, f1, f2, f3, f4)
110
111
112 # On cre une liste d'equations
113 Eq = z+zprime+ztilde
114
115 # On cre une liste de rsultats voulues
116 Res = z0+z1+z2
117
118 # On sait x3 = x24 = x45 = 1, sont les variables connues, on cre M1 la liste des monmes
119 # de degr 1 sans ses 3 variables.
120 M1 = [v[i] for i in range(3)]+[v[i] for i in range(4,24)]+[v[i] for i in range(25,45)]+[v[i] for i in
    range(46,64)]
121
122 # On cre M la liste de tous les monmes possibles.
123 M = [M1[i] for i in range(61)]+[M1[i]*M1[j] for i in range(18) for j in range(i+1,18)]+[M1[i]*M1[j]
    for i in range(18,39) for j in range(i+1,39)]+[M1[i]*M1[j] for i in range(39,61) for j in range(
    i+1,61)]
124
125 L = len(M)

```

```

126
127 # On cre la matrice Mat de taille N*L dans F2.
128 Mat = matrix(GF(2),3*N,L)
129
130 # On cre le vecteur de longueur N.
131 Vec = vector(GF(2),3*N)
132
133
134 # On utilise les indications de l'nonc.
135 for i in range(3*N):
136     Vec[i] = Eq[i].monomial_coefficient(1)
137     for j in range(L):
138         Mat[i,j] = Eq[i].monomial_coefficient(M[j])
139
140
141 # On rajoute le Vecteur aux rsultats prvues
142 Right = [Vec[i]+Res[i] for i in range(3*N)]
143
144 Sol = Mat.solve_right(Right)
145
146 #On a besoin que les 61 premiers bits du sol pour rcuprer
147 # R1, R2 et R3 :
148 R1 = Sequence([Sol[i] for i in range(3)]+[1]+[Sol[i] for i in range(3,18)])
149 R2 = Sequence([Sol[i] for i in range(18,23)]+[1]+[Sol[i] for i in range(23,39)])
150 R3 = Sequence([Sol[i] for i in range(39,43)]+[1]+[Sol[i] for i in range(43,61)])
151 print("R1 =", R1)
152 print("R2 =", R2)
153 print("R3 =", R3)
154 print("R4 =", R4)
155 print("\n")
156
157 # Maintenant on calcul les autres Registres qui produisent les deux autres suites chiffrantes.
158 # On transforme les 4 registres trouvs en vecteur pour pouvoir calculer les autres registres.
159 X1 = vector(R1)
160 X2 = vector(R2)
161 X3 = vector(R3)
162 X4 = vector(R4)
163
164 # on calcul les registres qui produisent z1
165 X1prime = X1 + vector([0 for _ in range(18)]+[IVprime[21]])
166 X2prime = X2 + vector([0 for _ in range(21)]+[IVprime[21]])
167 X3prime = X3 + vector([0 for _ in range(22)]+[IVprime[21]])
168 X4prime = X4 + vector([0 for _ in range(16)]+[IVprime[21]])
169
170 # on met un seul bit 1 dans chaque registre :
171 X1prime[3] = 1
172 X2prime[5] = 1
173 X3prime[4] = 1
174 X4prime[6] = 1
175
176 # On le remet en sequence
177 R1prime = Sequence(X1prime)
178 R2prime = Sequence(X2prime)
179 R3prime = Sequence(X3prime)
180 R4prime = Sequence(X4prime)
181
182 print("R1prime =",R1prime)
183 print("R2prime =",R2prime)
184 print("R3prime =",R3prime)
185 print("R4prime =",R4prime)
186 print("\n")
187
188
189 # et celles qui produisentz2:
190 X1tilde = X1 + A1 * vector([0 for _ in range(18)]+[IVtilde[20]])
191 X2tilde = X2 + A2 * vector([0 for _ in range(21)]+[IVtilde[20]])
192 X3tilde = X3 + A3 * vector([0 for _ in range(22)]+[IVtilde[20]])
193 X4tilde = X4 + A4 * vector([0 for _ in range(16)]+[IVtilde[20]])

```

```

194
195 # on met un seul bit 1 dans chaque registre :
196 X1tilde[3] = 1
197 X2tilde[5] = 1
198 X3tilde[4] = 1
199 X4tilde[6] = 1
200
201 # On le remet en sequence
202 R1tilde = Sequence(X1tilde)
203 R2tilde = Sequence(X2tilde)
204 R3tilde = Sequence(X3tilde)
205 R4tilde = Sequence(X4tilde)
206
207 print("R1tilde =", R1tilde)
208 print("R2tilde =", R2tilde)
209 print("R3tilde =", R3tilde)
210 print("R4tilde =", R4tilde)
211 print("\n")
212
213 # on produit ztest et on teste si c'est gale au rsultat voulu
214 ztest = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
215 print("z0 = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4) est ", ztest == z0)
216
217 # On teste les deux suites chiffrantes avec leurs valeurs voulues
218 zprimetest = A5_2_productionv_2(N, R1prime, R2prime, R3prime, R4prime, f1, f2, f3, f4)
219 print("z1 A5_2_productionv_2(N, R1prime, R2prime, R3prime, R4prime, f1, f2, f3, f4) est ",
      zprimetest == z1)
220
221 ztildetest = A5_2_productionv_2(N, R1tilde, R2tilde, R3tilde, R4tilde, f1, f2, f3, f4)
222 print("z2 = A5_2_productionv_2(N, R1tilde, R2tilde, R3tilde, R4tilde, f1, f2, f3, f4) est ",
      ztildetest == z2)
223 print("\n")
224
225 # On voit bien qu'on a les bons registres.
226 # Maintenant on cherche la clef K
227 # Pour dclarer les 64 inconnues que l'on utilisera dans la clef K.
228 BPR = BooleanPolynomialRing(64, 'K')
229 v = BPR.gens()
230
231 # Comme dans l'algorithme de A5/2-init les registres sont vides des 4 LFSR sont nulles.
232 X1 = vector([0 for _ in range(19)]) # registre du LFSR1
233 X2 = vector([0 for _ in range(22)]) # registre du LFSR2
234 X3 = vector([0 for _ in range(23)]) # registre du LFSR3
235 X4 = vector([0 for _ in range(17)]) # registre du LFSR4
236
237 # On calcule les registres comme dans le rsultat du (9).
238 # Dans la premiere boucle for:
239 for i in range(64):
240     X1 = A1*X1+vector([0 for _ in range(18)]+[v[i]])
241     X2 = A2*X2+vector([0 for _ in range(21)]+[v[i]])
242     X3 = A3*X3+vector([0 for _ in range(22)]+[v[i]])
243     X4 = A4*X4+vector([0 for _ in range(16)]+[v[i]])
244
245 # Comme IV est nulle, dans la deuxime boucle for on multiplie pour chaque registre par leurs
    matrices
246 # de rtroaction qui sont mis la puissance 22.
247 X1 = A1**22*X1
248 X2 = A2**22*X2
249 X3 = A3**22*X3
250 X4 = A4**22*X4
251
252 # A la fin de cet algorithme, on met un seul bit 1 dans chaque registre :
253 X1[3] = 1
254 X2[5] = 1
255 X3[4] = 1
256 X4[6] = 1
257
258 # On fait une liste quation

```

```

259 L = list(X1)+list(X2)+list(X3)+list(X4)
260
261 # On concatne les registres trouvs dans le (8)
262 R = list(R1)+list(R2)+list(R3)+list(R4)
263
264 # On va calculer l'idal:
265 I = ideal([L[i] + R[i] for i in range(81)])
266
267 print("L'idal I :", I)
268 print("\n")
269
270 # On utilise la base groebner.
271 Iq = I.groebner_basis()
272 print("List(I.groebner_basis()) :", list(Iq))
273 print("\n")
274
275 # On cherche les coefficients qui additionnent les variables pour avoir une clef Kprime.
276 Kprime = [el.constant_coefficient() for el in Iq]
277 print("Kprime =", Kprime)
278 print("\n")
279
280 # On produit avec la clef et leurs IV respectifs
281 ztest1 = A5_2v_2(N,Kprime, IV)
282 zprimetest1 = A5_2v_2(N,Kprime, IVprime)
283 ztildetest = A5_2v_2(N,Kprime, IVtilde)
284
285 # on teste avec leurs valeurs voulus
286 print("z0 = A5_2v_2(N,Kprime, IV) est ", ztest1 == z0)
287 print("z1 = A5_2v_2(N,Kprime, IVprime) est ", zprimetest1 == z1)
288 print("z2 = A5_2v_2(N,Kprime, IVtilde) est ", ztildetest == z2)

```

Ce code est dans "*Exercice13.sage*".

## 14

. Si on veut attaquer sans connaître le registre  $R_4$  pour retrouver la clef  $K$  en sachant qu'il y'a une comparaison dans la phase de mise à jour des registres ( $A5/2 - step$ ), on doit faire une recherche exhaustive sur  $R_4$  en imposant que  $R_{4,6} = 1$ . Comme ce registre a 17 bits et avec ce qu'on impose, on a donc  $2^{16}$  possibilités pour  $R_4$ . Comme expliqué dans le (12), il y aura bien 2 étapes pour l'attaque.

Pour chaque  $R_4$  possibles, on fait la première partie comme expliquée au (12) sans oublier de calculer leurs  $R'_4$  et leurs  $\tilde{R}_4$ .

Si il y'a une solution à la matrice, on la teste comme la première partie expliqué dans le (12), on recommence jusqu'à avoir le bon résultat et on continue comme la deuxième étape du (12).



15

.

## Chapitre II

# Cryptanalyse de RSA

## 16

```

1  # On cre le Polynomial Ring de Z
2  Pr.<x,y,u> = PolynomialRing(ZZ)
3
4  def gen(k,gamma):
5      # On prend au hasard p le premier nombre premier.
6      p = random_prime(2**k,lbound = 2**(k-1), proof=false)
7
8      # On prend au hasard q le deuxime nombre premier.
9      q = random_prime(2**k,lbound = 2**(k-1), proof=false)
10
11     # Si P et Q sont gal, on recherche pour une autre valeur qui est
12     # premier.
13     if q==p:
14         q=next_prime(q)
15
16     N = p*q
17
18     # On calcule phi(N)= (p-1)*(q-1)
19     phi =(p-1)*(q-1)
20
21     # On calcule d'abord d
22     d = floor(N**gamma)
23
24     # On cherche le plus grand d possible inferieur N^gamma
25     # et qui soit premier avec phi
26     while gcd(phi,d) != 1:
27         d = d-1
28
29     # On calcule e=d^(-1)mod(phi)
30     e = d.inverse_mod(phi)
31     return N,e,d
32
33 print("gen(2048,0.5) =", gen(2048,0.5))

```

Ce code est dans "*Exercice16.sage*". Au cas que mes programmes en sage ne fonctionnent pas, je vous ai laissé même la version notebook jupiter, ce code est "*RSA.ipynb*" (Toute la partie *RSA*).

## 17

. D'après l'énoncé on a :

$$e \times d = 1 + x_0 \times \varphi(n) \text{ et } y_0 = -(p + q)$$

On a aussi  $A = N + 1$  et la fonction  $f$  :

$$f(x, y) = 1 + x \times (A + y) \bmod(e)$$

On commence par calculer  $x_0$ , on a :

$$x_0 = (e \times d - 1) \times \varphi(N)^{-1}$$

On remplace  $x_0$  et  $y_0$  dans la fonction  $f$  :

$$f(x_0, y_0) = 1 + x_0 \times (A + y_0)$$

$$\text{en sachant que } x_0 = (e \times d - 1) \times \varphi(N)^{-1} \text{ et } y_0 = -(p + q)$$

$\Leftrightarrow$

$$f(x_0, y_0) = 1 + (e \times d - 1) \times \varphi(N)^{-1} \times (A - (p + q))$$

$\Leftrightarrow$

$$f(x_0, y_0) = 1 + (e \times d - 1) \times \varphi(N)^{-1} \times (N + 1 - p - q)$$

or on sait que  $N = p \times q$

$\Leftrightarrow$

$$f(x_0, y_0) = 1 + (e \times d - 1) \times \varphi(N)^{-1} \times (p \times q - p - q + 1)$$

$\Leftrightarrow$

$$f(x_0, y_0) = 1 + (e \times d - 1) \times \varphi(N)^{-1} \times ((p - 1) \times (q - 1))$$

$\Leftrightarrow$

$$f(x_0, y_0) = 1 + (e \times d - 1) \times \varphi(N)^{-1} \times \varphi(N)$$

$\Leftrightarrow$

$$f(x_0, y_0) = 1 + (e \times d - 1)$$

$\Leftrightarrow$

$$f(x_0, y_0) = e \times d$$

$\Leftrightarrow$

$$f(x_0, y_0) = 0 \bmod(e)$$

$(x_0, y_0)$  est bien une racine du polynôme  $f(x, y)$ .

On suppose que :

$$e \approx \varphi(N) \approx N$$

On a que  $e \times d = 1 + x_0 \times \varphi(n)$ ,  $\Leftrightarrow$

$e \times d \approx x_0 \times \varphi(n)$  comme  $e \approx \varphi(N)$  on a donc  $d \approx x_0$  comme  $d = N^\delta$

et comme  $e \approx N$  donc  $d = e^\delta$

On a  $y_0 = -(p + q)$  et on sait d'après l'énoncé que  $p \approx N^{\frac{1}{2}}$  et  $q \approx N^{\frac{1}{2}}$  on a donc  $y_0 \approx -2 \times N^{\frac{1}{2}}$  et donc  $y_0 \approx -N^{\frac{1}{2}}$  or  $N \approx e$  on a donc  $y_0 \approx -e^{\frac{1}{2}}$ , on a bien  $|y_0| \approx e^{\frac{1}{2}}$ .

## 18

. On a que  $u_0 = 1 + x_0 y_0$ , et  $f(x, y) = 1 + x(A + y)$  avec le résultat en *modulo*  $e$ . et  $\bar{f}(u, x) := u + Ax$  donc que  $\bar{f}(u, x) = f(x, y)$ . Comme  $\bar{f}(u, x) = f(x, y)$ .

$\Leftrightarrow$

$$\bar{f}(u_0, x_0) = u_0 + Ax_0$$

$\Leftrightarrow$

et comme  $u_0 = 1 + x_0 y_0$  on a donc :

$$\bar{f}(u_0, x_0) = f(x_0, y_0)$$

or on sait  $x_0$  et  $y_0$  sont racines de  $f(x, y)$  en *modulo*  $e$ .

On a que  $\bar{g}_{i,k}(u_0, x_0) = x_0^i \bar{f}^k(u_0, x_0) e^{m-k}$  avec  $0 \leq k \leq m$  et  $0 \leq i \leq m - k$

on a bien que  $\bar{g}_{i,k}(u_0, x_0) = 0$  *modulo*  $e$  et donc  $(x_0, y_0)$  est bien une racine de  $\bar{g}_{i,k}(u, x)$ .

De même pour  $\bar{h}_{j,k}(u_0, x_0, y_0) = y_0^j \bar{f}^k(u_0, x_0) e^{m-k}$ , avec  $0 \leq j \leq t$  et  $\lfloor m/t \rfloor j \leq k \leq m - k$

comme  $\bar{f}(u_0, x_0) = 0$  *modulo*  $e$  on a donc  $\bar{f}(u_0, x_0) = cst \times e$  avec  $cst$  qui est une constante.

On a donc que :

$$\bar{h}_{j,k}(u_0, x_0, y_0) = y_0^j (cst \times e)^k e^{m-k}$$

$\Leftrightarrow$

$$\bar{h}_{j,k}(u_0, x_0, y_0) = y_0^j (cst)^k e^m$$

et donc :  $\bar{h}_{j,k}(u_0, x_0, y_0) = 0$  *modulo*  $e$  on a bien que  $(u_0, x_0, y_0)$  est une racine de  $\bar{h}_{j,k}(u, x, y)$

## 19

.  $M = \{x^{k-i}u^i \text{ pour } k \in \{0, \dots, m\} \text{ et } i \in \{0, \dots, k\}\} \cup \{y^j u^k \text{ pour } j \in \{1, \dots, t\} \text{ et } k \in \{\lfloor m/t \rfloor j, \dots, m\}\}$   
et son cardinal  $|M| = l$ .

Comme les formes de deux unions sont différentes, on peut écrire somme directe.

On a :

$l = l1 \oplus l2$  avec  $l1$  le cardinal de  $\{x^{k-i}u^i \text{ pour } k \in \{0, \dots, m\} \text{ et } i \in \{0, \dots, k\}\}$   
et  $l2$  le cardinal de  $\{y^j u^k \text{ pour } j \in \{1, \dots, t\} \text{ et } k \in \{\lfloor m/t \rfloor j, \dots, m\}\}$ .

On a donc que :

$$l1 = \sum_{k=0}^m \sum_{i=0}^k 1$$

et

$$l2 = \sum_{j=1}^t \sum_{k=\lfloor m/t \rfloor j}^m 1$$

. On compare  $l2$  avec le nombre de polynômes de  $\bar{h}$  et  $l1$  avec le nombres de polynômes de  $\bar{g}$ . Soit  $G$  le nombre de polynômes de  $\bar{g}$  et  $H$  le nombre de polynômes de  $\bar{h}$ . On a que :

$$G = \sum_{k=0}^m \sum_{i=0}^{m-k} 1$$

et

$$H = \sum_{j=1}^t \sum_{k=\lfloor m/t \rfloor j}^m 1$$

. On voit que  $H = l2$ , il ne reste plus qu'à voir  $G$  et  $l1$ .

Pour avoir l'égalité  $G$  et  $l1$ , il faudrait juste faire un changement de variables avec  $k' = m - k$  dans  $G$ . On a donc :

Si  $k = 0$  alors  $k' = m$  et si  $k = m$  alors  $k' = 0$ .

$$G = \sum_{k'=0}^m \sum_{i=0}^{k'} 1$$

On a donc trouvé que  $l = G + H$ , on a bien démontré qu'il y a bien en tout  $l$  polynômes  $\bar{g}$  et  $\bar{h}$ .

## 20

. On a que  $\bar{f}(u, x) = u + Ax$ , on a donc :

$$\bar{g}_{i,k}(u, x) = x^i \bar{f}^k(u, x) e^{m-k}$$

avec  $0 \leq k \leq m$  et  $0 \leq i \leq m - k$

$\Leftrightarrow$

$$\bar{g}_{i,k}(u, x) = x^i (u + Ax)^k e^{m-k}$$

$\Leftrightarrow$

$$\bar{g}_{i,k}(u, x) = x^i \sum_{k'=0}^k \binom{k}{k'} u^{k-k'} (Ax)^{k'} e^{m-k}$$

On voit que les puissances de  $x, ux$  sont inférieurs et égale à  $k$ , et comme on a des sommes de ces types de monômes de  $M$ , on a que pour chaque  $\bar{g}$  est une combinaison des monômes de  $M$ .

On pose  $xy = u - 1$  dans la fonction  $\bar{h}$ .

$$\bar{h}_{j,k}(u, x, y) = y^j \bar{f}^k(u, x) e^{m-k}$$

avec  $0 \leq j \leq t$  et  $\lfloor m/t \rfloor j \leq i \leq m$

$\Leftrightarrow$

$$\bar{h}_{j,k}(u, x, y) = y^j \sum_{k'=0}^k \binom{k}{k'} u^{k-k'} (Ax)^{k'} e^{m-k}$$

On voit qu'on peut mettre dans la somme  $y^j$  en remplaçant  $xy$  par  $u - 1$  et avoir la forme des monômes de  $M$ ,  $u$ ,  $ux$ ,  $uy$ .



## 21

```

1 load('Exercice16.sage')
2
3 def changevar1e(f):
4
5     # On recupere le degre du polynomes pour crer les listes de monmes.
6     d = f.degree()
7
8     # On recherche dans la fonction tous les coefficient des mnomes de la forme x**i*y**j avec i
9     # et j entre 1 et d
10    for i in range(1,d):
11        for j in range(1,d):
12            # On calcule tous les coefficients de chaque monme de la forme x**i*y**j
13            k = f.coefficient(x**i*y**j)
14            if i < j:
15                # Comme la puissance de x est suprieur la puissance de y.
16                f = f - k*((x**i*y**j) - ((u-1)**i*y**(j-i)))
17
18            if i == j :
19                #Comme la puissance de x est gale la puissance de y.
20                f = f - k*((x**i*y**j) - (u-1)**i)
21
22            if j < i :
23                # Comme la puissance de y est suprieur la puissance de x.
24                f = f - k*((x**i*y**j) - ((u-1)**j*x**(i-j)))
25
26    return f
27
28 def lattice(m,t,N,e,gamma):
29     e1 = ZZ(e)
30     # D'aprs l'nonc:
31     X = integer_ceil(e1**gamma)
32     Y = integer_ceil(e1**(0.5))
33     U = 1+X*Y
34
35     # On cre la fonction ft
36
37     ft = u + (N+1)*x
38
39     # On cre la liste de famille de polynomes ht
40     H = [y**j*ft**k*e**(m-k) for j in range(1,t+1) for k in [integer_floor(m/t)*j..m]]
41
42     # Pour chaque polynme ht on fait le changement de x*y par (u-1)
43     H1 = [changevar1e(i) for i in H]
44
45     # On cre la liste des deux familles de plynomes.
46     L = [x**i*ft**k*e**(m-k) for k in range(m+1) for i in range(m-k+1)]+H1
47
48     # On rajoute x*X,y*Y,u*U
49     L1 = [i.subs({x:x*X, y:y*Y, u:u*U}) for i in L]
50
51     # On cre la liste des monmes.
52     M = [x**(k-i)*u**i for k in range(m+1) for i in range(k+1)]+[y**j*u**k for j in range(1,t+1)
53         for k in [integer_floor(m/t)*j..m]]
54
55     # On cre la matrice avec les coefficients de chaques monmes par rapport aux polynomes.
56     Mat = matrix(ZZ,len(L1),len(M))
57     for i in range(len(L1)):
58         for j in range(len(M)):
59             Mat[i,j] = L1[i].monomial_coefficient(M[j])
60     return Mat
61
62 # On essaie avec un petit exemple

```

```

61 N,e,d=gen(5,0.7)
62 print("N = %d e = %d" %(N, e))
63 gamma = 0.7
64 X = integer_ceil(e**gamma)
65 Y = integer_ceil(e**(0.5))
66 U = 1+X*Y
67 print("X = %d Y = %d U = %d" %(X, Y, U))
68 print("\n")
69 print(" lattice(2, 1, %d, %d, %.2f) = " %(N, e, gamma))
70 print(lattice(2,1,N,e,gamma))

```

Ce code est dans "*Exercice21.sage*".

## 22

```
1 Pr.<x,y,u> = PolynomialRing(ZZ)
2
3 def changevar2(f):
4     #On change u en 1+x*y
5     f = f.substitute(u = 1+x*y)
6     return f
7
8 # On teste une petite fonction
9 f = x*u + y*u**2 + 2
10 print("f =",f)
11 print("changevar2(f) =", changevar2(f))
```

Ce code est dans "*Exercice22.sage*".

## 23

. On pose

$$\begin{aligned}
 P(x_0, y_0) &= \sum_{i=0}^{k'-1} \sum_{j=0}^{k-1} |P_{i,j} x_0^i y_0^j| \text{ avec } kk' = \omega \\
 &\leq \sum_{i=0}^{k'-1} \sum_{j=0}^{k-1} |P_{i,j}| x_0^i y_0^j \\
 &\leq \sum_{i=0}^{k'-1} \sum_{j=0}^{k-1} |P_{i,j}| X^i Y^j \\
 &= \langle (|P_{0,0}|, \dots, |P_{k'-1, k-1}|), (1, 1, \dots, 1) \rangle \\
 &\leq \|P(xX, yY)\| \times \|(1, 1, \dots, 1)\| \\
 &= \|P(xX, yY)\| \sqrt{\omega} < e^m
 \end{aligned}$$

La majoration du produit scalaire des normes correspond à l'inégalité de **Cauchy-Schwartz**.

Comme  $P(x_0, y_0) = 0 \pmod{e^m}$ , on a que :  $P(x_0, y_0) = cst \times e^m$  avec  $cst \in \mathbf{Z}$

mais comme  $|P(x_0, y_0)| < e^m$

on a que  $cst = 0$  et donc  $P(x_0, y_0) = 0 \in \mathbf{Z}$ .

## 24

```

1 load('Exercice21.sage')
2 load('Exercice22.sage')
3
4 def factorisation(N,e,gamma,m):
5     e1 = ZZ(e)
6     # D'après l'nonc:
7     X = integer_ceil(e1**gamma)
8     Y = integer_ceil(e1**(0.5))
9     U = 1+X*Y
10    t = integer_floor(m*(1-2*gamma))
11    Mat =lattice(m,t,N,e,gamma)
12    Mat1 = Mat.LLL()
13
14    Pr.<x,y,u> = PolynomialRing(QQ)
15    # On cre une liste de monomes
16    M =[(x/X)**(k-i)*(u/U)**i for k in range(m+1) for i in range(k+1)]+[(y/Y)**j*(u/U)**k for j
        in range(1,t+1) for k in [integer_floor(m/t)*j..m]]
17
18    # On cre les polynomes de chaque ligne de la matrice.
19    P=[sum(Mat1[i][j]*M[j] for j in range(len(M))) for i in range(len(M))]
20
21
22
23    # On remplace u par 1+x*y
24    Pchange = [changevar2(i) for i in P]
25
26
27
28    # on initialise i=0
29    i = 0
30    P1 = Pchange[i]
31    P2 = Pchange[i+1]
32
33    # On met Pxy un nouvelle anneau polynomial en fonction de x et y
34    Pxy.<x, y> = PolynomialRing(QQ)
35    # On caste P1 et P2 en Pxy
36
37    PP1 = Pxy(P1)
38    PP2 = Pxy(P2)
39
40
41    res = PP1.resultant(PP2,y)
42
43    # On met Px un nouvelle anneau polynomial en fonction de x
44    Px.<x> = PolynomialRing(QQ)
45
46    resx = Px(res)
47
48
49    while res.degree() < 0 and i!=len(M)-2:
50        i = i+1
51        P1 = Pchange[i]
52        P2 = Pchange[i+1]
53        PP1 = Pxy(P1)
54        PP2 = Pxy(P2)
55
56
57    # On met Px un nouvelle anneau polynomial en fonction de x
58
59    res = PP1.resultant(PP2,y)
60
61    Px.<x> = PolynomialRing(QQ)

```

```

62
63     resx = Px(res)
64
65
66
67
68
69     # On regarde sur on est sorti de la liste de fonctions
70     if i == len(M)-2:
71         return (-1)
72     # On cherche la premiere valeur
73     x0 = resx.roots()[0][0]
74
75
76     #On reinjecte cette valeur sur le premier polynme
77     PPP1 = P2.subs(x=x0)
78
79     # On met Py un nouvelle anneau polynomial en fonction de y
80     Py.<y> = PolynomialRing(QQ)
81
82     # On le caste en Py
83     equy = Py(PPP1)
84
85     # On cherche la racine de cette quation en fonction de y
86     y0 = equy.roots()[0][0]
87
88     # D'aprs les rsultats de la question 16 et 17
89     d = (x0*(N+1+y0)+1)/e1
90
91     return (d )
92
93
94 #premier exemple demand
95 N,e,d = gen(1024,0.2)
96 gamma = 0.2
97 m = 2
98
99 # On teste si on a trouv le bon d
100 d1 = factorisation(N, e, gamma, m)
101
102 print("d = factorisation(%d, %d,%.2f, %d) est %s" %(N, e, gamma, m, d==d1))
103
104 # Pour la deuxime exemple, je n'ai pas russit  debugger mon programme.

```

Mon programme ne fonctionne pas parfaitement car je ne savais pas utiliser le while pour les conditions données dans le (23), exactement pour  $|x_0| < X$  et de même pour  $|y_0| < Y$ .

Ce code est dans "*Exercice24.sage*".

## 25

. Après avoir normalement récupérer  $d$ , on peut grâce aux formule du (16) calculer  $\phi(N)$ .

On a que

$$ed = 1 + x_0\phi(n) \text{ et } y_0 = -(p + q)$$

donc :  $\phi(N) = \frac{ed-1}{x_0}$  et on sait que  $\phi(N) = (p-1)(q-1)$

On peut chercher  $p$  et  $q$  en résolvant cette équation :

$$x^2 + y_0x + \phi N$$

On trouvera  $p$  et  $q$  les diviseurs de  $N$  et on aura donc factoriser un  $N$  RSA.