

La sécurisation des messages à l'aide de la cryptographie sur les courbes elliptiques

13262 BAHOUS Youness

Épreuve de TIPE

Session 2021

Plan de l'exposé

- ① Logarithme Discret
- ② Courbes elliptiques sur un corps fini
- ③ Chiffrement de Bout en Bout
 - Encodage
 - Authentification
- ④ Conclusion

Logarithme Discret

Position du problème

- L'importance de sécurité cybernétique
- Sécuriser les communications dans un canal

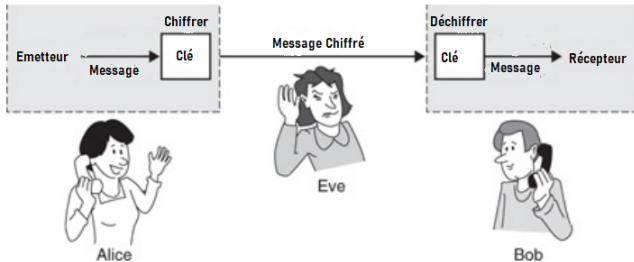


FIGURE – nécessité chiffrement à clé publique

Problème de Logarithme Discret

Problème du logarithme discret (PLD)

Soit G un groupe cyclique d'ordre m , $\alpha \in G$ un de ses générateurs pour tout $\beta \in G$ **On cherche l'unique a défini modulo m tel que**

$$\alpha^a = \beta$$

on le note $\log_\alpha(\beta)$

Logarithme Discret

Applications du Logarithme discret

Le problème du logarithme discret est particulièrement difficile sur des corps finis (Cycliques)

- *L'algorithme RSA* (PLD Dans $\mathbb{Z}/p\mathbb{Z}$) (corps multiplicatif fini)
- *Les courbes elliptiques* (PLD sur le Groupe de points d'une courbe elliptique sur un corps fini)

Logarithme Discret

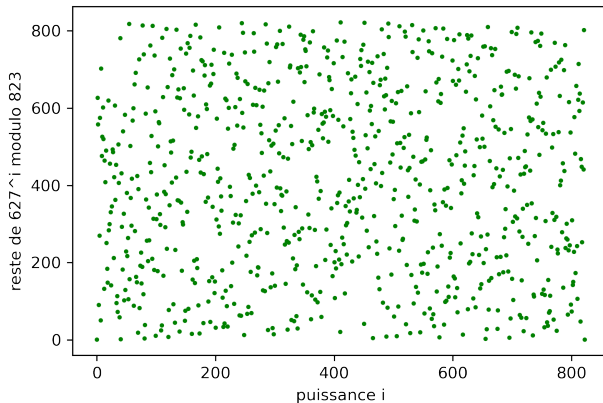


FIGURE – illustration de difficulté de résolution du PLD en $\mathbb{Z}/p\mathbb{Z}$

Courbes elliptiques sur un corps fini

Notion

Courbes elliptiques sur un corps fini

Une courbe elliptique sur un corps K noté $E(K)$ est l'ensemble des solutions (X, Y) dans K à **une équation de WIEISTRASS** la forme $Y^2 = X^3 + AX + B$, avec $(A, B) \in K$, tel que le **DISCRIMINANT** $\Delta = 4A^3 + 27B^2 \neq 0$

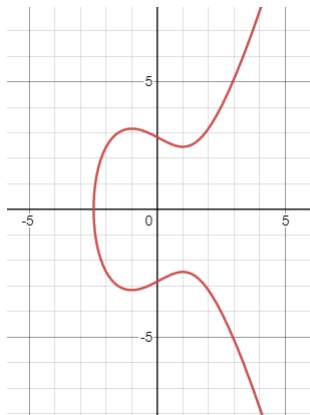


FIGURE – $y^2 = x^3 - 3x + 8$ sur \mathbb{R}

Loi de Groupe sur les courbes elliptiques Loi de composition des cordes-tangentes

soient $P(x_P, y_P)$, $Q(x_Q, y_Q)$ deux points d'une courbe elliptique E on définit la loi de composition interne $P + Q$ d'élément neutre \mathcal{O} telle que :

- ① Si $x_P \neq x_Q$, on pose :

$\lambda = \frac{y_p - y_q}{x_p - x_q}$

et

$\nu = \frac{x_P y_Q - x_Q y_P}{x_P - x_Q}$

et on calcule $P + Q = \lambda^2 - x_P - x_Q, -\lambda(\lambda^2 - x_P x_Q) - \nu$

- ② si $x_P = x_Q$ et $P \neq Q$: $P + Q = \mathcal{O}$

- ③ si $P = Q$ et $y_P \neq 0$ posons

$\lambda = \frac{3x_p^2 + a}{2y_p}$

et

$\mu = \frac{-x_p^3 + ax_p + 2b}{2y_p}$

et on applique la formule de sommation

- ④ si $P = Q$ et $y_P = 0$: alors

$$2P = \mathcal{O}$$

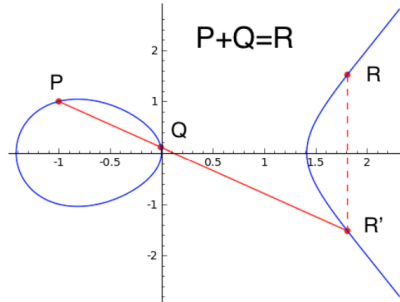


FIGURE – somme de deux points sur la courbe $y^2 = x^3 - 2x$

la loi du groupe I

```
p = int(input('donner le cardinal de votre corps cyclique de base'))
a,b = input('La courbe de la forme Y**2=X**3+A*X+B donnez (A,B)').split(',')
a=int(a)
b=int(b)
# L'element neutre Pour la loi du groupe
O='infini'
def Point(x,y):
    return (x,y)
def Appartient_courbe(P):
    return (P == O or ((P[1]**2 - (P[0]**3 + a*P[0] + b)) % p == 0
                        and 0 <= P[0] < p
                        and 0 <= P[1] < p))
def inv_mod_p(x):
    if x % p == 0:
        raise ZeroDivisionError("Impossible inverse")
    return pow(x, p-2, p)
def ec_inv(P):
    """
    Inverse d'un point P la courbe  $y^2 = x^3 + ax + b$ .
    """
    if P == O:
        return P
    return Point(P[0], (-P[1])%p)
```

la loi du groupe II

```
def ec_add(P, Q):
    if not (Appartient_courbe(P) and Appartient_courbe(Q)):
        raise ValueError("Entree invalide")
    if P == 0:
        return Q
    elif Q == 0:
        return P
    elif Q == ec_inv(P):
        return 0
    else:
        if P == Q:
            s = (3 * P[0]**2 + a) * inv_mod_p(2 * P[1])
        else:
            s = (Q[1] - P[1]) * inv_mod_p(Q[0] - P[0])
        x = (s**2 - P[0] - Q[0]) % p
        y = (s * (P[0] - x) - P[1]) % p
        somme = Point(x, y)
    assert Appartient_courbe(somme)
    return somme
```

Problème :

Soient \mathbb{K} un corps fini et E une courbe elliptique définie sur \mathbb{K} , Soit P un point de la courbe $E(\mathbb{K})$. Trouver un entier n , s'il existe, tel que

$$nA = P$$

Présentation de processus

Encryption du texte à l'aide des courbes elliptiques



Utilisation de ASCII pour écrire du texte comme entier

Soit Un message $M = (m_1, \dots, m_n)$, on associe à chaque caractère m_k son code dans le schéma Unicode a_k ainsi M est représenté par (a_1, \dots, a_n) et en considérant $b = 2^{16}$ la base de codage :

$$m = \sum_{k=0}^n a_k b^{k-1}$$

Réciproquement, sachant m on peut retrouver les a_k à l'aide de la formule :

$$a_k = \left\lfloor \frac{m}{b^{k-1}} \right\rfloor \bmod b$$

Limitations techniques imposées par les prochaines étapes de chiffrement

Contraintes techniques

Pour chiffrer le message m obtenu par l'encodage on doit avoir un nombre premier $p > m$ on trouve ainsi des contraintes sur la taille de m , on se limite aujourd'hui à des blocs de texte de **160** caractères.

Code Python

```
def codage_uni(M): #M= le message texte #n nombre de blocs #l=longueur de bloc
    m,b=0,1
    for i in range(len(M)):
        m+=b*ord(M[i])
        b=b*2**16
    return m

#####

def codage_multi(M,n):
    return [codage_unibloc(M[k:k+n]) for k in range(0,len(M),n)]
#####

def decodage(N):
    M=""
    b=1
    a=2**16
    while N//b!=0:
        M+=chr((N//b)%a)
        b=b*a
    return M

#####

def decodage_multi(L):
    return ''.join(decodage(N) for N in L)
```


Utilisation du Codage de KOBLITZ

Conditions

Codage de KOBLITZ : Les Conditions

soit E une courbe elliptique d'équation $y^2 = x^3 + Ax + B$ définie sur $\mathbb{Z}/p\mathbb{Z}$ on choisit p tel que :

- ❶ p ne divise pas $-16(4A^3 + 27B^2)$
- ❷ $p \equiv 3 \pmod{4}$
- ❸ p est au moins de 2560 bits

Un Lemme clé

soit $n \in \mathbb{Z}$ et p premier vérifiant $p \equiv 3 \pmod{4}$ alors

$$y^2 \equiv s \pmod{p} \iff s^{\frac{p+1}{2}} \equiv s \pmod{p} \text{ et } y \equiv s^{\frac{p+1}{2}}$$

Utilisation du Codage de KOBLITZ

Algorithme

Codage de KOBLITZ : L'Algorithme

- ❶ on calcule $m = \sum_{k=0}^n a_k b^{k-1}$
- ❷ on fixe $d \leq \frac{p}{m}$
- ❸ pour $i \in \{1, \dots, d-1\}$
 - ❶ On calcule l'abscisse $x_i = (dm + i) \bmod p$ d'un point de la courbe avec $m = \lfloor \frac{x_i}{d} \rfloor$
 - ❷ On calcule $s_i = (x_i^3 + Ax_i + B) \bmod p$
 - ❸ Si $(s_i)^{\frac{p+1}{2}} \equiv s_i \bmod p$ on définit l'ordonnée $y_i = (s_i)^{\frac{p+1}{4}} \bmod p$

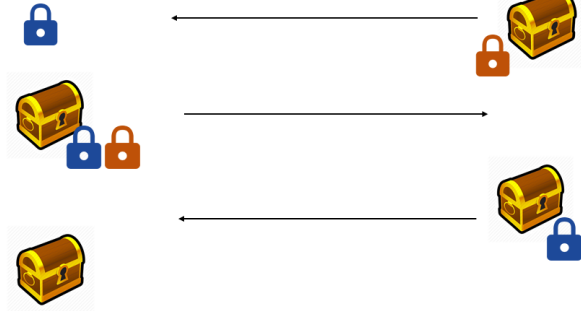
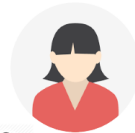
Code Python

```
C=[A,B,p,N] #p est le cardinal du groupe de base
           #N est la cardinalité de la courbe
def koblitz_codage(C,M):
    b = 216
    A = C[0] # les coefficients de la courbe elliptique
    B = C[1]
    p = C[2] # Cardinalité (Nombre de Points De La Courbe)
    n = len(M)
    m = sum([ord(M[i])*b**i for i in range(n)])
    d = min(floor(p/m),100)
    for j in range (p):
        x = (d*m + j) % p
        s = (x3 + A*x + B) % p
        print(s)
        if s==pow(s,int((p+1)/2),p):
            y = pow(s,int((p+1)/4), p)
            return (x,y)
def koblitz_decode(P):
    b = 216
    d = 100
    lst = []
    m = floor((P[0])/d)
    while m != 0:
        lst.append(chr(m%b)) #convertir m to en liste de caractères
        m = m//b #replace Nmb by partie entière de (Nmb/b)
    return "".join(lst)
```




Massey Omura

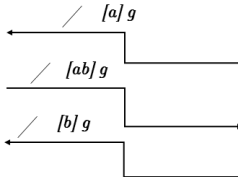
présentation




Massey Omura protocole

un groupe abélien G d'un
corps fini.

- 
3. Bob Choisit sa clé secrete b avec $b \wedge |G| = 1$, et $[ba]g = [ab]g$ et l'envoie à Alice
 5. Bob calcule $b^{-1} = \beta \in G$ et calcule $[\beta b]g = g$.



- 
1. représente son message sous la forme d'un élément $g \in G$
 2. Alice choisit un entier secret a avec $a \wedge |G| = 1$, calcule $[a]g$
 4. Alice Calcule $a^{-1} = \alpha \in G$ elle calcule $[aab]g = [b]g$ et l'envoie à Bob

Code Python

```
def cle(C): # C désignant la un tableau représentant la courbe à 4 paramètres [p,A,B,N]
    N = C[3]
    #On choisit de manière aléatoire la clé d'encryption
    a = random()
    while (gcd(a,N) != 1):
        a = random()
    return a

def massey_omura(C, Message):
    a = cle(C)
    #On code le message sur un point
    P = koblitz_codage(C,Message)
    Q = multipP(a,P)
    return ([a, Q])

def inverse(a, C):
    N = # nombre de points sur la courbe
    alpha = power_mod(a, -1, N)
    return alpha
```

Exemple d'exécution sur SageMath

Transmission du message 'Grandes écoles'

#Définition de la courbe

```
p = 0x01fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
K = GF(p)
a = K(0x01fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
b = K(0x0051953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef109e15619395
E = EllipticCurve(K, (a, b))
G = E(0x00c6858e06b70404e9cd9e3ecb662395b4429c648139053fb521f828af606b4d3dbaa14b5e7
E.set_order(0x01fffffffffffffffffffffffffffffffffffffffffffffffffffffffffa5
```


Exemple d'exécution sur SageMath

```
: M = 'Grandes ecoles'  
string = M
```

```
: [a,aP]=massey_omura(E, string)  
  
print(a)
```

```
8859848207944786693473325788800329000351473306724326927294788158997959460785836067105207897766235873201138125414307672249382631  
06912691694748248601456864060
```

```
: print(aP)  
  
(417990132747365192186104208866071773248074487536464422529740386181093173931025663600145272628726039520291445134328978805339607  
3767680663868058506236801746400 : 295908138361705067931327178675379591987792281654698426895623582496883210632344955760923789896  
893830421553988558788068233123438049295289715665755884415883943 : 1)
```

```
: b=cle(E)  
print(b)  
  
4511101062141242565342529544308622943636433456028304095397740398963798806973403215367167112452777738639705543466879323896879540  
622958838378773322264604713424
```

Exemple d'exécution sur SageMath

```
alpha=inverse(a,E)
print(a)
```

885984820794478669347332578880032900035147330672432692729478815899795946078583606710520789776623587320113812541430767224938263106912691694748248601456864060

```
beta=inverse(b,E)
print(b)
```

451110106214124256534252954430862294363643345602830409539774039896379880697340321536716711245277738639705543466879323896879540622958838378773322264604713424

```
baP=b*aP
```

```
baP
```

(1859693054357198464611576100944939915086424349582058228308097553008733492117426006638249787753865195139437609451793481855459752185510588596110308711769575557 : 6081993022101120316675418086751073203465289387740781098623313331218761843053103234205607543918782067839807860388344887687829195200995746683524725567505042533 : 1)

Exemple d'exécution sur SageMath

```
bP
```

```
(3670545349542034433435828907767893740706863425648326279372768135682654625404287234587540555142652797678586804186687929511583161107621820058155204893695015842 : 4773090944835148240922543999120568480121313433246739845422723936584952609820679393249801100388545761775104516402410288769914388604658229436941410073179279166 : 1)
```

```
P=beta*bP
```

```
P
```

```
(4730889002060445489644681308073808735147345385853276794143860333501 : 29775698249141323677718606628560252218514442199084112520413716670115373440745370063315431659316377546615831757862282211008984110671577553539524635386631581 : 1)
```

```
koblitz_decodage(P)
```

```
'Grandes ecoles'
```

Conclusion

Réponse à la problématique

La manipulation de la loi de groupe sur les points d'une courbe elliptique sur un corps fini permet ainsi le chiffrement de bout en bout des messages textuels