Université de technologie de
Belfort-Montbéliard

RS40 : TP1

Le RSA

Chausson
Thibault
Promo : 21

Professeur :
Abdeljalil
Abbas-Turki

15 mai 2022

Résumé

Le TP considère un message envoyé de Bob vers Alice, où chacun dispose d'une paire de clés (publique, privée) pour le chiffrement RSA. Bob chiffre le message avec la clé publique d'Alice. Il procède aussi à la signature de l'empreinte numérique du message avec sa clé privée. Alice reçoit le message le déchiffre et vérifie la signature de Bob (ce n'est pas sécurisé).

Table des matières

1 Préliminaires **1**

1.1 home_mod_expnoent 1

1.2 home_ext_euclide 2

2 Amélioration du md5 **3**

3 Amélioration du RSA avec le lemme chinois **4**

4 Exécution pas à pas **5**

Table des images **7**

Liste des algorithmes **7**

Liste des codes sources **7**

1 Préliminaires

1.1 home_mod_expnoent

Nous allons réaliser un premier programme basé sur la méthode suivante : soit e l'exposant et a notre valeur

- si e est pair, $a^e = (a^{\frac{e}{2}})^2$
- si e est impair, $a^e = (a^{\frac{e}{2}})^2 \times a$

```python
def home_mod_expnoent(x, y, n):  # exponentiation modulaire (on prend x puissance y)
    (r1,r2)=(1,x)
    while (y>0):
        if (y%2==1):
            r1=(r1*r2)%n
        r2=(r2**2)%n
        y=y//2
    return(r1)
```

Code source 1 – Exponentiation modulaire (parité)

Après vérification de la fonction « home_mod_expnoent », en la comparant à la fonction « pow() » du module « math » de Python, nous pouvons conclure qu'elle fonction correctement. Mais, en l'incluant dans la suite du code un problème apparait, donc j'ai codé une fonction en utilisant le binaire de l'exposant.

Voici l'algorithme donné par le professeur :

Algorithme 1 *Exponentiation rapide binaire* $y = x^p \mod n$

Donnée(s) : $n \geq 2$, $x > 0$, $p \geq 2$
Résultat(s) : $y = x^p \mod n$

1: $p \leftarrow (d_{k-1}, ..., d_0)$
2: $R_1 \leftarrow 1$
3: $R_2 \leftarrow x$
4: **pour** i **allant de** 0 **à** $(k-1)$ **faire**
5:   **si** $d_i = 1$ **alors**
6:     $R_1 \leftarrow R_1 \times R_2 \mod n$
7:   **fin si**
8:   $R_2 \leftarrow R_2^2 \mod n$
9: **fin pour**

```
def home_mod_expnoent(x, y, n):  # exponentiation modulaire (
   on prend x puissance y)
    p = decimal_to_bin(y) #la puissance en binaire
    R1 = 1
    R2 = x
    for i in range(len(p)):
        if p[len(p)-i-1] == "1":
            R1 = (R1*R2) % n
        R2 = (R2**2) % n
    return(R1)
```

Code source 2 – Exponentiation modulaire (binaire)

Mais le problème persiste, donc dans le suite du code j'utiliserai : « pow() ». De plus, ce problème est toujours présent avec la correction publiée par le professeur qui est :

```
def home_mod_expnoent(x, y, n):  # exponentiation modulaire
    (res1,res2)=(1,x)
    while y>0:
        if y%2==1:
            res1=(res1*res2)%n
        res2=(res2*res2)%n
        y=y//2
    return res1
```

Code source 3 – Exponentiation modulaire (parité correction du professeur)

1.2 home_ext_euclide

Pour implémenter la fonction « home_ext_euclide », j'utilise l'affectation multiple de Python qui est très pratique ici.

Soit a et b appartenant à $\mathbb{Z}$ En utilisant le cours de mathématiques on sait qu'il y a deux suites (u_k) et (v_k), tel que $\forall k \in \mathbb{N}$:

$$r_0 = r_1 q_1 + r_2$$

$$r_{k-2} = r_{k-1} q_{k-1} + r_k$$

Et il existe un $n \in \mathbb{N}$ tel que :

$$r_n = 0$$

Donc,

$$d = pgcd(a, b) = r_n \ et \ au_n + bv_n = d$$

```
def home_ext_euclide(a, b):  # algorithme d'euclide étendu
    pour la recherche de l'exposant on a r=au+bv
    (r,u,v,rp,up,vp)=(a,1,0,b,0,1)
    while rp!=0:
        q=r//rp
        (r, u, v, rp, up, vp) = (rp , up , vp , r-q*rp , u-q*
    up , v-q*vp)
    return (v)
```

Code source 4 – Euclide étendu

2 Amélioration du md5

Dans le but d'améliorer le hash réalisé par « md5 », nous utiliserons par la suite « sha256 ». Pour se faire, nous devons utiliser des nombres premiers plus grands, par exemple avec 60 chiffres (en utilisant bigprimes). De plus, nous devons aussi choisir une clé publique e avec $1 < e < \Phi(n)$ tel que $pgcd(e, \Phi(n)) = 1$.

Et voici le changement apporter à la ligne du hash :

```
Bhachis0 = hashlib.sha256(secret.encode(encoding='UTF-8',
    errors='strict')).digest()
```

Code source 5 – Passage au SHA256

Maintenant réfléchissons dans quelle mesure nous pouvons augmenter le nombre de caractères du message à hasher.

Nous pouvons utiliser toutes les tailles de message pour SHA256.

3 Amélioration du RSA avec le lemme chinois

Voici un extrait du cours :

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 $m = (m_q + h \times q)\%n$

IMAGE 1 – CRT

Nous utiliserons le lemme chinois quand nous devons décoder un message grâce à notre clef privée, ou signer le message, pour protéger la clef privée qui est sensibles. Pour résumer, nous utilisons le CRT des que nous manipulons une clé privée.

Commençons par déterminer les variables préliminaires :

```
def calculPreliminaire(xi, xj,d):
    n=xi*xj
    #phiq = (xi-1)*(xj-1)
    if (xi<xj):
        q=xi
        p=xj
    else:
        q=xj
        p=xi
    qPrime=home_ext_euclide(p,q)
    dq=d%(q-1)
    dp=d%(p-1)
    return(qPrime,dq,dp,q,p,n)
```

Code source 6 – Préliminaire du CRT

Utilisons le lemme chinois :

```python
def CRT(xi,xj, d, c): #c correspond au message
    (qPrime, dq, dp, q, p,n)=calculPreliminaire(xi,xj, d)
    mq=home_mod_expnoent(c,dq,q)
    mp=home_mod_expnoent(c,dp,p)
    h=((mp-mq)*qPrime)%p
    m=(mq+h*q)%n
    return m
```

Code source 7 – Le CRT

Par exemple, pour déchiffrer un message reçu par Alice de la part de Bob :

```python
print("Passons au déchiffrement par le CRT")
dechiffreCRT=CRT(x1a, x2a, da, chif)
print("Alice déchiffre son fameux message avec la clé de Bob,
    ce qui donne : ")
print(home_int_to_string(dechiffreCRT))
```

Code source 8 – Affichage du résultat

Où x_{1a}, x_{2a} sont les deux grands nombres premiers p et q d'Alice et d_a est la clef secrète d'Alice.

4 Exécution pas à pas

Premièrement nous avons un récapitulatif des information près définies :

```
Vous êtes Bob, vous souhaitez envoyer un secret à Alice
voici votre clé publique que tout le monde a le droit de consulter (de Bob)
n = 222620053029647070278426411556944546090455246701787467224116204407488132055461248737855893754586744313510063895027607849
exposant : 23
voici votre précieux secret
d = -161318879006990630636540877939814888471344381667961932771097992477937092142919279972932742008142223173889554263267431 33
*******************************************************************
Voici aussi la clé publique d'Alice que tout le monde peut consulter
n = 249375946457480010561536247157173851445104162012347634308971363317487195295552520916380298195191694428330826583377724123
exposent : 17
*******************************************************************
il est temps de lui envoyer votre secret
*******************************************************************
appuyer sur entrer
```

IMAGE 2 – Première étape

Entrons un message à chiffrer : « Bonjour, comment vas tu ? »

```
donner un secret de 32 caractères au maximum : Bonjour, comment vas tu ?
*******************************************************************
voici la version en nombre décimal de  Bonjour, comment vas tu ?  :
396253296868106187003715443474031959065209140975805423054658
voici le message chiffré avec la clé publique d'Alice :
68264276728338118016191215469651243984510660423125021803688248562431441523409895485121377382939461683885656388513362210
*******************************************************************
On utilise la fonction de hashage sha256 pour obtenir le hash du message Bonjour, comment vas tu ?
voici le hash en nombre décimal
23781438080638634351628413342840606090910543284805723527851712913151311088299931152750647006773179950074322496
voici la signature avec la clé privée de Bob du hachis
51807308386103931913343633025120031833273382254196341746617529083333038084182220728666776614639406164605995802983927965
voici la signature avec la clé privée de Bob du hachis avec le CRT
51807308386103931913343633025120031833273382254196341746617529083333038084182220728666776614639406164605995802983927965
*******************************************************************
Bob envoie
     1-le message chiffré avec la clé public d'Alice
 68264276728338118016191215469651243984510660423125021803688248562431441523409895485121377382939461683885656388513362210
     2-et le hash signé
 51807308386103931913343633025120031833273382254196341746617529083333038084182220728666776614639406164605995802983927965
*******************************************************************
appuyer sur entrer
```

Image 3 – Nous entrons le message, le chiffrons et le signons

Nous déchiffrons et vérifions la signature :

```
Alice déchiffre le message chiffré
 68264276728338118016191215469651243984510660423125021803688248562431441523409895485121377382939461683885656388513362210
ce qui donne
Bonjour, comment vas tu ?
Passons au déchiffrement par le CRT
Alice déchiffre son fameux message avec la clé de Bob, ce qui donne :
Bonjour, comment vas tu ?
*******************************************************************
Alice déchiffre la signature de Bob
 51807308386103931913343633025120031833273382254196341746617529083333038084182220728666776614639406164605995802983927965
 ce qui donne  en décimal
23781438080638634351628413342840606090910543284805723527851712913151311088299931152750647006773179950074322496
Alice déchiffre la signature CRT de Bob
 51807308386103931913343633025120031833273382254196341746617529083333038084182220728666776614639406164605995802983927965
 ce qui donne  en décimal
23781438080638634351628413342840606090910543284805723527851712913151311088299931152750647006773179950074322496
Alice vérifie si elle obtient la même chose avec le hash de  Bonjour, comment vas tu ?
23781438080638634351628413342840606090910543284805723527851712913151311088299931152750647006773179950074322496
La différence = 0
Alice : Bob m'a envoyé :  Bonjour, comment vas tu ?
La différence pour le CRT = 0
Alice : Bob m'a envoyé :  Bonjour, comment vas tu ?
```

Image 4 – Déchiffrer et vérifier la signature

Table des images

1 CRT 4
2 Première étape 5
3 Nous entrons le message, le chiffrons et le signons 6
4 Déchiffrer et vérifier la signature 6

Liste des algorithmes

1 *Exponentiation rapide binaire* $y = x^p \mod n$ 1

Liste des codes sources

1 Exponentiation modulaire (parité) 1
2 Exponentiation modulaire (binaire) 2
3 Exponentiation modulaire (parité correction du professeur) . . 2
4 Euclide étendu 3
5 Passage au SHA256 3
6 Préliminaire du CRT 4
7 Le CRT 5
8 Affichage du résultat 5