Rapport du Projet de Cryptanalyse

Sid Ali Zitouni Terki

Décembre 2020

Table des matières

Chapitre I

Cryptanalyse de A5/2

1

.

```
def LFSR_step(state,f):
    L = len(state)
    out = state[0]
    state = state[1:]+[sum(f[j+1]*state[L−1−j] for j in range(L))]
    return out,state


def A5_2_init(K, IV, f1, f2, f3, f4):

    # On initialise les registres  0.
    R1 = Sequence([GF(2)(0) for _ in range(19)])
    R2 = Sequence([GF(2)(0) for _ in range(22)])
    R3 = Sequence([GF(2)(0) for _ in range(23)])
    R4 = Sequence([GF(2)(0) for _ in range(17)])


    for i in range(64):
        # On met  jour chaque registre avec son polynome
        # de rtroaction respectif.
        _, R1 = LFSR_step(R1, f1)
        _, R2 = LFSR_step(R2, f2)
        _, R3 = LFSR_step(R3, f3)
        _, R4 = LFSR_step(R4, f4)



        # On Xor pour chaque derniers bits de chaque registre
        # avec l'indice i (qui est le nombre tours faits)
        # de la clef K.
        R1[18] = R1[18] + K[i]
        R2[21] = R2[21] + K[i]
        R3[22] = R3[22] + K[i]
        R4[16] = R4[16] + K[i]




    for i in range(22):

        # On met  jour chaque registre avec son polynome
        # de rtroaction respectif.
        _, R1 = LFSR_step(R1, f1)
        _, R2 = LFSR_step(R2, f2)
        _, R3 = LFSR_step(R3, f3)
        _, R4 = LFSR_step(R4, f4)



        # On Xor pour chaque derniers bits de chaque registre
        # avec l'indice i du vecteur initial IV.
        R1[18] = R1[18] + IV[i]
        R2[21] = R2[21] + IV[i]
        R3[22] = R3[22] + IV[i]
        R4[16] = R4[16] + IV[i]



    # On fixe certains bits des registres.
    R1[3] = 1
    R2[5] = 1
    R3[4] = 1
    R4[6] = 1
```

```python


    return R1, R2, R3, R4


def Maj(a, b, c):
    # 3 cas possibles pour la fonction majorit.
    m = 0
    if a == b:
        m = a

    if a == c:
        m = a

    if b == c :
        m = b

    return m


def A5_2_step(R1, R2, R3, R4, f1, f2, f3, f4):

    # On calcule la fonction majorit
    # sur les indices 6, 13 et 9 de R4.
    m = Maj(R4[6], R4[13], R4[9])


    # On a 3 choix possibles.
    # Si  l'indice 6 de R4 est gale  m
    # on met  jour R1 en ignorant sa sortie.

    if R4[6] == m :
        _, R1 = LFSR_step(R1, f1)



    # Si  l'indice 13 de R4 est gale  m
    # on met  jour R2 en ignorant sa sortie.
    if R4[13] == m :
        _, R2 = LFSR_step(R2, f2)



    # Si  l'indice 9 de R4 est gale  m
    # on met  jour R3 en ignorant sa sortie.
    if R4[9] == m :
        _, R3 = LFSR_step(R3, f3)





    # on met  jour R4 en ignorant la sortie.
    _, R4 = LFSR_step(R4, f4)

    # On calcule y1, y2 et y3 avec les formules donnes.
    y1 = R1[0] + Maj(R1[3], R1[4] + 1, R1[6])
    y2 = R2[0] + Maj(R2[8], R2[5] + 1, R2[12])
    y3 = R3[0] + Maj(R3[4], R3[9] + 1, R3[6])

    # On ressort le rsultat de l'additions des 3 variables
    return (y1 + y2 + y3),(R1, R2, R3, R4)


def A5_2_production(R1, R2, R3, R4, f1, f2, f3, f4):

    # On excute 99 fois la fonction "A5_2_step" en ignorant
    # son bit de sortie
```

```
    for i in range(99):
        _, (R1 ,R2 ,R3 ,R4) = A5_2_step(R1, R2, R3, R4, f1, f2, f3, f4)

    # On prend z une suite iniatilis  une liste vide.
    z = []

    # On excute 228 fois la fonction "A5_2_step" en utilisant son
    # bit de sortie.
    for i in range(228):
        zi, (R1 ,R2 ,R3 ,R4) = A5_2_step(R1, R2, R3, R4, f1, f2, f3, f4)
        z.append(zi)


    return z


def A5_2(K, IV):

    # On cre les polynomes de rtroaction
    PR.<X> = PolynomialRing(GF(2))
    f1 = X**19 + X**18 + X**17 + X**14 + 1
    f2 = X**22 + X**21 + 1
    f3 = X**23 + X**22 + X**21 + X**8 + 1
    f4 = X**17 + X**12 + 1

    # Phase d'initialisation.
    R1, R2, R3, R4 = A5_2_init(K, IV, f1, f2, f3, f4)


    # Phase de productions.
    return A5_2_production(R1, R2, R3, R4, f1, f2, f3, f4)


load('A5_2-test-vector.sage')

# On va tester si notre fonction nous donne les bonnes rsultats
# prvues .
zprime = A5_2(K, IV)

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 connaitre $R4$ qui est le registre du $LFSR4$. Comme on connait $R4$, 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 :
$R1 = (x_0, ..., x_{18})$
$R2 = (x_{19}, ..., x_{40})$
$R3 = (x_{41}, ..., x_{63})$

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 :
$x_0 = x_0, x_1 = x_1, ..., x_{18} = x_{18}$

Pour le premier tour :
$x_0 = x_1, x_1 = x_2, ..., x_{18} = x_{18} + x_{17} + x_{16} + x_{13}$

Après les k tours :
$x_0 = x'_0, x_1 = x'_1, ..., x_{18} = x'_{18}$

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

.

```
load('Exercice1.sage')
# On cre les matrices de rtroaction avec cette fonction:
def matf(f):# matrice de transition
    n = f.degree()
    A = Matrix(GF(2),n,n)
    for i in range(n-1):
        A[i,i+1] = 1
    tmp = f.list()[1:] # c_1,c_2,...,c_L
    tmp.reverse() # c_L,c_{L-1}, ... , c_1
    A[n-1] = vector(tmp)
    return A

def A5_2_stepret(R1, R2, R3, R4, f1, f2, f3, f4):
    #nos matrices de retroactions:
    A1 = matf(f1)
    A2 = matf(f2)
    A3 = matf(f3)
    A4 = matf(f4)


    # On calcule la fonction majorit
    # sur les indices 6, 13 et 9 de R4.
    m = Maj(R4[6], R4[13], R4[9])


    # On a 3 choix possibles.
    # Si  l'indice 6 de R4 est gale  m
    # on met  jour R1 en ignorant sa sortie.

    if R4[6] == m :
        x1 =vector(R1)
        x1 = A1*x1
        R1 = Sequence(x1)



    # Si  l'indice 13 de R4 est gale  m
    # on met  jour R2 en ignorant sa sortie.
    if R4[13] == m :
        x2 = vector(R2)
        x2 = A2*x2
        R2 = Sequence(x2)



    # Si  l'indice 9 de R4 est gale  m
    # on met  jour R3 en ignorant sa sortie.
    if R4[9] == m :
        x3 = vector(R3)
        x3 = A3*x3
        R3 = Sequence(x3)





    # on met  jour R4 en ignorant la sortie.
    x4 = vector(R4)
    x4 = A4*x4
    R4 = Sequence(x4)

    # On calcule y1, y2 et y3 avec les formules donnes.
```

```
    y1 = R1[0] + Maj(R1[3], R1[4] + 1, R1[6])
    y2 = R2[0] + Maj(R2[8], R2[5] + 1, R2[12])
    y3 = R3[0] + Maj(R3[4], R3[9] + 1, R3[6])

    # On ressort le rsultat de l'additions des 3 variables
    return (y1 + y2 + y3),(R1, R2, R3, R4)



# On suppose qu'on connait R4 et donc on va utiliser les K et IV donnes pour l'exemple du (1)

# La clef K
K = Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

# IV
IV = Sequence([GF(2)(1), 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

# On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
PR.<X> = PolynomialRing(GF(2))
f1 = X**19 + X**18 + X**17 + X**14 + 1
f2 = X**22 + X**21 + 1
f3 = X**23 + X**22 + X**21 + X**8 + 1
f4 = X**17 + X**12 + 1

# On cre le R4 que l'on connait.
_, _, _, R4 = A5_2_init(K, IV, f1, f2, f3, f4)


# Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
BPR = BooleanPolynomialRing(64, 'x')
v = BPR.gens()

# On cre les R1, R2 et R3 avec les inconnues associs.
R1 = Sequence([v[i] for i in range(19)])
R2 = Sequence([v[i] for i in range(19,41)])
R3 = Sequence([v[i] for i in range(41,64)])

# On excute 99 fois la fonction "A5_2_step" en ignorant
# son bit de sortie
for i in range(99):
     _, (R1 ,R2 ,R3 ,R4) = A5_2_stepret(R1, R2, R3, R4, f1, f2, f3, f4)

# On excute 228 fois la fonction "A5_2_step" en utilisant en ignorant
# bit de sortie.
for i in range(228):
    _, (R1 ,R2 ,R3 ,R4) = A5_2_stepret(R1, R2, R3, R4, f1, f2, f3, f4)

# On cre une liste d'quations linaires.

Eql = [R1[i] for i in range(19)] + [R2[i] for i in range(22)] + [R3[i] for i in range(23)]
print("Taille de la liste d'quation =", len(Eql))

# On voit les 64 quations.
print("Les 64 quations sont : ")
for i in range(64):
    print("xprime[%d] = %s" %(i, Eql[i]))
```

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	`Maj`$(a,\, b,\, c)$
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 :
`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 :

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

et on a que

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

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 :

$$y_1 + y_2 + y_3 =$$

$$R_{1,0}+R_{2,0}+R_{3,0}+R_{1,3}+R_{1,6}+R_{2,8}+R_{2,12}+R_{3,4}+R_{3,6}+R_{1,3}*R_{1,4}+R_{1,4}*R_{1,6}+R_{1,3}*R_{1,6}$$

$$+R_{2,8} * R_{2,5} + R_{2,5} * R_{2,12} + R_{2,8} * R_{2,12} + R_{3,4} * R_{3,9} + R_{3,9} * R_{3,6} + R_{3,4} * R_{3,9}$$

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

$$y_1 + y_2 + y_3 =$$

$$x_0 + x_{19} + x_{41} + x_3 + x_6 + x_{27} + x_{31} + x_{45} + x_{47} + x_3 * x_4 + x_4 * x_6 + x_3 * x_6$$

$$+x_{27} * x_{24} + x_{24} * x_{31} + x_{27} * x_{31} + x_{45} * x_{50} + x_{50} * x_{47} + x_{45} * x_{47}$$

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

.

```
load('Exercice1.sage')

def Maj_al_n(a,b,c): # fonction majorit en forme algbrique normale.
    return a*b +b*c +a*c


def A5_2_stepv_2(R1, R2, R3, R4, f1, f2, f3, f4):
    # On utilise la fonction majorit en forme algbrique normale.

    # On calcule la fonction majorit
    # sur les indices 6, 13 et 9 de R4.
    m = Maj_al_n(R4[6], R4[13], R4[9])


    # On a 3 choix possibles.
    # Si  l'indice 6 de R4 est gale  m
    # on met  jour R1 en ignorant sa sortie.

    if R4[6] == m :
        _, R1 = LFSR_step(R1, f1)



    # Si  l'indice 13 de R4 est gale  m
    # on met  jour R2 en ignorant sa sortie.
    if R4[13] == m :
        _, R2 = LFSR_step(R2, f2)



    # Si  l'indice 9 de R4 est gale  m
    # on met  jour R3 en ignorant sa sortie.
    if R4[9] == m :
        _, R3 = LFSR_step(R3, f3)





    # on met  jour R4 en ignorant la sortie.
    _, R4 = LFSR_step(R4, f4)

    # On calcule y1, y2 et y3 avec les formules donnes.
    y1 = R1[0] + Maj_al_n(R1[3], R1[4] + 1, R1[6])
    y2 = R2[0] + Maj_al_n(R2[8], R2[5] + 1, R2[12])
    y3 = R3[0] + Maj_al_n(R3[4], R3[9] + 1, R3[6])

    # On ressort le rsultat de l'additions des 3 variables
    return (y1 + y2 + y3),(R1, R2, R3, R4)


def A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4): # On utilise la fonction majorit en forme algbrique normale
    # Dans cette version on rajoute N pour pourvoir utiliser avec n'importe quel N.
    # On excute 99 fois la fonction "A5_2_step" en ignorant
    # son bit de sortie
    for i in range(99):
        _, (R1 ,R2 ,R3 ,R4) = A5_2_stepv_2(R1, R2, R3, R4, f1, f2, f3, f4)

    # On prend z une suite iniatilis  une liste vide.
    z = []

```

```
    # On excute N fois la fonction "A5_2_step" en utilisant son
    # bit de sortie.
    for i in range(N):
        zi, (R1 ,R2 ,R3 ,R4) = A5_2_stepv_2(R1, R2, R3, R4, f1, f2, f3, f4)
        z.append(zi)


    return z


def A5_2v_2(N,K, IV):

    # On cre les polynomes de rtroaction
    PR.<X> = PolynomialRing(GF(2))
    f1 = X**19 + X**18 + X**17 + X**14 + 1
    f2 = X**22 + X**21 + 1
    f3 = X**23 + X**22 + X**21 + X**8 + 1
    f4 = X**17 + X**12 + 1

    # Phase d'initialisation.
    R1, R2, R3, R4 = A5_2_init(K, IV, f1, f2, f3, f4)


    # Phase de productions.
    return A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)

# On suppose qu'on connait R4 et donc on va utiliser les K et IV donnes pour l'exemple du (1)

# On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
PR.<X> = PolynomialRing(GF(2))
f1 = X**19 + X**18 + X**17 + X**14 + 1
f2 = X**22 + X**21 + 1
f3 = X**23 + X**22 + X**21 + X**8 + 1
f4 = X**17 + X**12 + 1

# On cre le R4 que l'on connait.
_, _, _, R4 = A5_2_init(K, IV, f1, f2, f3, f4)


# Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
BPR = BooleanPolynomialRing(64, 'x')
v = BPR.gens()

# On cre les R1, R2 et R3 avec les inconnues associs.
R1 = Sequence([v[i] for i in range(19)])
R2 = Sequence([v[i] for i in range(19,41)])
R3 = Sequence([v[i] for i in range(41,64)])

# On veut N =228
N = 228

# On calcule la suite chiffr z avec ses 64 inconnues.
zprime = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)

# On a cre une liste d'quations quadratiques et on l'affiche les 228 quations.
print("Taille de la liste d'quation =", len(zprime))

# On voit les 228 quations.
print("Les %d quations sont : " %(N))
for i in range(228):
    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 $R1$: $153 + 18 = 171$ monômes possibles. de même pour $R2$:

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

et e même pour $R3$:

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

On a

$$171 + 231 + 253 = 655$$

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

```
# Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
BPR = BooleanPolynomialRing(64, 'x')
v = BPR.gens()

# On sait x3 = x24 = x45 = 1, sont les variables connues, on cre M1 la liste des monmes
# de degr 1 sans ses 3 variables.
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)]
len(M1)

# On cre M la liste de tous les monmes possibles.
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)]
print("M =", M)
print("La longueur de M est :", len(M))
# On voit que sa longueur est bien 655
N = 228
L = 655
```

Ce code est dans "*Exercice*6.*sage*".

7

.

```
load('Exercice5.sage')
load('Exercice6.sage')

# On suppose qu'on connait R4 et donc on va utiliser les K et IV donnes pour l'exemple du (1)

# On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
PR.<X> = PolynomialRing(GF(2))
f1 = X**19 + X**18 + X**17 + X**14 + 1
f2 = X**22 + X**21 + 1
f3 = X**23 + X**22 + X**21 + X**8 + 1
f4 = X**17 + X**12 + 1

# On cre le R4 que l'on connait.
_, _, _, R4 = A5_2_init(K, IV, f1, f2, f3, f4)


# Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
BPR = BooleanPolynomialRing(64, 'x')
v = BPR.gens()

# On cre les R1, R2 et R3 avec les inconnues associs et leurs bits connus.
R1 = Sequence([v[i] for i in range(3)]+[1]+[v[i] for i in range(4,19)])
R2 = Sequence([v[i] for i in range(19,24)]+[1]+[v[i] for i in range(25,41)])
R3 = Sequence([v[i] for i in range(41,45)]+[1]+[v[i] for i in range(46,64)])

# On calcule la suite chiffr z avec ses 64 inconnues.
zprimev2 = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)

# On a cre une liste d'quations quadratiques et on l'affiche les 228 quations.
print("Taille de la liste d'quation =", len(zprimev2))

# On voit les 228 quations.
print("Les %d quations sont : " %(N))
for i in range(228):
    print("zprime%d = %s" %(i, zprimev2[i]))

print("\n")
# On cre la matrice Mat de taille N*L dans F2.
Mat = matrix(GF(2),N,L)

# On cre le vecteur de longueur N.
Vec = vector(GF(2),N)

# On utilise le zprime calcul dans le (5).
# On utilise les indications de l'nonc.
for i in range(N):
    Vec[i] = zprimev2[i].monomial_coefficient(1)
    for j in range(L):
        Mat[i,j] = zprimev2[i].monomial_coefficient(M[j])

print("Vec =", Vec)
print("\n")
print("Mat =", Mat)
```

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

8

.

```
load('Exercice5.sage')
load('A5_2-700.sage')

N = 700
# On considre une excution de A5/2 donnant N = 700 bits de suite chiffrante avec
# On a la valeur de R4 aprs la phase d'initialisation
# et z la suite chiffrante de 700 bits

# Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2 et R3.
BPR = BooleanPolynomialRing(64, 'x')
v = BPR.gens()

# On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
PR.<X> = PolynomialRing(GF(2))
f1 = X**19 + X**18 + X**17 + X**14 + 1
f2 = X**22 + X**21 + 1
f3 = X**23 + X**22 + X**21 + X**8 + 1
f4 = X**17 + X**12 + 1

# On cre les R1, R2 et R3 avec les inconnues associs et leurs bits connus.
R1 = Sequence([v[i] for i in range(3)]+[1]+[v[i] for i in range(4,19)])
R2 = Sequence([v[i] for i in range(19,24)]+[1]+[v[i] for i in range(25,41)])
R3 = Sequence([v[i] for i in range(41,45)]+[1]+[v[i] for i in range(46,64)])

# On calcule la suite chiffr z avec ses 64 inconnues.
zprime = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)

# On a cre une liste d'quations quadratiques et on l'affiche les 700 quations.
print("Taille de la liste d'quation =", len(zprime))

# On voit les 700 quations.
print("Les %d quations sont : " %(N))
for i in range(N):
    print("zprime[%d] = %s" %(i, zprime[i]))

print("\n")

# On cre la matrice Mat de taille N*L dans F2.
Mat = matrix(GF(2),N,L)

# On cre le vecteur de longueur N.
Vec = vector(GF(2),N)

# On utilise le zprime .
# On utilise les indications de l'nonc.
for i in range(N):
    Vec[i] = zprime[i].monomial_coefficient(1)
    for j in range(L):
        Mat[i,j] = zprime[i].monomial_coefficient(M[j])

print("Vec =", Vec)
print("\n")
print("Mat =", Mat)
print("\n")

# Pour rsoudre l'quation linaire, on va utiliser la fonction solve_right() donne dans l'nonc.
# On additionne vec  z.
Right = [Vec[i] + z[i] for i in range(N)]
#on rsoud l'quation grce  la fonction, en mettant tous les inconnues  gauche de l'galit
# et le reste  droite, on pose sol le rsultat de la fonction:
Sol = Mat.solve_right(Right)
print("Sol =",Sol)
```

```
print("\n")

#On a besoin que les 61 premiers bits du sol pour rcuprer
# R1, R2 et R3 :
R1 = Sequence([Sol[i] for i in range(3)]+[1]+[Sol[i] for i in range(3,18)])
R2 = Sequence([Sol[i] for i in range(18,23)]+[1]+[Sol[i] for i in range(23,39)])
R3 = Sequence([Sol[i] for i in range(39,43)]+[1]+[Sol[i] for i in range(43,61)])
print("R1 =", R1)
print("R2 =", R2)
print("R3 =", R3)
print("R4 =", R4)
print("\n")

# On teste si a nous donne le bon rsultat:
ztest = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
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

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

Soit A_1 la matrice de rétroaction du $LFSR1$. Dans la boucle `for`, à $i = 0$:
On met à jour $R1$:

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

On rajoute K_0, On a que :

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

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

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

De même pour $i = 2$:

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

$\Leftrightarrow$

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

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

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

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

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

$\Leftrightarrow$

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

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

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

au temps $i = 1$:

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

$\Leftrightarrow$

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

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

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

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

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

$\Leftrightarrow$

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

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

10

.

```
load('Exercice5.sage')
load('A5_2-700.sage')
# On recupre les Ri de l'exercice 8:
R1 = [1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]
R2 = [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1]
R3 = [1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0]
R4 = [1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1]

# Son IV est nulle
IV = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


def matf(f):# matrice de transition
    n = f.degree()
    A = Matrix(GF(2),n,n)
    for i in range(n-1):
        A[i,i+1] = 1
    tmp = f.list()[1:] # c_1,c_2,...,c_L
    tmp.reverse() # c_L,c_{L-1}, ... , c_1
    A[n-1] = vector(tmp)
    return A


# On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
PR.<X> = PolynomialRing(GF(2))
f1 = X**19 + X**18 + X**17 + X**14 + 1
f2 = X**22 + X**21 + 1
f3 = X**23 + X**22 + X**21 + X**8 + 1
f4 = X**17 + X**12 + 1

# On construit les matrices de retroactions pour chaque fonction de rtroaction
A1 = matf(f1)
A2 = matf(f2)
A3 = matf(f3)
A4 = matf(f4)


# Pour dclarer les 64 inconnues que l'on utilisera dans la clef K.
BPR = BooleanPolynomialRing(64, 'K')
v = BPR.gens()

# Comme dans l'algorithme de A5/2-init les registres sont vides des 4 LFSR sont nulles.
X1 = vector([0 for _ in range(19)]) # registre du LFSR1
X2 = vector([0 for _ in range(22)]) # registre du LFSR2
X3 = vector([0 for _ in range(23)]) # registre du LFSR3
X4 = vector([0 for _ in range(17)]) # registre du LFSR4

# On calcule les registres comme dans le rsultat du (9).
# Dans la premire boucle for:
for i in range(64):
    X1 = A1*X1+vector([0 for _ in range(18)]+[v[i]])
    X2 = A2*X2+vector([0 for _ in range(21)]+[v[i]])
    X3 = A3*X3+vector([0 for _ in range(22)]+[v[i]])
    X4 = A4*X4+vector([0 for _ in range(16)]+[v[i]])

# Comme IV est nulle, dans la deuxime boucle for on multiplie pour chaque registre par leurs matrices
# de rtroaction qui sont mis  la puissance 22.
X1 = A1**22*X1
X2 = A2**22*X2
X3 = A3**22*X3
X4 = A4**22*X4
```

```

# A la fin de cet algorithme, on met un seul bit  1 dans chaque registre :
X1[3] = 1
X2[5] = 1
X3[4] = 1
X4[6] = 1

# On fait une liste quation
L = list(X1)+list(X2)+list(X3)+list(X4)

# On concatne les registres trouvs dans le (8)
R = list(R1)+list(R2)+list(R3)+list(R4)

# On va calculer l'idal :
I = ideal([L[i] + R[i] for i in range(81)])
print("I lidal :",I)
print("\n")
# On utilise la base groebner.
Iq = I.groebner_basis();
print("list(II.groebner_basis() :", list(Iq))
print("\n")

# On cherche les coefficients qui additionnent les variables pour avoir une clef Kprime.
Kprime = [el.constant_coefficient() for el in Iq]
print("Kprime =", Kprime)
print("\n")

#On calcule avec Kprime pour chercher les Riprime avec 1<=i<=4
R1prime, R2prime, R3prime,R4prime =A5_2_init(Kprime, IV, f1, f2, f3, f4)

# On teste si a nous donne les bons rsultats:
print("R1prime = R1 est ", R1prime == R1)
print("R2prime = R2 est ", R2prime == R2)
print("R3prime = R3 est ", R3prime == R3)
print("R4prime = R4 est ", R4prime == R4)
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` :

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

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` :

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

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

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

Pour pouvoir différencier les R_i des trois suites chiffrante :
pour $z0$ ça sera R_i,
pour $z1$ ça sera R_i',
pour $z2$ ça sera $\tilde{R_i}$.

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

$$R_i' = R_i + \begin{pmatrix} 0 \\ . \\ . \\ . \\ 0 \\ 1 \end{pmatrix}$$

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

$$\tilde{R_i} = R_i + A_i \times \begin{pmatrix} 0 \\ . \\ . \\ . \\ 1 \\ 0 \end{pmatrix}$$

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 $1 \leq i \leq 4$

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

$$\tilde{R_i} = R_i + A_i \times \begin{pmatrix} 0 \\ . \\ . \\ . \\ 1 \\ 0 \end{pmatrix}$$

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

$$R1 = (x_0, ..., x_{18})$$
$$R2 = (x_{19}, ..., x_{40})$$
$$R3 = (x_{41}, ..., x_{63})$$

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 $z0$, $z1$ et $z2$, 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 $F2$ 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 :

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

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 :

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

$$R_2 = [Y_{18}, ..., Y_{22}, 1, Y23, ..., Y_{38}]$$

$$R_3 = [Y_{39}, ..., Y_{42}, 1, Y43, ..., Y_{60}]$$

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

.

```
load('Exercice5.sage')
load('A5_2-3frames.sage')

IV= Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]) # pour z0
IVprime = Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]) # pour z1
IVtilde = Sequence([GF(2)(0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]) # pour z2

# On pose les registres des LFSRi Ri pour z0, Riprime pour z1 et Ritilde pour z2 avec 1 <= i <= 4
# Pour dclarer les 64 inconnues que l'on utilisera dans les registres R1, R2, R3, R1prime, R2prime, R3prime
# R1tilde, R2tilde et R3tilde
BPR = BooleanPolynomialRing(64, 'x')
v = BPR.gens()

# On cre les R1, R2 et R3 avec les inconnues associs et leurs bits connus.
R1 = Sequence([v[i] for i in range(3)]+[1]+[v[i] for i in range(4,19)])
R2 = Sequence([v[i] for i in range(19,24)]+[1]+[v[i] for i in range(25,41)])
R3 = Sequence([v[i] for i in range(41,45)]+[1]+[v[i] for i in range(46,64)])

print("R1 =",R1)
print("R2 =",R2)
print("R3 =",R3)
print("R4 =",R4)
print("\n")

def matf(f):# matrice de transition
    n = f.degree()
    A = Matrix(GF(2),n,n)
    for i in range(n-1):
        A[i,i+1] = 1
    tmp = f.list()[1:] # c_1,c_2,...,c_L
    tmp.reverse() # c_L,c_{L-1}, ... , c_1
    A[n-1] = vector(tmp)
    return A


# On utilise les mmes polynmes de rtroactions que l'algorithme "A5_2_init"
PR.<X> = PolynomialRing(GF(2))
f1 = X**19 + X**18 + X**17 + X**14 + 1
f2 = X**22 + X**21 + 1
f3 = X**23 + X**22 + X**21 + X**8 + 1
f4 = X**17 + X**12 + 1

# On construit les matrices de retroactions pour chaque fonction de rtroaction
A1 = matf(f1)
A2 = matf(f2)
A3 = matf(f3)
A4 = matf(f4)

# Pour faire les calculs je transforme les registres en vecteur.
X1 = vector(R1)
X2 = vector(R2)
X3 = vector(R3)
X4 = vector(R4)

# On calcul les registres des autres suites chiffrantes.
# On commence par celle de z1:
X1prime = X1 + vector([0 for _ in range(18)]+[IVprime[21]])
X2prime = X2 + vector([0 for _ in range(21)]+[IVprime[21]])
X3prime = X3 + vector([0 for _ in range(22)]+[IVprime[21]])
X4prime = X4 + vector([0 for _ in range(16)]+[IVprime[21]])
```

```
# on met un seul bit  1 dans chaque registre :
X1prime[3] = 1
X2prime[5] = 1
X3prime[4] = 1
X4prime[6] = 1

# On le remet en sequence
R1prime = Sequence(X1prime)
R2prime = Sequence(X2prime)
R3prime = Sequence(X3prime)
R4prime = Sequence(X4prime)

print("R1prime =",R1prime)
print("R2prime =",R2prime)
print("R3prime =",R3prime)
print("R4prime =",R4prime)
print("\n")

# et celle de z2:
X1tilde = X1 + A1 * vector([0 for _ in range(18)]+[IVtilde[20]])
X2tilde = X2 + A2 * vector([0 for _ in range(21)]+[IVtilde[20]])
X3tilde = X3 + A3 * vector([0 for _ in range(22)]+[IVtilde[20]])
X4tilde = X4 + A4 * vector([0 for _ in range(16)]+[IVtilde[20]])

# on met un seul bit  1 dans chaque registre :
X1tilde[3] = 1
X2tilde[5] = 1
X3tilde[4] = 1
X4tilde[6] = 1

# On le remet en sequence
R1tilde = Sequence(X1tilde)
R2tilde = Sequence(X2tilde)
R3tilde = Sequence(X3tilde)
R4tilde = Sequence(X4tilde)

print("R1tilde =",R1tilde)
print("R2tilde =",R2tilde)
print("R3tilde =",R3tilde)
print("R4tilde =",R4tilde)
print("\n")

# On pose N = 228
N = 228
# On calcule les z thoriques
z = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
zprime = A5_2_productionv_2(N, R1prime, R2prime, R3prime, R4prime, f1, f2, f3, f4)
ztilde = A5_2_productionv_2(N, R1tilde, R2tilde, R3tilde, R4tilde, f1, f2, f3, f4)


# On cre une liste d'quations
Eq =z+zprime+ztilde

# On cre une liste de rsultats voulues
Res = z0+z1+z2

# On sait x3 = x24 = x45 = 1, sont les variables connues, on cre M1 la liste des monmes
# de degr 1 sans ses 3 variables.
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)]

# On cre M la liste de tous les monmes possibles.
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)]

L = len(M)
```

```

# On cre la matrice Mat de taille N*L dans F2.
Mat = matrix(GF(2),3*N,L)

# On cre le vecteur de longueur N.
Vec = vector(GF(2),3*N)


# On utilise les indications de l'nonc.
for i in range(3*N):
    Vec[i] = Eq[i].monomial_coefficient(1)
    for j in range(L):
        Mat[i,j] = Eq[i].monomial_coefficient(M[j])


# On rajoute le Vecteur aux rsultats prvues
Right = [Vec[i]+Res[i] for i in range(3*N)]

Sol = Mat.solve_right(Right)

#On a besoin que les 61 premiers bits du sol pour rcuprer
# R1, R2 et R3 :
R1 = Sequence([Sol[i] for i in range(3)]+[1]+[Sol[i] for i in range(3,18)])
R2 = Sequence([Sol[i] for i in range(18,23)]+[1]+[Sol[i] for i in range(23,39)])
R3 = Sequence([Sol[i] for i in range(39,43)]+[1]+[Sol[i] for i in range(43,61)])
print("R1 =", R1)
print("R2 =", R2)
print("R3 =", R3)
print("R4 =", R4)
print("\n")

# Maintenant on calcul les autres Registres qui produisent les deux autres suites chiffrantes.
# On transforme les 4 registres trouvs en vecteur pour pouvoir calculer les autres registres.
X1 = vector(R1)
X2 = vector(R2)
X3 = vector(R3)
X4 = vector(R4)

# on calcul les registres qui produisent z1
X1prime = X1 + vector([0 for _ in range(18)]+[IVprime[21]])
X2prime = X2 + vector([0 for _ in range(21)]+[IVprime[21]])
X3prime = X3 + vector([0 for _ in range(22)]+[IVprime[21]])
X4prime = X4 + vector([0 for _ in range(16)]+[IVprime[21]])

# on met un seul bit  1 dans chaque registre :
X1prime[3] = 1
X2prime[5] = 1
X3prime[4] = 1
X4prime[6] = 1

# On le remet en sequence
R1prime = Sequence(X1prime)
R2prime = Sequence(X2prime)
R3prime = Sequence(X3prime)
R4prime = Sequence(X4prime)

print("R1prime =",R1prime)
print("R2prime =",R2prime)
print("R3prime =",R3prime)
print("R4prime =",R4prime)
print("\n")


# et celles qui produisentz2:
X1tilde = X1 + A1 * vector([0 for _ in range(18)]+[IVtilde[20]])
X2tilde = X2 + A2 * vector([0 for _ in range(21)]+[IVtilde[20]])
X3tilde = X3 + A3 * vector([0 for _ in range(22)]+[IVtilde[20]])
X4tilde = X4 + A4 * vector([0 for _ in range(16)]+[IVtilde[20]])
```

```

# on met un seul bit  1 dans chaque registre :
X1tilde[3] = 1
X2tilde[5] = 1
X3tilde[4] = 1
X4tilde[6] = 1

# On le remet en sequence
R1tilde = Sequence(X1tilde)
R2tilde = Sequence(X2tilde)
R3tilde = Sequence(X3tilde)
R4tilde = Sequence(X4tilde)

print("R1tilde =",R1tilde)
print("R2tilde =",R2tilde)
print("R3tilde =",R3tilde)
print("R4tilde =",R4tilde)
print("\n")

# on produit ztest et on teste si c'est gale au rsultat voulu
ztest = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4)
print("z0 = A5_2_productionv_2(N, R1, R2, R3, R4, f1, f2, f3, f4) est ", ztest ==z0)

# On teste les deux suites chiffrantes avec leurs valeurs voulues
zprimetest = A5_2_productionv_2(N, R1prime, R2prime, R3prime, R4prime, f1, f2, f3, f4)
print("z1 A5_2_productionv_2(N, R1prime, R2prime, R3prime, R4prime, f1, f2, f3, f4) est ",
    zprimetest == z1)

ztildetest = A5_2_productionv_2(N, R1tilde, R2tilde, R3tilde, R4tilde, f1, f2, f3, f4)
print("z2 = A5_2_productionv_2(N, R1tilde, R2tilde, R3tilde, R4tilde, f1, f2, f3, f4) est ",
    ztildetest == z2)
print("\n")

#On voit bien qu on a les bons registres.
# Maintenant on cherche la clef K
# Pour dclarer les 64 inconnues que l'on utilisera dans la clef K.
BPR = BooleanPolynomialRing(64, 'K')
v = BPR.gens()

# Comme dans l'algorithme de A5/2-init les registres sont vides des 4 LFSR sont nulles.
X1 = vector([0 for _ in range(19)]) # registre du LFSR1
X2 = vector([0 for _ in range(22)]) # registre du LFSR2
X3 = vector([0 for _ in range(23)]) # registre du LFSR3
X4 = vector([0 for _ in range(17)]) # registre du LFSR4

# On calcule les registres comme dans le rsultat du (9).
# Dans la premire boucle for:
for i in range(64):
    X1 = A1*X1+vector([0 for _ in range(18)]+[v[i]])
    X2 = A2*X2+vector([0 for _ in range(21)]+[v[i]])
    X3 = A3*X3+vector([0 for _ in range(22)]+[v[i]])
    X4 = A4*X4+vector([0 for _ in range(16)]+[v[i]])

# Comme IV est nulle, dans la deuxime boucle for on multiplie pour chaque registre par leurs
    matrices
# de rtroaction qui sont mis  la puissance 22.
X1 = A1**22*X1
X2 = A2**22*X2
X3 = A3**22*X3
X4 = A4**22*X4

# A la fin de cet algorithme, on met un seul bit  1 dans chaque registre :
X1[3] = 1
X2[5] = 1
X3[4] = 1
X4[6] = 1

# On fait une liste quation
```

```
L = list(X1)+list(X2)+list(X3)+list(X4)

# On concatne les registres trouvs dans le (8)
R = list(R1)+list(R2)+list(R3)+list(R4)

# On va calculer l'idal:
I = ideal([L[i] + R[i] for i in range(81)])

print("L'idal I :", I)
print("\n")

# On utilise la base groebner.
Iq = I.groebner_basis()
print("List(I.groebner_basis()) :" ,list(Iq))
print("\n")

# On cherche les coefficients qui additionnent les variables pour avoir une clef Kprime.
Kprime = [el.constant_coefficient() for el in Iq]
print("Kprime =", Kprime)
print("\n")

# On produit avec la clef et leurs IV respectifs
ztest1 = A5_2v_2(N,Kprime, IV)
zprimetest1 = A5_2v_2(N,Kprime, IVprime)
ztildetest = A5_2v_2(N,Kprime, IVtilde)

# on teste avec leurs valeurs voulus
print("z0 = A5_2v_2(N,Kprime, IV) est ", ztest1 == z0)
print("z1 = A5_2v_2(N,Kprime, IVprime) est ", zprimetest1 == z1)
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 $R4$ 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

.

```
# On cre le Polynomial Ring de Z
Pr.<x,y,u> = PolynomialRing(ZZ)

def gen(k,gamma):
    # On prend au hasard p le premier nombre premier.
    p = random_prime(2**k,lbound = 2**(k-1), proof=false)

    # On prend au hasard q le deuxime nombre premier.
    q = random_prime(2**k,lbound = 2**(k-1), proof=false)

    # Si P et Q sont gal, on recherche pour une autre valeur qui est
    # premier.
    if q==p:
        q=next_prime(q)

    N = p*q

    # On calcule phi(N)= (p-1)*(q-1)
    phi =(p-1)*(q-1)

    # On calcule d'abord d
    d = floor(N**gamma)

    # On cherche le plus grand d possible inferieur  N^gamma
    # et qui soit premier avec phi
    while gcd(phi,d) != 1:
        d = d-1

    # On calcule e=d^(-1)mod(phi)
    e = d.inverse_mod(phi)
    return N,e,d

print("gen(2048,0.5) =", gen(2048,0.5))
```

Ce code est dans "*Exercice*16.*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 :
$e \times d = 1 + x_0 \times \varphi(n)$ et $y_0 = -(p+q)$
On a aussi $A = N + 1$ et la fonction f :

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

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 :

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

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

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

$\Leftrightarrow$

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

$\Leftrightarrow$

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

$\Leftrightarrow$

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

$\Leftrightarrow$

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

$\Leftrightarrow$

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

(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 $u_0 = 1+x_0y_0$, 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 $u_0 = 1 + x_0y_0$ 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 $\bar{h}_{j,k}(u_0,x_0,y_0) = y_0^i\bar{f}^k(u_0,x_0)e^{m-k}$, 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 :
$\bar{h}_{j,k}(u_0,x_0,y_0) = y_0^i(cst \times e)^k e^{m-k}$
$\Leftrightarrow$
$\bar{h}_{j,k}(u_0,x_0,y_0) = y_0^i(cst)^k e^m$
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

. $M=\left\{x^{k-i}u^i \; pour \; k\in\{0,...,m\} \; et \; i\in\{0,...,k\}\right\}\bigcup\left\{y^ju^k \; pour \; j\in\{1,...,t\}\, etk\in\{\lfloor m/t\rfloor j,...,m\}\right\}$
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 $\left\{x^{k-i}u^i \; pour \; k\in\{0,...,m\} \; et \; i\in\{0,...,k\}\right\}$
et $l2$ le cardinal de $\left\{y^ju^k \; pour \; j\in\{1,...,t\} \; et \; k\in\{\lfloor m/t\rfloor j,...,m\}\right\}$.
On a donc que :

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

et

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

. 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

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

.
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 $\lfloor m/t \rfloor j \leq i \leq m$
$\Leftrightarrow$

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

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

.

```
load('Exercice16.sage')

def changevar1e(f):

    # On recupre le degre du polynmes pour crer les listes de monmes.
    d = f.degree()

    # On recherche dans la fonction tous les coefficient des mnomes de la forme x**i*y**j avec i et j entre 1 et d
    for i in range(1,d):
        for j in range(1,d):
        # On calcule tous les coefficients de chaque monme de la forme x**i*y**j
            k = f.coefficient(x**i*y**j)
            if i < j:
                # Comme la puissance de x est suprieur  la puissance de y.
                f = f - k*((x**i*y**j) - ((u-1)**i*y**(j-i)))

            if i ==j :
                #Comme la puissance de x est gale  la puissance de y.
                f = f - k*((x**i*y**j) - (u-1)**i)

            if j<i :
                # Comme la puissance de y est suprieur  la puissance de x.
                f = f - k*((x**i*y**j) - ((u-1)**j*x**(i-j)))

    return f

def lattice(m,t,N,e,gamma):
    e1 = ZZ(e)
    # D'aprs l'nonc:
    X = integer_ceil(e1**gamma)
    Y = integer_ceil(e1**(0.5))
    U = 1+X*Y

    # On cre la fonction ft

    ft = u + (N+1)*x

    # On cre la liste de famille de polynomes ht
    H = [y**j*ft**k*e**(m-k) for j in range(1,t+1) for k in [integer_floor(m/t)*j..m]]

    # Pour chaque polynme ht on fait le changement de x*y par (u-1)
    H1 = [changevar1e(i) for i in H]

    # On cre la liste des deux familles de plynomes.
    L =[x**i*ft**k*e**(m-k) for k in range(m+1) for i in range(m-k+1)]+H1

    # On rajoute x*X,y*Y,u*U
    L1 = [i.subs({x:x*X, y:y*Y, u:u*U}) for i in L]

    # On cre la liste des monmes.
    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) for k in [integer_floor(m/t)*j..m]]

    # On cre la matrice avec les coefficients de chaques monmes par rapport aux polynmes.
    Mat = matrix(ZZ,len(L1),len(M))
    for i in range(len(L1)):
        for j in range(len(M)):
            Mat[i,j] = L1[i].monomial_coefficient(M[j])
    return Mat

# On essaie avec un petit exemple
```

```
N,e,d =gen(5,0.7)
print("N = %d e =%d" %(N, e))
gamma = 0.7
X = integer_ceil(e**gamma)
Y = integer_ceil(e**(0.5))
U = 1+X*Y
print("X = %d Y = %d U = %d" %(X, Y, U))
print("\n")
print(" lattice(2, 1, %d, %d, %.2f) = " %(N, e, gamma))
print(lattice(2,1,N,e,gamma))
```

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

22

.

```
Pr.<x,y,u> = PolynomialRing(ZZ)

def changevar2(f):
    #On change u en 1+x*y
    f = f.substitute(u = 1+x*y)
    return f

# On teste une petite fonction
f = x*u +y*u**2 +2
print("f =",f)
print("changevar2(f) =", changevar2(f))
```

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

23

. On pose

$$P(x_0,y_0)=\sum_{i=0}^{k^{'}-1}\sum_{j=0}^{k-1}|P_{i,j}x_0^iy_0^j|\ \ avec\ kk^{'}=\omega$$

$$\leq \sum_{i=0}^{k^{'}-1}\sum_{j=0}^{k-1}|P_{i,j}|x_0^iy_0^j$$

$$\leq \sum_{i=0}^{k^{'}-1}\sum_{j=0}^{k-1}|P_{i,j}|X^iY^j$$

$$=<(|P_{0,0}|,...,|P_{k^{'}-1,k-1}|),(1,1,...,1)>$$

$$\leq ||P(xX,yY)||\times||(1,1,...,1)||$$

$$=||P(xX,yY)||\sqrt{\omega}\ <\ e^m$$

La majoration du produit scalaire des normes correspond à l'inégalité de `Cauchy-Schwartz`.
Comme $P(x_0,y_0)=0\ (mod\ e^m)$, 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

.

```
load('Exercice21.sage')
load('Exercice22.sage')

def factorisation(N,e,gamma,m):
    e1 = ZZ(e)
   # D'aprs l'nonc:
    X = integer_ceil(e1**gamma)
    Y = integer_ceil(e1**(0.5))
    U = 1+X*Y
    t = integer_floor(m*(1-2*gamma))
    Mat =lattice(m,t,N,e,gamma)
    Matl = Mat.LLL()

    Pr.<x,y,u> = PolynomialRing(QQ)
    # On cre une liste de monmes
    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]]

    # On cre les polynomes de chaque ligne de la matrice.
    P=[sum(Matl[i][j]*M[j] for j in range(len(M))) for i in range(len(M))]



    # On remplace u par 1+x*y
    Pchange = [changevar2(i) for i in P]



    # on initialise i=0
    i = 0
    P1 = Pchange[i]
    P2 = Pchange[i+1]

    # On met Pxy un nouvelle anneau polynomial en fonction de x et y
    Pxy.<x, y> = PolynomialRing(QQ)
    # On caste P1 et P2 en Pxy

    PP1 = Pxy(P1)
    PP2 = Pxy(P2)


    res = PP1.resultant(PP2,y)

     # On met Px un nouvelle anneau polynomial en fonction de x
    Px.<x> = PolynomialRing(QQ)

    resx = Px(res)


    while res.degree() < 0 and i!=len(M)-2:
        i = i+1
        P1 = Pchange[i]
        P2 = Pchange[i+1]
        PP1 = Pxy(P1)
        PP2 = Pxy(P2)


        # On met Px un nouvelle anneau polynomial en fonction de x

        res = PP1.resultant(PP2,y)

        Px.<x> = PolynomialRing(QQ)
```

```
        resx = Px(res)





    # On regarde sur on est sorti de la liste de fonctions
    if i == len(M)-2:
        return (-1)
    # On cherche la premiere valeur
    x0 = resx.roots()[0][0]


    #On reinjecte cette valeur sur le premier polynme
    PPP1 =P2.subs(x=x0)

    # On met Py un nouvelle anneau polynomial en fonction de y
    Py.<y> = PolynomialRing(QQ)

    # On le caste en Py
    equy = Py(PPP1)

    # On cherche la racine de cette quation en fonction de y
    y0 =equy.roots()[0][0]

    # D'aprs les rsultats de la question 16 et 17
    d = (x0*(N+1+y0)+1)/e1

    return (d )


#premier exemple demand
N,e,d =gen(1024,0.2)
gamma = 0.2
m = 2

# On teste si on a trouv le bon d
d1 = factorisation(N, e, gamma, m)

print("d = factorisation(%d, %d,%.2f, %d) est %s" %(N, e, gamma, m, d==d1))

# 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

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

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.