Compte rendu RSA

Réalisé par : ZELLOU Wiam

Définition :

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

Le 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.

Algorithme :

Alice veut envoyer M à Bob.

- M un entier représentant un message.

  Génération des clés:

- Bob choisit p et q deux nombres entiers premiers et on note n leur produit (n = p.q)
- Bob choisit e un entier premier avec: f(n)=(p-1)(q-1)
- On a f(n) = (p-1)(q-1) donc e est premier avec f(n) et on obtient (via Bezout) qu'il est inversible modulo f(n), i.e. il existe un entier d tel que e.d = 1 (mod f(n)).
- Bob calcule la cl é d de déchiffrage qui doit satisfaire l'équation: d.e = 1 (mod(f(n)).

(n, e) est appelé clef publique

(n, d) est appelé clef privée.

Implémentation :

- `isprem(n)` retourne True si l'entier n est premier, False dans le cas contraire.

```python
def isprem(n):

    """retourne True si n est premier, False dans le cas contraire.
    n doit etre un entier"""
    if n == 1 or n == 2:
        return True

    if n%2 == 0:

        return False

    r = n**0.5

    if r == int(r):

        return False

    for x in range(3, int(r), 2):

        if n % x == 0:

            return False

    return True
```

- `coupcoup(k, long)` découpe des blocs de longueur long dans une chaine de caractères k et retourne une liste des blocs

• Quand le message est représenté par M < n le chiffrement se fait en une seule étape.
• Pour chiffrer de plus long messages il suffit de le diviser en blocs M=M1 M2 ... Mt de sorte que Mi < n pour tout 1≤i≤t

```python
def coupcoup(k, long):

    """découpe des blocs de longueur long dans une chaine de caractères k
    d , f = 0 , long

    l = []

    while f <= len(k):

        l.append(k[d:f])

        d , f = f , f + long

    m = len(k)%long

    if m != 0:

        l.append(k[len(k)-m:])

    return l
```

- `pgcd(a,b)` retourne le plus grand dénominateur commun de a et b

```
def pgcd(a,b):

    """retourne le plus grand dénominateur commun de a et b"""

    while (b>0):

        r=a%b

        a,b=b,r

    return a
```

- `pgcde(a, b)` pgcd étendu avec les 2 coefficients de Bézout u et v
  Entrée : a, b entiers
  Sorties : r = pgcd(a,b) et u, v entiers tels que a*u + b*v = r

```
def pgcde(a, b):

    """ pgcd étendu avec les 2 coefficients de bézout u et v
    Entrée : a, b entiers
    Sorties : r = pgcd(a,b) et u, v entiers tels que a*u + b*v = r
    """
    r, u, v = a, 1, 0
    rp, up, vp = b, 0, 1

    while rp != 0:
        q = r//rp
        rs, us, vs = r, u, v
        r, u, v = rp, up, vp
        rp, up, vp = (rs - q*rp), (us - q*up), (vs - q*vp)

    return (r, u, v)
```

- `key()` retourne un dictionnaire contenant la clé privée et la clé publique sous forme de tuples: {priv:(clé privée),pub:(clé publique)}

```
def key():

    """retourne un dictionnaire contenant la clé privée et la clé publique sous forme de tuples: {priv:(clé pri

    #choix au hasard de deux entiers premiers (n et q)
    p = np.random.choice(1000,1)
    q = np.random.choice(1000,1)

    while isprem(p) is False:
        p = np.random.choice(1000,1)

    while isprem(q) is False:
        q = np.random.choice(1000,1)
    #calcul de n et m
    n = p*q
    m = (p-1)*(q-1)

    #recherche de c premier de m (c'est a dire tel que pgcd(m,c)=1 ) et de d = pgcde(m,c) tel que 2 < d < m
    r = 10
    d = 0
    while r != 1 or d <= 2 or d >= m:
        c = np.random.choice(1000,1)
        r, d, v = pgcde(c,m)

    n, c, d = int(n), int(c), int(d)
    return {"priv":(n,c), "pub":(n,d)}
```

- `chiffre(n, c, msg)` : (n,c) est la clé publique et msg est le message à crypter.

```
def chiffre(n, c, msg):

    #conversion du message en codes ascii
    asc = [str(ord(j)) for j in msg]

    #ajout de 0 pour avoir une longueur fixe (3) de chaque code asccii
    for i, k in enumerate(asc):
        if len(k) < 3:

            while len(k) < 3:

                k = '0' + k

            asc[i] = k

    #formation de blocs de taille inferieure a n (ici blocs de 4)
    ascg = ''.join(asc)

    d , f = 0 , 4

    while len(ascg)%f != 0: #on rajoute eventuellement des 0 a la fin de ascg de maniere a ce que len(ascg) soit un multiple de f

        ascg = ascg + '0'

    l = []

    while f <= len(ascg):

        l.append(ascg[d:f])
        d , f = f , f + 4
    #chiffrement des groupes
    crypt = [str(((int(i))**c)%n) for i in l]

    return crypt
```

- `dechiffre(n, d, *crypt)` : (n,d) est la clé privée, et *crypt est une liste des blocks à déchiffrer

```python
def dechiffre(n, d, *crypt):

    """*crypt est une liste des blocks à déchiffrer"""

    #dechiffrage des blocs
    resultat = [str((int(i)**d)%n) for i in crypt]

    #on rajoute les 0 en debut de blocs pour refaire des blocs de 4
    for i, s in enumerate(resultat):

        if len(s) < 4:
            while len(s) < 4:
                s = '0' + s
            resultat[i] = s

    #on refait des groupes de 3 et on les convertie directement en ascii
    g = ''.join(resultat)
    asci = ''
    d , f = 0 , 3

    while f < len(g):
        asci = asci + chr(int(g[d:f])) #conversion ascii
        d , f = f , f + 3

    return asci
```

Teste :

```python
message = "Message : bonjour je m'appelle wiam "
key= key()
```

```python
print(key)
```

```
{'priv': (103331, 629), 'pub': (103331, 18269)}
```

```python
k_priv=key["priv"]
print(k_priv)
k_pub=key["pub"]
print(k_pub)
```

```
(103331, 629)
(103331, 18269)
```

```python
sortie=chiffre(k_pub[0],k_pub[1],message)
print(sortie)
```

```
['38855', '9158', '8463', '10859', '78534', '91310', '96728', '57832', '58731', '52100', '50083', '39203', '54466', '56279', '96371', '
42865', '6087', '28504', '69137', '63520', '13520', '61673', '88179', '39015', '46691', '68132', '22890']
```

```python
dechiffre(k_priv[0],k_priv[1],*sortie)
```

```
"Message : bonjour je m'appelle wiam"
```

Lien GITHUB : https://github.com/wiamzellou/RSA.py/blob/main/RSA.ipynb