

PROJET TUTORÉ :

GÉNÉRATION ALÉATOIRE DE NOMBRE PREMIERS.

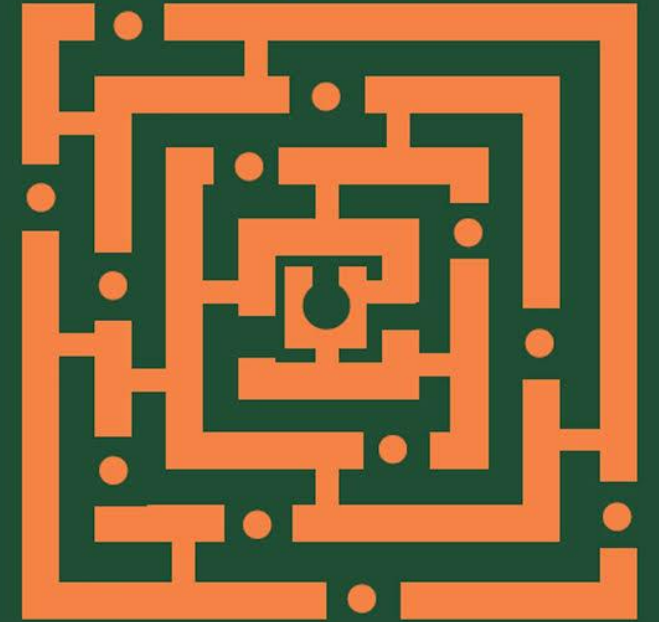
Toky RANDRIAMALALA
Elias DEBEYSSAC
Pierre GRABER

Année 2019-2020

Introduction

- Contexte
- Motivations du projet
- Liens direct avec le cours : Arithmétique et Cryptologie
 - Modules RSA, Signature Digitale

HANDBOOK of APPLIED CRYPTOGRAPHY



Alfred J. Menezes
Paul C. van Oorschot
Scott A. Vanstone

 CRC Press
Taylor & Francis Group

Sommaire

Partie 1 : SageMath, Aléatoire, Tests de primalité



```
graph TD; A[Partie 1 : SageMath, Aléatoire, Tests de primalité] --> B[Partie 2 : Nos Algorithmes]; B --> C[Partie 3 : Expérimentations, Tests d'efficacité]; C --> D[Conclusion];
```

Partie 2 : Nos Algorithmes

Partie 3 : Expérimentations, Tests d'efficacité

Conclusion



PREMIÈRE PARTIE



Pourquoi SageMath ?

- Langage Python :
 - Haut niveau
 - Déjà manipulé
- Logiciel Libre de calcul Mathématique
- Fonctions algébriques déjà implémentées
 - Exponentiation modulaire
 - Tests de primalité, Pseudo-Primalité

Les Tests de Primalité :

Divisions Successives

Miller-Rabin

Tests fournis par Sage
:

- `is_prime()`
- `is_pseudoprime()`



SECONDE PARTIE

Premier Algorithme : Recherche Aléatoire

Algorithme
simple

Boucle "Tant
que" :

- Tirage aléatoire
- Test de Primalité

Génération de "Strong Prime"

Algorithme de Gordon

Premiers forts :

- $(p-1)$ et $(p+1)$ ont des "grands" facteurs premiers
- T_e

Besoin de modules RSA surs

- Eviter la factorisation de $(p-1)$ et $(p+1)$

Génération de "Strong Prime"

Algorithme de Gordon

Premiers forts :

- $(p-1)$ et $(p+1)$ ont des "grands" facteurs premiers
- T_e

Besoin de modules RSA surs

- Eviter la factorisation de $(p-1)$ et $(p+1)$

