Rapport

Sonny Klotz - Idir Hamad - Younes Benyamna - Malek Zemni

Projet M1 Informatique
Primalité

18/02/2018

Module *TER*

Table des matières

Introduction

Ce document est le compte-rendu final de notre projet sur les tests de primalité qui s'inscrit dans le cadre du module *TER* du M1 informatique de l'*UVSQ*.

Les tests de primalité sont des algorithmes qui permettent de savoir si un nombre entier est premier. Ces tests sont indispensables pour la cryptographie à clé publique.

Il existe plusieurs algorithmes de tests de primalité. L'efficacité de ces algorithmes est particulièrement liée au cryptosystème utilisé.

Notre travail consiste donc à implémenter différents tests de primalité et de comparer leurs performances.

Dans la première partie de ce document, on présentera l'architecture de notre application, illustrée par un organigramme.
Ensuite, on parlera des principaux cryptosystèmes faisant appel à des tests de primalité.
La troisième partie traitera des différents algorithmes de tests de primalité implémentés.
Les mesures de performance et le comparatif des tests de primalités seront détaillés dans la quatrième partie.
Finalement, on établira un bilan technique de notre projet, quant à l'application, à l'organisation interne au sein du groupe et aux coûts.

1 Architecture de l'application

1.1 Organigramme et données échangées

Cet organigramme représente la décomposition en modules de l'application, ainsi que les informations qui circulent entre ces modules.

1.2 Fonctionnalités des modules

1.3 Outils et langages de programmation

Notre application va être implémentée dans le langage `C`. Le langage `C` possède plusieurs types pour représenter des nombre entiers. Cependant, tous ces types ont une précision fixe et ne peuvent pas dépasser un certain nombre d'octets. Le type le plus grand est le `long long int` qui peut contenir des entiers d'un taille maximale de 64 bits. Or, tous ces types sont beaucoup trop courts pour les applications cryptographiques qui nécessitent la manipulation de données d'au moins 512 bits.

`GNU MP` pour `GNU Multi Precision`, souvent appelée `GMP` est une bibliothèque `C`/`C++` de calcul multiprécision sur des nombres entiers, rationnels et à virgule flottante qui permet en particulier de manipuler de très grand nombres.

2 Cryptosystèmes

Les tests de primalité sont des algorithmes indispensables pour la cryptographie à clé publique. Ces tests sont couramment utilisés par les cryptosystèmes ***RSA*** et ***ElGamal***.
Pour *RSA*, les tests sont effectués lors la phase de génération de clés. Pour *ElGamal*, ils sont effectués lors de l'établissement d'un échange de clés.

2.1 RSA

2.2 ElGamal

3 Tests de primalité

Les tests de primalité sont des algorithmes qui permettent de savoir si un nombre entier est premier. Dans le cas où le nombre n'est pas premier, il est dit ***composé***. Dans cette partie, on va détailler les différents algorithmes de tests de primalité traités. Pour chaque test, on donnera un bref historique, l'algorithme du test, sa complexité et sa preuve, puis son implémentation.
Les tests de primalité peuvent être

— **déterministes :** fournissent toujours la même réponse pour un nombre donné

— **probabilistes :** peuvent fournir des réponses différentes pour un même nombre (utilisent des données tirées aléatoirement)

3.1 Test de Fermat

Le test de Fermat est un test de primalité probabiliste basé sur le *petit théorème de Fermat*.

> **Petit théorème de Fermat.** *si p est un nombre premier, alors pour tout nombre entier a premier avec p*
>
> $$a^{p-1} \equiv 1 \pmod{p}$$

3.1.1 Algorithme

Le théorème de Fermat décrit une propriété commune à tous les nombres premiers qui peut être utilisée pour détecter si un nombre est premier ou bien composé.
En effet, si pour un entier a premier avec n :

— $a^{n-1} \not\equiv 1 \pmod{n}$ alors n est surement composé.

— $a^{n-1} \equiv 1 \pmod{n}$, on ne peut pas conclure avec certitude que n est premier puisque la réciproque du théorème de Fermat est fausse. Un nombre n vérifiant cette équation peut être premier, mais aussi composé, dans ce cas n est dit ***pseudo-premier*** *de base* a.

Les nombres pseudo-premiers sont relativement rares. On peut donc envisager d'adopter ce critère pour un test probabiliste de primalité, qui est le test de Fermat.

Algorithme 1 : Test de Fermat

Données : un entier n et le nombre de répétitions souhaitée k

pour $i = 1$ *jusqu'à* k **faire**

  Choisir aléatoirement a tel que $1 < a < n-1$;

  si $a^{p-1} \not\equiv 1 \pmod{n}$ **alors**

    retourner composé;

  fin

fin

retourner premier;

3.1.2 Complexité

3.1.3 Preuve

3.2 Test de Miller-Rabin

3.3 AKS

4 Mesures de performance et comparatifs

5 Bilan technique du projet

Notre produit final, c'est à dire l'application, se comporte comme prévu : l'application est fonctionnelle, la liaison entre ses différents modules réussit bien et les différentes fonctionnalités fournissent le résultat attendu.

5.1 Problèmes rencontrés

Problèmes résolus :

Lors de la réalisation de l'application, on a été confrontés à plusieurs problèmes et points délicats, principalement des verrous techniques, qui ont perturbé le bon déroulement de notre travail :

Problèmes non résolus :

Certains problèmes rencontrés n'ont pas été entièrement résolus. Ces problèmes ne sont pas déterminants pour l'acceptabilité de notre produit.

5.2 Organisation interne du groupe

Assignation des modules pour chaque membre du groupe : Cette répartition a été parfaitement respectée. Elle nous a permis de travailler efficacement et assez indépendamment, ce qui prouve que l'assignation des modules a été judicieusement faite. Nous sommes également restés en contact pendant toute la phase de développement pour s'entraider pour la prise en main des nouveaux outils.

5.3 Coûts

Ce tableau indique les coûts estimés et les coûts finaux, en nombre de lignes de code et pour chaque module :

Conclusion