TIPE: Fonctionnement et sécurité du cryptosystème d'ElGamal lors d'un cryptage de numéro de carte bancaire

LE GALL Mathis

Épreuve de TIPE

Session 2020/2021

Enjeux sociétaux

Considérons ce numéro de carte bancaire :

4970 9424 1234 5678

Quelle est la taille d'un groupe cyclique permettant un échange de numéro de carte bancaire sécurisé via le cryptosystème d'ElGamal ?

Plan de l'exposé

Introduction

Position du problème

Le système de chiffrement d'ElGamal :

- **Problème du logarithme discret**

  Dans un groupe cyclique G d'ordre n, pour $y_a \in \mathrm{G}$, trouver $x_a \in \mathbb{Z}$ tel que

$$g^{x_a} = y_a \iff x_a = \log_g(y_a)$$

Introduction

Notations

- p est le nombre premier utilisé.
- g est le générateur choisi de $(\mathbb{Z}/p\mathbb{Z})^{\times}$.
- x_a est la clé privée du récepteur.
- $y_a = g^{x_a}$.
- c_a est le triplet (p, g, y_a).
- x_b est la clé personelle de l'émetteur.
- $y_b = g^{x_b}$.
- m est le message à transmettre.
- m_c est le message crypté.
- c_b est le doublet (m_c, y_b).

Chiffrement d'ElGamal

Principe

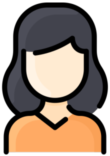

FIGURE – Récepteur

- Choix d'un nombre p premier très grand. $\boxed{p = 13}$
- Calcul du générateur g. $\boxed{g = 2}$
- Choix d'une clé privée $x_a \leq p - 1$. $\boxed{x_a = 7}$
- $y_a = g^{x_a} \pmod{p}$. $\boxed{y_a = 2^7 = 11 \pmod{13}}$
- Génération du triplet $c_a = (p, g, y_a)$. $\boxed{(13, 2, 11)}$

Chiffrement d'ElGamal

Principe

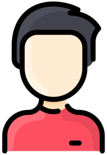

Figure – Emetteur

- Réception du triplet c_a
- Choix d'une clé personnelle $x_b \leq p - 1$ $\boxed{x_b = 9}$
- $m_c = m \cdot y_a^{x_b} \pmod p$ $\boxed{m_c = 12 \cdot 11^9 = 5 \pmod{13}}$
- $y_b = g^{x_b} \pmod p$ $\boxed{y_b = 2^9 = 5 \pmod{13}}$
- Génération du doublet $c_b = (m_c, y_b)$ $\boxed{(5, 5)}$

Chiffrement d'ElGamal

Principe

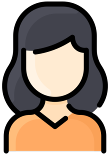

Figure – Récepteur

- Déchiffrage du message :

$$\frac{m_c}{y_b^{x_a}} = \frac{m \cdot y_a^{x_b}}{g^{x_b \cdot x_a}} = \frac{m \cdot g^{x_b \cdot x_a}}{g^{x_b \cdot x_a}} = m \qquad \boxed{\frac{5}{5^7} = \frac{12 \cdot 11^9}{2^{9 \cdot 7}} = \frac{12 \cdot 2^{9 \cdot 7}}{2^{9 \cdot 7}} = 12}$$

Figures extraites du site www.flaticon.com

Introduction

Choix du groupe

- $(\mathbb{Z}/p\mathbb{Z})^\times$ est l'ensemble des éléments **inversibles** de $(\mathbb{Z}/p\mathbb{Z})$.
- $|(\mathbb{Z}/p\mathbb{Z})^\times| = \varphi(p) = p - 1$.
- $(\mathbb{Z}/p\mathbb{Z})$ est un corps $\iff p$ **premier**.
- p premier $\Rightarrow ((\mathbb{Z}/p\mathbb{Z})^\times, \times)$ **cyclique.**

Introduction

Choix du groupe

On se place dans $(\mathbb{Z}/p\mathbb{Z})^{\times}$ avec p un grand nombre premier.

Intérêts :

- Sécurité
- Transmission d'un plus grand nombre de caractères

A la recherche d'un grand nombre premier

Algorithme de Rabin Miller

Soit p un nombre impair.

- On décompose $p - 1$ sous la forme $p - 1 = 2^s \times m$.

Pour tout $a \in (\mathbb{Z}/p\mathbb{Z})^\times$, trois cas possibles :

- $a^m \equiv 1 \pmod p$.
- $\exists i \in [\![1\ ;\ s-1]\!]$ tel que $a^{2^i m} \equiv -1 \pmod p$.
- Aucun des deux cas précédents n'est réalisé.

A la recherche d'un grand nombre premier

Algorithme de Rabin Miller

- p est un **témoin de Miller** s'il passe le test.
- Probabilité que p soit composé : $\frac{1}{4^k}$ (k tests).
- **Complexité temporelle** en $\mathcal{O}(k \times (\log(p))^3)$.
- 45 secondes pour trouver $p = 10000000000000000000000000000000000000000000000000000$ $000000000000000000000000000000000000000000000000000043723$

Recherche d'un générateur

Principe

- **Propriété** : Soit $g \in (\mathbb{Z}/p\mathbb{Z})^\times$ avec $\frac{p-1}{2}$ premier.
  g est générateur $\iff g^{\frac{p-1}{2}} \equiv -1 \pmod p$.

Preuve :

- Petit théorème de **Fermat**.
- $(\mathbb{Z}/p\mathbb{Z})^\times$ est un **corps**.
- Théorème de **Lagrange**.

Connaissant $p-1 = \prod_{i=1}^{n} p_i^{k_i}$,

- **Propriété** : g est générateur $\iff \forall i \in [\![0 \ ; \ n]\!] \quad g^{\frac{p-1}{p_i}} \neq 1$.

Exemple de fonctionnement

Cryptage de numéro carte bancaire

Bob souhaite envoyer son numéro de carte bancaire à Alice.
Génération du triplet c_a :

- $p = 10^{100} + 43723$
- Générateur $g = 2$
- Clé privée $x_a = 79884$
- $c_a = (10^{100} + 43723, 2, 909193757763375166213052361815724131285427082552257488173418251006292213002096757445085541670313582 8)$

Exemple de fonctionnement

Cryptage de numéro de carte bancaire

Chiffrement du message de Bob et génération du doublet c_b :

- Message : $m = 4970942412345678$
- Clé personnelle : $x_b = 124521$
- $c_b = (m_c, y_b) = (8289379584784680188195132401055011781$
$67360662021885737412138493427634478498106882974703795563$
$015744337211057806404 0, 6978355027099320812093141289568 01$
$53835548035798990233159963074681173927408427736370878620$
$92010286366)$

Exemple de fonctionnement

Cryptage de numéro de carte bancaire

Décryptage du numéro de carte bancaire par Alice :

$$\frac{m_c}{y_b^{x_a}} = \frac{m \cdot {y_a}^{x_b}}{g^{x_b \cdot x_a}} = \frac{m \cdot g^{x_b \cdot x_a}}{g^{x_b \cdot x_a}} = 4970942412345678$$

Attaque de Pohlig-Hellman

Principe

- Décomposition de l'ordre :

$$p-1 = p_1^{k_1} \times \ldots \times p_n^{k_n} = \prod_{i=1}^{n} p_i^{k_i} \qquad \boxed{12 = 2^2 \times 3}$$

- Recherche des $x_i = x_a \pmod{p_i^{k_i}}$ en base p_i :

$$x_i = \sum_{j=0}^{k_i-1} d_j \times p_j^j \qquad \boxed{x_1 = d_0 + d_1 2^1 \text{ en base } 2}$$

- Résolution du problème en $\mathcal{O}(\max_i p_i \times k_i)$.

Attaque de Pohlig-Hellman

Principe

- Recherche de d_0 :
  $k = \frac{p-1}{p_i}$ $\boxed{k = \frac{12}{2} = 6}$
  $y_a^k = g^{x_a k} = (g^k)^{d_0}$ $\boxed{11^6 = 12\ [13]}$; $\boxed{(2^6)^{d_0} = 12^{d_0}}$
- On obtient d_0 en testant des valeurs entières p_i fois :
  $\boxed{12 \neq 12^0\ ;\ 12 = 12^1} \Longrightarrow \boxed{d_0 = 1}$
- Recherche de d_1 :
  $k' = \frac{k}{p_i}$ et $y_a' = \frac{y_a}{g^{d_0}} = g^{x_a - d_0}$ $\boxed{k' = \frac{6}{2} = 3}$; $\boxed{y_a' = 12}$
  ${y_a'}^{k'} = (g^{x_a - d_0})^{k'} = (g^k)^{d_1}$ $\boxed{12^3 = 12\ [13]}$; $\boxed{(2^6)^{d_1} = 12^{d_1}}$
  $\Longrightarrow \boxed{\mathrm{d}_1 = 1}$

Attaque de Pohlig-Hellman

Principe

- Obtention des x_i $\boxed{x_1 = 1 + 1 \times 2 = 3}$; $\boxed{x_2 = 1}$
- Obtention de x_a grâce au **théorème des restes chinois** :

Pour tout $i \in [\![0 \; ; \; n]\!]$, $x_a = x_i \pmod{p_i^{k_i}}$

$\boxed{x_a = 3 \pmod{2^2}}$; $\boxed{x_a = 1 \pmod{3}}$

$$\Longrightarrow \boxed{x_a = 7}$$

Attaque de Pohlig-Hellman

Exemple

- $(p, g, y_a) = (27581023696414335939127381214338628659 3, 5, 12)$
- $p - 1 = 2^9 \times 3^7 \times 7^8 \times 11^5 \times 13^{11} \times 23^6$
- $x_2 = 222, \quad x_3 = 1789, \quad x_7 = 4628322,$ $x_{11} = 141123, \quad x_{13} = 122262334834, \quad x_{23} = 121441255$
- **Théorème des restes chinois** :

$$x_a = 991856713263528528631959634457873231 02$$

Conclusion

Cryptosystème d'ElGamal

Sécurité assurée via :

- L'utilisation d'un grand nombre p premier.
- Le respect de la condition $\frac{p-1}{2}$ premier.

Code Python

Exponentiation rapide et PGCD

```
def puiss_rec(a,n,p):
    if n == 0:
        return 1
    else:
        if n % 2 == 0:
            b = a*a % p
            return puiss_rec(b,n//2,p)
        else:
            b = a*a % p
            return a*puiss_rec(b,(n-1)//2,p) % p


def pgcd(a,b):
    if a < b :
        return pgcd(b,a)
    elif a % b == 0:
        return b
    else:
        return pgcd(b, a % b)
```

Code Python

Euclide étendu

```
def euclide_rec(a,b):
    if b == 0 :
        return 1,0
    else :
        u,v = euclide_rec(b, a%b)
        return (v, u - (a//b)*v)


def inverse_modulaire(x,p):
    u,v = euclide_rec(x,p)
    return u % p
```

Code Python

Nombres premiers (1)

```
def valuation(p):
    s = 0
    m = p - 1
    while m % 2 == 0:
        s += 1
        m = m//2
    return (s,m)

def Temoin_Miller(a,p) :
    s,m = valuation(p)
    x = puiss_rec(a,m,p)
    if x == 1 :
            return True
    else:
        for i in range(s) :
            t = puiss_rec(a,m,p)
            if t == -1 % p:
                return True
            else :
                m = 2*m
    return False
```

Code Python

Nombres premiers (2)

```
def Rabin_Miller(p,k):
    for i in range(k):
        a = random.randrange(2,p-2)
        if not Temoin_Miller(a,p):
            return False
    return True


def premier(n):
    while not (Rabin_Miller(n,30) and Rabin_Miller((n-1)//2,30)):
        n += 2
    return n


def recherchep(z):
    m,n,k,u,v,w = 3,3,3,3,3,3
    L = [m,n,k,u,v,w]
    a = (2**L[0])*(3**L[1])*(7**L[2])*(11**L[3])*(13**L[4])*(23**L[5])
    while (a < 10**30 or not(Rabin_Miller((a+1),z))) :
        i = random.randint(0,5)
        L[i] += 1
        a = (2**L[0])*(3**L[1])*(7**L[2])*(11**L[3])*(13**L[4])*(23**L[5])
    return a
```

Code Python

Générateur

```
## Generateur (p-1/2 premier)

def est_generateur(a,p):
    return(puiss_rec(a,(p-1)//2,p) == p-1)

def generateur(p):
    g = 2
    while not est_generateur(g,p):
        g = g + 1
    return(g)

## Generateur (facteurs premiers)

def testgen(g,p):
    X = decompo_finale(p-1)
    for i in range (len (X)):
        if puiss_rec(g,((p-1)//X[i][0]),p) == 1 :
            return False
    return True

def gen(p):
    g = 2
    while not testgen(g,p):
        g += 1
    return g
```

Code Python

Décomposition en facteurs premiers (1)

```
def facteurs_premiers(N):
    L = []
    p = 2
    while p*p <= N:
        while (N % p) == 0:
            L.append(p)
            N //= p
        p += 1
    if N > 1:
       L.append(N)
    return L

def count_puiss (p,L) :
    a=0
    n=len(L)
    for i in range (0,n):
        if L[i] == p :
            a+=1
    return a
```

Code Python

Décomposition en facteurs premiers (2)

```
def decompo_finale (N) :
    P= facteurs_premiers (N)
    n=len(P)
    a=0
    X=[]
    i=0
    while i<n :
        a=count_puiss (P[i],P)
        X.append([P[i],a])
        i+=a
    return X
```

Code Python

Décomposition du problème du logarithme discret (1)

```
def deci(p,g,y,q,m):
    L = []
    x = (p-1)//q
    a = puiss_rec(y,x,p)
    d = 0
    while (puiss_rec(g,x*d,p)) != a :
        d += 1
    L.append(d)
    if m > 1 :
        for i in range(1,m):
            k = 0
            x = x//q
            y = y*inverse_modulaire(puiss_rec(g,q**(i-1)*L[i-1],p),p) % p
            u = puiss_rec(y,x,p)
            while puiss_rec(g,((p-1)//q)*k,p) != u:
                k += 1
            L.append(k)
    return L
```

Code Python

Décomposition du problème du logarithme discret (2)

```
def rassemble_decimal (g,y,p) :
    J = decompo_finale(p-1)
    D = []
    for i in range (len(J)) :
        L = deci(p,g,y,J[i][0],J[i][1])
        xi = 0
        for j in range (J[i][1]) :
            xi += L[j]*(J[i][0]**j)
        D.append(xi)
    return D


def reste_chinois(g,y,p) :
    P = decompo_finale(p-1)
    D = rassemble_decimal(g,y,p)
    s = 0
    e = []
    for i in range (len(D)) :
        ni = P[i][0]**P[i][1]
        mi = (p-1)//ni
        s += D[i]*mi*inverse_modulaire(mi,ni)
    return (s % (p-1))
```