RS40 : TP -PROJET RSA

Rapport TP RS40

TRABOULSI Rawan

Pour le : LUNDI 16 MAI 2022

Introduction :

Dans ce projet là on a fait une application sur le RSA (exponentation modulaire, inverse modulaire, chiffrement, déchiffrement, théorème de reste chinois crt , comment augmenter la taille du message et signature).

Pour expliquer bien cela on vous représente un partage de message entre deux personnes ALICE et BOB.

A. Les variables

Voici ce que nous avons vue dans le cours ;

Choix des clefs :

Bob

Si Bob désire que l'on puisse communiquer avec lui de façon secrète, il procède de la manière suivante :

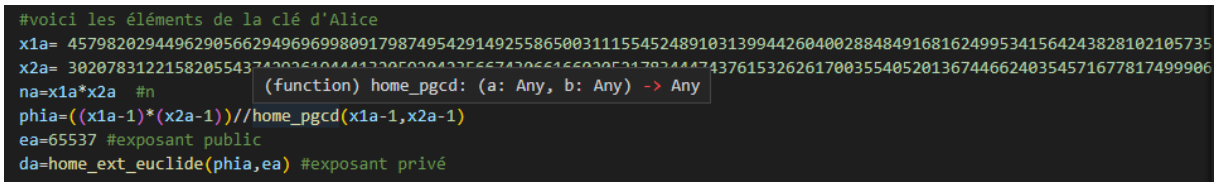
1. Bob engendre deux grands nombres premiers p et q (test de primalité).
2. Il calcule $n = p \times q$, donc $\phi(n) = (p-1)(q-1)$ où ϕ est l'indicateur d'Euler.
3. Il choisit un nombre aléatoire e avec $1 < e < \phi(n)$ tel que $\text{pgcd}(e, \phi(n)) = 1$.
4. Il calcule l'inverse de e modulo $\phi(n)$, noté d, c'est-à-dire : $d \times e \equiv 1 \mod \phi(n)$ (Algorithme d'Euclide généralisé).

Voici ci-dessous les différentes variables utilisées pour Bob dans le programme ;

```
def home_crt(x, d, n, p, q):
    inverse_de_q = home_ext_euclide(q,1)
    dq = home_mod_expnoent(d,1,q-1)
    dp = home_mod_expnoent(d,1,p-1)
    mq = home_mod_expnoent(x,dq,q)
    mp = home_mod_expnoent(x,dp,p)
    h = home_mod_expnoent(((mp-mq)*inverse_de_q),1,p)
    return home_mod_expnoent(mq+ h*q,1,n)
```

- x1b et x2b sont les deux grand nombre premiers générées par Bob. Pour notre cas leurs taille sont de 280.
- la variable nb correspond donc à n=p*q et phib est l'indicateur d'Euler
- eb correspond à l'exposant public il est égale à 65537
- db correspond à la clef secrète

Pour Alice nous avons les mêmes variables identiques.



B. Les fonctions

i. home_pgcd

Cette fonction prend en paramètre deux entier a et b et nous retourne leur plus grand diviseur commun.

```
def home_pgcd(a,b): #recherche du pgcd
    if(b==0):
        return a
    else:
        return home_pgcd(b,a%b)
```

ii. home_string_to_int et home_int_to_string

Cette fonction prend en paramètre une chaine de caractère et le converti en entier

```python
def home_string_to_int(x): # pour transformer un string en int
    z=0
    for i in reversed(range(len(x))):
        z=int(ord(x[i]))*pow(2,(8*i))+z
    return(z)
```

La deuxième fonction est une fonction réciproque elle convertie un nombre entier en chaine de caractère

```python
def home_int_to_string(x): # pour transformer un int en string
    txt=''
    res1=x
    while res1>0:
        res=res1%(pow(2,8))
        res1=(res1-res)//(pow(2,8))
        txt=txt+chr(res)
    return txt
```

iii. home_mod_expnoent(x,y,n)

La fonction home_mod_expnoent(x,y,n) doit permettre de calculer x^y%n. Cette fonction sert à chiffrer un nombre x avec une clé

Pseudo-code: Exponentiation modulaire rapide

Algorithme 1 Calcul de $y = x^p \mod (n)$

Entrées: $n \geq 2, x > 0, p \geq 2$

Sortie: $y = x^p \mod (n)$

Début

$p = (d_{k-1}; d_{k-2}; \cdots; d_1; d_0)$ % Écriture de p en base 2

$R_1 \leftarrow 1$

$R_2 \leftarrow x$

Traitement

Pour $t = 0; \cdots; k-1$ **Faire**

  Si $d_i == 1$ **Alors**

    $R_1 \leftarrow R_1 \times R_2 \mod (n)$ % Calcul de la colonne 4 du tableau si le bit est 1

  Fin Si

  $R_2 \leftarrow R_2^2 \mod (n)$ % carré modulo n de la colonne 3 du tableau

Fin Pour

publique (y, n) afin de compléter cette fonction, j'ai principalement utilisé l'algorithme de l'exponentiation modulaire rapide. Voici l'algorithme :

En effet, cet algorithme à été adapté en fonction du langage python, voici ce qu'on obtient :

```
def home_mod_expnoent(x,y,n):
    y=format(y, 'b')   #on transforme en binaire l'exposant
    k=(len(y)) #longueur du bit
    R1=1
    R2=x
    result=[]
    for i in range(0, len(y), 1):
       result.append(int(y[i : i + 1]))     # stockage des bits de manière séparée dans un tableau
    for i in range(k-1,-1,-1):     # attention ici on dit qu'on part de k-1 jusqu'a -1 car si on met 0, le 0 ne sera pas p
        if result[i]==1:           #si le bit est égale à  1
            R1=(R1*R2)%n           # R1 prend la valeur de R1 multiplié par R2 si le bit est 1 si non on ne touche pas à R
        R2=(R2*R2)%n               #R2 change  de valeur quelque soit la valeur du bit 0 ou 1
    return R1
```

iv. home_ext_euclide

Cette fonction sert à calculer l'inverse modulaire, c'est-à-dire l'inverse de y (mod b). On applique donc l'algorithme d'Euclide étendu avec le théorème de Bézout que nous avons étudié dans le cours.

```
def home_ext_euclide(y, b):  # algorithme d'euclide  pour la recherche de l'exposant secret
    (r,nouvr,t,nouvt)=(y,b,0,1)
    whil (variable) quotient: Any
        quotient=r//nouvr
        (r, nouvr) = (nouvr, r-quotient*nouvr)
        (t, nouvt) = (nouvt, t-quotient*nouvt)
    return nouvt%y
```

v. longueur Maximale

La fonction longueur Maximale prend en paramètre les deux grands nombres générés par Bob et indique le nombre de caractères maximale que l'utilisateur peut saisir. Le processus de calcul se fait de la manière suivante :

```
def longueurMaximale(x1,x2): #entrer le secret
    i=1
    while(2**i<=(x1*x2)):
        i=i+1
    nbCaracteresMaxi= int (i/8)
    secret=input("donner un secret de "+str(nbCaracteresMaxi) + " caractères au maximum : ")
    while(len(secret)>i):
        secret=input("c'est beaucoup trop long,"+str(nbCaracteresMaxi)+" caractères S.V.P : ")
    return secret
```

Ici on cherche simplement le nombre de bits qui correspond au message, pour cela on utilise les puissances de 2 avec une boucle qui a pour condition d'arrêt n=p*q (ici c'est x1*x2). Et puisque les caractères sont codés sur 8 bits, on doit diviser le nombre qu'on

obtient par 8 afin d'envoyer à l'utilisateur le nombre de caractères maximale qu'il peut saisir.

vi. home_crt

Le théorème du reste chinois permet de calculer $x^d (mod\ n)$. C'est donc une alternative à l'exponentiation modulaire. Le théorème du reste chinois est plus léger que l'exponentiation modulaire et possède l'avantage également de se protéger de l'attaque par canal auxiliaire

Pour implémenter le théorème du reste chinois j'utilise l'algorithme ci-dessous :

Algorithme de calcul $m = c^d \% n$ en utilisant CRT

Calcul préalable :

1- Avec $n = x_i x_j$ prendre $q = x_i$ et $p = x_j$ tel que $x_i < x_j$
2- Calculer q^{-1} dans $\mathbb{Z}_p$
3- Calculer d_q=d%(q-1) et d_p=d%(p-1)

Ces calculs sont réalisés **qu'une seule fois** et les valeurs de q^{-1}, d_q et d_p sont gardées secrètement.

A la réception d'un message c, effectuer les opérations suivantes :

1- Calculer $m_q = c^{d_q} \% q$ et $m_p = c^{d_p} \% p$
2- Calculer $h = \left((m_p - m_q) q^{-1}\right) \% p$
3- Calculer $m = \left(m_q + h \times q\right) \% n$

Voici la fonction python qui correspond à cet algorithme :

```
60  def home_crt(x, d, n, p, q):
61      inverse_de_q = home_ext_euclide(q,1)
62      dq = home_mod_expnoent(d,1,q-1)
63      dp = home_mod_expnoent(d,1,p-1)
64      mq = home_mod_expnoent(x,dq,q)
65      mp = home_mod_expnoent(x,dp,p)
66      h = home_mod_expnoent(((mp-mq)*inverse_de_q),1,p)
67      return home_mod_expnoent(mq+ h*q,1,n)
68
```

On remplace donc la fonction home_mod_expnoent par la fonction home_crt pour le déchiffrement du message, de la manière suivante :

```
print("Alice déchiffre le message chiffré envoyé par Bob  \n",chif,"\nce qui donne ")
dechif=home_int_to_string(home_mod_expnoent(chif, da, na))
print(dechif)
```

C. Hachage

La fonction de SHA256 est une fonction de hachage qui n'est plus considérée comme sécurisée. Il existe des fonctions de hachage qui sont meilleurs, comme SHA-256 ou encore SHA-512. Ici, j'ai décidé de remplacer md5 par SHA-256. Donc on l'utilise pour le chiffrement et le déchiffrement comme ci-dessous :

```
print("**************************************************************************")
print("On utilise la fonction de hashage SHA256 pour obtenir le hash du message",secret)
Bhachis0=hashlib.sha256(secret.encode(encoding='UTF-8',errors='strict')).digest() #SHA256 du message
print("voici le hash en nombre décimal ")
Bhachis1=binascii.b2a_uu(Bhachis0)
Bhachis2=Bhachis1.decode() #en string
Bhachis3=home_string_to_int(Bhachis2)
print(Bhachis3)
print("voici la signature avec la clé privée de Bob du hachis")
signe=home_mod_expnoent(Bhachis3, db, nb)
print(signe)
```

FIN