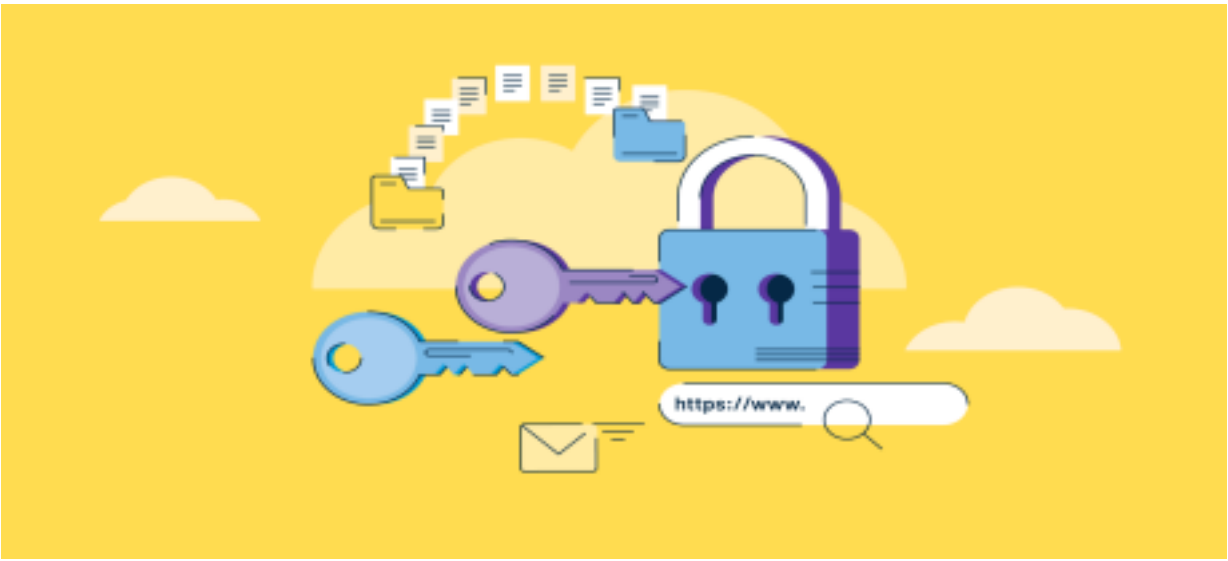
Cryptographie avancé

Compte Rendu

Courbes Elliptiques



Année Universitaire: 2023/2024

LEKOUAGHET imad eddine

Encryption et Decryption à l'aide des courbes elliptiques:

Les courbes elliptiques ont une large gamme d'applications en cryptographie moderne en raison de leurs propriétés mathématiques uniques, qui offrent des mécanismes de chiffrement et de déchiffrement sécurisés. Le code présenté ici illustre les opérations fondamentales sur les courbes elliptiques, ainsi que la mise en œuvre d'un système de chiffrement basé sur ces courbes.Les principales fonctions incluses dans ce code sont : classe point , fonction inv,Fonction IsPoint, fonction addPoint, fonction mul_scalaire ,fonction encrypt , fonction decrypt.

1-Fonction point:

Cette classe est utilisée pour représenter un point sur la courbe elliptique. Elle est initialisée avec deux coordonnées, x et y.

```
class Point:
    def __init__(self,x_init,y_init):
        self.x = x_init
        self.y = y_init
```

2- Fonction inv:

Cette fonction calcule l'inverse modulaire de n modulo p. Elle itère à travers les valeurs possibles et retourne la première valeur pour laquelle le produit de n et i est congruent à 1 modulo p.

```
def inv(n, p):
    for i in range(p):
        if (n * i) % p == 1:
            return i
```

3-Fonction AddPoints:

Vérifie si un point donné appartient à la courbe elliptique définie par les coefficients a et b.

```python
def isPoint(a1, a, b, p):
    y = a1.y * a1.y
    y %= p
    x = (a1.x * a1.x * a1.x + a * a1.x + b)
    x %= p
    if x == y:
        return True
    else:
        return False
```

4-Fonction AddPoints:

Cette fonction effectue l'addition de deux points sur la courbe elliptique. Elle prend comme paramètres deux points a1 et a3, ainsi que les coefficients de la courbe (a et p).

Selon les cas où les points sont égaux ou distincts, elle calcule la pente de la tangente ou de la sécante et trouve le troisième point d'intersection. Le calcul de l'addition est réalisé selon les règles suivantes :

Si les points sont égaux, elle calcule la tangente à la courbe au point et trouve le point d'intersection suivant.Si les points sont distincts, elle calcule la pente de la sécante passant par les deux points et trouve le troisième point d'intersection.

```python
def addPoints(a1, a3, a, p):
    if a1.x == a3.x and (a1.y != a3.y):
        return(a1)
    if a1.x == a3.x:
        l = ((3 * a1.x * a1.x  + a ) * inv(2 * a1.y, p)) % p
        xr= (l*l - a1.x - a3.x) % p
        yr= (l*(a1.x-xr)-a1.y)%p
        a3=Point(xr, yr)
        return(a3)
    else:
        l = ((a3.y - a1.y) * inv(a3.x - a1.x, p)) % p
        xr= (l*l - a1.x - a3.x) % p
        yr= (l*(a1.x-xr)-a1.y)%p
        a3=Point(xr, yr)
        return(a3)
```

5-Fonction Mul_scalaire:

Cette fonction effectue la multiplication scalaire d'un point sur la courbe elliptique par un scalaire. Elle prend comme paramètres le scalaire et un point générant, ainsi que les coefficients de la courbe (a et p). Elle utilise l'addition de points répétée pour effectuer la multiplication scalaire. En utilisant l'algorithme double and add, elle multiplie le point générant par le scalaire donné, en effectuant les additions nécessaires de manière efficace.

```
def mul_scalaire(na, genr, a, p):
    g2=genr
    for k in range(1, na):
        g2 = addPoints(g2, genr, a, p)
    return(g2)
```

6-Fonction encrypt:

Cette fonction chiffre un message en utilisant le chiffrement Elliptic Curve Integrated Encryption Scheme (ECIES). Elle prend comme paramètres le message à chiffrer (Pm), le nombre aléatoire k, le nombre secret na, le point de génération (genr), et les coefficients de la courbe (a et p). Pour chiffrer le message, elle génère un point aléatoire pa en multipliant le point de génération par le nombre secret na. Ensuite, elle multiplie le point pa par le nombre aléatoire k, puis ajoute le message Pm chiffré à ce résultat. Le résultat final est un couple de points qui représente le message chiffré.

```
def encrypt(Pm,k, na, genr, a, p):
    pa = mul_scalaire(na, genr, a, p)
    print(pa.x,pa.y)
    c2 = mul_scalaire(k, pa, a, p)
    c2 = addPoints(c2, Pm, a, p)
    return pa, c2
```

7-Fonction Decrypt:

Cette fonction déchiffre un message chiffré en utilisant le chiffrement ECIES. Elle prend comme paramètres le nombre secret na, le nombre aléatoire k, les points chiffrés a3 et a4, et les coefficients de la courbe (a et p). Pour déchiffrer le message, elle commence par multiplier le point chiffré a3 par le nombre secret na pour récupérer un point intermédiaire h1. Ensuite, elle multiplie ce point par le nombre aléatoire k et inverse l'ordonnée du résultat. Enfin, elle ajoute ce point au point chiffré a4 pour obtenir le message original.

```
def decrypt(na,k, a3, a4, a, p):
    print(a3.x,a3.y)
    h1 = mul_scalaire(k, a3, a, p)
    print(h1.x,h1.y)
    h1.y = -1 * h1.y
    h2 = addPoints(a4, h1, a, p)
    return h2
```

L'Exécution:

on supposant :

a = -5

b = 3

p = 31

x = 9

y = 6

```
esr ce que ca appartient a la courbe ?
True
Sum :
30 21
19 3
Encrypted message:
(30, 21)
19 3
9 6
Decrypted message:
12 11
```

Signature à l'aide des courbes elliptiques:

1-Fonction point_add :

Cette fonction prend deux points de la courbe elliptique en tant qu'entrées (p1 et p2). Elle vérifie d'abord si l'un des points est un point à l'infini (représenté par None). Ensuite, elle vérifie si les points sont égaux ; dans ce cas, elle calcule la tangente à la courbe au point, sinon elle calcule la pente de la sécante passant par les deux points. En utilisant cette pente, elle calcule ensuite les coordonnées du troisième point résultant de l'addition.

```
def point_add(p1, p2):
    if p1 is None:
        return p2
    if p2 is None:
        return p1
    x1, y1 = p1
    x2, y2 = p2
    if x1 == x2 and y1 != y2:
        return None
    if p1 == p2:
        m = (3 * x1 * x1 + a) * pow(2 * y1, p - 2, p) % p
    else:
        m = (y1 - y2) * pow(x1 - x2, p - 2, p) % p
    x3 = (m * m - x1 - x2) % p
    y3 = (m * (x1 - x3) - y1) % p
    return (x3, y3)
```

2-Fonction Generate_keys:

Cette fonction génère une paire de clés privée/publique pour la cryptographie à courbes elliptiques. Elle commence par choisir une clé privée aléatoire dans l'intervalle [1, n-1], où n est l'ordre du groupe de points sur la courbe. Ensuite, elle multiplie le point générateur G par cette clé privée pour obtenir la clé publique correspondante.

```python
def generate_keys():
    private_key = random.randint(1, n - 1)
    public_key = point_mul(G, private_key)
    return private_key, public_key
```

3-Fonction sign_message

Cette fonction signe un message en utilisant l'algorithme ECDSA. Elle commence par hacher le message pour obtenir une valeur z. Ensuite, elle choisit aléatoirement un nombre k dans l'intervalle [1, n-1] et calcule le point de signature r en multipliant le point générateur G par k. Elle calcule ensuite la composante s de la signature en utilisant des opérations modulo et d'inversion modulaire.

```python
def sign_message(private_key, message):
    z = int.from_bytes(hashlib.sha256(message.encode()).digest(), 'big')
    k = random.randint(1, n - 1)
    r_x = point_mul(G, k)
    if isinstance(r_x, tuple):
        r = r_x[0] % n  # Récupérer la coordonnée x de r
    else:
        r = 0
    s = pow(k, -1, n) * (z + r * private_key) % n
    return r, s
```

4-Fonction Verify_Signature

Cette fonction vérifie la validité d'une signature donnée pour un message donné en utilisant la clé publique correspondante. Elle commence par vérifier si les composantes r et s de la signature sont dans l'intervalle valide. Ensuite, elle récupère la valeur hachée z du message. Elle calcule ensuite plusieurs valeurs intermédiaires pour vérifier si le point de signature r est bien le résultat de l'addition de deux points sur la courbe.

```python
def verify_signature(public_key, message, signature):
    r, s = signature
    if r < 1 or r > n - 1 or s < 1 or s > n - 1:
        return False
    z = int.from_bytes(hashlib.sha256(message.encode()).digest(), 'big')
    w = pow(s, -1, n)
    u1 = z * w % n
    u2 = r * w % n
    x, y = point_add(point_mul(G, u1), point_mul(public_key, u2))
    if x is None or r != x % n:
        return False
    return True
```

L’Exécution:

```
Clé privée: 1053547783019019042072602048816688661585765622057020558991422588352910968505 80
Clé publique: (774112650088409720642545108928293961877193112142460631145862382759202524590 20, 864544032858584575884239280898231394842555215760329973916207984415039176943 83)
Signature: (107156972510281575948673592922309124721995524201425447266211882115277415739932, 398657514547497235301402884108051395327055228785863239629744500534612379704 62)
Signature validée avec succès.
```