A toy implementation of ECPP in Sage

Yannick Méheut

Agence Nationale de la Sécurité des Systèmes d'Informations
Laboratoire de Cryptographie

September 19, 2013

Plan

1 Introduction

2 Proving a number's primality
- Compositeness tests
- Primality tests

3 Author's implementation
- Objectives
- Specifications
- Encountered problems
- Demonstration

4 Conclusion
- Results
- Summary

Presentation of the internship

Origin of the internship

Context Prime numbers are intensively used in asymmetric cryptography (encryption, signature, ...). The ECPP algorithm is, for large numbers, the fastest way to prove a number's primality.
Observation There's no free (as in speech), efficient implementation of this algorithm.
Goal Study, implement and improve the ECPP algorithm.

Supervisors Jean-Pierre Flori, Jérôme Plût, Jean-René Reinhard
Internship duration 6 months (March 25 → September 25)

Plan

1. Introduction
2. Proving a number's primality
   - Compositeness tests
   - Primality tests
3. Author's implementation
   - Objectives
   - Specifications
   - Encountered problems
   - Demonstration
4. Conclusion
   - Results
   - Summary

Differences between compositeness and primality tests

Compositeness tests

Principle

$\mathcal{P}(N) \implies N$ composite

Properties

- Very fast: $\tilde{O}(\log^2 N)$
- We're not sure the number is prime

Example : Miller-Rabin test

Primality tests

Principle

$\mathcal{P}(N) \implies N$ prime

Properties

- Slower: $\tilde{O}(\log^5 N)$
- A number declared prime is inevitably so

Examples : $N-1$, ECPP

Differences between compositeness and primality tests

Compositeness tests

Principle

$\mathcal{P}(N) \implies N$ composite

Properties

- Very fast: $\tilde{O}(\log^2 N)$
- We're not sure the number is prime

Example : Miller-Rabin test

Primality tests

Principle

$\mathcal{P}(N) \implies N$ prime

Properties

- Slower: $\tilde{O}(\log^5 N)$
- A number declared prime is inevitably so

Examples : $N-1$, ECPP

The $N - 1$ primality test

Theorem (Lucas – 1876)

Let $N \in \mathbb{N}$*. Let* $g \in (\mathbb{Z}/N\mathbb{Z})^\times$*. If*

- $g^{N-1} \equiv 1 \bmod N$ *;*
- *for every prime* p *dividing* $N - 1$*,* $g^{\frac{N-1}{p}} \not\equiv 1 \bmod N$ *;*

then,

- g *has a multiplicative order of* $N - 1$ *in* $(\mathbb{Z}/N\mathbb{Z})^\times$ *;*
- *the cardinality of the multiplicative group is therefore* $N - 1$ *;*
- N *is therefore a prime number.*

The $N-1$ primality test

Theorem (Lucas – 1876)

Let $N \in \mathbb{N}$. *Let* $g \in (\mathbb{Z}/N\mathbb{Z})^\times$. *If*

- $g^{N-1} \equiv 1 \bmod N$;
- *for every prime* p *dividing* $N-1$, $g^{\frac{N-1}{p}} \not\equiv 1 \bmod N$;

then,

- g *has a multiplicative order of* $N-1$ *in* $(\mathbb{Z}/N\mathbb{Z})^\times$;
- *the cardinality of the multiplicative group is therefore* $N-1$;
- N *is therefore a prime number.*

The Pocklington theorem allows us to verify the above theorem for a single, large enough, divisor p.

Probable factorization

Definition (Smooth number)

A number $s \in \mathbb{N}$ is said B-**smooth** if, and only if none of its prime factors is greater than B.

Definition (Probably factored number)

A number $m \in \mathbb{N}$ is said **probably factored** if, and only if it can be written as:

$$m = s \times N'$$

where s is a B-smooth number for some B, and N' is probably prime.

Practical implementation of the $N-1$ test

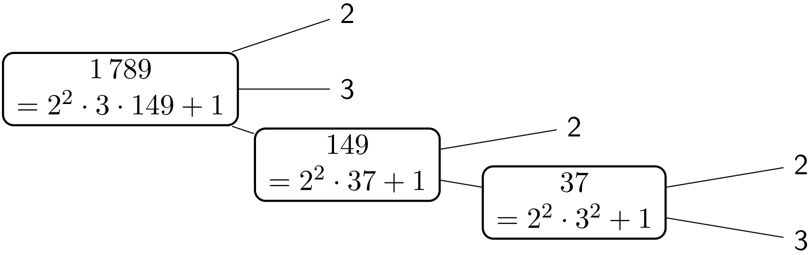
In practice

We want $N-1$ to be probably factored. We then check the previous theorem for p dividing s, then for $p = N'$.

Then, we have to check whether N' is indeed prime.

Creation of a certificate

Smoothness bound: $B = 3$



Original idea behind ECPP

Theorem (Goldwasser-Killian – 1986)

Let $N \in \mathbb{N}$. Let $\mathcal{E}$ be an elliptic curve over $\mathbb{Z}/N\mathbb{Z}$, $G \in \mathcal{E}$ and $N - 2\sqrt{N} \leq m \leq N + 2\sqrt{N}$ an integer. If

- $[m]G = \mathcal{O}_{\mathcal{E}}$;
- *for every prime p dividing m,* $\left[\frac{m}{p}\right] G \neq \mathcal{O}_{\mathcal{E}}$;

then G has an order of m in $\mathcal{E}$. The cardinality of the curve is therefore m, and N is prime.

Original idea behind ECPP

Theorem (Goldwasser-Killian – 1986)

Let $N \in \mathbb{N}$. Let $\mathcal{E}$ be an elliptic curve over $\mathbb{Z}/N\mathbb{Z}$, $G \in \mathcal{E}$ and $N - 2\sqrt{N} \leq m \leq N + 2\sqrt{N}$ an integer. If

- $[m]G = \mathcal{O}_{\mathcal{E}}$;
- *for every prime p dividing m,* $\left[\frac{m}{p}\right] G \neq \mathcal{O}_{\mathcal{E}}$;

then G has an order of m in $\mathcal{E}$. The cardinality of the curve is therefore m, and N is prime.

There is a version of this thereom analog to the Pocklington's theorem.

ECPP: first version (Goldwasser-Killian)

repeat
    Generate an elliptic curve $\mathcal{E}$ and compute its cardinality m over $\mathbb{Z}/N\mathbb{Z}$.
until m is probably factored
Write $m = s \times N'$ where s is smooth and N' is probably prime.
if we find G verifying the conditions of the Goldwasser-Killian's theorem **then**
    N is prime only if N' is. We verify the primality of N' by the same way.
else
    N is composite.
end if

Possible improvement

Computing the cardinality of an elliptic curve: Schoof's algorithm
Complexity of $O(\log^5 N) \implies$ ECPP's complexity is $O(\log^6 N)$.

Possible improvement

Computing the cardinality of an elliptic curve: Schoof's algorithm
Complexity of $O(\log^5 N) \implies$ ECPP's complexity is $O(\log^6 N)$.

Using complex multiplication (Atkin-Morain)

- allows us to construct elliptic curves over $\mathbb{Z}/N\mathbb{Z}$ with predictible cardinalities.
- possible using parameters D compatible with N: $4N = A^2 + DB^2$ with $A, B \in \mathbb{Z}$
- we can construct an elliptic curve with cardinality $m = N + 1 - A$

Idea: we try different D until we find a probably factored m
$\implies$ ECPP's complexity is $\tilde{O}(\log^5 N)$.

ECPP: second version (Atkin-Morain)

1: $k \leftarrow 0; N_0 \leftarrow N$

2: **while** $N_k > B$ **do**
3: **for** $-D_j$ fundamental discriminant **do**
4: **if** $4N_k = A_j^2 + D_j B_j^2$ **and** $m_j = N_k + 1 - A_j$ is probably factored **then**
5: Write $m_j = s \times N'$
6: $D^k \leftarrow D_j; m^k \leftarrow m_j$
7: Stock $\{k, N_k, D^k, m^k\}$
8: $k \leftarrow k + 1; N_k \leftarrow N'$
9: Return to step 2
10: **end if**
11: **end for**
12: **end while**
13: **for** i=k-1 **to** 0 **do**
14: Compute the equation of the elliptic curve $\mathcal{E}_i$, of cardinality m^i
15: **if** we find G verifying the conditions of the Goldwasser-Killian's theorem **then**
16: Continue $\{N_i$ is prime$\}$
17: **else**
18: $k \leftarrow i - 1$
19: Return to step 2 $\{N_i$ is composite$\}$
20: **end if**
21: **end for**

ECPP: second version (Atkin-Morain)

1: $k \leftarrow 0; N_0 \leftarrow N$

2: **while** $N_k > B$ **do**
3:     **for** $-D_j$ fundamental discriminant **do**
4:         **if** $4N_k = A_j^2 + D_j B_j^2$ **and** $m_j = N_k + 1 - A_j$ is probably factored **then**
5:             Write $m_j = s \times N'$
6:             $D^k \leftarrow D_j; m^k \leftarrow m_j$
7:             Stock $\{k, N_k, D^k, m^k\}$
8:             $k \leftarrow k + 1; N_k \leftarrow N'$
9:             Return to step 2
10:         **end if**
11:     **end for**
12: **end while**

Downrun phase

13: **for** i=k-1 **to** 0 **do**
14:     Compute the equation of the elliptic curve $\mathcal{E}_i$, of cardinality m^i
15:     **if** we find G verifying the conditions of the Goldwasser-Killian's theorem **then**
16:         Continue $\{N_i$ is prime$\}$
17:     **else**
18:         $k \leftarrow i - 1$
19:         Return to step 2 $\{N_i$ is composite$\}$
20:     **end if**
21: **end for**

Proof phase

Plan

1. Introduction
2. Proving a number's primality
   - Compositeness tests
   - Primality tests
3. Author's implementation
   - Objectives
   - Specifications
   - Encountered problems
   - Demonstration
4. Conclusion
   - Results
   - Summary

Objectives of the implementation

- Use of the Sage mathematical library (Python)
- Compatibility with the closed-source program Primo
- Modularity of the implementation
- Parallelization of the implementation

Some numbers

9 modified files

- `module_list.py`
- `rings/arith.py`
- `schemes/elliptic_curves/constructor.py`
- `libs/pari/decl.pxi`
- `libs/pari/gen.pxd`
- `quadratic_forms/quadratic_form__local_representation_conditions.py`
- `rings/finite_rings/integer_mod.pxd`
- `rings/finite_rings/integer_mod.pyx`
- `rings/integer.pyx`

Some numbers

9 added files

- sage/quadratic_forms/first_fundamental_discriminants.py
- sage/quadratic_forms/fundamental_discriminants_generator.py
- sage/quadratic_forms/quadratic_form__fundamental_discriminants.pyx
- sage/rings/elliptic_curve_primality_proving.py
- sage/rings/polynomial/polynomial_root_finding.pyx
- sage/schemes/elliptic_curves/basic_elliptic_curve.pxd
- sage/schemes/elliptic_curves/basic_elliptic_curve.pyx
- sage/schemes/elliptic_curves/basic_point.pxd
- sage/schemes/elliptic_curves/basic_point.pyx

2,614 lines

Dependencies

Patches

- Trac #14817: efficient copy of Pari's objects
- Trac #14818: access to Pari finite field functions
- Trac #14832: construction of polynomials over finite fields
- Trac #14833: construction of finite fields using irreducible polynomials
- Trac #12142: speed up Pari finite field operations
- Trac #11802: generation of Lucas sequences modulo an integer

Libraries

- GMP
- Pari
- CM

Dependencies

Patches

- Trac #14817: efficient copy of Pari's objects ✓
- Trac #14818: access to Pari finite field functions ✓
- Trac #14832: construction of polynomials over finite fields ✓
- Trac #14833: construction of finite fields using irreducible polynomials ✓
- Trac #12142: speed up Pari finite field operations ✓
- Trac #11802: generation of Lucas sequences modulo an integer

Libraries

- GMP
- Pari
- CM

Dependencies

Patches

- Trac #14817: efficient copy of Pari's objects ✓
- Trac #14818: access to Pari finite field functions ✓
- Trac #14832: construction of polynomials over finite fields ✓
- Trac #14833: construction of finite fields using irreducible polynomials ✓
- Trac #12142: speed up Pari finite field operations ✓
- Trac #11802: generation of Lucas sequences modulo an integer

Libraries

- GMP
- Pari
- CM *(modified)*

Cornacchia's algorithm

Solve $N = A^2 + DB^2$ *with* N, D *coprime*

1: Find x_0 a solution of $x^2 \equiv -D \bmod N$ with $N > x_0 > \frac{N}{2}$
2: Develop $\frac{N}{x_0}$ as a continued fraction:

$$\begin{aligned} N &= q_0 x_0 + q_1 \\ x_0 &= q_1 x_1 + q_2 \\ &\vdots \\ x_r &= q_{r+1} x_{r+1} + q_{r+2} \end{aligned}$$

and stop when $x_r^2 < N \leq x_{r-1}^2$
3: Let $A = x_r$ and $B = \sqrt{\frac{N - x_r^2}{D}}$
4: **if** B is not an integer **then**
5: N can't be written as $A^2 + DB^2$
6: **else**
7: Return (A, B)
8: **end if**

Cornacchia's algorithm

Solve $N = A^2 + DB^2$ *with* N, D *coprime*

1: Find x_0 a solution of $x^2 \equiv -D \bmod N$ with $N > x_0 > \frac{N}{2}$ — Modular square root

2: Develop $\frac{N}{x_0}$ as a continued fraction:

$$\left.\begin{aligned} N &= q_0 x_0 + q_1 \\ x_0 &= q_1 x_1 + q_2 \\ &\vdots \\ x_r &= q_{r+1} x_{r+1} + q_{r+2} \end{aligned}\right\} \text{Half Euclid's algorithm}$$

and stop when $x_r^2 < N \leq x_{r-1}^2$

3: Let $A = x_r$ and $B = \sqrt{\frac{N - x_r^2}{D}}$

4: **if** B is not an integer **then**

5: $\quad N$ can't be written as $A^2 + DB^2$

6: **else**

7: $\quad$ Return (A, B)

8: **end if**

Cornacchia's algorithm

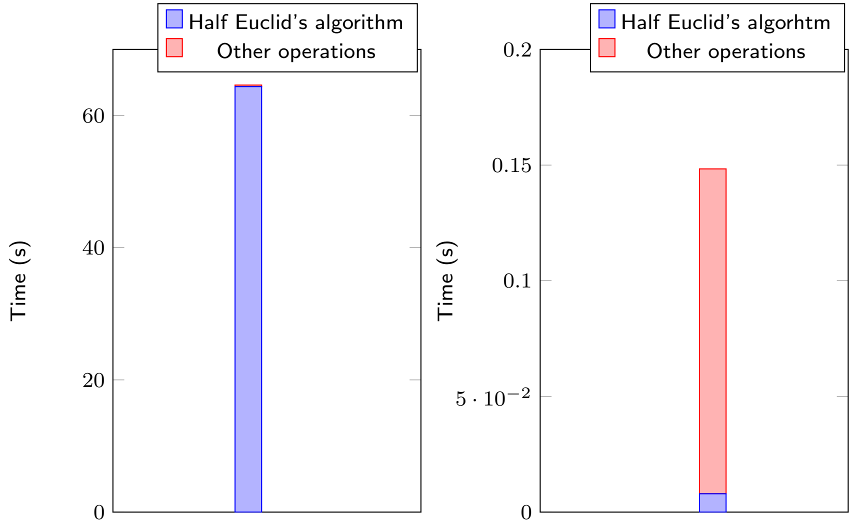
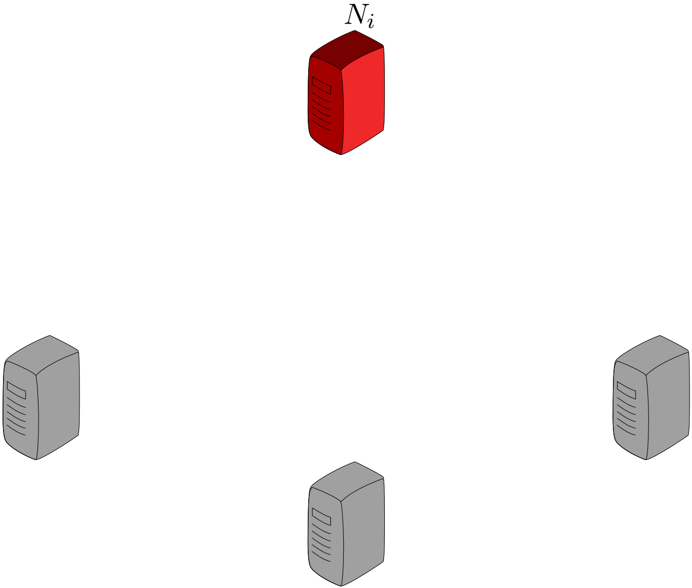
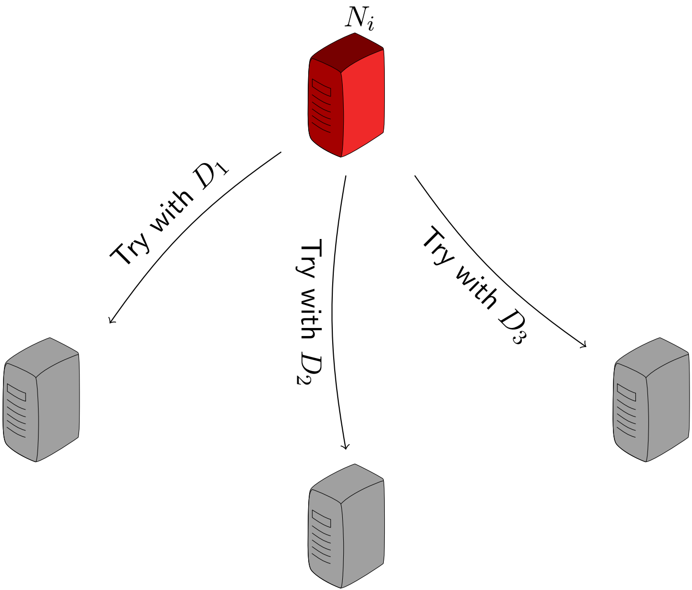


Figure: Time spent in Cornacchia's algorithm during the downrun phase: Python version (left) and Cython version (right)

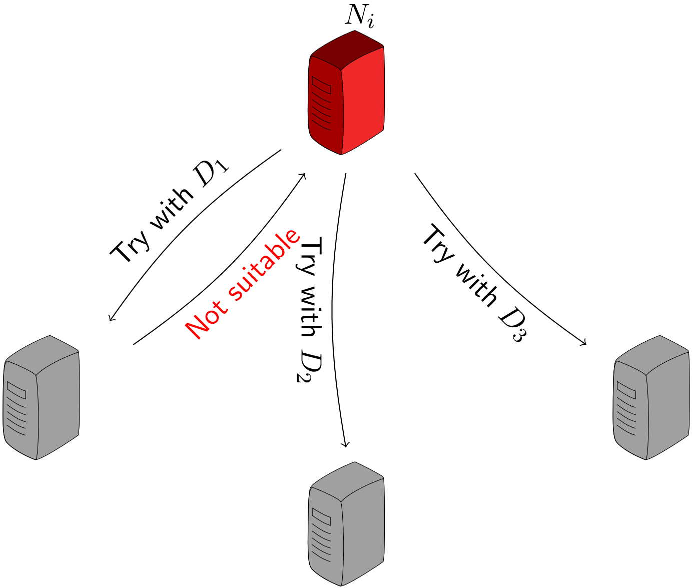
Parallelization



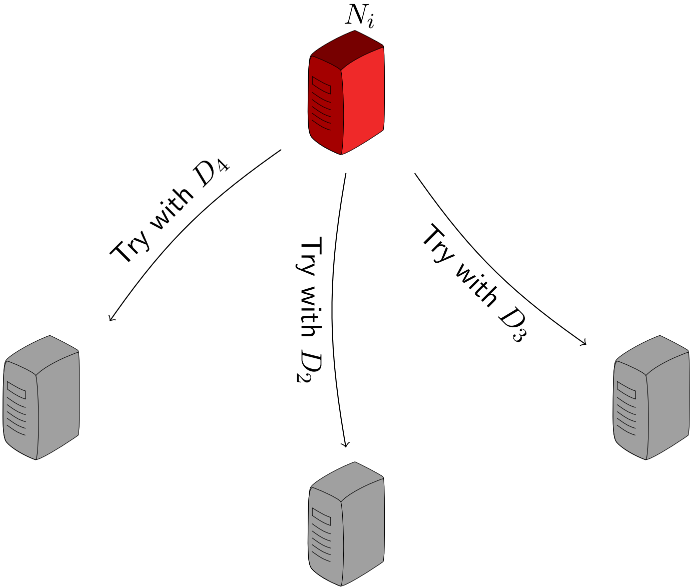
Parallelization



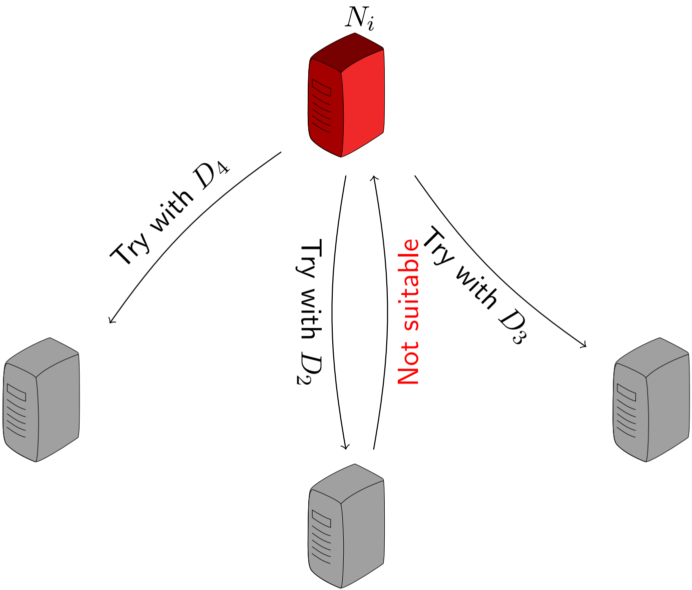
Parallelization



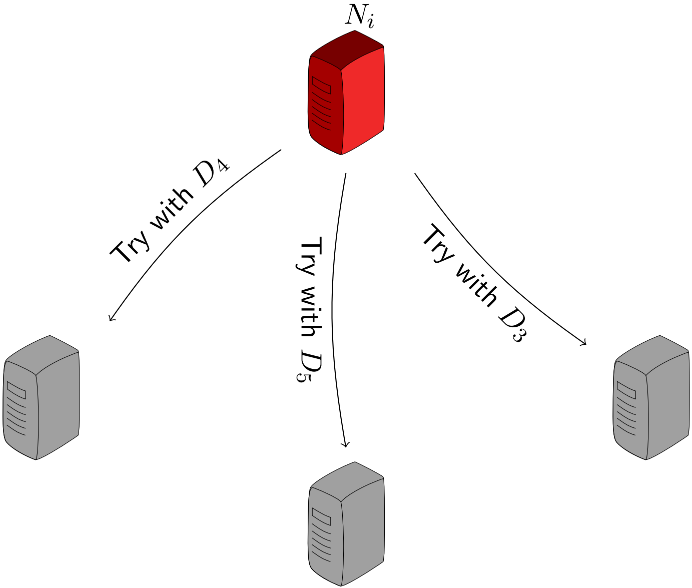
Parallelization



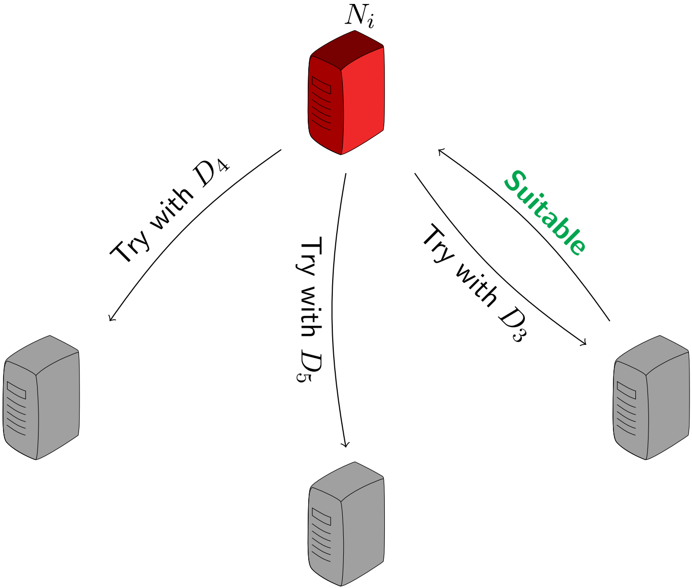
Parallelization



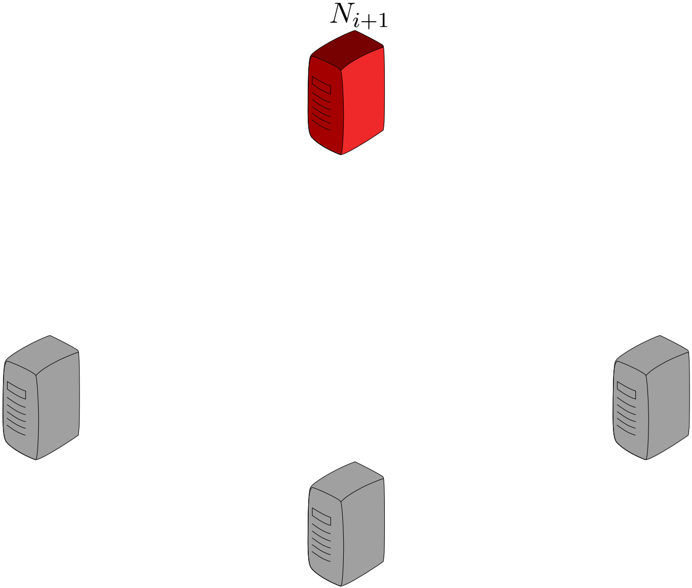
Parallelization



Parallelization



Parallelization

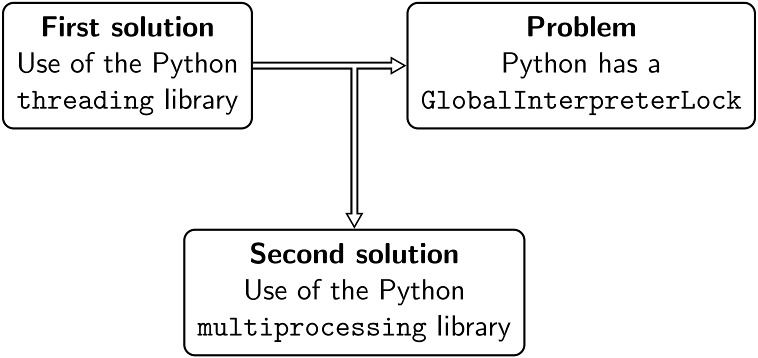


Parallelization

First solution

Use of the Python `threading` library

Parallelization



Sage verifies the primality of (almost) every parameter

Necessary operations

Building elliptic curves over $\mathbb{Z}/N\mathbb{Z}$
Computing square roots modulo N $\Longrightarrow$ Building $\mathbb{Z}/N\mathbb{Z}$

Sage verifies the primality of (almost) every parameter

Necessary operations

Building elliptic curves over $\mathbb{Z}/N\mathbb{Z}$
Computing square roots modulo N $\Longrightarrow$ Building $\mathbb{Z}/N\mathbb{Z}$

⇨ Sage will prove the primality of N

Solution: calling C libraries, creating a `BasicEllipticCurve` class

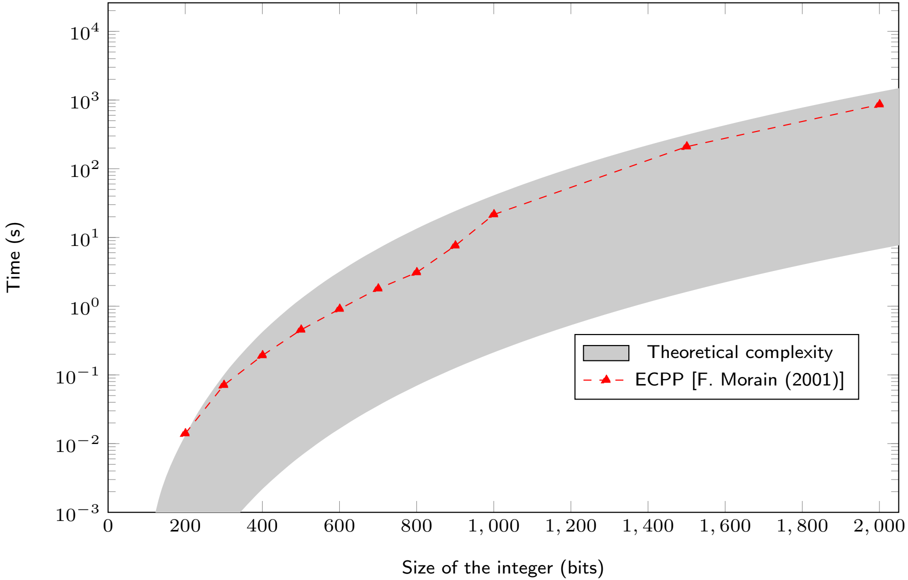
Demonstration

Demonstration of Sage ECPP

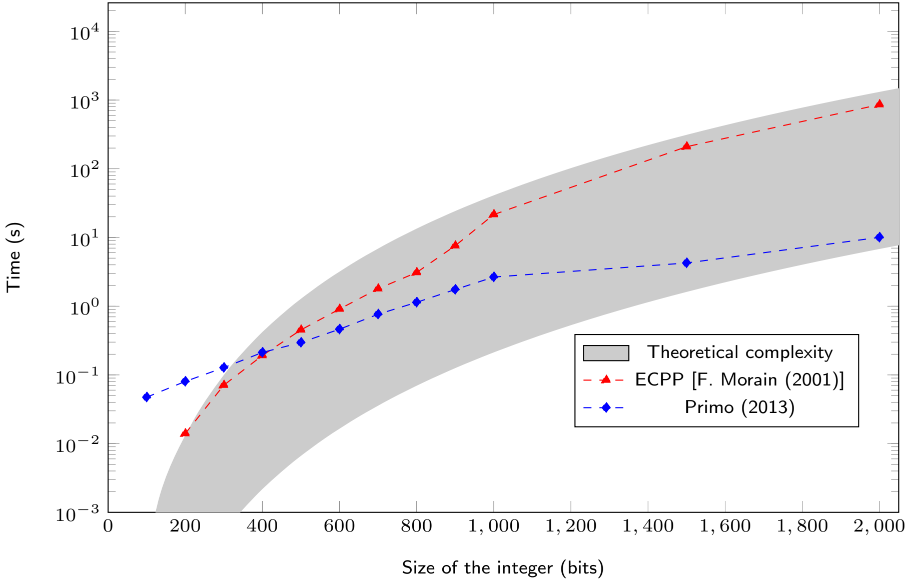
Plan

1. Introduction
2. Proving a number's primality
   - Compositeness tests
   - Primality tests
3. Author's implementation
   - Objectives
   - Specifications
   - Encountered problems
   - Demonstration
4. Conclusion
   - Results
   - Summary

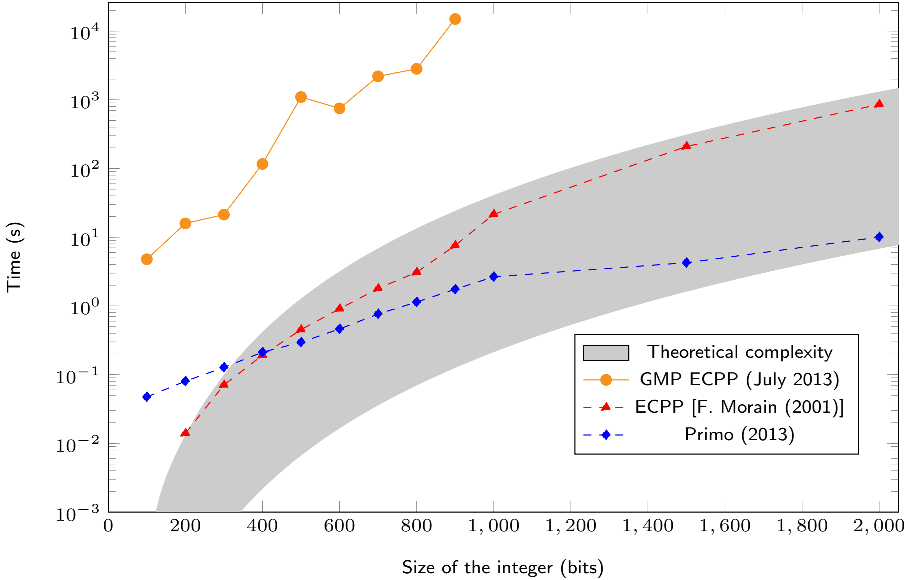
Comparison with existing implementations



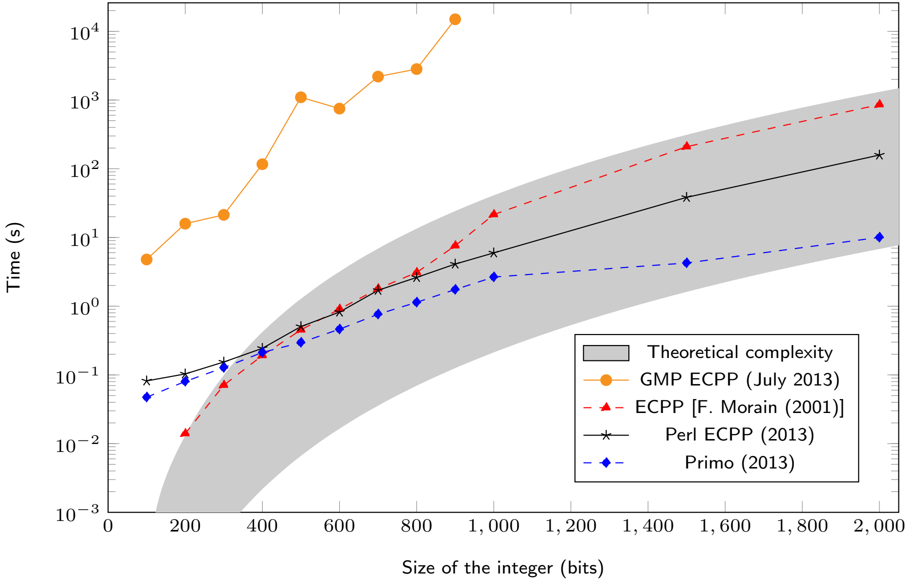
Comparison with existing implementations



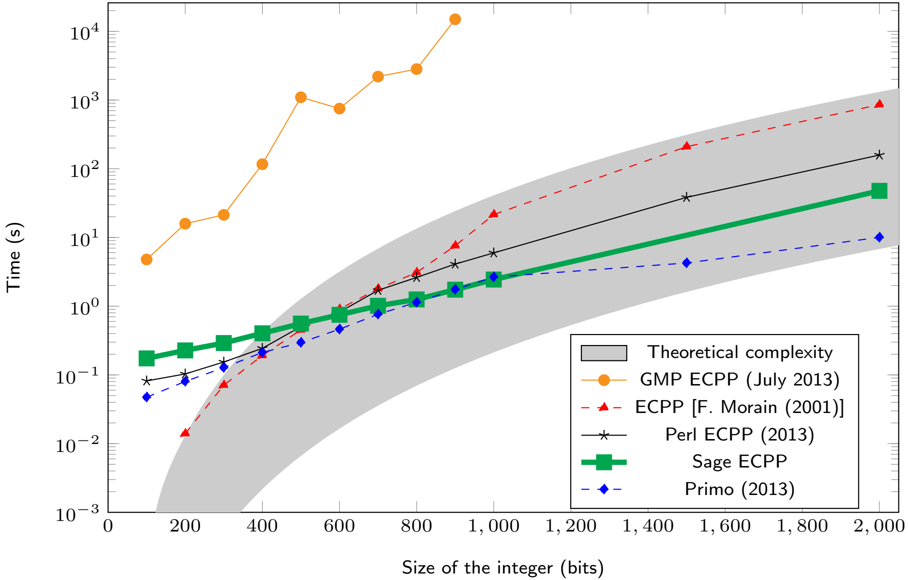
Comparison with existing implementations



Comparison with existing implementations



Comparison with existing implementations



Summary of the internship

Contributions

- Free, modular implementation of the ECPP algorithm
- The most efficient free implementations (parallelization, rewriting in Cython of critical code, calling of C libraries)
- Compatibility with the closed-source program Primo
- Integration in Sage **(work in progress)**

Summary of the internship

Contributions

- Free, modular implementation of the ECPP algorithm
- The most efficient free implementations (parallelization, rewriting in Cython of critical code, calling of C libraries)
- Compatibility with the closed-source program Primo
- Integration in Sage **(work in progress)**

Possible improvements

- Implementation of FastECPP
- Modifying the complex multiplication method
- Using a network of machines
- Port to C