D34: Méthodes de calcul efficaces
Exponentiation modulaire

1 Exercices

Exercice 1. Exponentiation

Soit x un élément quelconque d'un groupe G.

1. Détailler les étapes de calcul de x^{123} en utilisant les algorithmes `square-and-multiply`, `NAF`, 3-`NAF`.
2. On suppose x fixé. Détailler les étapes de précalcul puis de calcul de x^{123} à l'aide de l'algorithme de Yao.

Exercice 2. Exponentiation ternaire

Dans certains groupe, l'élévation au cube est une opération plus efficace que le carré.

1. Écrire un algorithme d'exponentiation `cube-and-multiply` sur le schéma de l'algorithme binaire mais exploitant la représentation ternaire de l'exposant.
2. Quelle est son nombre d'opération moyen?

Exercice 3. Densité moyenne de la représentation w`NAF`

Le but est de prouver que la densité moyenne de chiffres non-nuls de la représentation w`NAF` est de $\frac{1}{w+1}$.

1. Démontrer la formule classique:

$$\sum_{i=0}^{+\infty} \frac{i}{2^i} = 2.$$

(Indice: étudier la somme finie et remarquer que $\frac{1}{2^i} = \frac{1}{2^{i-1}} - \frac{1}{2^i}$.

2. Considérons une suite de bits aléatoire (chaque bit a une probabilité 1/2 d'être à 1). Pour tout $n \geq 0$, calculer p_n la probabilité qu'une telle suite commence par exactement n zéros.
3. En déduire que le nombre moyen de bits consécutifs à zéro d'un suite binaire aléatoire est de 1.
4. En déduire que le nombre de zéros consécutifs après un chiffre non nul dans la représentation w`NAF` est en moyenne de w.
5. En déduire finalement la densité moyenne de la représentation.

Exercice 4. Fenêtre fractionnée

L'algorithme w`NAF` n'est pas très flexible en terme de mémoire. En effet, passer de d'une taille de fenêtre w à $w+1$ demande de doubler la quantité de mémoire pour stocker les précalculs. Pour pallier ce problème, on introduit alors le concept de fenêtre fractionné. Ainsi, on utilise pour effectuer le recodage d'un exposant k l'ensemble suivant $\{\pm 1, \pm 3, \cdots \pm (2^w + m)\}$ avec $1 \leq m \leq 2^w - 3$.

1. Proposer un algorithme de recodage sur le modèle du wNAF. On pourra considérait k par paquet de $w+2$ bits et traiter les cas suivants:
   - $k \mod 2^{w+2} = 0$
   - $1 \leq k \mod 2^{w+2} \leq 2^w + m$
   - $2^w + m < k \mod 2^{w+2} \leq 3 \times 2^w - m$
   - $3 \times 2^w - mk \mod 2^{w+2} < 2^{w+2}$
2. Montrer que la densité moyenne de chiffres non nuls d'un telle représentation est:

$$\frac{1}{w + \frac{m+1}{2^w} + 2}.$$

Exercice 5. Représentation de Zeckendorf

On rappelle la définition de la suite de Fibonacci (F_n): $F_0 = 0, F_1 = 1, \forall n \geq 2, F_n = F_{n-1} + F_{n-2}$. On admettra que tout entier k peut se représenter comme somme d'éléments de la suite de Fibonacci. Ex: $19 = 13 + 5 + 1 = F_7 + F_5 + F_2$. On adoptera une représentation binaire pour décrire un entier donné sous cette forme (on pourra omettre F_0 et F_1). Ex $19 = (101001)_{\mathcal{F}}$. Dans la suite de l'exercice, x est un élément d'un groupe G quelconque.

1. Soit $n \geq 2$. Décrire un algorithme simple permettant de calculer x^{F_n} en effectuant uniquement des multiplications (exceptée l'élévation au carré initiale).
2. En déduire un algorithme type `double-and-add` pour calculer x^k, où $k = (k_{l-1} \dots k_0)_{\mathcal{F}}$.
3. Montrer que $k = (\underbrace{11\dots11}_{l \text{ fois}})_{\mathcal{F}} = (\underbrace{1010\dots10b}_{l \text{ fois}})_{\mathcal{F}}$, où b vaut 0 ou 1 selon la parité de l.
4. En déduire un algorithme de recodage de k tel qu'il n'y a jamais de 1 adjacent dans la représentation obtenue. Quelle est alors la compléxité, en nombre de multiplications, de l'algo de la question 2 dans le pire cas (pour cela on admettra que $F_n \sim \phi^n$ et $log(2)/log(\phi) = 1.44$, où ϕ est le nombre d'or).

2 Programmation

Exercice 1.

L'algorithme CRT-RSA est une version de RSA utilisant le théorème des restes chinois afin de découper l'exponentiation modulo n en deux exponentiations modulo p et q. En effet, d'après le CRT (Chinese Remainder Theorem), $\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q$. La fonction RSA calcule $x^e \mod n$ pour un $x \in \mathbb{Z}_n$. Soient $y_p = x^e \mod p$ et $y_q = x^e \mod q$, on peut retrouver l'unique entier $y \in \mathbb{Z}_n$ tel que $y \equiv y_p \mod p$ et $y \equiv y_q \mod q$ grâce à la formule:

$$y = y_p \, |q|_p^{-1} \, q + y_q \, |p|_q^{-1} \, p \mod n.$$

De plus, p (resp. q) étant premier, le sous groupe $\mathbb{Z}_{\text{ı}}$ (resp. $\mathbb{Z}_q$) est d'ordre $p-1$ (resp. $q-1$). On peut donc réduire le calcul de y_p (resp. y_q) à $x_p^{e_p} \mod p$ (resp. $x_q^{e_q} \mod q$) où $x_p \equiv x \mod p$ (resp. $x_q \equiv x \mod q$) et $e_p \equiv e \mod p-1$ (resp. $e_q \equiv e \mod q-1$).

1. Implanter l'algorithme de chiffrement/déchiffrement RSA à l'aide de la bibliothèque GMP. On utilisera l'algorithme `square-and-multiply` pour l'exponentitation modulaire. On vérifiera la validité des calculs à l'aide de la fonction `gmp_powm` de GMP.
2. Implanter l'algorithme de chiffrement CRT-RSA.
3. Comparer l'efficacité des deux algorithmes. Quel gain obtient-on? Cela est-il cohérent?
4. Améliorer la vitesse d'éxécution à l'aide de l'algorithme de Brauer.
5. Implanter une version multithread de l'algorithme. Comparer cette version avec l'implantation mono thread.