Efficient Quaternion Algorithms for the Deuring Correspondence and Applications to the Evaluation of Modular Polynomials

Antonin Leroux

ANTS XVII

IRMAR, Université de Rennes, France

The effective Deuring correspondence in a few words

Deuring's Theorem:

$E/\mathbb{F}_{p^2}$ supersingular $\Leftrightarrow$ $\mathrm{End}(E)$ is a maximal order in $B_{p,\infty}$.

E, E' supersingular $\Leftrightarrow$ $\mathrm{Hom}(E, E')$ is an ideal of max. orders in $B_{p,\infty}$.

We can exploit this result to navigate the supersingular isogeny graph with efficient quaternion operations.

The effective Deuring correspondence in a few words

Deuring's Theorem:

$E/\mathbb{F}_{p^2}$ supersingular $\Leftrightarrow$ $\mathrm{End}(E)$ is a maximal order in $B_{p,\infty}$.

E, E' supersingular $\Leftrightarrow$ $\mathrm{Hom}(E, E')$ is an ideal of max. orders in $B_{p,\infty}$.

The effective Deuring correspondence in a few words

Deuring's Theorem:

$E/\mathbb{F}_{p^2}$ supersingular $\Leftrightarrow$ $\mathrm{End}(E)$ is a maximal order in $B_{p,\infty}$.

E, E' supersingular $\Leftrightarrow$ $\mathrm{Hom}(E, E')$ is an ideal of max. orders in $B_{p,\infty}$.

We can exploit this result to navigate the supersingular isogeny graph with efficient quaternion operations.

Some applications

Cryptography

The knowledge of $\mathrm{End}(E)$ becomes a trapdoor enabling the use of the Deuring correspondence.

This idea is behind **SQIsign**, the most compact post-quantum signature scheme.

Some applications

Cryptography

The knowledge of $\mathrm{End}(E)$ becomes a trapdoor enabling the use of the Deuring correspondence.
This idea is behind **SQIsign**, the most compact post-quantum signature scheme.

Algorithmic Number Theory

The Deuring correspondence allows to compute efficiently j-invariants with specific properties.
This led to recent algorithms to compute and evaluate Hilbert Class polynomial, and modular polynomials [**L**23, CE**L**MP25].

So how does it work?

Easy! It's just a bunch of linear algebra and quadratic equation resolution, right?

So how does it work?

Easy! It's just a bunch of linear algebra and quadratic equation resolution, right?

In theory: yes! But in **practice**?

Cryptography and high performance algorithms require precision.

Back to the applications: SQIsign

Quaternions are defined over $\mathbb{Q}$ with no obvious upper-bound.

The NIST submission implementation:

1. based on `GMP` to handle big integers
2. uses generic algorithms to handle all the quaternion arithmetic

As a consequence:

1. very big integers involved (thousands of bits)
2. some obstacles to getting a constant-time implementation
3. a cost that is becoming less and less negligible as other parts of the scheme are improved.

Back to the applications: modular polynomials

The C++ implementation of one of the algorithms from [CE**L**MP25]:

1. based on `NTL`
2. initially used generic algorithms which were later improved with some of the algorithms presented in this work.

Back to the applications: modular polynomials

The C++ implementation of one of the algorithms from [CE**L**MP25]:

1. based on `NTL`
2. initially used generic algorithms which were later improved with some of the algorithms presented in this work.

The quaternion operations turned out to be **by far** the bottleneck. The practical performances were not as good as predicted by the asymptotic complexity and not competitive at all with the state of the art.

Contributions

Simple and optimized algorithms to perform operations on the quaternion ideals stemming from the Deuring correspondence, with tight and optimal bounds on the integers involved in the computation.

Contributions

Simple and optimized algorithms to perform operations on the quaternion ideals stemming from the Deuring correspondence, with tight and optimal bounds on the integers involved in the computation.

An improved implementation of [CELMP25] replacing (almost everywhere) the use of `NTL:ZZ` by the native 64-bits `long` integers. Together with various other ideas: improved overall performances by a factor ≈ 20 for the largest levels.

New quaternion algorithms

Some notations

Fix $p = 3 \mod 4$[1].

The quaternion algebra is $B_{p,\infty} = \mathbb{Q}\langle 1, i, j, k\rangle$ with $i^2 = -1, j^2 = -p, ij = -ji = k$.

[1] This simplifies everything due to the prence of a squareroot of -1 in $B_{p,\infty}$

Some notations

Fix $p = 3 \mod 4$[1].

The quaternion algebra is $B_{p,\infty} = \mathbb{Q}\langle 1, i, j, k\rangle$ with $i^2 = -1, j^2 = -p, ij = -ji = k$.

We restrict to left $\mathcal{O}_0$-ideals where $\mathcal{O}_0 = \mathbb{Z}\langle 1, i, \frac{i+j}{2}, \frac{1+k}{2}\rangle$ is isomorphic to $\mathrm{End}(E_0)$, for E_0 the supersingular curve $y^2 = x^3 + x$ of j-invariant 1728.

[1] This simplifies everything due to the prence of a squareroot of -1 in $B_{p,\infty}$

The structural result

The main inspiration of the new algorithms is a structural result on left $\mathcal{O}_0$- ideals of prime norm.

Theorem
For an odd prime $N \neq p$: all ideals of norm N (except at most 2 ideals[2]) are of the form:

$$I = \mathbb{Z}\left\langle N, Ni, \frac{x + iy + j}{2}, i\frac{x + iy + j}{2} \right\rangle$$

When that's the case, we say that I is inert.

Each inert ideal I corresponds to a ***unique pair*** $0 \leq x, y < 2N$.

[2] corresponding to split $\mathbb{Z}[i]$-ideals

Fast basis computation

Goal: compute a basis of the ideal[3] $I = \mathcal{O}_0\alpha + \mathcal{O}_0 N$.

Generic method: linear elimination on the eight basis elements of $\mathcal{O}_0\alpha, \mathcal{O}_0 N$.

[3] all $\mathcal{O}_0$-ideals can be defined like this

Fast basis computation

Goal: compute a basis of the ideal[3] $I = \mathcal{O}_0\alpha + \mathcal{O}_0 N$.

Generic method: linear elimination on the eight basis elements of $\mathcal{O}_0\alpha, \mathcal{O}_0 N$.

New lemma: When I is inert, there always exists an $\alpha = a + ib + (c + id)j$ such that $c^2 + d^2 \wedge N = 1$.

Computation of x, y:

$$x + iy + j = 2\left(\frac{c - id}{c^2 + d^2}\alpha \mod N\mathcal{O}_0\right)$$

[3] all $\mathcal{O}_0$-ideals can be defined like this

Fast basis computation

Goal: compute a basis of the ideal[3] $I = \mathcal{O}_0\alpha + \mathcal{O}_0 N$.

Generic method: linear elimination on the eight basis elements of $\mathcal{O}_0\alpha, \mathcal{O}_0 N$.

New lemma: When I is inert, there always exists an $\alpha = a + ib + (c + id)j$ such that $c^2 + d^2 \wedge N = 1$.

Computation of x, y:

$$x + iy + j = 2\left(\frac{c - id}{c^2 + d^2}\alpha \mod N\mathcal{O}_0\right)$$

Requires only a few operations modulo N.

[3] all $\mathcal{O}_0$-ideals can be defined like this

Intersection for composite norms

Any I of norm $N_1 N_2$ with $N_1 \wedge N_2 = 1$ is $I_1 \cap I_2$ where $n(I_i) = N_i$.

Generic method: from the formula $\Lambda_1 \cap \Lambda_2 = (\Lambda_1^* + \Lambda_2^*)^*$

Intersection for composite norms

Any I of norm $N_1 N_2$ with $N_1 \wedge N_2 = 1$ is $I_1 \cap I_2$ where $n(I_i) = N_i$.

Generic method: from the formula $\Lambda_1 \cap \Lambda_2 = (\Lambda_1^* + \Lambda_2^*)^*$

New method with inert ideals: The intersection of two inert ideals is also inert, and we have

$$x = \mathrm{CRT}([x_1, x_2], [N_1, N_2])$$
$$y = \mathrm{CRT}([y_1, y_2], [N_1, N_2])$$

Right order computation

Lemma
Let I be an inert ideal defined with $I = \mathcal{O}_0\langle \alpha_I, N\rangle$, where $\alpha_I = \frac{x+iy+j}{2}$ then:

$$\mathcal{O}_R(I) = \frac{\bar{I} \cdot I}{N} = \mathbb{Z}\left\langle \frac{1+Nk}{2}, \frac{1+ai+bj}{2N}, Aj+ck, (N/A)k \right\rangle$$

where $A = x \wedge N$.

Right order computation

Lemma
Let I be an inert ideal defined with $I = \mathcal{O}_0\langle \alpha_I, N\rangle$, where $\alpha_I = \frac{x+iy+j}{2}$ then:

$$\mathcal{O}_R(I) = \frac{\bar{I} \cdot I}{N} = \mathbb{Z}\left\langle \frac{1+Nk}{2}, \frac{1+ai+bj}{2N}, Aj+ck, (N/A)k \right\rangle$$

where $A = x \wedge N$.

Computation of a, b, c from simple formulas, with only a few operations mod N^2.

Computation and Evaluation of modular polynomials

A quick overview of the CRT approach

Goal: compute $\Phi_\ell(X, Y)$ or $\Phi_\ell(X, j)$ over $\mathbb{Z}$ or $\mathbb{F}_q$.

1. Select a set of primes $p_1, \ldots, p_n$
2. For $m = 1$ to n:
   (2a) For all j^* in a set of well-chosen j-invariants over $\mathbb{F}_{p_m^k}$:
      i. Compute $j_1, \ldots j_{\ell+1}$ all ℓ-isogenous j-invariants to j^*
      ii. Interpolate $\Phi_\ell(j^*, X) = \prod_{k=1}^{\ell+1}(X - j_k) \mod p_m$
   (2b) Reconstruct the result from the $\Phi_\ell(j^*, X)$
3. Reconstruct the final result using CRT.

State of the art CRT approach

Paper		Type of curves	Time	Memory
[S13] (extends [BLS12])	Main	Ordinary	$O(\ell^{3+\varepsilon})$	$O(\ell^{2+\varepsilon})$
	Hybrid		$O(\ell^{3+\varepsilon^{++}})$	$O(\ell^{1+\varepsilon})$
[CK**L**MP25] (extends [**L**23])	BigChar	Supersingular	$O(\ell^{3+\varepsilon})$	$O(\ell^{1+\varepsilon})$
	BigLevel		$O(\ell^{2+\varepsilon})$	$O(\ell^{1+\varepsilon})$

Table 1: Comparison of the state of the art CRT methods[4]

[4] There is also an algorithm outlined in the preprint [R23] based on the deformation method from [KR24], but it was not never properly analyzed and implemented.

State of the art CRT approach

Paper		Type of curves	Time	Memory
[S13] (extends [BLS12])	Main	Ordinary	$O(\ell^{3+\varepsilon})$	$O(\ell^{2+\varepsilon})$
	Hybrid		$O(\ell^{3+\varepsilon^{++}})$	$O(\ell^{1+\varepsilon})$
[CK**L**MP25] (extends [**L**23])	BigChar	Supersingular	$O(\ell^{3+\varepsilon})$	$O(\ell^{1+\varepsilon})$
	BigLevel		$O(\ell^{2+\varepsilon})$	$O(\ell^{1+\varepsilon})$

Table 1: Comparison of the state of the art CRT methods[4]

Implementation result:

[Main, S13] > [BigChar, CK**L**MP25] > [BigLevel, CK**L**MP25]

[BigChar, CK**L**MP25] couldn't compete with [Main, S13].

Bottleneck: quaternion operations.

[4] There is also an algorithm outlined in the preprint [R23] based on the deformation method from [KR24], but it was not never properly analyzed and implemented.

A focus on the BigChar algorithm from [CKLMP25]

Main idea: minimize elliptic curve operations with the Deuring correspondence.

With $p_m = O(\ell)$, to compute $\Phi_\ell(X, j) \mod p_m$:

1. Compute $\mathrm{End}(E)$ for all supersingular $E/\mathbb{F}_{p_m^2}$ and store as a dictionnary: $O(\ell) = O(p_m)$
2. Use ideals of norm ℓ and the dictionnary to compute all necessary pairs of ℓ-isogenous j-invariants: $O(\ell^2)$.
3. Interpolate the result: $O(\ell^2)$

A focus on the BigChar algorithm from [CKLMP25]

Main idea: minimize elliptic curve operations with the Deuring correspondence.

With $p_m = O(\ell)$, to compute $\Phi_\ell(X, j) \mod p_m$:

1. Compute $\mathrm{End}(E)$ for all supersingular $E/\mathbb{F}_{p_m^2}$ and store as a dictionnary: $O(\ell) = O(p_m)$
2. Use ideals of norm ℓ and the dictionnary to compute all necessary pairs of ℓ-isogenous j-invariants: $O(\ell^2)$.
3. Interpolate the result: $O(\ell^2)$

Requires:

- Ideal arithmetic (Basis, intersection, right order computation)
- Invariants for maximal order type (Lattice reduction)

In more details

Assume the dictionnary $\mathcal{M} : E \leftrightarrow \mathrm{End}(E)$ is computed.

1. Compute $I_{0,0}, \ldots, I_{\ell+1,0}$, $\mathcal{O}_0$-inert ideals in distinct ideal classes.
2. Compute $J_1, \ldots, J_\ell + 1$, all the $\mathcal{O}_0$-ideals of norm ℓ.
3. For $i = 0$ to $\ell + 1$:

   (3a) Compute $\mathcal{O}_R(I_{i,0})$ and recover j_i from $\mathcal{M}$

   (3b) For $k = 1$ to $\ell + 1$:

      i. Compute $I_{i,k} = I_{i,0} \cap J_k$.

      ii. Compute $\mathcal{O}_R(I_{i,k})$ and recover $j_{i,k}$[5] from $\mathcal{M}$.

   (3c) Interpolate $\Phi_\ell(j_i, X) = \prod_{k=1}^{\ell+1}(X - j_{i,k})$

[5] subtlety to distinguish between $j_{i,k}$ and $j_{i,k}^{p^m}$

Improving the implementation with the new method

The C++ implementation from [CK**L**MP25] is based on the `NTL:ZZ` arbitrary-size integer library.

The tight control on the size of the integers given by the new algorithms allowed to use only the native `long` integers.

[6] which were partly included in [CK**L**MP25]

Improving the implementation with the new method

The C++ implementation from [CK**L**MP25] is based on the `NTL:ZZ` arbitrary-size integer library.

The tight control on the size of the integers given by the new algorithms allowed to use only the native `long` integers.

1. Estimated improvement of the new algorithms compared to the generic algorithms $\approx \times 4$ [6].
2. Improvement to quaternion part of BigChar: $\approx \times 30$.
3. Overall improvement to BigChar $\approx \times 20$

[6] which were partly included in [CK**L**MP25]

Final results

Table 2: Experimental data: CPU time consumed by runs of the Weber variant of our implementation of BigChar compared to the one of [CK**L**MP25] for various levels ℓ, fixing the characteristic $p = 2^{31} - 1$

Level ℓ	Ours	[CK**L**MP25]	Ratio
211	8.558 s	18.191 s	2.1
419	16.205 s	40.702 s	2.5
811	35.543 s	2.00 m	3.4
2003	2.05 m	15.13 m	7.4
4003	7.52 m	1.65 h	13.1
8011	34.40 m	12.13 h	21.15
11681	1.65 h	1.56 d	22.6

Analysis of performance for $\ell = 11681$

For $\ell = 11681$, our implementation took $1.65h$. The time was divided in three roughly equivalent parts:

- The computation of the dictionnary: $O(p_m) = O(\ell)$
- The quaternion operations: $O(\ell^2)$
- The interpolation step: $O(\ell^2)$

Comparison with [S13]

[S13] reports $\approx 2h$ for $\ell = 11681$ with bigger prime char.

$\Rightarrow$ the algorithm from [S13] is likely faster in practice than BigChar.

[7] which are 2 times bigger

Comparison with [S13]

[S13] reports $\approx 2h$ for $\ell = 11681$ with bigger prime char.
$\Rightarrow$ the algorithm from [S13] is likely faster in practice than BigChar.

Some elements of explanations:

- [S13] use $2x$ less CRT primes[7].
  While $p_m \ll 2^{64}$: no gain in using smaller primes.
- [S13] works only over $\mathbb{F}_p$: $\times 3$ improvement compared to $\mathbb{F}_{p^2}$ for us.
- The linear precomputation part of the dictionnary is still not negligible at $\ell = 11681$.

[7] which are 2 times bigger

Future work

Our improvement of the [CK**L**MP25] implementation proved that using appropriate algorithm to handle quaternions ideals can make a big difference in practice by using optimized algorithm with fixed-size integers.

Future work

Our improvement of the [CK**L**MP25] implementation proved that using appropriate algorithm to handle quaternions ideals can make a big difference in practice by using optimized algorithm with fixed-size integers.

The most interesting application of the Deuring correspondence remains cryptography and in particular the SQIsign signature algorithm.

This would require to remove the GMP dependency from SQIsign's code, and to generalize some of our algorithms to a slightly more generic setting (non $\mathcal{O}_0$-ideals).