COMPLEX – Complexité, algorithmes randomisés et approchés
Semaine 6

Année 2014–2015

Bruno Escoffier
Safia Kedad-Sidhoum
Fanny Pascual
Ludovic Perret

1 Projet sur les Tests de Primalité

1.1 Cadre applicatif

La cryptographie à clef publique utilise des fonctions à sens unique avec trappe. De manière informelle, une telle fonction est une application facile à évaluer mais difficile à inverser sauf si l'on possède une information supplémentaire : la *trappe*. Pour construire ces fonctions, on utilise des problèmes difficiles (typiquement, de la classe NP). Le célèbre chiffrement à clef publique RSA tire sa sécurité du problème suivant :

`FACT`

Entrée : un entier $N \in \mathbb{N}^*$.

Question : trouver un diviseur *non-trivial* p de N.

Plus précisément, la clef publique dans RSA est donnée par $N = p \cdot q$ avec p et q des premiers. La trappe consiste ici en la connaissance des facteurs premiers p et q. Pour garantir un bon niveau de sécurité, il faut pouvoir générer des grands premiers. En effet, la complexité du meilleur algorithme pour `FACT` est sous-exponentielle en la taille de N. En pratique, pour générer une clef publique RSA, on 1) tire aléatoirement des grands nombres et 2) on vérifie que ces nombres sont premiers. La génération nécessite donc de résoudre efficacement le problème `PREMIER`.

1.2 Modèle

Le problème de primalité est le suivant :

`PREMIER`

Entrée : un entier $N \in \mathbb{N}^*$.

Question : N est-il premier ?

Nous allons implémenter un test de primalité naïf déterministe dont la complexité est exponentielle. Ensuite, nous allons implémenter deux tests probabilistes de primalité efficaces mais qui se trompent de temps en temps lorsque N est premier. Ainsi, si le test probabiliste retourne `premier` alors l'entier est premier avec une certaine probabilité. En revanche, si N est composé alors le test probabiliste retourne toujours `composé`.

1.3 Mise en oeuvre

Le choix du langage de programmation pour implémenter les tests est libre. En revanche, on souhaite tester la primalité de grands nombres. Il faut donc faire attention de choisir un langage permettant de gérer ces nombres. Par exemple, l'utilisation du package `gmp` (`https://gmplib.org/`) en `C` permet de gérer les grands nombres.

1.4 Organisation du travail

– Le travail est à effectuer en binôme.
– Les projets doivent être rendus le 25 Novembre 2014 au plus tard par mail à votre chargé de TD (le sujet de l'email doit être de la forme, [CPLX] PROJET 2, NOMbinôme1-NOMbinôme2). Votre livraison sera constituée d'une archive tar.gz qui doit comporter un rapport décrivant vos choix d'implémentation et votre code ainsi que la description des tests de validation et les réponses aux questions.
– Une soutenance est prévue lors des TD de la semaine du 24 Novembre 2014.

1.5 Travail à réaliser

Exercice 1 Arithmétique dans $\mathbb{Z}_N$

Dans un premier temps, nous allons écrire des fonctions qui permettent de travailler modulo un entier N.

Q 1.1 Écrire une fonction `my_pgcd` qui prend comme paramètres des entiers a, b et retourne $\text{pgcd}(a, b)$.

Q 1.2 Écrire une fonction `my_inverse` qui prend comme paramètres des entiers a, N et retourne – s'il existe – un entier b tel que $a\,b \equiv 1 \bmod N$. L'entier b est donc l'inverse modulo N de a. La fonction `my_inverse` retournera un message d'erreur si a n'est pas inversible modulo N.

Q 1.3 Écrire la fonction `expo_mod` qui prend comme paramètres des entiers m, e, N et retourne $m^e \bmod N$. On vous demande de faire une implémentation de l'exponentiation rapide (et de bien réduire modulo N à chaque étape).

Soit $e = \sum_{i=0}^{\text{Nb}-1} e_i 2^i$ (avec $e_{\text{Nb}-1} = 1$). L'idée de l'exponentiation rapide est de remarquer que

$$m^e \equiv \prod_{i=0}^{\text{Nb}-1} (m^{2^i})^{e_i} \bmod N.$$

FastExp(m,e)
$U \leftarrow 1$ $T \leftarrow m$,
Pour $i = 0$ à $\text{Nb} - 1$ **Faire**
  Si $e_i = 1$ **alors** $U \leftarrow T \cdot U \bmod N$
  $T \leftarrow T \cdot T \bmod N$
Retourner U

Exercice 2 Test Naïf

Un algorithme déterministe simple permettant de vérifier la primalité de N consiste à tester si N est divisible par un entier $k, 1 < k \leq \lfloor\sqrt{N}\rfloor$. Ainsi, N est premier s'il n'est pas divisible par aucun $k, 1 < k \leq \lfloor\sqrt{N}\rfloor$. Il est facile de voir que la complexité de cette approche est exponentielle en la taille de N.

Q 2.1 Écrire une fonction `first_test` qui permet d'effectuer le test naïf de primalité sur un entier N.

Q 2.2 Utiliser votre fonction pour compter les nombre d'entiers premiers $\leq 10^5$ (il y en a 9592).

Q 2.3 Dans votre rapport, vous préciserez la taille du plus grand entier qu'il est possible de tester avec votre fonction en ≈ 1 minute.

Exercice 3 Nombres de Carmichaël

Q 3.1 Écrire une fonction `Is_Carmichael` qui prend comme entrée un entier N et teste si N est de

Carmichaël (tester votre fonction avec $N = 561 = 3 \times 11 \times 17$, c'est le plus petit nombre de Carmichaël).

Q 3.2 Dans votre rapport, vous préciserez le plus grand nombre de Carmichaël trouvé avec votre fonction en ≈ 1 minute.

Q 3.3 Proposer et écrire une fonction `Gen_Carmichael` qui permet de générer un nombre de Carmichaël avec 3 facteurs premiers (vous pouvez utiliser votre fonction `first_test` pour générer des premiers).

Q 3.4 Écrire une fonction qui permet de lister les nombres de Carmichaël $\leq 10^5$ (il y en a 16).

Q 3.5 Dans votre rapport, vous préciserez le plus grand nombre de Carmichaël trouvé avec votre implémentation de `Gen_Carmichael` en ≈ 5 minutes.

Exercice 4 Test de Fermat

Q 4.1 Écrire une fonction `TestFermat` qui prend comme entrée un entier N et implémente le test de Fermat du cours.

Q 4.2 Pour tester votre fonction, vous utiliserez 1) des nombres retournés par `Gen_Carmichael` , 2) des nombres composés, 3) des nombres tirés aléatoirement.

Q 4.3 On vous demande de faire un fonction qui permet d'estimer expérimentalement la probabilité d'erreur de `TestFermat` pour des entiers $\leq 10^5$.

Q 4.4 Dans votre rapport, vous préciserez la taille du plus grand nombre qu'il est possible de tester avec votre implémentation du test de Fermat en ≈ 5 minutes.

Exercice 5 Test de Rabin et Miller

Q 5.1 Écrire une fonction `TestRabinMiller` qui prend comme entrée un entier N et implémente le test de Rabin-Miller comme dans le cours.

Q 5.2 Pour tester votre fonction, vous utiliserez 1) des nombres retournés par `Gen_Carmichael` , 2) des nombres composés, 3) des nombres tirés aléatoirement.

Q 5.3 On vous demande de faire un fonction qui permet d'estimer expérimentalement la probabilité d'erreur de `TestRabinMiller` pour des entiers $\leq 10^5$.

Q 5.4 Écrire une fonction `GenPKRSA` qui prend comme entrée un entier $t \geq 0$ et retourne $N = p \cdot q$, avec p et q deux entiers $< 2^t$ qui ne sont pas composés d'après les tests de Fermat et Rabin-Miller.

Q 5.5 Dans votre rapport, vous préciserez le plus grand t pour lequel votre fonction retourne une clef publique RSA $N = p \cdot q$ en ≈ 1 minute.