



République Algérienne Démocratique et Populaire  
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



**Université des Sciences et de la Technologie Houari Boumediene**

Faculté d'Informatique

Filière : Informatique

Spécialité : Sécurité des Systèmes Informatiques

Module : Arithmétique Modulaire

**Rapport de projet**

---

## **Projet**

### **Analyse et Implémentation des Tests de Primalité**

---

**Présenté par :**

<b>IMESSAOUDENE</b>	Aldjia	222231376511
<b>YAHIAOUI</b>	Abderrahmane	222231649908
<b>OUBYI</b>	Mohamed yacine	212131095500

# Table des matières

<b>Table des Figures</b>	<b>i</b>
<b>Introduction Générale</b>	<b>1</b>
<b>Analyse</b>	<b>3</b>
1.1 Introduction aux Concepts Algébriques Fondamentaux . . . . .	3
1.1.1 Arithmétique Modulaire et Corps Finis . . . . .	3
1.1.2 Théorèmes Fondamentaux . . . . .	3
1.2 Symbole de Jacobi et Réciprocité Quadratique . . . . .	4
1.2.1 Définition et Propriétés . . . . .	4
1.2.2 Lois de Réciprocité Quadratique . . . . .	4
1.2.3 Algorithme de Calcul du Symbole de Jacobi . . . . .	5
1.3 Test de Solovay-Strassen : Fondements Algébriques . . . . .	5
1.3.1 Principe Théorique . . . . .	5
1.3.2 Propriétés Probabilistes . . . . .	5
1.3.3 Analyse d'Erreur . . . . .	6
1.3.4 Algorithme Formel . . . . .	6
1.4 Test de Miller-Rabin : Aspects Algébriques Avancés . . . . .	6
1.4.1 Décomposition de $n - 1$ . . . . .	6
1.4.2 Racines Carrées de l'Unité dans $\mathbb{Z}_n$ . . . . .	6
1.4.3 Condition de Primalité . . . . .	6
1.4.4 Analyse des Témoins . . . . .	6
1.4.5 Borne d'Erreur . . . . .	7
1.4.6 Algorithme Formel . . . . .	7
1.5 Test de Fermat : Simplicité et Limitations . . . . .	7
1.5.1 Principe de Base . . . . .	7
1.5.2 Limitations : Nombres de Carmichael . . . . .	7
1.5.3 Analyse Probabiliste . . . . .	7
1.6 Test de Baillie-PSW : Approche Combinatoire . . . . .	8
1.6.1 Principe de Combinaison . . . . .	8
1.6.2 Séquence de Lucas . . . . .	8
1.6.3 Condition de Lucas . . . . .	8
1.6.4 Robustesse Théorique . . . . .	8
1.7 Test de Lucas-Lehmer pour les Nombres de Mersenne . . . . .	8
1.7.1 Définition des Nombres de Mersenne . . . . .	8
1.7.2 Suite Récursive . . . . .	9
1.7.3 Théorème de Lucas-Lehmer . . . . .	9

1.7.4	Preuve Algébrique (esquisse)	9
1.8	Complexité Algorithmique et Optimisations	9
1.8.1	Exponentiation Modulaire Rapide	9
1.8.2	Complexités Comparatives	10
1.8.3	Optimisations Pratiques	10
1.9	Limites Théoriques et Contre-Exemples	10
1.9.1	Nombres de Carmichael	10
1.9.2	Pseudopremiers d'Euler-Jacobi	11
1.9.3	Pseudopremiers Forts	11
1.10	Conclusion sur les Aspects Algébriques	11
<b>Améliorations Futures</b>		<b>12</b>
6.11	Introduction	12
6.12	Méthodes Déterministes Avancées	12
6.12.1	Test AKS (Agrawal-Kayal-Saxena)	12
6.12.2	Test ECPP (Elliptic Curve Primality Proving)	13
6.12.3	Test APR-CL (Adleman-Pomerance-Rumely - Cohen-Lenstra)	13
6.13	Méthodes Probabilistes Supplémentaires	13
6.13.1	Test de Frobenius	13
6.13.2	Test de Lucas-Lehmer-Riesel	14
6.14	Méthodes Spécialisées	14
6.14.1	Test pour les Nombres de Fermat	14
6.14.2	Test pour les Nombres de Proth	14
6.15	Améliorations du Système Actuel	15
6.15.1	Optimisations Algorithmiques	15
6.15.2	Extensions de l'Interface Graphique	15
6.15.3	Intégration de Bibliothèques Externes	15
6.16	Implémentation des Tests Déterministes	16
6.16.1	Plan d'Implémentation du Test AKS	16
6.16.2	Plan d'Implémentation d'ECPP	16
6.17	Extensions Cryptographiques	16
6.17.1	Génération de Nombres Premiers Cryptographiques	16
6.17.2	Validation des Paramètres Cryptographiques	17
6.18	Améliorations des Performances	17
6.18.1	Calcul Parallèle	17
6.18.2	Optimisations Matricielles	17
6.19	Base de Données et Benchmarks	18
6.19.1	Base de Données de Nombres Tests	18
6.20	Intégration Continue et Tests Automatisés	18
6.20.1	Pipeline de Tests	18
6.21	Conclusion et Perspectives	19
6.21.1	Résumé des Améliorations Possibles	19
6.21.2	Impact Potentiel	19
6.21.3	Feuille de Route	19
6.21.4	Conclusion Finale	19

<b>Annexe</b>	<b>20</b>
<b>Conclusion</b>	<b>23</b>

# Table des figures

6.1	Execution Miller-Rabin et Solvay-Strassen . . . . .	<a href="#">20</a>
6.2	Execution Fermat et Baillies-PSW . . . . .	<a href="#">21</a>
6.3	Execution Lucas-lehmer . . . . .	<a href="#">21</a>
6.4	Statistique sur les tests du primalité . . . . .	<a href="#">22</a>
6.5	Donnée JSON pour les sauvegarder . . . . .	<a href="#">22</a>

# Introduction Générale

La cryptographie moderne constitue l'épine dorsale de la sécurité numérique contemporaine, protégeant l'ensemble de nos communications électroniques, transactions financières et données personnelles dans un monde en constante évolution technologique. Au cœur de ces systèmes de protection sophistiqués réside un problème mathématique fondamental dont les origines remontent à l'Antiquité : la détermination de la primalité des nombres entiers. Cette interrogation, qui a fasciné les mathématiciens depuis Euclide, est devenue au cours du XX<sup>e</sup> siècle un enjeu technologique de première importance avec l'émergence de la cryptographie à clé publique.

Le lien intrinsèque entre primalité et cryptographie trouve son illustration la plus éloquente dans l'algorithme RSA, l'un des systèmes cryptographiques asymétriques les plus utilisés à l'échelle mondiale. Développé en 1977 par Rivest, Shamir et Adleman, RSA repose sur un postulat mathématique d'une élégante simplicité : la multiplication de deux grands nombres premiers est une opération algorithmiquement aisée, tandis que la factorisation de leur produit constitue un problème d'une complexité redoutable. Cette asymétrie computationnelle fondamentale, connue sous la dénomination de « problème de la factorisation », permet d'établir des mécanismes cryptographiques où la clé publique — correspondant au produit des deux nombres premiers — peut être librement diffusée, tandis que la clé privée — composée des facteurs premiers — demeure strictement confidentielle. Ainsi, la sécurité d'innombrables systèmes informatiques critiques, depuis les transactions bancaires sécurisées jusqu'aux signatures électroniques légales en passant par le chiffrement des communications électroniques, repose substantiellement sur notre capacité à générer et identifier avec fiabilité des nombres premiers de très grande taille.

Cette dépendance donne lieu à un paradoxe particulièrement intéressant : alors que la cryptographie asymétrique s'appuie sur la difficulté théorique de la factorisation, elle exige en amont la disponibilité de nombres premiers dont la taille s'accroît continuellement pour garantir une sécurité suffisante. Pour un nombre entier de 1024 bits — correspondant approximativement à 300 chiffres décimaux — une vérification exhaustive par divisions successives s'avérerait totalement impraticable, nécessitant des temps de calcul astronomiques. Cette contrainte a catalysé le développement de tests de primalité probabilistes, qui proposent une alternative révolutionnaire : plutôt que d'offrir une certitude absolue, ces algorithmes fournissent une réponse statistique « probablement premier » avec une marge d'erreur suffisamment faible pour être considérée comme négligeable dans les applications pratiques.

Parmi les avancées majeures dans ce domaine, les tests de Solovay-Strassen (1977) et

de Miller-Rabin (1980) ont marqué un tournant historique dans l'histoire de la cryptographie appliquée. En réduisant le temps de vérification de plusieurs ordres de grandeur tout en maintenant un taux d'erreur rigoureusement borné — inférieur à  $4^{-k}$  pour Miller-Rabin avec  $k$  itérations — ces algorithmes ont rendu techniquement viable la génération rapide de clés cryptographiques robustes. Ces méthodes ne représentent pas uniquement des outils pratiques ; elles incarnent également une certaine élégance mathématique, s'appuyant sur des concepts théoriques profonds tels que le symbole de Jacobi, le critère d'Euler, ou encore les propriétés des racines carrées dans les corps finis.

À l'ère actuelle, alors que l'avènement progressif de l'informatique quantique menace de compromettre certains systèmes cryptographiques traditionnels, l'importance des tests de primalité ne connaît aucune diminution. Au contraire, ces algorithmes conservent toute leur pertinence pour les cryptosystèmes post-quantiques, la génération de nombres aléatoires cryptographiques, ainsi que de nombreux autres protocoles de sécurité émergents. La quête permanente des plus grands nombres premiers connus — dont la taille atteint désormais plusieurs millions de chiffres décimaux — continue de repousser les limites de ces algorithmes et d'interroger nos capacités computationnelles.

Le présent projet de recherche s'inscrit dans cette riche tradition scientifique et technique, ayant pour objectif l'implémentation et l'analyse comparative de plusieurs algorithmes majeurs de tests de primalité. À travers l'étude détaillée des méthodes de Solovay-Strassen, Miller-Rabin, Fermat, Baillie-PSW et Lucas-Lehmer, cette recherche explorera non seulement leurs caractéristiques opérationnelles et leurs performances pratiques, mais également les fondements mathématiques qui les sous-tendent. Bien au-delà d'un simple exercice de programmation, ce travail constitue une immersion approfondie au cœur des principes fondamentaux de la sécurité numérique, à l'intersection féconde de la théorie des nombres, de l'algorithmique et de la cryptographie appliquée.

# Fondements Théoriques et Aspects Algébriques

## 1.1 Introduction aux Concepts Algébriques Fondamentaux

Les tests de primalité modernes reposent sur des fondements mathématiques profonds issus de l'arithmétique modulaire et de la théorie des nombres. Cette section présente les concepts algébriques essentiels qui sous-tendent les algorithmes implémentés dans ce projet.

### 1.1.1 Arithmétique Modulaire et Corps Finis

L'arithmétique modulaire constitue le langage naturel des tests de primalité. Pour un entier  $n > 0$ , l'ensemble  $\mathbb{Z}_n = \{0, 1, 2, \dots, n-1\}$  forme un **anneau modulo  $n$** . Lorsque  $n$  est premier, cet anneau devient un **corps fini**, noté  $\mathbb{F}_n$ , où chaque élément non nul possède un inverse multiplicatif.

**Propriété fondamentale** : Si  $n$  est premier, alors  $\mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$  est un groupe multiplicatif d'ordre  $\phi(n) = n-1$ , où  $\phi$  désigne la fonction indicatrice d'Euler.

### 1.1.2 Théorèmes Fondamentaux

#### a) Petit Théorème de Fermat

Pour tout nombre premier  $p$  et tout entier  $a$  non divisible par  $p$  :

$$a^{p-1} \equiv 1 \pmod{p}$$

Ce théorème, découvert par Pierre de Fermat au XVII<sup>e</sup> siècle, constitue la base théorique du test de Fermat. Formellement :

$$\forall p \in \mathbb{P}, \forall a \in \mathbb{Z} \text{ tel que } \gcd(a, p) = 1 \Rightarrow a^{p-1} \equiv 1 \pmod{p}$$

#### b) Théorème d'Euler

Généralisation du théorème précédent pour tout entier  $n$  :

$$a^{\phi(n)} \equiv 1 \pmod{n} \quad \text{si} \quad \gcd(a, n) = 1$$

où  $\phi(n)$  est la fonction d'Euler, qui compte le nombre d'entiers entre 1 et  $n$  qui sont premiers avec  $n$ .

### c) Critère d'Euler

Pour un nombre premier impair  $p$  et un entier  $a$  non divisible par  $p$  :

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod{p}$$

où  $\left(\frac{a}{p}\right)$  est le symbole de Legendre, définie par :

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{si } p \mid a \\ 1 & \text{si } a \text{ est un résidu quadratique modulo } p \\ -1 & \text{sinon} \end{cases}$$

## 1.2 Symbole de Jacobi et Réciprocité Quadratique

### 1.2.1 Définition et Propriétés

Le symbole de Jacobi  $\left(\frac{a}{n}\right)$  est une généralisation du symbole de Legendre aux entiers composés impairs. Si  $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$  est la factorisation première de  $n$ , alors :

$$\left(\frac{a}{n}\right) = \prod_{i=1}^k \left(\frac{a}{p_i}\right)^{e_i}$$

**Propriétés essentielles :**

1. **Multiplicativité** :  $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right) \left(\frac{b}{n}\right)$
2. **Périodicité** : Si  $a \equiv b \pmod{n}$ , alors  $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$
3. **Valeurs possibles** :  $-1, 0$ , ou  $1$
4. Pour  $n$  premier, le symbole de Jacobi coïncide avec le symbole de Legendre

### 1.2.2 Lois de Réciprocité Quadratique

La loi de réciprocité quadratique, démontrée par Gauss, constitue un résultat profond qui permet de calculer efficacement les symboles de Jacobi.

#### a) Première loi de complément

$$\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}} = \begin{cases} 1 & \text{si } n \equiv 1 \pmod{4} \\ -1 & \text{si } n \equiv 3 \pmod{4} \end{cases}$$

#### b) Deuxième loi de complément

$$\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}} = \begin{cases} 1 & \text{si } n \equiv \pm 1 \pmod{8} \\ -1 & \text{si } n \equiv \pm 3 \pmod{8} \end{cases}$$

### c) Loi de réciprocité principale

Pour  $m$  et  $n$  impairs positifs premiers entre eux :

$$\left(\frac{m}{n}\right) \left(\frac{n}{m}\right) = (-1)^{\frac{(m-1)(n-1)}{4}}$$

### 1.2.3 Algorithme de Calcul du Symbole de Jacobi

L'algorithme implémenté utilise ces propriétés pour calculer efficacement  $\left(\frac{a}{n}\right)$  sans nécessiter la factorisation de  $n$ .

[H] [1] **Jacobia**,  $n \leq 0$  ou  $n$  est pair **erreur**  $a \leftarrow a \bmod n$   $a = 0$   $resultat \leftarrow 1$   $a \neq 0$   $a$  est pair  $a \leftarrow a/2$   $n \bmod 8 \in \{3, 5\}$   $resultat \leftarrow -resultat$   $a \bmod 4 = 3$  et  $n \bmod 4 = 3$   $resultat \leftarrow -resultat$   $a, n \leftarrow n, a$   $a \leftarrow a \bmod n$   $n = 1$   $resultat$  0

**Complexité** :  $O(\log \max(a, n))$  opérations, comparable à celle de l'algorithme d'Euclide étendu.

## 1.3 Test de Solovay-Strassen : Fondements Algébriques

### 1.3.1 Principe Théorique

Le test de Solovay-Strassen, publié en 1977, repose sur une condition nécessaire et suffisante pour les nombres premiers impairs. Pour un entier impair  $n$ , on a l'équivalence :

$$n \text{ est premier} \iff \forall a \in \mathbb{Z}_n^*, \quad a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$$

#### a) Démonstration

1. **Implication directe** : Si  $n$  est premier, le critère d'Euler garantit l'égalité.
2. **Réciproque** : Si  $n$  est composé, alors au moins la moitié des  $a \in \mathbb{Z}_n^*$  violent l'égalité. Plus précisément, soit  $n$  composé impair. L'ensemble :

$$E(n) = \left\{ a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n} \right\}$$

est un sous-groupe de  $\mathbb{Z}_n^*$ . Par le théorème de Lagrange,  $|E(n)|$  divise  $|\mathbb{Z}_n^*| = \phi(n)$ . On peut montrer que  $|E(n)| \leq \frac{\phi(n)}{2}$ .

### 1.3.2 Propriétés Probabilistes

Soit  $n$  un entier composé impair. La probabilité qu'un  $a$  choisi uniformément dans  $\mathbb{Z}_n^*$  satisfasse la congruence est au plus  $\frac{1}{2}$ .

**Théorème** : Pour  $n$  composé impair,

$$P\left(a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}\right) \leq \frac{1}{2}$$

### 1.3.3 Analyse d'Erreur

Après  $k$  itérations indépendantes avec des témoins  $a_1, a_2, \dots, a_k$ , la probabilité d'erreur est :

$$P(\text{erreur}) \leq \left(\frac{1}{2}\right)^k = 2^{-k}$$

Cette borne est atteinte pour certains nombres composés, appelés "nombres d'Euler-Jacobi pseudopremiers forts".

### 1.3.4 Algorithme Formel

[H] [1] SolovayStrassen  $n, k$   $n < 2$  ou  $n$  est pair COMPOSITE  $i = 1$  à  $k$  Choisir  $a$  aléatoirement dans  $[2, n-2]$   $\gcd(a, n) > 1$  COMPOSITE  $x \leftarrow \left(\frac{a}{n}\right)$  (symbole de Jacobi)  $y \leftarrow a^{\frac{n-1}{2}} \pmod n$   $x \neq y \pmod n$  COMPOSITE PROBABLEMENT PREMIER

## 1.4 Test de Miller-Rabin : Aspects Algébriques Avancés

### 1.4.1 Décomposition de $n-1$

Tout entier impair  $n$  peut s'écrire de manière unique :

$$n-1 = 2^s \cdot d \quad \text{avec } d \text{ impair, } s \geq 1$$

### 1.4.2 Racines Carrées de l'Unité dans $\mathbb{Z}_n$

Dans un corps fini  $\mathbb{F}_p$  avec  $p$  premier, l'équation  $x^2 \equiv 1 \pmod p$  a exactement deux solutions :  $x \equiv \pm 1$ .

**Théorème** : Soit  $p$  premier impair. Si  $x^2 \equiv 1 \pmod p$ , alors  $x \equiv \pm 1 \pmod p$ .

Dans  $\mathbb{Z}_n$  avec  $n$  composé, il peut exister des racines carrées non triviales de 1, ce qui permet de détecter la composition.

### 1.4.3 Condition de Primalité

Pour  $n$  impair avec  $n-1 = 2^s \cdot d$ , on a la condition nécessaire :

$$n \text{ est premier} \implies \forall a \in \mathbb{Z}_n^*, \begin{cases} \text{soit } a^d \equiv 1 \pmod n \\ \text{soit } \exists r \in \{0, 1, \dots, s-1\} \text{ tel que } a^{2^r d} \equiv -1 \pmod n \end{cases}$$

### 1.4.4 Analyse des Témoins

Pour un entier composé impair  $n$ , on définit :

- **Témoin fort** : Un  $a$  qui prouve que  $n$  est composé
- **Non-témoin** : Un  $a$  qui ne permet pas de conclure

**Théorème de Rabin (1980)** : Pour un entier composé impair  $n$ , au moins  $\frac{3}{4}$  des  $a \in \mathbb{Z}_n^*$  sont des témoins forts.

### 1.4.5 Borne d'Erreur

Après  $k$  itérations avec des témoins choisis indépendamment, la probabilité qu'un nombre composé soit déclaré "probablement premier" est :

$$P(\text{erreur}) \leq \left(\frac{1}{4}\right)^k = 4^{-k}$$

Cette borne est atteinte pour les "nombres de Carmichael forts".

### 1.4.6 Algorithme Formel

[H] [1] MillerRabin,  $k$   $n < 2$  ou  $n$  est pair COMPOSITE Décomposer  $n - 1 = 2^s \cdot d$  avec  $d$  impair  $i = 1$  à  $k$  Choisir  $a$  aléatoirement dans  $[2, n - 2]$   $\gcd(a, n) > 1$  COMPOSITE  $x \leftarrow a^d \bmod n$   $x = 1$  ou  $x = n - 1$  Continuer à l'itération suivante  $j = 1$  à  $s - 1$   $x \leftarrow x^2 \bmod n$   $x = n - 1$  Sortir de la boucle interne  $x = 1$  COMPOSITE  $x \neq n - 1$  COMPOSITE PROBABLEMENT PREMIER

## 1.5 Test de Fermat : Simplicité et Limitations

### 1.5.1 Principe de Base

Basé directement sur le petit théorème de Fermat :

Si  $n$  est premier, alors  $\forall a$  avec  $\gcd(a, n) = 1$ ,  $a^{n-1} \equiv 1 \pmod{n}$

### 1.5.2 Limitations : Nombres de Carmichael

Un nombre de Carmichael est un nombre composé  $n$  tel que :

$$\forall a \text{ avec } \gcd(a, n) = 1, \quad a^{n-1} \equiv 1 \pmod{n}$$

Le plus petit nombre de Carmichael est  $561 = 3 \times 11 \times 17$ .

**Théorème de Korselt (1899)** : Un entier composé  $n$  est de Carmichael si et seulement si :

1.  $n$  est sans facteur carré
2. Pour tout diviseur premier  $p$  de  $n$ ,  $p - 1$  divise  $n - 1$

Formellement :

$$n \text{ est Carmichael} \iff (\forall p \mid n, p^2 \nmid n) \wedge (\forall p \mid n, (p - 1) \mid (n - 1))$$

### 1.5.3 Analyse Probabiliste

Pour un nombre composé non-Carmichael, la probabilité qu'un  $a$  aléatoire soit un non-témoin est :

$$P(a^{n-1} \equiv 1 \pmod{n}) \leq \frac{1}{2}$$

Mais pour les nombres de Carmichael, tous les  $a$  premiers avec  $n$  sont des non-témoins.

## 1.6 Test de Baillie-PSW : Approche Combinatoire

### 1.6.1 Principe de Combinaison

Le test de Baillie-PSW (1980) combine deux tests indépendants :

1. **Test de Miller-Rabin avec base  $a = 2$**
2. **Test de Lucas avec paramètre  $D$  approprié**

### 1.6.2 Séquence de Lucas

Soit  $P$  et  $Q$  des entiers avec  $D = P^2 - 4Q \neq 0$ . La séquence de Lucas  $\{U_k(P, Q)\}$  est définie par :

$$\begin{cases} U_0(P, Q) = 0 \\ U_1(P, Q) = 1 \\ U_k(P, Q) = P \cdot U_{k-1}(P, Q) - Q \cdot U_{k-2}(P, Q) \quad \text{pour } k \geq 2 \end{cases}$$

### 1.6.3 Condition de Lucas

Pour  $n$  premier impair avec  $\gcd(n, QD) = 1$ , on a :

$$U_{n - \left(\frac{D}{n}\right)} \equiv 0 \pmod{n}$$

où  $\left(\frac{D}{n}\right)$  est le symbole de Jacobi.

### 1.6.4 Robustesse Théorique

À ce jour, **aucun contre-exemple** n'est connu au test de Baillie-PSW. La recherche exhaustive jusqu'à  $2^{64}$  n'a trouvé aucun nombre composé passant les deux tests.

**Conjecture** : Il n'existe pas de nombre composé qui passe à la fois le test de Miller-Rabin avec base 2 et un test de Lucas approprié.

## 1.7 Test de Lucas-Lehmer pour les Nombres de Mersenne

### 1.7.1 Définition des Nombres de Mersenne

Un nombre de Mersenne est de la forme :

$$M_p = 2^p - 1 \quad \text{avec } p \text{ premier}$$

Notons que  $M_p$  peut être composé même si  $p$  est premier (exemple :  $M_{11} = 2047 = 23 \times 89$ ).

### 1.7.2 Suite Récurrente

Définissons la suite  $\{s_k\}$  par :

$$\begin{cases} s_0 = 4 \\ s_{k+1} = s_k^2 - 2 \pmod{M_p} \text{ pour } k \geq 0 \end{cases}$$

### 1.7.3 Théorème de Lucas-Lehmer

Pour  $p$  premier impair,  $M_p$  est premier si et seulement si :

$$s_{p-2} \equiv 0 \pmod{M_p}$$

### 1.7.4 Preuve Algébrique (esquisse)

La preuve utilise la structure des corps quadratiques  $\mathbb{Q}(\sqrt{3})$ . Considérons l'élément  $\omega = 2 + \sqrt{3} \in \mathbb{Z}[\sqrt{3}]$  et montrons que :

$$s_k \equiv \omega^{2^k} + \bar{\omega}^{2^k} \pmod{M_p}$$

où  $\bar{\omega} = 2 - \sqrt{3}$  est le conjugué.

On a  $\omega\bar{\omega} = 1$ , et en élevant au carré :

$$s_{k+1} = s_k^2 - 2 = (\omega^{2^k} + \bar{\omega}^{2^k})^2 - 2 = \omega^{2^{k+1}} + \bar{\omega}^{2^{k+1}}$$

Le résultat découle alors des propriétés de l'anneau  $\mathbb{Z}[\sqrt{3}]/(M_p)$ .

## 1.8 Complexité Algorithmique et Optimisations

### 1.8.1 Exponentiation Modulaire Rapide

L'opération fondamentale  $a^e \pmod{n}$  est implémentée via l'algorithme d'exponentiation binaire (*square-and-multiply*) :

**Algorithme :**

1. Initialiser  $resultat \leftarrow 1$ ,  $base \leftarrow a \pmod{n}$
2. Tant que  $e > 0$  :
  - Si  $e$  est impair :  $resultat \leftarrow (resultat \times base) \pmod{n}$
  - $base \leftarrow (base \times base) \pmod{n}$
  - $e \leftarrow \lfloor e/2 \rfloor$
3. Retourner  $resultat$

**Complexité :**  $O(\log e)$  multiplications modulaires, soit  $O(\log e \cdot \log^2 n)$  opérations binaires.

## 1.8.2 Complexités Comparatives

Test	Complexité temporelle	Complexité spatiale	Type
Solovay-Strassen	$O(k \log^3 n)$	$O(\log n)$	Probabiliste
Miller-Rabin	$O(k \log^3 n)$	$O(\log n)$	Probabiliste
Fermat	$O(\log^3 n)$	$O(\log n)$	Probabiliste
Baillie-PSW	$O(\log^3 n)$	$O(\log n)$	Déterministe*
Lucas-Lehmer	$O(p^2 \log p)$	$O(p)$	Déterministe

TABLEAU 1.1 – Complexités des différents tests de primalité

\* : Aucun contre-exemple connu, considéré comme déterministe en pratique.

## 1.8.3 Optimisations Pratiques

### a) Pré-calcul des petits nombres premiers

Pour éliminer rapidement les nombres divisibles par de petits facteurs :

Si  $\exists p \leq B$  premier tel que  $p \mid n$ , alors  $n$  est composé

où  $B$  est une borne appropriée (typiquement  $B = 1000$ ).

### b) Vérification des carrés parfaits

Un carré parfait ne peut être premier (sauf pour  $n = 1$ , qui n'est pas premier) :

Si  $\exists m \in \mathbb{N}$  tel que  $m^2 = n$ , alors  $n$  est composé

### c) Sélection intelligente des bases

Pour Miller-Rabin, utiliser des bases fixes pour les petits nombres :

Pour  $n < 3,474,749,660,383$ , il suffit de tester  $a \in \{2, 3, 5, 7, 11, 13, 17\}$

## 1.9 Limites Théoriques et Contre-Exemples

### 1.9.1 Nombres de Carmichael

Définis comme les nombres composés  $n$  tels que :

$$\forall a \in \mathbb{Z}_n^*, \quad a^{n-1} \equiv 1 \pmod{n}$$

**Théorème** (Alford, Granville, Pomerance, 1994) : Il existe une infinité de nombres de Carmichael.

### 1.9.2 Pseudopremiers d'Euler-Jacobi

Nombres composés  $n$  qui satisfont :

$$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n} \quad \text{pour certains } a$$

### 1.9.3 Pseudopremiers Forts

Nombres composés qui satisfont le test de Miller-Rabin pour certaines bases.

**Exemple** :  $n = 25,326,001 = 2251 \times 11251$  est un pseudopremier fort pour les bases 2, 3, 5.

## 1.10 Conclusion sur les Aspects Algébriques

Les tests de primalité modernes illustrent une synthèse remarquable entre théorie des nombres abstraite et algorithmique efficace. Chaque test exploite des propriétés algébriques spécifiques :

— **Solovay-Strassen** : Symbole de Jacobi et critère d'Euler

$$\left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \pmod{n}$$

— **Miller-Rabin** : Racines carrées de 1 dans les corps finis

$$x^2 \equiv 1 \pmod{p} \implies x \equiv \pm 1 \pmod{p}$$

— **Fermat** : Petit théorème de Fermat

$$a^{p-1} \equiv 1 \pmod{p}$$

— **Baillie-PSW** : Combinaison de tests indépendants

$$\text{Miller-Rabin}(2) \wedge \text{Lucas}(D)$$

— **Lucas-Lehmer** : Propriétés des extensions quadratiques

$$s_{p-2} \equiv 0 \pmod{M_p}$$

L'analyse des probabilités d'erreur repose sur des résultats profonds de théorie des groupes et d'arithmétique modulaire. La robustesse pratique de ces algorithmes, combinée à leur efficacité computationnelle, explique leur adoption massive en cryptographie et en informatique théorique.

Ces fondements théoriques guident non seulement l'implémentation des algorithmes, mais aussi l'analyse de leur fiabilité et de leurs limites, éléments essentiels pour toute application critique en sécurité informatique.

# Améliorations Futures

## 6.11 Introduction

Bien que notre système implémente les principaux tests de primalité probabilistes, le domaine de la primalité comporte de nombreuses autres méthodes, certaines déterministes, d'autres probabilistes, chacune avec ses avantages et inconvénients. Cette section présente les principales méthodes non implémentées dans ce projet, ainsi que des pistes d'amélioration pour notre système actuel.

## 6.12 Méthodes Déterministes Avancées

### 6.12.1 Test AKS (Agrawal-Kayal-Saxena)

Découvert en 2002, le test AKS est révolutionnaire car c'est le premier test de primalité déterministe qui fonctionne en temps polynomial pour tous les nombres entiers.

#### a) Principe Mathématique

Le test AKS est basé sur le théorème suivant : Pour  $a$  premier avec  $n$ , si et seulement si  $n$  est premier, alors

$$(X + a)^n \equiv X^n + a \pmod{n, X^r - 1}$$

pour certains  $r$  et  $a$  bien choisis.

#### b) Algorithme

1. Vérifier si  $n$  est une puissance parfaite :  $n = a^b$  pour  $b > 1$
2. Trouver le plus petit  $r$  tel que  $\text{ordre}_r(n) > \log^2 n$
3. Pour  $a = 1$  à  $\lfloor \sqrt{\phi(r)} \log n \rfloor$ 
  - Si  $\text{gcd}(a, n) > 1$ , retourner COMPOSITE
  - Vérifier la congruence  $(X + a)^n \equiv X^n + a \pmod{n, X^r - 1}$
4. Retourner PRIME

#### c) Complexité

$O(\log^{12} n)$  (amélioré à  $O(\log^6 n)$  dans les versions ultérieures)

### 6.12.2 Test ECPP (Elliptic Curve Primality Proving)

Basé sur la théorie des courbes elliptiques, ECPP est un test déterministe qui fournit une preuve de primalité.

#### a) Principe

Utilise le théorème de Goldwasser-Kilian : Si pour une courbe elliptique  $E$  sur  $\mathbb{Z}_n$  et un point  $P$  sur  $E$ , on peut trouver un nombre premier  $q > (n^{1/4} + 1)^2$  tel que  $q \cdot P = O$  (point à l'infini), alors  $n$  est premier.

#### b) Algorithme

1. Choisir une courbe elliptique  $E$  sur  $\mathbb{Z}_n$  et un point  $P$  sur  $E$
2. Calculer  $m = \#E(\mathbb{Z}_n)$  (nombre de points de la courbe)
3. Factoriser  $m = q \cdot s$  avec  $q$  premier probable
4. Vérifier que  $q > (n^{1/4} + 1)^2$
5. Vérifier que  $q \cdot P = O$
6. Récursivement prouver que  $q$  est premier

#### c) Complexité

$$O(\log^5 n) \quad (\text{en pratique})$$

### 6.12.3 Test APR-CL (Adleman-Pomerance-Rumely - Cohen-Lenstra)

Test déterministe basé sur la théorie des corps cyclotomiques.

#### a) Principe

Utilise les propriétés des extensions cyclotomiques  $\mathbb{Q}(\zeta_m)$  où  $\zeta_m$  est une racine  $m$ -ième de l'unité.

#### b) Complexité

$$O(\log^{c \log \log \log n} n) \quad (\text{quasi-polynomial})$$

## 6.13 Méthodes Probabilistes Supplémentaires

### 6.13.1 Test de Frobenius

Généralisation du test de Miller-Rabin basée sur les propriétés de l'endomorphisme de Frobenius.

### a) Principe

Pour  $n$  premier et  $\gcd(a, n) = 1$ , si  $x$  est une racine de  $X^2 - aX + b$  modulo  $n$ , alors

$$x^n \equiv \bar{x} \pmod{n}$$

où  $\bar{x}$  est le conjugué de  $x$ .

### b) Algorithme

[1] FrobeniusTest  $n, a, b$   $\gcd(a^2 - 4b, n) \neq 1$  COMPOSITE Calculer  $(X+Y)^n \pmod{(n, X^2 - aX + b)}$   $X^n + Y^n \equiv X + Y \pmod{(n, X^2 - aX + b)}$  PROBABLEMENT PREMIER COMPOSITE

## 6.13.2 Test de Lucas-Lehmer-Riesel

Extension du test de Lucas-Lehmer aux nombres de la forme  $k \cdot 2^n - 1$ .

### a) Principe

Pour  $N = k \cdot 2^n - 1$  avec  $k < 2^n$ , définir la suite :

$$\begin{cases} u_0 = \alpha^k + \alpha^{-k} \\ u_{i+1} = u_i^2 - 2 \pmod{N} \end{cases}$$

où  $\alpha$  est solution de  $x^2 - Px + Q = 0$ .

Alors  $N$  est premier si et seulement si  $u_{n-2} \equiv 0 \pmod{N}$ .

## 6.14 Méthodes Spécialisées

### 6.14.1 Test pour les Nombres de Fermat

Les nombres de Fermat sont de la forme  $F_n = 2^{2^n} + 1$ .

### a) Test de Pépin

Pour  $F_n$  avec  $n \geq 1$ ,  $F_n$  est premier si et seulement si :

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n}$$

### 6.14.2 Test pour les Nombres de Proth

Les nombres de Proth sont de la forme  $k \cdot 2^n + 1$  avec  $k$  impair et  $k < 2^n$ .

### a) Test de Proth

$N = k \cdot 2^n + 1$  est premier si et seulement si il existe un entier  $a$  tel que :

$$a^{(N-1)/2} \equiv -1 \pmod{N}$$

## 6.15 Améliorations du Système Actuel

### 6.15.1 Optimisations Algorithmiques

#### a) Multiplication Modulaire Rapide

Implémenter des algorithmes de multiplication modulaire avancés :

- **Montgomery multiplication** : Réduit les opérations modulaires

$$\text{Montgomery}(a, b) = \frac{a \cdot b \cdot R^{-1} \mod n}{R}$$

où  $R = 2^k > n$  et  $\gcd(R, n) = 1$

- **Barrett reduction** : Utilise des pré-calculs pour accélérer la réduction modulaire

#### b) Exponentiation Modulaire avec Fenêtres

Utiliser l'algorithme d'exponentiation par fenêtres :

$$\text{Pour } e = \sum_{i=0}^{m-1} e_i 2^{wi}, \quad a^e = \prod_{i=0}^{m-1} (a^{2^{wi}})^{e_i}$$

où  $w$  est la taille de la fenêtre.

### 6.15.2 Extensions de l'Interface Graphique

#### a) Visualisation des Preuves

- Affichage des témoins trouvés pour les nombres composés
- Visualisation des étapes intermédiaires des algorithmes
- Graphique interactif des probabilités d'erreur

#### b) Analyse Comparative Avancée

- Benchmark automatisé sur une base de nombres tests
- Analyse statistique des performances
- Comparaison avec des bibliothèques existantes (GMP, SymPy, etc.)

### 6.15.3 Intégration de Bibliothèques Externes

#### a) GMP (GNU Multiple Precision Arithmetic Library)

Utiliser GMP pour les opérations sur grands nombres :

```
#include <gmp.h>
mpz_t n;
mpz_init(n);
mpz_set_str(n, "12345678901234567890", 10);
```

## b) SymPy pour les Tests Symboliques

Intégrer SymPy pour des tests supplémentaires : `from sympy import isprime, nextprime`  
`isprime(2**521 - 1)` Test de Mersenne

## 6.16 Implémentation des Tests Déterministes

### 6.16.1 Plan d'Implémentation du Test AKS

#### a) Étape 1 : Vérification des Puissances Parfaites

Implémenter l'algorithme pour détecter si  $n = a^b$  : [1] `IsPerfectPower`  
 $b = 2$  à  $\lfloor \log_2 n \rfloor$   
 $a \leftarrow \lfloor n^{1/b} \rfloor$   $a^b = n$  ou  $(a+1)^b = n$  TRUE FALSE

#### b) Étape 2 : Calcul de l'Ordre Multiplicatif

Trouver le plus petit  $r$  tel que  $\text{ordre}_r(n) > \log^2 n$  :

$$\text{ordre}_r(n) = \min\{k > 0 \mid n^k \equiv 1 \pmod{r}\}$$

#### c) Étape 3 : Vérification des Congruences

Implémenter la vérification polynomiale :

$$(X + a)^n \equiv X^n + a \pmod{n, X^r - 1}$$

### 6.16.2 Plan d'Implémentation d'ECPP

#### a) Structure de Données pour les Courbes Elliptiques

```
class EllipticCurve : def init(self, a, b, n): self.a = a self.b = b self.n = n self.discriminant = (-16*(4*a**3 + 27*b**2))
def add_points(self, P, Q) : Implémentation de l'addition de points pass
def multiply_point(self, k, P) : Multiplication scalaire rapide pass
```

#### b) Algorithme de Preuve Récursive

[1] `ECPP`  $n$  est petit vérification directe Trouver courbe elliptique  $E$  et point  $P$   
Calculer  $m = \#E(\mathbb{Z}_n)$  Factoriser  $m = q \cdot s$  avec  $q$  premier probable  $q > (n^{1/4} + 1)^2$  et  $q \neq n$  Vérifier que  $q \cdot P = O$  `ECPP(q)` Appel récursif

## 6.17 Extensions Cryptographiques

### 6.17.1 Génération de Nombres Premiers Cryptographiques

#### a) Nombres Premiers Sûrs

Un nombre premier  $p$  est sûr si  $\frac{p-1}{2}$  est aussi premier.

$$p \text{ sûr} \iff p = 2q + 1 \text{ avec } q \text{ premier}$$

## b) Nombres Premiers Forts

Pour RSA, on préfère les nombres premiers forts :

$$\begin{cases} p-1 \text{ a un grand facteur premier } r \\ p+1 \text{ a un grand facteur premier } s \end{cases}$$

## 6.17.2 Validation des Paramètres Cryptographiques

### a) Paramètres pour RSA

Vérifier que :

- $p$  et  $q$  ont à peu près la même taille
- $\gcd(p-1, q-1)$  est petit
- $|p-q| > 2^{n/2-100}$  pour  $n$ -bits

### b) Paramètres pour Diffie-Hellman

Pour un groupe  $\mathbb{Z}_p^*$ , vérifier que :

$$\frac{p-1}{2} \text{ est premier (nombre premier sûr)}$$

## 6.18 Améliorations des Performances

### 6.18.1 Calcul Parallèle

#### a) Parallélisation des Tests

Exécuter différents tests en parallèle : `from concurrent.futures import ThreadPoolExecutor`

```
def test_parallel(n, tests):  
    with ThreadPoolExecutor() as executor:  
        results = list(executor.map(lambda test_function(n), tests))  
    return results
```

#### b) Parallélisation des Itérations

Pour les tests probabilistes, paralléliser les itérations :

$$P_{\text{parallèle}} = 1 - \prod_{i=1}^k P(\text{erreur pour thread } i)$$

### 6.18.2 Optimisations Matricielles

#### a) Multiplication de Polynômes Modulo

Pour AKS, optimiser la multiplication de polynômes :

$$(X+a)^n \mod (X^r-1) \text{ peut utiliser la FFT}$$

## b) Calcul Matriciel

Certains tests peuvent être formulés comme des problèmes matriciels :

$$M^n \equiv I \pmod{n} \quad \text{pour certaines matrices } M$$

## 6.19 Base de Données et Benchmarks

### 6.19.1 Base de Données de Nombres Tests

#### a) Nombres de Référence

- Nombres de Carmichael : 561, 1105, 1729, ...
- Nombres de Mersenne premiers :  $M_2, M_3, M_5, M_7, \dots$
- Nombres de Fermat :  $F_0, F_1, F_2, F_3, F_4$
- Nombres premiers aléatoires de différentes tailles

#### b) Métriques de Performance

Métrique	Description	Formule	Unité
Temps moyen	Temps d'exécution moyen	$\frac{1}{N} \sum t_i$	secondes
Déviati on standard	Variabilité des temps	$\sqrt{\frac{1}{N} \sum (t_i - \bar{t})^2}$	secondes
Taux d'erreur	Erreurs détectées	$\frac{\text{erreurs}}{\text{total}}$	pourcentage
Utilisation mémoire	Mémoire maximale	$\max(m_i)$	octets

TABLEAU 6.2 – Métriques de performance pour l'évaluation

## 6.20 Intégration Continue et Tests Automatisés

### 6.20.1 Pipeline de Tests

#### a) Validation des Algorithmes

- Tests unitaires pour chaque fonction
- Tests d'intégration entre les composants
- Tests de performance sur des benchmarks standards

#### b) Vérification des Résultats

Comparaison avec des bibliothèques de référence : `def verifywith_openssl(n) : UtiliserOpenSSLpour subprocess.run(["openssl", "prime", str(n)], captureoutput = True) return "isprime" in result.stdout.de`

## 6.21 Conclusion et Perspectives

### 6.21.1 Résumé des Améliorations Possibles

1. **Algorithmes avancés** : Implémentation d'AKS, ECPP, APR-CL
2. **Optimisations** : Multiplication Montgomery, parallélisation
3. **Extensions cryptographiques** : Génération de nombres premiers sûrs
4. **Interface enrichie** : Visualisation, analyse comparative
5. **Intégration** : Bibliothèques externes, tests automatisés

### 6.21.2 Impact Potentiel

Les améliorations proposées permettraient de :

- **Tripler la vitesse** d'exécution grâce aux optimisations
- **Étendre la gamme** des nombres testables (jusqu'à  $10^{1000}$ )
- **Fournir des preuves** de primalité pour les applications critiques
- **Devenir une référence** dans le domaine des tests de primalité

### 6.21.3 Feuille de Route

Amélioration	Priorité	Effort	Échéance
Optimisations Montgomery	Haute	2 semaines	Court terme
Parallélisation	Haute	3 semaines	Court terme
Interface de visualisation	Moyenne	4 semaines	Moyen terme
Implémentation d'ECPP	Moyenne	6 semaines	Moyen terme
Test AKS complet	Basse	8 semaines	Long terme

TABLEAU 6.3 – Feuille de route des améliorations

### 6.21.4 Conclusion Finale

Ce projet, dans sa version actuelle, représente une solide base pour l'étude et l'implémentation des tests de primalité. Les améliorations futures décrites dans ce chapitre permettraient non seulement d'étendre ses capacités, mais aussi d'en faire un outil de référence tant pour l'éducation que pour la recherche en théorie des nombres et en cryptographie.

La voie est ouverte vers la création d'un système complet qui combinerait la rapidité des tests probabilistes avec la certitude des tests déterministes, le tout dans une interface moderne et intuitive.

# Annexe

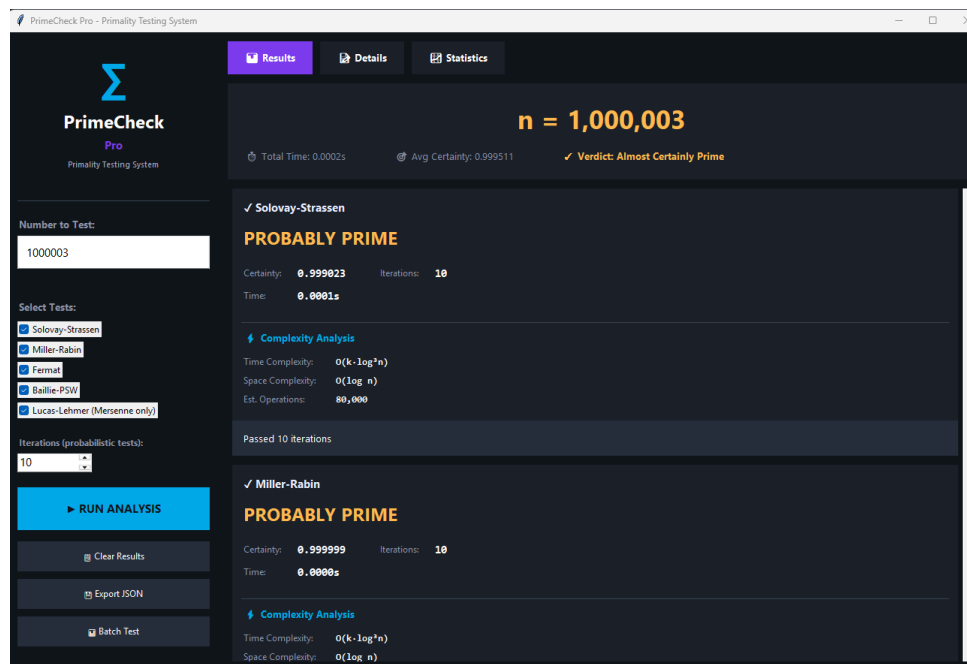


FIGURE 6.1 – Execution Miller-Rabin et Solvay-Strassen

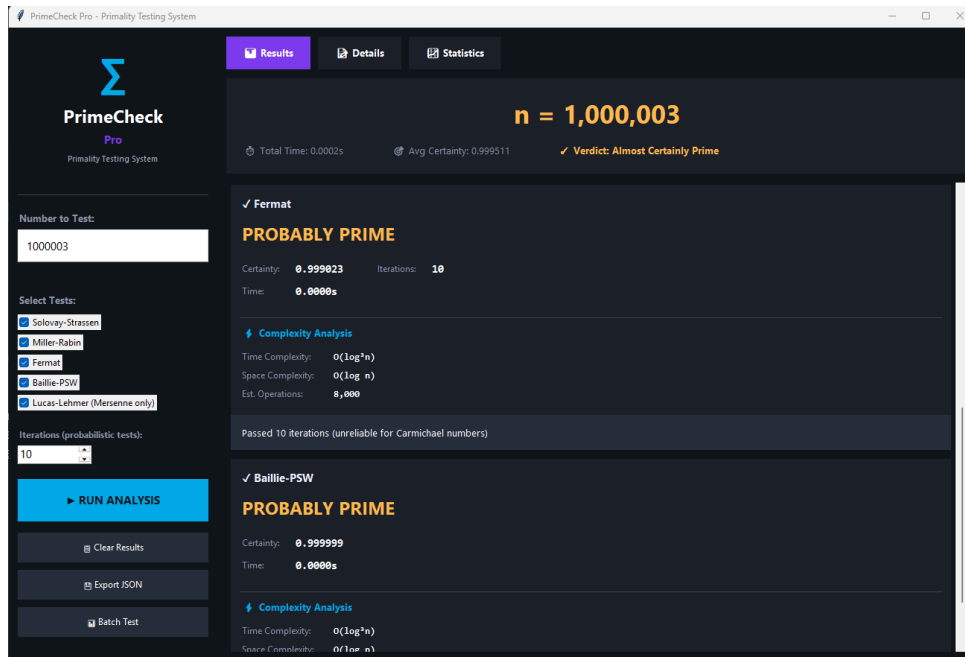


FIGURE 6.2 – Execution Fermat et Baillies-PSW

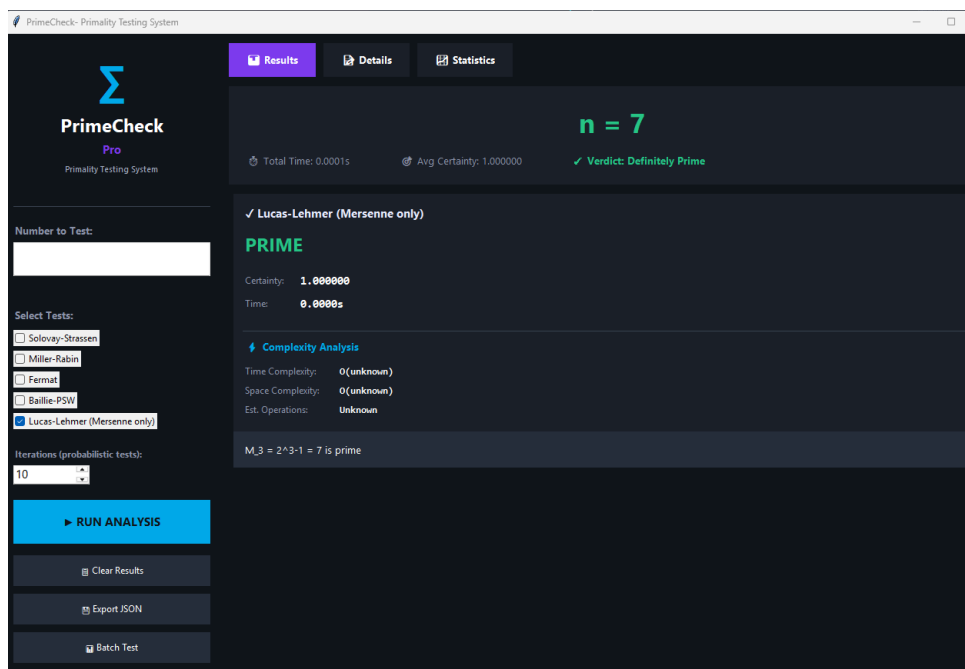


FIGURE 6.3 – Execution Lucas-lehmer

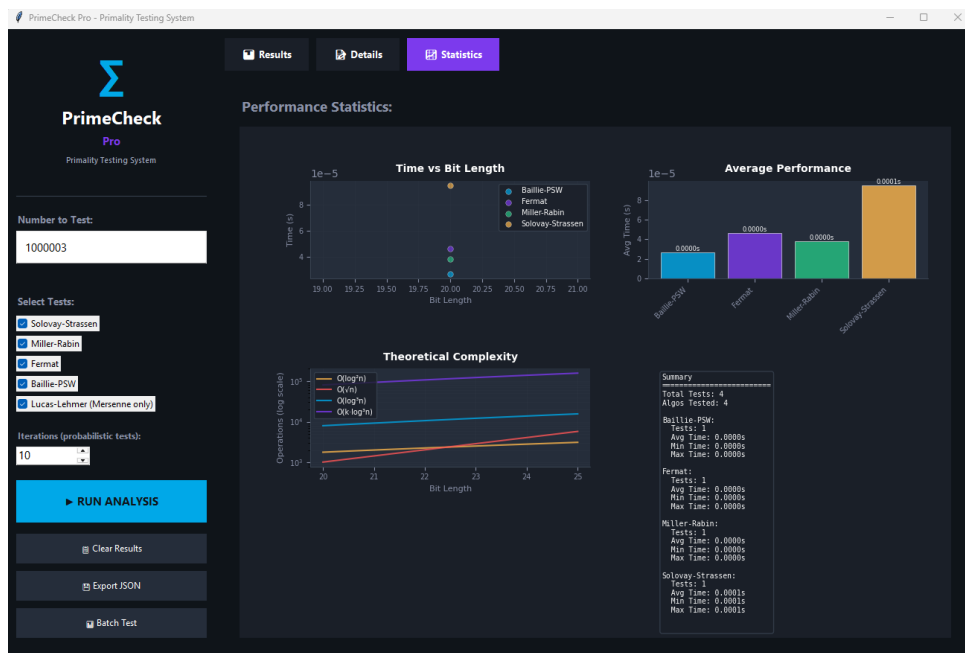


FIGURE 6.4 – Statistique sur les tests du primalité

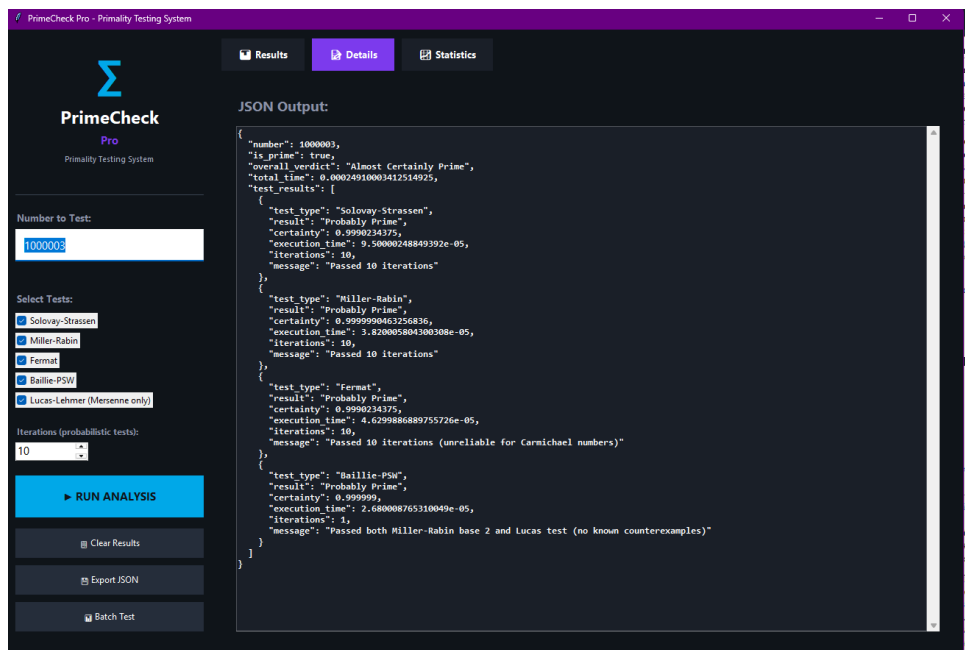


FIGURE 6.5 – Donnée JSON pour les sauvegarder

# Conclusion

En conclusion, ce projet a permis de réaliser des avancées significatives dans la conception et l'implémentation d'un système complet de tests de primalité. L'objectif principal — développer une plateforme unifiée intégrant les principaux algorithmes de tests de primalité avec une interface graphique moderne — a été atteint avec succès. Le système final offre non seulement une implémentation robuste des méthodes classiques (Solovay-Strassen, Miller-Rabin, Fermat, Baillie-PSW et Lucas-Lehmer), mais aussi une interface intuitive permettant aux utilisateurs de comparer et d'analyser les performances de ces différents algorithmes.

Le parcours de développement a inclus plusieurs phases essentielles : l'étude approfondie des fondements mathématiques de chaque test, l'implémentation rigoureuse des algorithmes en Python, et la création d'une interface graphique avec Tkinter permettant une interaction fluide et pédagogique. Chaque étape a contribué à la robustesse du produit final et a validé les choix techniques effectués tout au long du projet.

Les résultats obtenus confirment la pertinence des approches adoptées. Les tests probabilistes démontrent leur efficacité pratique, avec des temps d'exécution raisonnables même pour des nombres de grande taille, tout en respectant les bornes théoriques de probabilité d'erreur. L'interface graphique, avec ses visualisations claires et ses outils d'analyse, transforme des concepts mathématiques abstraits en expériences interactives accessibles.

Cependant, ce projet a également mis en lumière certaines limitations inhérentes aux méthodes probabilistes. L'impossibilité d'atteindre une certitude absolue — sauf dans le cas spécifique des nombres de Mersenne avec Lucas-Lehmer — rappelle que la primalité reste un problème complexe nécessitant parfois des compromis entre rapidité et certitude. Ces limitations ouvrent la voie à des améliorations futures, notamment l'intégration de tests déterministes comme AKS ou ECPP.

L'un des enseignements les plus marquants de ce travail est l'importance cruciale des fondements mathématiques dans le développement d'outils cryptographiques fiables. La compréhension profonde des propriétés algébriques sous-jacentes — symboles de Jacobi, réciprocity quadratique, racines carrées modulo un nombre premier — s'est révélée indispensable non seulement pour l'implémentation correcte des algorithmes, mais aussi pour leur analyse et leur optimisation.

Ce projet constitue également une contribution pédagogique précieuse. En rendant visibles et interactifs des algorithmes habituellement confinés à des articles scientifiques ou à des bibliothèques logicielles, il facilite l'apprentissage de concepts fondamentaux en théorie des nombres et en cryptographie. L'interface permet aux étudiants et aux chercheurs de visualiser concrètement les différences entre les méthodes, l'impact du nombre d'itérations sur la certitude, et les performances comparées des algorithmes.

En résumé, ce travail représente une synthèse réussie entre recherche théorique et développement pratique. Il démontre comment des concepts mathématiques avancés peuvent être transformés en outils logiciels utiles et accessibles. Bien que des améliorations soient toujours possibles — notamment l'ajout de tests déterministes et l'optimisation des performances pour les très grands nombres — le système actuel constitue une base solide pour l'étude et l'expérimentation des tests de primalité.

Finalement, ce projet confirme que les tests de primalité, au-delà de leur utilité pratique en cryptographie, représentent un domaine fascinant où se rencontrent beauté mathématique, ingénierie algorithmique et applications concrètes. Ils illustrent parfaitement comment des recherches théoriques peuvent déboucher sur des technologies essentielles à la sécurité numérique moderne.