Cryptographie à clé publique, partie théorie

1 Divisibilité et nombres premiers

Définition 1. *(Division euclidienne).* Soient deux entiers naturels non nuls a et b, il existe deux entiers naturels q et r tels que $a = b \cdot q + r$ et $r < b$. On définit la *division euclidienne* de a par b comme l'opération qui à (a, b) associe (q, r).

q est appelé le *quotient* et r le *reste* de la division euclidienne de a par b. Lorsque $r = 0$, on dit que b *divise* a, ou que b est un *diviseur* de a.

Notations : Soient des entiers naturels non nuls a, a', b, q, q' et un entier naturel positif r tels que $a = b \cdot q + r$ et $a' = bq' + r$.

- L'opération *modulo*, notée $\mod$, associe à deux entiers naturels le reste de leur division euclidienne : $a \mod b = a' \mod b = r$.
- Deux entiers a et a' sont dits *congrus modulo* b si le reste de la division euclidienne de a par b est égal à celui de la division de a' par b, on notera : $a \equiv a' \mod b$.

Définition 2. *(Nombres premiers).* Un entier positif p est dit *premier* lorsque ses seuls diviseurs sont 1 et p.

Définition 3. *(Plus grand commun diviseur).* Soit deux entiers non nuls a et b. Le *plus grand commun diviseur* (PGCD) de a et b, noté $\mathsf{pgcd}(a, b)$ est le plus grand entier divisant a et b.

En posant $a = qb + r$, on remarque que $\mathsf{pgcd}(a, b) = \mathsf{pgcd}(b, r)$. En effet, si un diviseur d est commun à a et b, alors d divise aussi $r = a - bq$, et si un diviseur d est commun à b et r, alors d divise aussi $a = bq + r$. Ainsi, les diviseurs communs de a et b sont les mêmes que ceux de b et r. On peut utiliser cette propriété pour calculer facilement le PGCD à l'aide de l'algorithme d'Euclide :

```
fonction euclide(a, b)
    tant que (b != 0)
        tmp := b
        b := a mod b
        a := tmp
    retourner a
```

Exemples : Calcul du PGCD de 60 et 42 :
$60 = 42 \times 1 + 18$
$42 = 18 \times 2 + 6$
$18 = 6 \times 3 + 0$
$\mathsf{pgcd}(60, 42) = 6$.

Calcul du PGCD de 47 et 7 :
$47 = 7 \times 6 + 5$
$7 = 5 \times 1 + 2$
$5 = 2 \times 2 + 1$
$2 = 1 \times 2 + 0$
$\mathsf{pgcd}(47, 7) = 1$.

Définition 4. *(Nombres premiers entre eux).* Deux entiers non nuls a et b sont dits *premiers entre eux* lorsque $\mathsf{pgcd}(a, b) = 1$.

Théorème 1. *(de Bachet-Bézout).* Soit deux entiers a et b , il existe un couple $(u, v) \in \mathbb{Z}$ tel que $au + bv = \mathsf{pgcd}(a, b)$. En particulier, si a et b sont premiers entre eux, alors il existe deux entiers u et v, appelés coefficients de Bézout, tels que $au + bv = 1$.

Le calcul des coefficients de Bézout (u et v tel que $au + bv = 1$) peut être réalisé à l'aide d'une généralisation de l'algorithme d'Euclide, qu'on appelle l'algorithme d'Euclide étendu (ce qui démontre le théorème) :

```
fonction euclide-étendu(a, b)
    si b = 0 alors
          retourner (a, 1, 0)
    sinon
          (g, u, v) := euclide-étendu(b, a mod b)
          retourner (g, v, u - (a/b)v)               //(division d'entiers)
```

Exemple : Calcul des coefficients de Bézout de 47 et 7 :
$47 = 7 \times 6 + 5 \rightarrow 5 = 47 - 7 \times 6$
$7 = 5 \times 1 + 2 \rightarrow 2 = 7 - 5 \times 1$
$5 = 2 \times 2 + 1 \rightarrow 1 = 5 - 2 \times 2$
$2 = 1 \times 2 + 0$

$1 = 1 \times 5 - 2 \times 2$
$1 = 5 - 2 \times (7 - 5 \times 1) = 5 - 7 \times 2 + 5 \times 2 = 3 \times 5 - 2 \times 7$
$1 = 3 \times (47 - 7 \times 6) - 2 \times 7 = 3 \times 47 - 3 \times 6 \times 7 - 2 \times 7 = 3 \times 47 - 20 \times 7$

$u = 3; v = -20$.

Avec l'algorithme :

```
a = 47; b = 7
    a = 7; b = 5
        a = 5; b = 2
            a = 2; b = 1
                a = 1; b =0
            g = 1, u = 1, v = 0
        g = 1, u = 0, v = 1
    g = 1, u = 1, v = 0-(5/2) = -2
g = 1, u = -2, v = 1 - (7/5) (-2) = 3

g = 1, u = 3, v = -2 - (47/5) 3 = -20
```

Théorème 2. *(de Bézout).* Deux entiers positifs a et b sont premiers entre eux si et seulement si il existe deux entiers u et v tels que : $au + bv = 1$.

Démonstration : On a vu que si a et b sont premiers entre eux, alors il existe u et v tels que $au+bv = 1$. Supposons que qu'il existe u et v tels que $au + bv = 1$, alors tout diviseur commun de a et b divise 1, donc a et b sont premiers entre eux. □

Lemme 1. *(de Gauss).* Soient a, b et c trois entiers. Si a divise bc et est premier avec b, alors a divise c.

Démonstration : Il existe u et v tels que a $au + bv = \mathsf{pgcd}(a, b) = 1$, donc $auc + bvc = c$. On sait que a divise ac et a divise bc, donc a divise $uac + vbc$, donc a divise c. □

Lemme 2. *(d'Euclide).* Soient p, b et c trois entiers. Si p est premier et divise bc, alors p divise b ou c. Plus généralement, si p divise un produit de nombres entiers, alors p divise un de ces nombres.

Démonstration : Soit p divise b, soit il est premier avec b et il divise c (d'arpès Gauss). □

Théorème 3. *(fondamental de l'arithmétique).* Tout entier strictement positif a peut être écrit comme un produit de nombres premiers d'une façon unique, à l'ordre près des facteurs. Autrement dit, en posant p_i le i-ème nombre premier, il existe un ensemble d'entiers positifs $\{k_i\}_{1 \leq i \leq n}$ tel que :

$$a = \prod_{i=1}^{n} p_i^{k_i}.$$

Démonstration : Existence par construction. Si a n'est pas premier, il existe p et q tels que $1 < p < n$ et $1 < q < n$ et $n = pq$. On peut appliquer récursivement cette propriété sur p et q. L'algorithme se termine car à chaque étape, les facteurs sont strictement inférieurs à leur produit, et sont minorés par 1.

Unicité par l'absurde.

— **cas 1** Supposons qu'on ait deux décompositions en facteurs premiers distinctes :

$$a = \prod_{i=1}^{n} p_i^{k_i} = \prod_{i=1}^{m} q_i^{l_i},$$

telles qu'il existe un indice j tel que pour tout indice i, $p_j \neq q_i$. On remarque que p_j divise $a = \prod_{i=1}^{m} q_i^{l_i}$. D'après le lemme d'Euclide, p_j divise donc un des q_j, ce qui est absurde.

— **cas 2** Supposons qu'on ait deux décompositions avec les mêmes facteurs premiers mais des exposants différents :

$$a = \prod_{i=1}^{n} p_i^{k_i} = \prod_{i=1}^{n} p_i^{l_i},$$

telles qu'il existe un indice j vérifiant $k_j > l_j$. Comme $p_j^{k_j}/p_j^{l_j} = p_j^{k_j - l_j}$, on a :

$$p_j^{k_j - l_j} \prod_{i=1; i \neq j}^{n} p_i^{k_i} = \prod_{i=1; i \neq j}^{n} p_i^{l_i}.$$

D'après le lemme d'Euclide, p_j divise donc un des p_i, ce qui est absurde. On traite de façon symétrique le cas où $k_j < l_j$.

On conclut que la décomposition est unique. □

Théorème 4. Il existe une infinité de nombres premiers.

Démonstration : Par l'absurde. Supposons que l'ensemble des nombres premiers soit un ensemble fini $S = \{p_i\}_{1 \leq i \leq n}$. On construit :

$$p = \left(\prod_{i=1}^{n} p_i \right) + 1.$$

On remarque que pour tout i :

— $p > p_i$ donc $p \notin S$, et

— $p \equiv 1 \mod p_i$, donc p n'est pas divisible par p_i, et donc p n'a pas de facteur premier ; il est donc premier (théorème fondamental d'arithmétique).

Ces deux propriétés sont contradictoires. □

Définition 5. *(Indicatrice d'Euler).* L'indicatrice d'Euler est la fonction $\phi : \mathbb{N}^+ \to \mathbb{N}$ qui à tout entier naturel N non nul associe le nombre d'entiers compris entre 1 et N (inclus) et premiers avec N.

Théorème 5. Soit a un entier positif dont la décomposition en produit de facteurs premiers est :

$$a = \prod_{i=1}^{n} p_i^{k_i},$$

alors :

$$\phi(a) = \prod_{i=1}^{n} (p_i - 1) p_i^{k_i - 1}.$$

En particulier, pour n'importe quelle paire de nombres premiers distincts (p, q), on aura $\phi(pq) = (p - 1)(q - 1)$.

Démonstration : Admis dans le cas général. Cas particulier : $\phi(pq)$ est le nombre d'entiers compris entre 1 et pq qui sont premiers avec pq. On pose $a = pq$. On liste tous les nombres qui ne sont pas premiers avec a. On a :

- les nombres n vérifiant $\mathsf{pgcd}(a, n) = p$, c'est à dire les entiers $n = pb$ où b est un entier compris entre 1 et $q - 1$;
- les nombres n vérifiant $\mathsf{pgcd}(a, n) = q$, c'est à dire les entiers $n = bq$ où b est un entier compris entre 1 et $p - 1$;
- le nombres n vérifiant $\mathsf{pgcd}(a, n) = pq$, c'est à dire a lui-même ;

On dénombre donc $(p - 1) + (q - 1) + 1 = p + q - 1$ nombres qui ne sont pas premiers avec a. On en déduit :

$$\phi(a) = a - (p + q - 1) = pq - p - q + 1 = (p - 1)q - p + 1 = (p - 1)q - (p - 1) = (p - 1)(q - 1).$$

□

2 Groupes finis

Définition 6. Un *groupe* est un couple $(\mathbb{G}, *)$ où $\mathbb{G}$ est un ensemble et $*$ est une opération de $\mathbb{G}^2 \to \mathbb{G}$ qui combine deux éléments a et b de $\mathbb{G}$ en un élément de $\mathbb{G}$ noté $a * b$, vérifiant les propriétés suivantes :

- **Associativité** Pour tout a, b et c dans $\mathbb{G}$, $a * (b * c) = (a * b) * c$
- **Élément neutre** Il existe un unique *élément neutre* $e \in \mathbb{G}$ tel que $\forall\ a \in \mathbb{G}, e * a = a * e = a$
- **Inversibilité** Pour tout $a \in \mathbb{G}$ il existe un unique *élément inverse* $a^{-1} \in \mathbb{G}$ tel que $a^{-1} * a = a * a^{-1} = e$

Un groupe est dit *commutatif* si $\forall\ a$ et b dans $\mathbb{G}$, on a $a * b = b * a$. Dans ce cours, on ne considérera que des groupes commutatifs. Par convention :

- On appellera groupe *additif* un groupe dont l'opération est $+$. Dans ce cas, l'élément neutre est noté 0 et l'inverse d'un élément a est noté $-a$.
- On appellera groupe *multiplicatif* un groupe dont l'opération est $\cdot$ ou $\times$. Dans ce cas, l'élément neutre est noté 1 et l'inverse d'un élément a est noté a^{-1} ou $\frac{1}{a}$.

Notations et vocabulaire :

- Soit $(\mathbb{G}, \cdot)$ un groupe, tout groupe $(H, \cdot)$ vérifiant $H \subseteq \mathbb{G}$ est appelé *sous-groupe* de $\mathbb{G}$.
- $|\mathbb{G}|$ correspond au nombre d'éléments dans un groupe, appelé *l'ordre* du groupe $\mathbb{G}$ ou le *cardinal* de l'ensemble $\mathbb{G}$.
- Un groupe *fini* (resp. *infini*) est un groupe dont l'ordre est fini (resp. infini)
- Par convention, $\mathbb{Z}_n$ désigne l'ensemble des entiers entre 0 et $n - 1$ (c'est à dire les entiers modulo n) et $\mathbb{Z}_n^*$ désigne l'ensemble des entiers entre 1 et $n - 1$.

Exemples de groupes :

- $(\mathbb{Z}, +)$
- $(\mathbb{Z}_n, +)$ où $+$ désigne l'addition modulo n
- Attention, $(\mathbb{Z}_n^*, \cdot)$ ou $\cdot$ désigne la multiplication modulo n n'est pas forcément un groupe (exemple : $(\mathbb{Z}_4^*, \cdot)$).

$(\mathbb{Z}_n, +)$:

Associativité. Pour tout x, y, et z dans $\mathbb{Z}_n$, $x + (y + z) = x + y + z = (x + y) + z$.

Élément neutre. Pour tout $x \in \mathbb{Z}_n$, on a $x + 0 = 0 + x = x$.

Inversibilité. Pour tout $x \in \mathbb{Z}_n$, on a $n - x \in \mathbb{Z}_n$ et $x + (n - x) = (n - x) + x = n \equiv 0 \mod n$.

$(\mathbb{Z}_4^*, \cdot)$: l'élément neutre pour la multiplication est 1. Or, 2 n'a pas d'inverse dans $\mathbb{Z}_4^* = \{1, 2, 3\}$, car :

- $2 \cdot 1 \equiv 2 \mod 4$ et $2 \neq 1$;
- $2 \cdot 2 \equiv 0 \mod 4$ et $0 \neq 1$;
- $2 \cdot 3 \equiv 2 \mod 4$ et $2 \neq 1$;

Définition 7. *(Exponentiation et logarithme discret).* Soit $(\mathbb{G}, \cdot)$ un groupe fini. On notera g^x *l'exponentiation discrète de base* g *et d'exposant* x où $g \in \mathbb{G}$ et $x \in \mathbb{N}$ de la façon suivante :

$$g^x = \underbrace{g \cdot g \cdot g \cdots g}_{x \text{ fois}}.$$

Par convention, on pose $g^0 = 1$ et $g^{-x} = (g^x)^{-1}$. L'inverse de cette opération est *le logarithme discret* et est noté $\log_g$:

$$\log_g(g^x) = x.$$

Définition 8. *(Générateur et groupe cyclique).* Soit $(\mathbb{G}, \cdot)$ un groupe fini. Un *générateur* de $\mathbb{G}$ est un élément $g \in \mathbb{G}$ vérifiant

$$\mathbb{G} = \{g^x | x \in \mathbb{Z}\}.$$

Un groupe qui admet un générateur est appelé un *groupe cyclique.*

Exemples de groupes cycliques : $(\mathbb{Z}_n, +)$ ou $+$ désigne l'addition modulo n est un groupe cyclique généré par 1.
On a $\mathbb{Z}_n = \{0, 1, 2, \cdots, n-1\}$ et :
$1 = 1$
$1 + 1 = 2$
$1 + 1 + 1 = 3$
$\vdots$
$1 + 1 + \cdots (n - 1 \text{ fois}) \cdots + 1 = n - 1$
$1 + 1 + \cdots (n \text{ fois}) \cdots + 1 = n \equiv 0 \mod n$
$1 + 1 + \cdots (n + 1 \text{ fois}) \cdots + 1 = n + 1 \equiv 1 \mod n$
$\vdots$

Définition 9. *(Ordre d'un élément).* Soit $(\mathbb{G}, \cdot)$ un groupe fini et $a \in \mathbb{G}$ un élément. Le plus petit entier positif non nul i vérifiant $a^i = 1$ est appelé l'*ordre de* a.

Théorème 6. Soit $(\mathbb{G}, \cdot)$ un groupe fini et $a \in \mathbb{G}$ un élément d'ordre n. $\{a^x | x \in \mathbb{Z}\}$ est un sous-groupe de $\mathbb{G}$ d'ordre n.

Démonstration : $\cdot$ est une lois interne pour $\langle a \rangle = \{a^x | x \in \mathbb{Z}\}$ puisque pour tout x et y dans $\mathbb{Z}$, on a $x + y \in \mathbb{Z}$ et $a^x \cdot a^y = a^{x+y} \in \langle a \rangle$. L'associativité découle de l'associativité de $\mathbb{G}$, et $a^0 = 1 \in \langle a \rangle$ est l'élément neutre. Pour tout $a^x \in \langle a \rangle$, on a $a^{-x} \in \langle a \rangle$ et $a^x \cdot a^{-x} = a^{x-x} = a^0 = 1$, donc a^{-x} est l'inverse de a^x. On a montré que $\langle a \rangle$ est un groupe.

Pour tout $x \in \mathbb{Z}$, on note respectivement q et r le quotient et le reste de la division de x par n. On a $x = q \cdot n + r$, donc $a^x = a^{q \cdot n + r} = (a^n)^q \cdot a^r = 1^q \cdot a^r = a^r$, on en déduit que $\langle a \rangle = \{a^x | x \in \mathbb{Z}_n\}$, et donc que l'ordre de $\langle a \rangle$ est au plus n.

Soient x et y dans $\mathbb{Z}_n$ tels que $x > y$, supposons que $a^x = a^y$. Dans ce cas, $a^x \cdot (a^y)^{-1} = a^y \cdot (a^y)^{-1} = 1$, et donc $a^x \cdot (a^y)^{-1} = a^{x-y} = 1$, ce qui implique que l'ordre de a est au plus $x - y$. Or $x - y < n$, ce qui est contradictoire puisque a est d'ordre n. On en déduit que l'ordre de $\langle a \rangle$ est au moins n, et donc $|\langle a \rangle| = n$. □

Définition 10. *(Sous-groupe engendré par un élément).* Soit $(\mathbb{G}, \cdot)$ un groupe fini et $a \in \mathbb{G}$ un élément. Le groupe $\{a^x | x \in \mathbb{Z}\}$ est appelé le *sous-groupe de* $\mathbb{G}$ *engendré par* a et est noté $\langle a \rangle$.

Lemme 3. Soit $(\mathbb{G}, \cdot)$ un groupe fini et H un sous-groupe de $\mathbb{G}$. Pour tout $x \in \mathbb{G}$, on a $|H| = |xH|$ (où $xH = \{x \cdot h | h \in H\}$).

Démonstration : On pose la fonction surjective $f : H \to xH$ définie par $f(h) = x \cdot h$. Soit $(h_1, h_2) \in H^2$, On a :

$$f(h_1) = f(h_2) \Leftrightarrow x \cdot h_1 = x \cdot h_2 \Leftrightarrow x^{-1} \cdot x \cdot h_1 = x^{-1} \cdot x \cdot h_2 \Leftrightarrow h_1 = h_2.$$

La fonction est donc injective, et donc bijective. On en déduit que $|H| = |xH|$. □

Lemme 4. Soit $(\mathbb{G}, \cdot)$ un groupe fini et H un sous-groupe de $\mathbb{G}$. Pour tout $(x, y) \in \mathbb{G}^2$, soit $xH = yH$, soit $xH \cap yH = \emptyset$.

Démonstration : Supposons $xH \cap yH \neq \emptyset$, on va montrer que $xH = yH$. Soit $a \in xH \cap yH$, il existe $h_1 \in H$ et $h_2 \in H$ tels que $a = x \cdot h_1 = y \cdot h_2$. On en déduit que $x = y \cdot h_2 \cdot h_1^{-1}$. Pour tout $b \in xH$, il existe h tel que $b = x \cdot h$, or $b = (y \cdot h_2 \cdot h_1^{-1}) \cdot h = y \cdot (h_2 \cdot h_1^{-1} \cdot h) \in yH$, donc $b \in yH$. De même, pour tout $b \in yH$, on montre de la même façon que $b \in xH$. On en déduit que $xH = yH$. □

Théorème 7. *(de Lagrange).* Soit G un groupe et H un sous-groupe de G. L'ordre de H divise l'ordre de G.

Démonstration : On a :

$$\bigcup_{x \in \mathbb{G}} xH = \mathbb{G}.$$

On pose l'ensemble $S \subseteq \mathbb{G}$ tels que pour tout x et y dans S, on a $xH \neq yH$ et :

$$\bigcup_{x \in S} xH = \mathbb{G}.$$

D'après le lemme 3, $xH \neq yH$ implique $xH \cap yH = \emptyset$, donc les xH pour $x \in S$ forment une partition de $\mathbb{G}$, et d'après le lemme 4, pour chaque $x \in S$, on a $|xH| = |H|$. On en déduit que :

$$|\mathbb{G}| = |S| \times |H|.$$

Donc $|H|$ divise $|\mathbb{G}|$. □

Remarque 1. Comme tout élément d'ordre n engendre un sous groupe d'ordre n (Théorème 6), le théorème de Lagrange implique que dans un groupe, l'ordre d'un élément divise l'ordre du groupe.

Lemme 5. Si p est premier, alors $(\mathbb{Z}_p^*, \cdot)$ est un groupe.

Démonstration : On montre les propriétés suivantes :

Associativité. Pour tout x, y, et z dans $\mathbb{Z}_p^*$, $x \cdot (y \cdot z) = x \cdot y \cdot z = (x \cdot y) \cdot z$.

Élément neutre. Pour tout $x \in \mathbb{Z}_p^*$, $1 \cdot x = x \cdot 1 = x$.

Inversibilité. Pour tout $x \in \mathbb{Z}_p^*$, x est premier avec p car p est premier, donc il existe u et v tels que $x \cdot u + p \cdot v = 1$. On en déduit $x \cdot u = 1 - p \cdot v \equiv 1 \mod p$. Donc x admet u pour inverse.

□

Théorème 8. *(de Fermat).* Si p est premier, alors pour tout entier a non divisible par p, on a $a^{p-1} \equiv 1 \mod p$.

$(\mathbb{Z}_p^*, \cdot)$ est un groupe d'ordre $(p-1)$. On note H le sous-groupe de $\mathbb{Z}_p^*$ engendré par a. On note h l'ordre de H. On a $a^h \equiv 1 \mod p$. D'après Lagrange, h divise $(p-1)$, donc il existe un entier k tel que $(p-1) = kh$. On déduit :

$$a^{p-1} \equiv a^{kh} \equiv (a^h)^k \equiv 1^k \equiv 1 \mod p.$$

□

Théorème 9. Tout élément non neutre d'un groupe d'ordre premier est générateur de ce groupe.

Démonstration : Soit $\mathbb{G}$ un groupe d'ordre premier $|\mathbb{G}| = p$. Soit un élément h non neutre du groupe, on note H le sous-groupe engendré par h. L'ordre de H divise l'ordre du groupe, donc l'ordre de H est 1 ou p. Comme h n'est pas l'élément neutre, il est d'ordre supérieur à 1, donc l'ordre de H est $|H| = p$. On a $H \subseteq \mathbb{G}$ et $|H| = |\mathbb{G}|$ donc $H = \mathbb{G}$ et h est générateur de $\mathbb{G}$. □

Théorème 10. *(d'Euler).* Soit N un entier positif. Pour tout entier a premier avec N, on a $a^{\phi(N)} \equiv 1 \mod N$.
En particulier, si N est le produit de deux nombres premiers distincts, pour tout entier a premier avec N, on a $a^{\phi(N)} \equiv 1 \mod N$.

Démonstration : Admis dans le cas général. Cas $N = pq$ avec p et q premiers :
a n'est pas divisible par p. On a, d'après Fermat :

$$a^{\phi(N)} \equiv a^{(p-1)(q-1)} \equiv (a^{p-1})^{q-1} \equiv 1^{q-1} \equiv 1 \mod p \Rightarrow a^{\phi(N)} - 1 \equiv 0 \mod p.$$

On en déduit que p divise $a^{\phi(N)} - 1$. De même, a n'est pas divisible par p, donc $a^{\phi(N)} - 1 \equiv 0 \mod q$ et q divise $a^{\phi(N)} - 1$. Finalement, $N = pq$ divise $a^{\phi(N)} - 1$, on en déduit :

$$a^{\phi(N)} - 1 \equiv 0 \mod N \Rightarrow a^{\phi(N)} \equiv 1 \mod N.$$

□

Théorème 11. Soit N un produit de deux nombres premiers distincts, pour tout entier a non divisible par N et tout entier k, on a $a^{k\phi(N)+1} \equiv a \mod N$.

Démonstration : On pose $N = pq$. Si a est premier avec N, conséquence d'Euler. Il reste deux cas :

— **Si q divise a,** alors p ne divise pas a. Dans ce cas, $a \equiv 0 \mod q$, donc $a^{k\phi(N)+1} \equiv 0 \mod q$ et $a^{k\phi(N)+1} - a \equiv 0 \mod q$. D'autre part, d'après Fermat :

$$a^{k\phi(N)+1} \equiv aa^{k(p-1)(q-1)} \equiv a(a^{p-1})^{k(q-1)} \equiv a \cdot 1^{k(q-1)} \equiv a \mod p \Rightarrow a^{k\phi(N)+1} - a \equiv 0 \mod p.$$

On en déduit que q et p divisent $a^{k\phi(N)+1} - a$, donc $N = pq$ divise $a^{k\phi(N)+1} - a$. On a donc :

$$a^{\phi(N)+1} - a \equiv 0 \mod N \Rightarrow a^{\phi(N)+1} \equiv a \mod N.$$

— **Si p divise a,** alors q ne divise pas a. de façon analogue, on déduit $a^{k\phi(N)+1} \equiv a \mod N$.
Dans tous les cas, on a bien $a^{k\phi(N)+1} \equiv a \mod N$. □

3 Algorithmes pour la cryptographie

Définition 11. *(Hypothèse de la factorisation).* L'hypothèse de la factorisation stipule qu'il n'existe pas d'algorithme permettant de factoriser un produit $N = pq$ de deux facteurs premiers p et q choisis aléatoirement en temps polynomial en $|N|$.

Test de primalité de Fermat : Le théorème de Fermat montre que si $a < p$ et $a^{p-1} \not\equiv 1 \mod p$, alors p n'est pas premier. La réciproque n'est pas vraie. Par exemple, $2^{340} \equiv 1 \mod 341$, pourtant $341 = 11 \times 31$. Les nombres non premiers vérifiant $a^{p-1} \not\equiv 1 \mod p$ sont appelés *nombres pseudo-premiers de Fermat de base a*. Les nombres pseudo-premiers de Fermat sont rares, et leur nombre tend rapidement vers zéro lorsque l'on augmente la taille des nombres testés. Par exemple, en prenant $a = 2$, la probabilité qu'un nombre p de plus de 540 bits vérifiant $a^{p-1} \not\equiv 1 \mod p$ soit pseudo-premier est inférieure à 10^{-20}. Pour des nombres de plus de 1024 bits, elle descend en dessous de 10^{-40}. Même si ce critère ne permet pas d'assurer qu'un nombre est premier avec certitude, il est tout à fait raisonnable de l'utiliser en pratique pour vérifier la parité de nombres assez grands.

Nous aurons besoin, dans la suite de ce cours, de générer de grands nombres premiers aléatoires. Pour cela, on peut se servir du test de primalité de Fermat. La méthode suivante permet de trouver assez rapidement un nombre premier de n bits :

— Choisir aléatoirement un nombre p de n bits ;
— Si $2^{p-1} \equiv 1 \mod p$, retourner p ;
— Sinon, recommencer ;

Pour diminuer la probabilité de générer un nombre pseudo-premier, on peut utiliser plusieurs bases différentes :

— Choisir k "petits" entiers a_k ;
— Choisir aléatoirement un nombre p de n bits ;
— Si pour tout k, $a_k^{p-1} \equiv 1 \mod p$, retourner p ;
— Sinon, recommencer avec un nouveau p ;

Définition 12. *(Hypothèse du logarithme discret).* L'hypothèse du logarithme discret dans un groupe multiplicatif $\mathbb{G}$ d'ordre premier p et généré par $g \in \mathbb{G}$ stipule qu'il n'existe pas d'algorithme permettant de calculer le logarithme discret $x = \log_g(h)$ d'un élément $h \in \mathbb{G}$ choisi aléatoirement en temps polynomial en $|\mathbb{G}|$.

Génération de groupes d'ordre premier : La méthode suivante permet de générer un groupe multiplicatif d'ordre premier dans lequel le problème du logarithme discret est difficile :

— Choisir p et q deux premiers tels que $q = 2p + 1$ ("premiers sûrs").
— $(\mathbb{Z}_q^*, \cdot)$ est un groupe et est d'ordre $2p$, ses éléments sont d'ordre p, 2, ou $2p$.
— Trouver g un élément d'ordre p (ces éléments sont très nombreux dans $\mathbb{Z}_q^*$)
— g génère le groupe $\mathbb{G} = \{g^x | x \in \mathbb{Z}\}$ d'ordre p, qui est premier

Exponentiation rapide : Soit x et e deux entiers positifs. On souhaite calculer x^e. La méthode naïve consiste à multiplier e fois x avec lui-même. La complexité de cette méthode est linéaire en e. Cependant, cette méthode est impraticable avec des paramètres cryptographiques. Il existe une autre méthode beaucoup plus efficace dont la complexité est logarithmique en e : *l'exponentiation rapide*. Cette méthode utilise la propriété suivante :

Proposition 1. Soit x et e deux entiers positifs. On pose $l = \log_2(e)$ et on note $b_l b_{l-1} \cdots b_1 b_0$ l'écriture binaire de e. On pose :

$$\begin{cases} x_0 = x \\ \forall i > 0, x_{i+1} = x_i^2. \end{cases}$$

On a :

$$x^e = \prod_{i=0}^{l} x_i^{b_i}.$$

Démonstration : On a :

$$e = \sum_{i=0}^{l} b_i 2^i.$$

On montre par récurrence que $x_i = x^{2^i}$ pour tout $i \geq 0$: la propriété est vraie pour $x_0 = x^1 = x^{2^0}$, et si $x_i = x^{2^i}$ alors $x_{i+1} = x_i^2 = (x^{2^i})^2 = x^{2^i \cdot 2} = x^{2^{i+1}}$. Finalement :

$$x^e = x^{\left(\sum_{i=0}^{l} b_i 2^i\right)} = \prod_{i=0}^{l} x^{b_i 2^i} = \prod_{i=0}^{l} (x^{2^i})^{b_i} = \prod_{i=0}^{l} x_i^{b_i}.$$

□

On en déduit l'algorithme d'exponentiation suivant, dont la complexité en temps est en $\log_2(e)$:

```
fonction exponentiation-rapide(x, e)
    y := 1
    tant que e != 0
        si (e mod 2 = 1)
            y := y * x
        x := x * x
        e := e / 2          //(division d'entiers)
    retourner y
```

Si on travaille avec des entiers modulo n, on peut adapter cet algorithme pour éviter d'avoir à manipuler de trop grands entiers :

```
fonction exponentiation-rapide(x, e, n)
    y := 1
    tant que e != 0
        si (e mod 2 = 1)
            y := (y * x) mod n
        x := (x * x) mod n
        e := e / 2          //(division d'entiers)
    retourner y
```

Exemple : Calcule de $3^{17} \mod 23$. Comme $17 = 16 + 1 = 2^4 + 2^0$, l'écriture binaire de 17 est 10001.
$3^2 = 9$
$9^2 = 81 \equiv 12 \mod 23$
$12^2 = 144 \equiv 6 \mod 23$
$6^2 = 36 \equiv 13 \mod 23$
Donc $3^{17} \mod 23 = 13 \times 3 \mod 23 = 39 \mod 23 = 16$

4 Cryptographie à clé publique

Définition 13. *(Protocole d'échange de clé).* Un protocole d'échange de clé est un protocole interactif entre deux parties (qu'on appellera Alice et Bob) dont l'objectif est d'établir une clé partagée entre Alice et Bob. Cette clé doit être secrète, dans le sens où un observateur malveillant ne doit pas pouvoir deviner la clé (en temps polynomial en un paramètre de sécurité qu'on notera l) même si il intercepte les messages échangés par Alice et Bob.

Définition 14. Le protocole d'échange de clé Diffie-Hellman entre Alice et Bob dans un groupe multiplicatif $\mathbb{G}$ d'ordre premier p (de l bits, l étant le paramètre de sécurité) et généré par $g \in \mathbb{G}$ est un protocole d'échange de clé défini comme suit :

— Alice choisit a dans $\mathbb{Z}_p^*$ et envoie $A = g^a$ à Bob.
— Bob choisit b dans $\mathbb{Z}_p^*$ et envoie $B = g^b$ à Alice.
— Alice calcule la clé $K = B^a$.
— Bob calcule la clé $K = A^b$.

Théorème 12. Alice et Bob partagent la même clé à la fin du protocole Diffie-Hellman.

Démonstration :

$$A^b = (g^a)^b = g^{ab} = g^{ba} = (g^b)^a = B^a$$

□

Remarque 2. Pour avoir un bon aléa de la clé K, on a besoin que le groupe soit d'ordre premier. En effet, si le groupe est d'ordre premier, tout élément est générateur, et est donc d'ordre p. Ainsi, A est d'ordre p, donc chaque b donnera un élément différent de $\mathbb{G}$, et donc une clé différente. On aura $p-1$ clés possibles (autant que d'éléments non neutres dans $\mathbb{G}$), et on évitera le cas $K = 1$ puisque $0 < b < p - 1$.
Pour aller plus loin : En utilisant l'algorithme d'attaque de Pohlig-Hellman (que nous ne décrirons pas dans ce cours), La difficulté du logarithme discret dans un groupe d'ordre n non premier revient à la difficulté du logarithme discret dans un groupe d'ordre p^k, ou p est le plus grand facteur premier de n et k est son exposant dans la décomposition en produit de facteurs premiers de n, d'où l'intérêt d'utiliser des groupes d'ordre premier.

Définition 15. *(Système de chiffrement à clé publique).* Un système de chiffrement à clé publique est défini par quatre algorithmes :

Algorithme Gen **de génération des clés publique et privée :** Cet algorithme prend en entrée un paramètre de sécurité et génère une clé publique pk et une clé privée sk.

Algorithme Enc **de chiffrement :** Cet algorithme prend en entrée un message M et une clé publique pk et renvoie un chiffré C.

Algorithme Den **de déchiffrement :** Cet algorithme prend en entée un chiffré C et une clé secrète sk et renvoie un message déchiffré M.

Pour tout paramètre l, tout message M, et tout $(\mathsf{pk}, \mathsf{sk})$ généré par $\mathsf{Gen}(l)$, on doit avoir $M = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, M))$. Tous les algorithmes doivent être polynomiaux en temps en l (efficacité), par contre, il ne doit pas être possible de déchiffrer un chiffré en temps polynomial en l (sécurité).

Définition 16. *(RSA).* le chiffrement RSA (pour Rivest-Shamir-Adleman) est un système de chiffrement à clé publique défini comme suit :

Génération des clés publique et privée depuis le paramètre l **:**
Choisir deux grands nombres premiers p et q de l bits au hasard (à l'aide du test de Fermat).
Poser $N = pq$ (on aura donc $\phi(N) = (p-1)(q-1)$).
Choisir un entier e au hasard tel que $0 \leq e \leq \phi(N) - 1$, et tel que e et $\phi(N)$ soient premiers entre eux (on vérifie qu'ils sont premiers entre eux avec l'algorithme d'Euclide).
Calculer (d, k) tels que $ed + \phi(N)k = 1$ (à l'aide de l'algorithme d'Euclide étendu).
La clé secrète est $d \pmod{\phi(N)}$, la clé publique est (N, e).

Chiffrement de $M < N$ **avec la clé publique** (e, N) **:** Calculer $C = M^e \mod N$.

Déchiffrement de C **avec la clé secrète** d **:** Calculer $M = C^d \mod N$.

Théorème 13. Le chiffrement RSA déchiffre correctement les messages chiffrés.

Démonstration : $ed + k\phi(N) = 1$ donc $ed = 1 - k\phi(N)$ et d'après Théorème 11 :

$$C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^{1-k\phi(N)} \equiv M \mod N.$$

□

Remarque 3. Le chiffrement RSA est déterministe, ce qui pose problème lorsque que l'ensemble des messages est petit. Par exemple, si l'ensemble des messages possibles d'un chiffré C est $\{\text{oui}, \text{non}\}$ (cas d'un vote), il suffit de rechiffrer "oui" et "non" avec la clé publique de chiffrement et de comparer le résultat avec C. Le chiffrement ElGamal, présenté ci-dessous, est probabiliste et évite ce problème.

Définition 17. *(ElGamal).* le chiffrement ElGamal dans un groupe multiplicatif $\mathbb{G}$ d'ordre premier p (de l bits, l étant le paramètre de sécurité) et généré par $g \in \mathbb{G}$ est un système de chiffrement à clé publique défini comme suit :

Génération des clés publique et privée : Choisir x dans $\mathbb{Z}_p^*$ et poser $y = g^x$.
La clé secrète est x, la clé publique est $(y, \mathbb{G}, g, p)$.

Chiffrement de M avec la clé publique $(y, \mathbb{G}, g, p)$ **:** Choisir r dans $\mathbb{Z}_p^*$, calculer $c_1 = g^r$ et $c_2 = y^r \cdot M$.
Le chiffré est $C = (c_1, c_2)$.

Déchiffrement de C avec la clé secrète x **:** Calculer $M = \frac{c_2}{c_1^x}$.

Théorème 14. Le chiffrement ElGamal déchiffre correctement les messages chiffrés.

Démonstration :

$$\frac{c_2}{c_1^x} = \frac{y^r \cdot M}{(g^r)^x} = \frac{(g^x)^r \cdot M}{g^{rx}} = \frac{g^{xr} \cdot M}{g^{rx}} = M$$

□

Définition 18. *(Système de signature).* Un système de signature est défini par quatre algorithmes :

Algorithme Gen **de génération des clés publique et privée :** Cet algorithme prend en entrée un paramètre de sécurité et génère une clé publique pk et une clé privée sk.

Algorithme Sig **de signature :** Cet algorithme prend en entrée un message M et une clé privée sk et renvoie une signature S.

Algorithme Ver **de vérification :** Cet algorithme prend en entée une clé publique pk, un message M, et une signature S, et renvoie un booléen indiquant si la signature est valide ou non.

Pour tout paramètre l, tout message M, et tout $(\mathsf{pk}, \mathsf{sk})$ généré par $\mathsf{Gen}(l)$, on doit avoir $\mathtt{True} = \mathsf{Ver}(\mathsf{pk}, M, \mathsf{Sig}(\mathsf{sk}, M))$. Tous les algorithmes doivent être polynomiaux en temps en l (efficacité), par contre, il ne doit pas être possible de générer une signature valide d'un message M sans connaître la clé sk en temps polynomial en l (sécurité).

Signature RSA :

Définition 19. la signature RSA est un système de signature défini comme suit :

Génération des clés publique et privée depuis le paramètre l :
Choisir deux grands nombres premiers p et q de l bits au hasard.
Poser $N = pq$.
Choisir un entier e au hasard tel que $0 \leq e \leq \phi(N) - 1$, et tel que e et $\phi(N)$ soient premiers entre eux.
Calculer (d, k) tels que $ed + \phi(N)k = 1$.
La clé secrète est $d \pmod{\phi(N)}$, la clé publique est (N, e).

Signature de M avec la clé secrète d **:** Calculer $S = M^d \mod N$.

Vérification de S avec la clé publique e **:** Vérifier $M = S^e \mod N$.

Théorème 15. La signature RSA authentifie correctement les messages signés.

5 Exercices théoriques

Exercice 1.

a. À l'aide de l'algorithme d'Euclide, calculer $\mathsf{pgcd}(96, 76)$, $\mathsf{pgcd}(50, 33)$, $\mathsf{pgcd}(306, 758)$ et $\mathsf{pgcd}(456, 43)$.

b. À l'aide de l'algorithme d'Euclide étendu, donner les coefficients de Bézout pour chaque couple (a, b) de la question précédente qui vérifie $\mathsf{pgcd}(a, b) = 1$.

c. Soient a et b deux entiers premiers entre eux. Montrer que $a + b$ et a sont premiers entre eux.

d. En déduire que $a + b$ et ab sont premiers entre eux.

e. Soient a, b et n trois entiers, montrer que $\mathsf{pgcd}(a, b)^n = \mathsf{pgcd}(a^n, b^n)$.

Exercice 2. Soit n un nombre entier.

a. Montrer que pour tout $a \in (\mathbb{Z}_n^*, \cdot)$ tel que $\mathsf{pgcd}(a, n) = 1$, a possède un inverse.

b. Montrer que pour tout $a \in (\mathbb{Z}_n^*, \cdot)$ tel que $\mathsf{pgcd}(a, n) \neq 1$, a ne possède pas d'inverse.

c. On pose $\mathbb{G} = \{a \in \mathbb{Z}_n^* | \mathsf{pgcd}(a, n) = 1\}$, montrer que $(\mathbb{G}, \cdot)$ est un groupe, et qu'il est le plus grand groupe compris dans $\mathbb{Z}_n^*$.

d. En déduire le plus grand groupe compris dans $(\mathbb{Z}_{12}^*, \cdot)$. Donner l'ordre et l'inverse de chaque élément, et l'ordre du groupe. Ce groupe est-il cyclique ?

Exercice 3.

a. Calculer $3^{48} \mod 97$ en utilisant l'exponentiation rapide, sachant que l'écriture binaire de 48 est 110000 (vous pouvez utiliser la calculatrice pour calculer les carrés et les modulos).

b. En déduire que 1 appartient au sous-groupe de $(\mathbb{Z}_{97}, +)$ engendré par 3 (qu'on notera $\mathbb{G}$).

c. En déduire que $\mathbb{G} = \mathbb{Z}_{97}$.

d. Résoudre le système suivant :

$$\begin{cases} 12x + 11y \equiv 40 \mod 97 \\ 24x + 7y \equiv 72 \mod 97 \end{cases}$$

e. Utiliser le test de Fermat en base 2 sur 97 (on utilisera l'exponentiation rapide, sachant que l'écriture binaire de 96 est 1100000). Que peut-on en conclure sur 97 ?

f. En sachant que 97 est premier, démontrer que le sous-groupe $\mathbb{H}$ de $(\mathbb{Z}_{97}, +)$ engendré par 87 vérifie $\mathbb{H} = \mathbb{Z}_{97}$.

Exercice 4.

a. Soient a et b deux entiers positifs non nuls tels que a n'est pas divisible par 11, montrer que $a^{900b} - 1$ est divisible par 11.

b. Soient p un nombre premier et a un entier inférieur à p, montrer que $a^p - a$ est divisible par p.

c. Soient p un nombre premier et a un entier inférieur à p, montrer que $a^p - a$ est divisible par $2p$.

d. Soient p un nombre premier et a et b deux entiers inférieurs à p, montrer que $a^{bp-b} - b^{ap-a}$ est divisible par p.

e. Soient p un nombre premier, a un entier inférieur à p et b un entier quelconque, montrer que $a^{b+p} + (p-1)a^{b+1}$ est divisible par p.

f. Soit p un entier tel que pour tout entier $1 < a < p$, $a^{p-1} \equiv 1 \mod p$. Montrer que p est premier.

Exercice 5.

a. On pose $p = 11$, $q = 13$ et $N = pq = 143$. On va utiliser ces nombres pour créer des clés de chiffrement pour RSA. $e = 12$ est-il un bon choix de clé publique ?

b. On choisit $e = 7$, déterminer la clé privée d.

c. En utilisant l'exponentiation rapide, chiffrer le message 10.

d. En utilisant l'exponentiation rapide, déchiffrer 2.

Exercice 6.

a. On pose $p = 23$ et $q = 11$. Quel est l'ordre de 3 dans $(\mathbb{Z}_{23}^*, \cdot)$? On note $\mathbb{G}$ le sous-groupe engendré par $g = 3$, on va mettre en place un système de chiffrement de ElGamal dans ce groupe.

b. On pose $\mathsf{sk} = 4$, calculer la clé publique pk.

c. Chiffrer le message 11 en utilisant $r = 2$ comme aléa.

d. Déchiffrer le message $(4, 12)$.

Exercice 7. Soit $(\mathbb{G}, \cdot)$ un groupe d'ordre premier p généré par g. Soit pk une clé publique pour un chiffrement de ElGamal.

a. On considère le chiffré (a, b) du message m avec la clé pk, on pose $x \in \mathbb{Z}_p$. Donner le déchiffrement de (a^x, b^x) et $(a \cdot g^x, b \cdot \mathsf{pk}^x)$ en fonction de m et x.

b. On considère le chiffré (a, b) du message m et le chiffré (c, d) du message n avec la clé pk, on pose $x \in \mathbb{Z}_p$. Donner le déchiffrement de $(a/c, b/d)$ en fonction de m et n.

Exercice 8. L'échange de clé Diffie-Hellman opère dans un groupe d'ordre premier. Dans cet exercice, on va utiliser ce protocole dans le groupe $(\mathbb{Z}_{23}, +)$.

a. Contrairement à l'usage, $(\mathbb{Z}_{23}, +)$ utilise l'addition et non la multiplication comme opération. Par quelle opération faudra-t-il remplacer l'exponentiation et le logarithme en base a dans le protocole ?

b. Montrer que $g = 5$ est un générateur de $(\mathbb{Z}_{23}, +)$.

c. Décrire l'échange de clé de Diffie-Hellman dans $(\mathbb{Z}_{23}, +)$ en utilisant le générateur $g = 5$.

d. On suppose que Alice envoie 14 et Bob envoie 21, calculer la clé partagée par Alice et Bob.

e. Cette version du protocole permet-elle un échange de clé sécurisé ?

Exercice 9. *Attaque sur la réutilisation de mêmes facteurs dans RSA.*

a. On considère N un produit de facteurs premiers, et deux exposants de chiffrement RSA e_1 et e_2 premiers entre eux. On notera d_1 et d_2 les exposants de déchiffrement respectifs. Soient deux chiffrés RSA $C_1 = M^{e_1} \mod N$ et $C_2 = M^{e_2} \mod N$ chiffrant le même message M. Montrer comment retrouver le message M sans utiliser d_1 et d_2 (on se servira des coefficients de Bézout de e_1 et e_2).

b. On considère deux chiffrés $C_1 = 18$ et $C_2 = 86$ chiffrant un même message secret pour les clés publiques RSA (e_1, N) et (e_2, N) telles que $e_1 = 5$, $e_2 = 7$ et $N = 221$. Sans factoriser N, déterminer le message M.

c. Quelle précaution doit-on prendre pour éviter ce genre d'attaques ?

6 Implémentation avec SageMath

Dans cette partie, on utilisera le logiciel SageMath. Ce logiciel permet, via l'interface de programmation du langage python, de réaliser diverses opérations arithmétiques de façon simple et optimisée. Pour commencer, installer le paquet `sagemath`. Pour lancer la console de SageMath, taper la commande `sage`. Dans cette console, vous pourrez exécuter des scripts via la commande `load("script.sage")`.

Exercice 10. Essayer, dans la console, les fonctions suivantes (on pourra vérifier les solutions des exercices de la partie théorique du TD) :

- `r = randrange(a, b)` : cette fonction génère un entier aléatoire compris entre a et b.
- `g = gcd(a, b)` : cette fonction renvoie le `pgcd` de a et b.
- `g, u, v = xgcd(a, b)` : cette fonction renvoie le `pgcd` g de a et b, ainsi que leurs coefficients de Bézout u et v.
- `y = pow(g, e, n)` : cette fonction envoie $g^e \mod n$ (et utilise l'exponentiation rapide).
- `b = is_prime(p)` : cette fonction indique si p est premier ou non via le booléen b.
- `p = next_prime(n)` : cette fonction renvoie le plus petit nombre premier p tel que $p > n$.
- `factor(n)` : cette fonction renvoie le produit de facteurs premiers de n.

Exercice 11. Dans ce qui suit, nous allons re-programmer les fonctions ci-dessus. Créer un script `crypto.sage` qui contiendra les fonctions que nous allons implémenter durant ce TD. Copier le code ci-dessous :

```
def my_gcd(a, b) :
    while (b != 0) :
        tmp = b
        b = a % b
        a = tmp
    return a
```

Charger le script dans la console Sage (`load("crypto.sage")`), et vérifier que la fonction fonctionne correctement en testant quelques valeurs et en comparant le résultat avec `gcd(a, b)` (on pourra utiliser l'instruction `my_gcd(a, b) == gcd(a, b)`).

a. En utilisant l'algorithme d'Euclide étendu, coder une fonction `my_xgcd` équivalente à la fonction `xgcd`. **Attention, en sage, l'opérateur / correspond à la division dans l'ensemble des rationnels. Pour récupérer le quotient de la division euclidienne, utiliser l'opérateur //.** Tester la fonction sur quelques valeurs et comparer le résultat avec celui de `xgcd`.

b. En utilisant l'algorithme d'exponentiation rapide, coder une fonction `my_pow` équivalente à la fonction `pow`. Tester la fonction sur quelques valeurs et comparer le résultat avec celui de `pow`.

c. En utilisant le test de Fermat en base 2, 3, 5 et 7, coder une fonction `my_is_prime` équivalente à la fonction `is_prime`. Tester la fonction sur quelques valeurs et comparer le résultat avec celui de `is_prime`.

d. Coder une fonction `my_next_prime` équivalente à la fonction `next_prime`. Tester la fonction sur quelques valeurs et comparer le résultat avec celui de `next_prime`.

Exercice 12. La fonction `factor` implémente un algorithme de factorisation optimisé. Malgré tout, comme à ce jour on ne connaît pas d'algorithme capable de factoriser de grands nombres en temps polynomial, cet algorithme atteint rapidement ses limites lorsque les facteurs grandissent. Pour constater cela, nous allons utiliser cette fonction :

```
import time
def test_factor() :
    l = 8
    while(l <= 111) :
```

```
rq = randrange(2, 2**l)
q = next_prime(rq)
rp = randrange(2, 2**l)
p = next_prime(rp)
N = p * q

start =  time.time()
factor(N)
stop = time.time()

print l, (stop - start) * 10^6
l = l + 1
```

a. Que fait la fonction `test_factor` ? Copier cette fonction dans le script et la tester.

b. Tracer, à l'aide de Gnuplot (installer le paquet "gnuplot" si besoin), une courbe décrivant le temps d'exécution de `factor` en fonction de la taille de ses facteurs en bits. Pour cela, copier l'affichage de `test_factor` dans un fichier `factor.dat`, lancer Gnuplot, et taper l'instruction `plot "factor.dat" with lines`.

c. Commenter le résultat.

Exercice 13. Dans cet exercice, on va implémenter le système de chiffrement RSA.

a. Écrire une fonction `rsa_gen(l)` qui prend un nombre entier l et qui génère la clé publique (e, N) et la clé privée d d'un système RSA. Pour ce faire, on générera deux nombres premiers p et q aléatoires de l bits, et on choisira un nombre e aléatoire inférieur à $\phi(pq)$ tel que e soit premier avec $\phi(pq)$.

b. Écrire une fonction `rsa_enc(e,N,M)` qui chiffre le message M avec la clé (e, N).

c. Écrire une fonction `rsa_dec(d,N,C)` qui déchiffre le chiffré C avec la clé d.

d. Tester vos fonctions pour $l = 512$, en chiffrant et déchiffrant quelques messages.

Exercice 14. Dans cet exercice, on va implémenter le système de chiffrement ElGamal.

a. Écrire une fonction `eg_set(l)` qui prend un nombre entier l et qui génère trois entiers q, p et g tels que :
- q est un nombre premier de l bits ;
- $p = 2q + 1$ et est aussi premier ;
- g engendre un sous-groupe de $\mathbb{Z}_p^*$ d'ordre q (qu'on appellera $\mathbb{G}$), et est le plus petit possible.

b. Écrire une fonction `eg_gen(q, p, g)` qui renvoie une paire de clés privée/publique ElGamal $(\mathsf{sk}, \mathsf{pk})$, en utilisant g comme base du groupe $\mathbb{G}$ d'ordre q.

c. Écrire une fonction `eg_enc(q, p, g, pk, M)` qui chiffre le message M avec la clé `pk`.

d. Écrire une fonction `eg_dec(q, p, g, sk, C)` qui déchiffre le chiffré C avec la clé `sk`.

e. Tester vos fonctions pour $l = 512$, en chiffrant et déchiffrant quelques messages.