Université de Caen Normandie

Master 1 Informatique

Graphes, recherche arborescente et complexité

Contrôle Continu I

(TP 4 : Problèmes difficiles et sécurité)

Rapport réalisé par :

Nom : MESSILI

Prénom : Islem

Numéro Etu : 22303045

Année : 2023/2024

L'algorithme RSA (Rivest, Shamir et Adelman) :

Question 1 - Un premier exemple

Génération de clés

On a **p=19** et **q=11** fixés.

1/Calcul de ϕ(n) :

n = p * q = **209**

ϕ(n) = (p−1) * (q−1) = (19−1) * (11−1) = **180**

2/Calcul une clé publique {c,n} :

Pour calculer la clé publique en commence par calculer l'exposant de chiffrement « c » tel que « c » est 1<c<ϕ(n) et il est premier avec ϕ(n) :

On peut prendre c = **3** et dans ce cas notre clé publique vas être {c,n} = **{3,209}**

3/Calcul de la clé privée correspondante {d,n} :

Pour calculer la clé privée en commence par calculer l'exposant de déchiffrement « d » qui est l'inverse de c modulo ϕ(n).

d = inverse_mod(c, ϕ(n)) = **18** et dans ce cas notre clé privée vas être {d,n} = **{18,209}**

Chiffrement et déchiffrement

Le message doit être chiffre en utilisant la clé publique avant d'être envoyé vers le récepteur et il doit être déchiffrer avec la clé prive avant d'être lu.

4/Coder le message M = 200 :

Le message chiffré envoyer est :

$C = M^c \pmod n$ ⬅➡ C = power_mod(M, c, n)

Dans notre cas C =

5/Décoder le message M = 200

Le message reçu est chiffré donc on le déchiffre de la manière suivante :

$D = C^d \pmod n$ ⟷ D = power_mod(C, d, n)

Dans notre cas D = M = 200

Question 2 - Générer une paire de clés d'une taille donnée

1/Écriture d'une fonction qui prend en entrée une taille donnée t et qui retourne un nombre premier de t bits choisi au hasard :

```
45  def nb_premier(t):
46      # Générer un nombre premier aléatoire de t bits
47      a = pow(2,t)
48      b = pow(2,t+1) - 1
49
50      numBP = next_prime(randrange(a,b))
51      while a > numPB > b:
52          numBP = next_prime(randrange(a,b))
53
54      return numB
55
```

2/Écriture d'une fonction paireCles qui prend en entrée une taille donnée t et qui engendre une paire de clés publique et privée avec le module n de t bits :

```
56
57  def paireCles(t):
58      # Générer deux nombres premiers de t/2 bits chacun
59      p = next_prime(2 ** (t // 2))
60      q = next_prime(2 ** (t // 2))
61
62      n = p * q
63      fn = (p - 1) * (q - 1)
64
65      c = 0
66      while gcd(fn,c) != 1 :
67          c = randrange(2,fn)
68
69      d = inverse_mod(c, fn)
70
71
72      cle_publique = (c, n)
73      cle_privee = (d, n)
74
75      return cle_publique, cle_privee
```

Sûreté de RSA ?

Si l'on sait factoriser n alors on sait casser RSA !

Question 3 - Reconstruction de la clé privée {d,n} à partir de la clé publique {c,n}, si l'on arrive à factoriser n

Soit {c,n}={32224920548891884277319616150846179881012498534099,5859752073467330615054108447928357270066490591132 3} la clé publique

1/Factorisation n :

La factorisation d'un nombre consiste à représenter ce nombre sous la forme d'un produit, dans notre cas « n » doit être factoriser en deux nombre « q » et « p » qui sont des nombres premiers

n = p * q

p = 7131431184351431593024423

q = 8216796771909460827720301

n = factor(n)[0] * factor(n)[1]

2/Calcul ϕ(n) :

ϕ(n) = (p−1) * (q−1)

ϕ(n) = 585975207346733061505410691310556164397724851666 00

3/Déduire la clé privée :

{d,n} = (28611098868280672528246029063936969199085701482899, 585975207346733061505410844792835727006649059113 23)

Le message codé
est 81297846809292584521768839267680509123077608826918129784680929258452176883926768050912307760882691

4/Retrouver le message initial :

Pour retrouver le message initial il faut passé par les étapes suivantes :

- Factoriser n
- Calculer $\phi(n)$
- Calculer l'exposant de déchiffrement d
- Trouver le message initial

En passant le message code et sa clé publique a la fonction suivante en obtiendra le message initial :

```python
def casserRSA(cle_pb,message_code):
    # Factoriser n
    n = cle_pb[1]
    p = factor(n)[0]
    q = factor(n)[1]

    # Calculer fn
    fn = (p[0] - 1) * (q[0] - 1)

    # Calculer d
    c = cle_pb[0]
    d = inverse_mod(c, fn)

    # Clé privée
    cle_pv = (d, n)

    # Déchiffrement
    message_initial = power_mod(message_code, d, n)

    return message_initial

```

```
message_code:  81297846809292584521768839267680509123077608826 91

message_initial:  12345678910111213141516171819202122232425262728 29
```

5/Généralisation :

Écrire une fonction casserRSA qui prend en entrée une clé publique quelconque et qui retourne la clé privée correspondante :

```python
def casserRSA(cle_pb,message_code):
    # Factoriser n
    n = cle_pb[1]
    p = factor(n)[0]
    q = factor(n)[1]

    # Calculer fn
    fn = (p[0] - 1) * (q[0] - 1)

    # Calculer d
    c = cle_pb[0]
    d = inverse_mod(c, fn)

    # Clé privée
    cle_pv = (d, n)

    # Déchiffrement
    message_initial = power_mod(message_code, d, n)

    return message_initial
```

Question 4 - Taille du module et sécurité

1/En utilisant la fonction paireCles, déterminer expérimentalement le coût de la fonction casserRSA en fonction de la taille de n :

2/Lorsque la taille du module (t) augmente, la fonction casserRSA devient exponentiellement plus coûteuse en termes de temps d'exécution donc elle devient plus en plus inefficace.

La taille du module n est à conseiller pour un minimum de sécurité pour RSA :

La taille du module n est important pour la sécurité de RSA. Il est recommandé la taille 2048 bits ou plus.

L’exécution de fichier algoRsa.py donne :