Projet de Code Correcteur d'Erreur :
Cryptosystéme de Mc Eliece

El Hadji Mamadou Dia

UFR Sciences Appliquées et Technologies
Universite Gaston Berger de Saint Loius
Master I Cryptologie,Codage et Application (CCA)

Avril 2022

Sommaire

Introduction

Dans le domaine de la théorie de l'information, les codes sont utilisés pour détecter et corriger les erreurs de transmission. En effet, lorsqu'une source souhaite transmettre un message à un destinataire, elle envoie ce message au travers d'un canal dans lequel peuvent apparaître des erreurs dues à des bruits.

Ainsi le message que le destinataire reçoit peut avoir été modifié. Afin de protéger l'information de ce genre d'erreurs, on ajoute au message de la redondance avant de le transmettre dans le canal. A la réception, le décodeur tente de récupérer le message originel à partir du message reçu.

Pour corriger ces erreurs, on utilise le code correcteur des erreurs qui est basé sur des problèmes difficiles qui sont : Le problème du décodage (déterminer le mot du code le plus proche de m : message clair) et le problème de déterminer sa distance minimale.

Mc Eliece utilise la théorie des codes correcteurs d'erreurs à des fins Cryptographiques, et plus précisément pour un algorithme de chiffrement asymétrique.

Leur principe est : Alice envoyer un message contenant un grand nombre d'erreurs, erreurs que seul Bob sait détecter et corriger.

Motivations

La plupart des cryptosystèmes à clé publique utilisés sont basés sur des problèmes de théorie des nombres. Il est important de connaître des systèmes alternatifs efficaces en pratique (réseaux, systèmes multivariés, codes correcteurs d'erreurs, fonctions de hachage. .).

Les avantages de la cryptographie basés sur les codes correcteurs d'erreurs: a priori sûre face à l'ordinateur quantique, des problèmes NP-complets bien connus, rapides et faciles à implanter.

Les désavantages de la cryptographie basés sur les codes correcteurs d'erreurs :grande taille de clé publique (des centaines de milliers de bits . . .).Récemment, les systèmes basés sur les codes correcteurs d'erreurs ont été présentés avec de petites tailles de clés (aux alentours de 30 000 bits pour le chiffrement).

Historique

C'est le plus ancien crypto système à clef publique utilisant des codes correcteurs d'erreurs. Il a été imaginé par **Mc Eliece en 1978**, à peu près en même temps que **RSA**. Comme tous les crypto systèmes à clef publique, ce système est constitué de 3 algorithmes :

- la génération de clefs.
- le chiffrement (utilisant la clef publique).
- le déchiffrement (utilisant la clef secrète).

Mc Eliece a suggéré d'utiliser les codes de Goppa, qui sont des codes linéaires avec un algorithme rapide de décodage.

Generation de cle

Nous déterminons une matrice generatrice **G** du code de Goppa avec un polynôme irréductible **g(x)** et un support **L**.
En réalité, la clef publique du système n'est rien d'autre que cette matrice génératrice du code de Goppa. C'est à dire la cle publique est **G**.
En plus la clé privé du systeme est composé du polynôme irréductible **g(x)** et le support **L** qu'on a généré aléatoirement.

Clé publique

Pub(G)

Clé privée

Priv(L, g)

Voici l'algorithme de génération de clé:

```
def genkey(n, m, t):
    k = n - (m*t)
    if (n < 2^m & n > m*t) :
        print("Il faut que n soit superieur a m facteur de 2t + 2")
    else :
#Generation d'un polynome irreductible g(x) tels que deg(g) = t
        F = GF(2^m, 'Z')
        D = GF(2)
        R = PolynomialRing(F, 'X')
        while(1):
            irreductible_poly = R.random_element(t)
            if irreductible_poly.is_irreducible():
                break
        g = irreductible_poly
        show("Nous avons generer le polynome irreductible ci-dessous\n")
        show("g = ",g)
#Generation du support L tels que Li != Lj et g(Li) != 0
        L = random_vector(F , n)
        i = 0
        while(i != n):
            for j in range(1 , n):
                while(L[i] != L[j]) & (g(L[j]) != 0):
                    break
            i = i + 1
        show("Nous avons generer le support ci-dessous\n")
        show("L = ",L)
        H = matrix(F, [[L[j]^i / g(L[j]) for i in [0..n-1]] for j in [0..t-1]])
        show("La matrice de parite est donne par:")
        show("H = ",H)
        Hb = matrix(D, m*H.nrows(), H.ncols())
```

```
#Projectons chaque element sur m ligne
        for i in range(H.nrows()):
            for j in range(H.ncols()):
                tab = bin(H[i,j].integer_representation())[2:]
                tab = tab[::-1]
                tab = tab + '0'*(m - len(tab))
                tab = list(tab)
                Hb[m*i:m*(i + 1), j] = vector(map(int, tab))
        show("Hb = ",Hb)
#Forme systematique
        Hp = Hb.transpose().kernel().basis_matrix()
        show("Hp = ",Hp)
        M = Hp.matrix_from_columns([t..n-1])
        show("M = ",M)
        G =  block_matrix([[identity_matrix(D,k)],[M.transpose()]])
        show("G = ",G)
    return G,L,g,H
B, T, s, A = genkey(10, 4, 2)
print("_____________________________________________________________________________")
show("La cle publique est : ")
show("Public Key = ",B)
print("_____________________________________________________________________________")
print("_____________________________________________________________________________")
print("_____________________________________________________________________________")
print("_____________________________________________________________________________")
show("La cle prive est : ")
show("Priv Key = ",(T,s))
```

Chiffrement

Pour chiffrer un message donné on multiplie ce dernier par la clé publique **G** et on l'additionne avec l'erreur **e** qu'on a générer aléatoirement.
voici l'algorithme de chiffrement:

```
16
17 def encrypt(message, m, n, t):
18     F = GF(2)
19     k = n - (m*t)
20     #Generation de l'erreur e tel que le poids de e est egale a t
21     e = random_vector(F , n)
22     while e.hamming_weight() != t:
23         e = random_vector(F , n)
24     show("e = ", e)
25     #Le messgae chiffrer est :
26     c = matrix((B*message) + e)
27     return c
28 o = encrypt(vector([1,0]),4, 10, 2)
29 print("_____
30 print("_____
31 show("Le message chiffrer est : ")
32 show("c = ",o)
33 print("_____
34 print("_____
```

Dechiffrement

Pour le déchiffrement nous avons utilisés l'algorithme de **Paterson**. Nous avons calculé le syndrome, ensuite l'équation cé pour déchiffrer le message.
Voici l'algoriyhme de déchiffrement :

```
def decrypt(m, n, t):
    F = GF(2^m, 'Z')
    D = GF(2, 'Z')
    Z = F.gen()
    R = PolynomialRing(F, 'X')
    X = R.gen()
#Construction du Syndrome
    syndrome = matrix(R, 1, len(T))
    for i in range(len(T)):
        syndrome[0,i] = (X - T[i]).inverse_mod(s)
    show("syndrome = ",syndrome)
    synd = syndrome*o.transpose()
    syndrome_poly = 0
    for i in range (synd.nrows()):
        syndrome_poly += synd[i,0]*X^i
    show("syndrome_poly = ", syndrome_poly)
    X = s.parent().gen()
#Square root
    (g0,g1) = split(s); sqrt_X = g0*inverse_g(g1);
    (T0,T1) = split(inverse_g(syndrome_poly) - X);
    R = (T0 + sqrt_X*T1).mod(s)
```

```
#Dechiffrement
    for i in range(len(T)):
        if sigma(T[i]) == 0:
            message_code[0,i] += 1
    return message_code
message_decode = decrypt(4, 10, 2)
print("_____________________________________________________________________")
print("_____________________________________________________________________")
print("Le message dechiffrer est : ")
show("message_decode = ",message_decode)
print("_____________________________________________________________________")
print("_____________________________________________________________________")
```