Université de technologie de Belfort Montbéliard

TP1 : RSA

Année 2022-2023

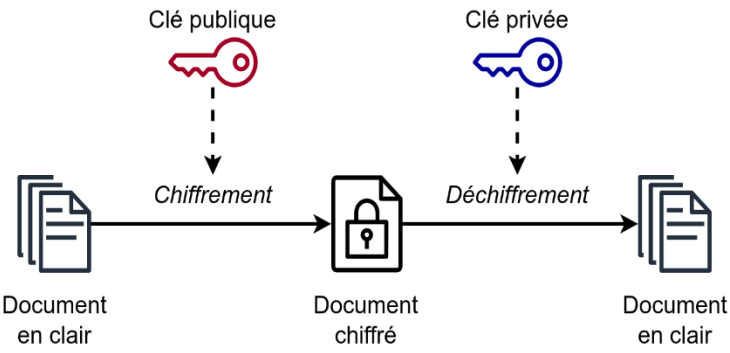
UV : RS40

Présenté par : Saad SBAT

Contents

Introduction

RSA est un système cryptographique, ou cryptosystème, pour le chiffrement à clé publique. Il est souvent utilisé pour la sécurisation des données confidentielles, en particulier lorsqu'elles sont transmises sur un réseau peu sûr comme Internet.



RSA a été décrit pour la première fois en 1977 par Ron Rivest, Adi Shamir et Leonard Adleman du MIT (Massachusetts Institute of Technology). Le chiffrement à clé publique, également appelé chiffrement asymétrique, utilise deux clés différentes, mais mathématiquement liées, une publique et l'autre privée. La clé publique peut être partagée avec quiconque, tandis que la clé privée doit rester secrète. Dans le chiffrement RSA, tant la clé publique que la clé privée peuvent servir à chiffrer un message. Dans ce cas, c'est la clé opposée à celle ayant servi au chiffrement qui est utilisée pour le déchiffrement. C'est notamment grâce à cette caractéristique que RSA est devenu l'algorithme asymétrique le plus répandu : il offre en effet une méthode permettant d'assurer la confidentialité, l'intégrité, l'authenticité et la non-répudiabilité des communications électroniques et du stockage de données.

De nombreux protocoles, tels que SSH, OpenPGP, S/MIME et SSL/TLS reposent sur RSA pour leurs fonctions de chiffrement et de signature numérique. Cet algorithme est également utilisé dans des logiciels : les navigateurs en sont un exemple flagrant, car établir une connexion sécurisée sur un réseau peu sûr comme Internet ou valider une signature numérique font partie de leurs attributions. La vérification de signature RSA constitue l'une des opérations les plus couramment réalisées en informatique.

Sujet

Le TP vise à illustrer le déploiement d'un ensemble de mécanismes cryptographiques pour sécuriser l'échange entre deux parties : Alice et Bob. 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.

RSA

Les mathématiques derrière RSA

Pour comprendre comment l'algorithme RSA fonctionne et le réaliser d'une manière optimisée on a besoin de définir des fonctions mathématiques telles que :

- home_mod_expnoent(x,y,n): La fonction qui permet de réaliser l'exponentiation modulaire x^y%n.

- home_ext_euclide(y,b) : La fonction permettant d'obtenir la clé secrète(inverse modulaire). Pour ce faire, il faut utiliser l'algorithme d'Euclide étendu.
- Crt(x1,x2,d,c) : La fonction qui permet de réaliser l'exponentiation modulaire c^d%(x1*x2) pour éviter 2 problèmes, Le premier est le temps de calculs qui est très long, le deuxième est dû à la possibilité d'écoute permettant de déduire la taille de la clé et le nombre de 1.

Home_mode_exponent(x,y,n)

Pour chiffrer et déchiffrer les messages, on a besoin de calculer $x^y[n]$ à chaque fois, pour ce faire j'ai créé la fonction home_mode_exponent(x,y,n) qui donne un résultat rapide de l'exponentiation modulaire car elle réduit le nombre des calcules en bénéficiant de la structure binaire du l'exposant

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 $i = 0; \cdots; k-1$ **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

Pseudo-code : Exponentiation modulaire rapide dans le cours

```
1.  def home_mod_exponent(x,y,n): #exponentiation modulaire
2.      exposant_binaire=decimalToBinary(y)#je transforme le puissance en binaire
3.      r1=1
4.      r2=x
5.      while (len(exposant_binaire)>=1):
6.          if(exposant_binaire[-1]=='1'):#si le bit=1
7.              r1=(r1*r2)%n #update r1
8.          r2=(r2*r2)%n   # update r2
9.          exposant_binaire=exposant_binaire[0:-1] #je supprime le bit déja traité pour
    lire le bit suivant
10.     return r1 #je retourne r1
```

Code en Python

home_ext_euclide(y,b)

Pour le choix des clés, on choisit une clé publique e et on calcule son inverse afin d'obtenir la clé privée.

Définition de l'inverse modulaire :

L'inverse **modulaire** d'un entier relatif pour la multiplication modulo est un entier satisfaisant l'équation *:*

$$ac \equiv 1[n]$$

Pour calculer l'inverse ***c*** de ***a*** modulo ***p***, il faut calculer le coefficient de Bézout ***u*** relatif à ***a*** (car ***a*** > ***p***) et faire ***c*** = ***u*** **mod** ***p***

```
1.  def home_ext_euclide(p,a):#inverse modulaire
2.
3.      r1, r2 = a, p
4.      u1, u2 = 1, 0 #  uk
5.      v1, v2 = 0, 1 #  vk
6.
7.      while r2 != 0:#quand le  dernier reste est 0 on arrête et on récupère le
    reste avant le dernier
8.          q = r1 // r2                # quotient
9.          r1, r2 = r2, r1 - q * r2    # update r1 et r2
10.         u1, u2 = u2, u1 - q * u2    # update uk
11.         v1, v2 = v2, v1 - q * v2    # update vk
12.
13.     if r1 == 1:                     # dernier reste avant que r=0
14.         return u1 % p
15.     else:
16.         return None
```

SHA256

La fonction de hachage SHA-256 est plus sécurisée que MD5. SHA-256 produit un hachage de 256 bits, qui est deux fois plus long que MD5. De plus, la probabilité de collision avec SHA-256 est plus faible qu'avec MD5.Afin d'améliorer le hash « md5 » on va utiliser le hash SHA-256 qui a comme sortie une chaine de 64 caractères, et pour garder la condition que le message doit être inférieur à n (M<n) et 1<e< φ(n) on va choisir eb=65573 et p et q deux nombres premiers de plus de 63 digits.

```
1.  Bhachis0=hashlib.sha256(secret.encode(encoding='UTF-8',errors='strict')).digest()
    #SH256 du message
```

Théorème du reste chinois

Mauvaise exponentiation modulaire de RSA

En calculant l'exponentiation modulaire $x^y[n]$ avec l'algorithme rapide, à chaque fois où un bit=1, il y'a une opération supplémentaire réalisée, Kevin peut deviner la clé privée en mesurant les opérations réalisées.

Pour éviter cette attaque, on va utiliser le théorème du reste chinois, qui va réduire le temps de calculs, et la possibilité d'écoute ne sera plus un problème.

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

Algorithme de calcul de l'expo modulaire avec CRT

```
1. def CRT(x1,x2,d,c):
2.     p,q=None,None
3.     if x1<x2:
4.         p,q=x1,x2
5.     else:
6.         p,q=x2,x1
7.     inv_q=home_ext_euclide(q,p)
8.     dq=d%(q-1)
9.     dp=d%(p-1)
10.
11.    mq=home_mod_expnoent(c,dq,q)
12.    mp=home_mod_expnoent(c,dp,p)
13.    h=((mp-mq)*inv_q)%p
14.    m=(mq+h*q)%(x1*x2)
15.    return m
```

Code CRT

Découpage du message + bourrage

Pour éteindre la taille du message il est possible de procéder au découpage du message par blocs et au bourrage.

Utilité du bourrage :

Imaginons que le message à envoyer par Bob est toujours un entier appartenant à l'ensemble E={1,2,3},Kevin (Middle man) peut récupérer le message chiffré 'C'

et il essaye de chiffré 1 puis 2 puis 3 par la clé public d'Alice et compare le résultat avec 'C' ce qui lui permettre de déduire le contenu du message.

Avec le bourrage ça sera impossible de réaliser cette attaque car Bob va procéder à la manière suivante :

- Définir une taille limite de chaque bloc que nous notons k
- Bob : chaque bloc ne doit pas contenir plus de 50% du message que nous notons mi et dont la taille en octets est j. Ainsi, $m=m0‖m1‖m2‖…‖mi‖…‖mn$ et $j≤k2/$.
- Bob : Générer $k−j−3$ octets non nuls. Bob utilise alea%255+1 pour chaque octet. Le nombre issu de cette génération est noté x.
- Bob : Constituer le bloc de la forme suivante : $00‖02‖x‖00‖mi‖$, avec 00, 02 sont des octets valant 0 et 2 en hexadécimal.
- Bob : Chiffrer le bloc avec RSA.
- Alice : Déchiffrer le bloc avec RSA
- Alice : Eliminer $00‖02‖x‖00$ de chaque bloc pour obtenir mi
- Alice : Concaténer l'ensemble de mi pour constituer le message initial.

Code en python

```python
def prefix(k,mi):
    i=len(mi)#i c'est la taille du message
    x=""
    for j in range(k-i-3):#generation de k-j-3 nombre aleatoire non nul
       x+=str(randint(255,263)%255+1)
    return str(0x00)+str(0x02)+x+str(0x00)+mi #00‖02‖x‖00‖m

def blocks(message,k=10):#k repésente la taille du bloc
    nbchar=(k//2)-1 #taille du message<(taille du bloc)/2
    taille=len(message)
    d=taille//nbchar #nombre des blocks de messages complets
    r=taille-d*nbchar #nb des octets qui construient pas un blocs de message complets
    liste=[]
    c=0
    j=0
    block=""
    while c<len(message):#découpage du message
        if(j<nbchar):
            block+=message[c]
            j+=1
            c+=1
        if j==nbchar :
            liste.append(block)
            block=""
            j=0
    if r!=0:
        liste.append(block)
    for l in range(len(liste)):
        liste[l]=prefix(k,liste[l])#concatenation de 00‖02‖x‖00 avec Mi
    return liste


#Elimination de 00‖02‖x‖00 de chaque bloc pour obtenir mi
def concat(liste):
    string=""
    for i in range(len(liste)):
        bloc=liste[i]
        j=1
        while bloc[j]!='0':
            j+=1
        string+=bloc[j+1:]

    return string
```

Jeu d'essai pour K=10,M='aaaaaaaaaa'

Blocks(M,K) -> ['022340aaaa', '027360aaaa', '0248480aaa']

```
PKCS.py > ...
45
46  liste_blocks=blocks("aaaaaaaaaa",10)
47  print("Liste_blocks:",liste_blocks,end="\n")
48  print("Elimination de 00‖02‖x‖00:",concat(liste_blocks))
49

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\saad_\OneDrive - Université De Technologie De Belfort-
 - Université De Technologie De Belfort-Montbeliard/Archive/UTB
Liste_blocks: ['022740aaaa', '025740aaaa', '0215840aaa']
Elimination de 00‖02‖x‖00: aaaaaaaaaa
```

Quand Alice va recevoir les messages, elle les déchiffre puis elle élimine les $00\|02\|x\|00$ de chaque bloc pour obtenir *mi* par la fonction *concat(liste)*

```
1.  #Elimination de 00‖02‖x‖00 de chaque bloc pour obtenir mi
2.  def concat(liste):
3.      string=""
4.      for i in range(len(liste)):
5.          bloc=liste[i]
6.          j=1
7.          while bloc[j]!='0':
8.              j+=1
9.          string+=bloc[j+1:]
10.
11.     return string
```

Exécution pas à pas

Premièrement nous avons un récapitulatif des paramètres déjà définies :

```
Vous êtes Bob, vous souhaitez envoyer un secret à Alice
voici votre clé publique que tout le monde a le droit de consulter
n = 838579839361665105294667123929533991567126101556268218458485073200016188784748264839164566670829119970611306062460507039
exposant : 65573
voici votre précieux secret
db = 39088028289218571581796624590770906397830218678522163233622893754539229853748498063901665548940773907307340713751576031 7
da = 98794450136332874420723206312955163541437207438334070786626456918694875126018610715611906759455912832602307782652098929
**********************************************************************
Voici aussi la clé publique d'Alice que tout le monde peut conslter
n = 20993820653970735814403681341502972252555406580645990042158218757645376449108174696722485117386106062817294991037077961 1
exposent : 17
**********************************************************************
il est temps de lui envoyer votre secret
**********************************************************************
appuyer sur entrer
```

Ensuite j'insère le message « *THE RS40 IS THE BEST UV IN UTBMMMM*» pour le chiffrer et le signer puis l'envoyer à Alice :

```
C:\Users\saad_\OneDrive - Université De Technologie De Belfort-Montbeliard\Archive\UTBM\RS40\TP1>C:/Users/saad_/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/saad_/OneDrive - Université De
chnologie De Belfort-Montbeliard/Archive/UTBM/RS40/TP1/RSA_B_A_SH256_CORR.py"
Vous êtes Bob, vous souhaitez envoyer un secret à Alice
voici votre clé publique que tout le monde a le droit de consulter
n = 838579839361665105294667123929533991567126101556268218458485073200016188784748264839164566670829119970611306062460507039
exposant : 65573
voici votre précieux secret
db = 39088028289218571581796624590770906397830218678522163233622893754539229853748498063901665548940773907307340713751576031 7
da = 98794450136332874420723206312955163541437207438334070786626456918694875126018610715611906759455912832602307782652098929
**********
Voici aussi la clé publique d'Alice que tout le monde peut conslter
n = 20993820653970735814403681341502972252555406580645990042158218757645376449108174696722485117386106062817294991037077961 1
exposent : 17
**********
il est temps de lui envoyer votre secret
**********
appuyer sur entrer
donner un secret:THE RS40 IS THE BEST UV IN UTBMMMM
**********
Liste des blocks avant le chiffrage:

['024460THE ', '023830RS40', '028580 IS ', '022290THE ', '022430BEST', '027920 UV ', '022560IN U', '027730TBMM', '02228980MM']
**********
**********
voici la listes des blocs de messages chiffrés avec la publique d'Alice :
[113244729320528646570527559130848687278427591241561814521293403059508762744994428751607856648213345821985132407302262003, 159972507575321374711041699264074090015785726554971055502656682238985390181229
979115770688304669938815211027585768 8, 163320415382438245457639430754846982915442662183510600713527567009102854131606818555778951576767980009318492874222082256, 159517279291637291209787084271686653912491
684147177452474193496841427394664360755287732283794967404632297015704637459, 78988205939340709705946129024138775923434503633223057759215088108193181354511847029091587558723885531834991394067122758, 1948
419834494630241918822950218048296805331090081351068226080783350076509330903401901486361758361106953371544475777 7, 626928025925708234561697485571450465064418589524470801941426364140938457567025302570969
944381592455406484989646076, 193631648524092863377467380247910940446941864452484006349654268870156986633434463613231645250826839983489718222354832 8, 923374693081461153109118034995064218428943316130195
047730158873053476185566540957412452899606160489832409020563059]
**********
**********
On utilise la fonction de hashage sha256 pour obtenir le hash du message THE RS40 IS THE BEST UV IN UTBMMMM
THE RS40 IS THE BEST UV IN UTBMMMM
voici le hash en nombre décimal
23781153628619965648106155408753843028295419696249007999851888693964966587872400937735147120034493168295095104
voici la signature avec la clé privée de Bob du hachis
sigature:
 41393542586201751154980284420996624204181722957944284756923890374857519879762791024835536800812674247684375898103128 8
**********
Bob envoie
        1-la liste des blocks chiffrés avec la clé public d'Alice
 [113244729320528646570527559130848687278427591241561814521293403059508762744994428751607856648213345821985132407302262003, 159972507575321374711041699264074090015785726554971055502656682238985390181229
197911577068830466993881521102758576 8, 163320415382438245457639430754846982915442662183510600713527567009102854131606818555778951576767980009318492874222082256, 159517279291637291209787084271686653912491
968414717745247419349684142739466436075528773228379496740463229701570463745 9, 789882059393407097059461290241387759234345036332230577592150881081931813545118470290915875587238855318349913940671227 8, 1948
341983449463024191882295021804829680533109008135106822608078335007650933090340190148636175836110695337154447577 7, 626928025925708234561697485571450465064418589524470801941426364140938457567025302570969
794438159245540648498964607 6, 193631648524092863377467380247910940446941864452484006349654268870156986633434463613231645250826839983489718222354832 8, 923374693081461153109118034995064218428943316130195
504773015887305347618556654095741245289960616048983240902056305 9]

**********
appuyer sur entrer
```

Après Alice déchiffre les blocs puis élimine les 00‖02‖x‖00 et concatènes les mi, ensuite elle vérifie la signature :

Fin