Illustration RSA

March 28, 2023

[1]:
```
from gmpy2 import gcdext # PGCD avec relation de Bézout assocciée :
# gcdext(a, b) returns a 3-element tuple (g, s, t) such that
# g == gcd(a, b) and g == a * s + b * t

from numpy import random as r
# Generates a random sample from a given 1-D array :
```

[2]:
```
# Exemple :
gcdext(12, 35)
# 1 = 3×12 + (-1)×35
```

[2]:
```
(mpz(1), mpz(3), mpz(-1))
```

1 Bob détermine ses clés publique et privée :

[3]:
```
# 1) Détermination des entiers premiers p et q, puis de n et de phi(n) :
p = 2447
q = 7069
n = p * q
phi_n = (p-1)*(q-1)

# 2) Détermination de la clé publique e, qui doit être pratique (historiquement␣
 ↪5, souvent 65537=2¹⁶+1) :
e = 65537

# 3) Détermination de la clé privée d avec l'algorithme d'Euclide étendu :
(g, d, k) = gcdext(e, phi_n)
```

[4]:
```
print(f"p: {p}, q: {q}, n:{n}, phi(n):{phi_n}, e:{e}, d:{d}, g:{g}")
```

```
p: 2447, q: 7069, n:17297843, phi(n):17288328, e:65537, d:-7245919, g:1
```

2 Bob comunique alors sa clé publique à Alice :

[5]:
```
print(f"Clé publique de Bob : (e={e}, n={n})")
```

```
Clé publique de Bob : (e=65537, n=17297843)
```

3 Alice veut envoyer le message “m” à Bob (m < n) :

3.1 Génération du message :

[6]:
```
m = int(r.choice(n, 1)[0])
print(f"Message : m={m}")
```

```
Message : m=860342
```

3.2 Alice chiffre son message en faisant `c = m^e[n]` :

[7]:
```
c = pow(int(m), e, n)
print(f"Chiffre : c={c}")
```

```
Chiffre : c=13236793
```

Rmq :`pow(m, e, n)` sera beaucoup plus efficace que `(m**e) % n` parce qu'il va réduire modulo `n` à chaque multiplication plutôt que de laisser un énorme nombre grossir avant de réduire le résultat

3.3 Bob déchiffre le message chiffré “c” en faisant `res = c^d[n] = m^(ed)[n] = m[n]` :

[8]:
```
res = pow(c, d, n)
```

3.4 On vérifie :

[9]:
```
print(res == m)
```

```
True
```