Cryptographie : le système RSA

Aurélien Blais, Nicolas Iung

February 5, 2019

1 Principe du chiffrement RSA

1.1 Question 1

1.1.1 Montre que $cd = 1 + k(p-1)(q-1)$

En utilisant le théorème de Bezout $au + bv = 1$
On pose

$$\begin{aligned} a &= c \\ u &= d \\ b &= \varphi(n) \\ v &= -k \end{aligned}$$

$$\begin{aligned} cd + -k\varphi(n) &= 1 \\ cd &= 1 + k\varphi(n) \\ \text{Or } \varphi(n) &= (p-1)(q-1) \\ cd &= 1 + k(p-1)(q-1) \end{aligned}$$

1.2 Question 2

1.2.1 Déduire que $cd \equiv 1 \pmod{\varphi(n)}$

On sait que

$$\begin{aligned} & a \equiv b \pmod{c} \\ \iff & a = b + kc \end{aligned}$$

On peut donc en déduire que

$$\begin{aligned} cd &= 1 + k(p-1)(q-1) \\ cd &= 1 + k\varphi(n) \\ cd &\equiv 1 \pmod{\varphi(n)} \end{aligned}$$

1.2.2 Conclure que si c et $\varphi(n)$ premiers entre eux, il existe toujours un entier d inverse de c modulo $\varphi(n)$

On a $cd \equiv 1 \pmod{\varphi(n)}$
Or on dit que a est l'inverse de $b \equiv \pmod{n}$
Si et seulement si $ab \equiv 1 \pmod{n}$
On a donc d inverse de $c \pmod{\varphi(n)}$

1.3 Question 3

1.3.1 Montrer que M est premier avec p et avec q

2 Premier exemple

2.1 Question 1

2.1.1 Calculer $n_1 = p_1 q_1$ et $\varphi(n_1) = (p_1 - 1)(q_1 - 1)$ avec $p_1 = 7307$ et $q_1 = 5923$

$$\begin{aligned}
n_1 &= p_1 q_1 \\
n_1 &= 7307 * 5923 \\
n_1 &= 43279361
\end{aligned}$$

$$\begin{aligned}
\varphi(n_1) &= (p_1 - 1)(q_1 - 1) \\
\varphi(n_1) &= (7307 - 1)(5923 - 1) \\
\varphi(n_1) &= 43266132
\end{aligned}$$

2.2 Question 2

2.2.1 Choisir un entier c_1 premier avec $\varphi(n_1)$ tel que $c_1 < \varphi(n_1)$

On sait que 2 nombres sont premiers entre eux si leur $PGCD$ est égal à 1
Pour déterminer rapidement ce chiffre, on utilise un algorithme simple, ici en Ruby.

```
(2..43266131).each do |i|
        if p.gcd(i) == 1
                puts i
                break
        end
end
```

L'algorithme itère simplement sur les entiers entre 2 et $\varphi(n_1)$ et retourne le plus petit entier pour lequel $PGCD(\varphi(n_1), i) = 1$ soit $c_1 = 5$.

3 Fonctions de base

3.1 Question 1

3.1.1 Implémenter *exponentiationModulaire(x, k, n)*

L'algorithme implémenté suit le pseudo-code suivant :
`https://en.wikipedia.org/wiki/Modular_exponentiation#Right-to-left_binary_method`
Ayant une complexité en $O(log(k))$

```
def self.exponentiation_modulaire(x, k, n)
  result = 1
  base   = x
  while k > 0
    if (k & 1) == 1
      result = (result * base) % n
    end
    k    = k >> 1
    base = (base * base) % n
  end
  result
end
```

3.2 Question 2

3.2.1 Implémenter *euclideEtendu(a, b)*

L'algorithme implémenté suit le pseudo-code suivant :
`https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide_%C3%A9tendu#L'algorithme`

```
def self.euclide_etendu(a, b)
  r, u, v, r2, u2, v2, q = a, 1, 0, b, 0, 1, 0
  while(r2 > 0) do
    q = r/r2
    r, u, v, r2, u2, v2 = r2, u2, v2, r-q*r2, u-q*u2, v-q*v2
  end
  {pgcd: r, u: u, v: v}
end
```

La fonction retourne un $Hash$, c'est à dire ensemble clé ⇒ valeur
Tel que $\{pgcd : valeur, u : valeur, v : valeur\}$

3.3 Question 3

3.3.1 Implémenter *inverseModulaire(a, N)*

La méthode retourne l'inverse modulaire de (a, N) et se base sur la définition fournie ici :
`https://fr.wikipedia.org/wiki/Inverse_modulaire#Algorithme_d'Euclide_%C3%A9tendu`
Si a et N ne sont pas premiers entre eux, la méthode lève une exception.

```
def self.inverse_modulaire(a, n)
  val = euclide_etendu a, n
  raise Exception.new("Can't find value for a: #{a} and n: #{n}") unless val[
  val[:u] % n
end
```

3.4 Question 4

3.4.1 Implémenter *generationExposants(p, q)*

On déclare $\varphi = (p-1)*(q-1)$ comme vu précédemment dans l'énoncé.
On déclare ensuite $c = 2$, afin de ne pas obtenir $c = 1$, qui est premier avec l'ensemble des entiers.
Pour obtenir c, on l'incrémente tant que $PGCD(c, \varphi)$ n'est pas égal à 1.
Enfin, on retourne un $Hash$ contenant la valeur de c et de d tel que $d = inverseModulaire(c, \varphi)$

```
def self.generation_exposants(p, q)
  phi = (p - 1) * (q - 1)
  c = 2
  while c < phi
    break if c.gcd(phi) == 1
    c += 1
  end
  {c: c, d: inverse_modulaire(c, phi)}
end
```