Algorithmic Number Theory Symposium XVII, Groningen, July 6-10, 2026

EFFICIENT QUATERNION ALGORITHMS FOR THE DEURING CORRESPONDENCE AND APPLICATION TO THE EVALUATION OF MODULAR POLYNOMIALS

ANTONIN LEROUX

Abstract. This work presents several algorithms to perform operations in the quaternion ideals and orders stemming from the Deuring correspondence. While most of the desired operations can be done with generic linear algebra, we show that they can be performed much more efficiently while maintaining a strict control over the size of the integers involved. This allows us to obtain a very efficient implementation with fixed sized integers of the effective Deuring correspondence.

We apply our new algorithms to improve greatly the practical performances of a recent algorithm by Corte-Real Santos, Eriksen, Leroux, Meyer, and Panny to evaluate modular polynomials. Our new implementation, including several other improvements, runs 20 times faster than before for the level $\ell = 11681$.

The Deuring correspondence also plays a central role in the most recent developments in isogeny-based cryptography, and in particular in the SQIsign signature scheme submitted to the NIST PQC competition. After the latest progresses, it appears that fixed-sized efficient quaternion operations is one of the main missing feature of the most recent implementations of SQIsign. We believe that several of our new algorithms could be very useful for that.

1. Introduction

The Deuring correspondence links supersingular elliptic curves and isogenies in characteristic p, with orders and ideals of $\mathcal{B}_{p,\infty}$ the quaternion algebra ramified at p and ∞. This result has recently gained much attention due to recent developments in isogeny-based cryptography. At first, the correspondence was studied in the hope of gaining a better understanding of the supersingular isogeny graphs upon which the security of isogeny-based cryptography relies. However, following the discovery of the KLPT algorithm [KLPT14], the Deuring correspondence has quickly outgrown its original purpose in this context by becoming a powerful means to navigate efficiently on the isogeny graph by replacing expensive isogeny computations with more efficient quaternion computations, enabling several new interesting applications for cryptography. In particular, the SQIsign signature scheme introduced in [DKL+20] is the most compact post-quantum signature scheme, and it was recently submitted to the NIST competition for standardization. A lot of other protocols are now built upon the Deuring correspondence: [BDF+24, DLRW24, Ler25, BBC+25, NO24] to cite only a few. SQIsign's main drawback used to be the heavy amount of isogeny computation still required by its core algorithms. This cost was gradually reduced with several new variants of the scheme: [DFLLW23, DLRW24, BDF+24, NOC+24, AAA+25], and there is the promise to reduce this cost further with the recent Qlapoti algorithm from

[BCRSE+25]. With all these works, the relative weight of the quaternion operations has been increasing to a point where they cannot be considered negligible anymore. Moreover, quaternion operations are the main obstacle to getting a constant-time implementation of SQIsign, which is now starting to be a concern [JMKR23, BHJ+25, KMJK25, HKK+25]. One of the difficulties related to quaternions in this matter is that the integers involved are quite big. The official SQIsign implementation relies on the arbitrary sized integers from GMP [Gt12] for this reason, and there have been some recent efforts to get rid of that [KLKL25, KMJK25].

Recently, Leroux proposed an application of the Deuring correspondence outside of cryptography for computing modular and Hilbert polynomials [Ler23]. This paper introduced most of the high-level ideas but lacked an implementation. This was later corrected and applied to the evaluation of modular polynomials in [CRSEL+25] by Corte-Real Santos, Eriksen, Leroux, Meyer, and Panny. The initial motivation behind the algorithms from [Ler23] and the ModularEvaluationBigCharacteristic algorithm from [CRSEL+25] was to trade many expensive isogeny operations for supposedly cheaper quaternion computations, thanks to the Deuring correspondence. This leads to several new heuristic algorithms matching or improving upon the state of the art either in time or memory complexity for a variety of cases. However, the implementation in [CRSEL+25] showed that the practical cost of the quaternion operations is in fact not at all negligible, and it even turns out to be the main bottleneck in ModularEvaluationBigCharacteristic. The implementation of the quaternion operations in [CRSEL+25] is based on arbitrary sized integers of the NTL library [S+01], and this was pointed out in [CRSEL+25] as one of the main directions to improve the practical efficiency of the implementation.

Aside of this issue of integers' sizes, there is also the question of how to perform the basic quaternion operations required by the Deuring correspondence. Most of the time, the Deuring correspondence manipulates orders and ideals of $\mathcal{B}_{p,\infty}$, which are $\mathbb{Z}$-lattices of dimension 4, and the typical applications require one to perform several basic operations on these lattices such as basis computation, basis reduction, intersection, multiplication, or isomorphism computation. While all these operations can be done quite easily with generic linear algebra, we have explained already that their practical efficiency is becoming critical, and so it is relevant to find the most efficient methods to perform them.

1.1. **Contributions.** In this paper, we show that several basic operations on quaternion ideals and orders in $\mathcal{B}_{p,\infty}$ can be significantly sped-up in some well-chosen special cases by exploiting the very specific structure of these lattices. We make explicit a few simple algorithms that severely outperform the naive algorithms while avoiding blowing-up the size of the integers involved.

We illustrate how our algorithms can make a huge difference in practice by improving the C++ implementation of the ModularEvaluationBigCharacteristic algorithm from [CRSEL+25] to evaluate modular polynomials. In fact, some of our algorithms were already part of the latest version of the implementation provided in [CRSEL+25] but were not discussed at all in the paper.

As we believe these improvements are interesting in their own right, we correct this oversight with the current paper. We also benefit from the better control on integer size provided by our new algorithms to replace the arbitrary-sized `NTL::ZZ` integers used in the implementation from [CRSEL+25] by 64-bits `long` integers

up to something like $\ell \approx 20000$, which we believe would have been completely impossible with more generic algorithms.

We also incorporate various additional improvements to the implementation from [CRSEL+25] allowing us to reach a final constant factor of improvement of roughly 30 for the implementation of the quaternion part of ModularEvaluationBigCharacteristic. It is to be noted that this factor 30 does not include our first improvement of the original unpublished implementation from [CRSEL+25]. We estimate this initial improvement (which included most notably an implementation with `NTL:ZZ` of the Algorithms 2 and 3, the CRT idea to perform intersection that we present in Section 2.3, and a slower version of Algorithm 4) to be a factor of 3 or 4, thus bringing the final improvement factor closer to something like 100.

We also improved various other aspects of the implementation from [CRSEL+25] (mainly finite field arithmetic) to obtain an overall improvement factor of more than 20 at the level $\ell = 11681$ for the whole implementation of ModularEvaluationBigCharacteristic. Our implementation is available at:

`https://github.com/tonioecto/modular-polynomial-computation-and-evaluation-from-supersingular-curves`

1.2. **Related work.** Our improvements are mainly inspired by a close inspection of the structure of the bases of the integral ideals whose left order is the maximal order $\mathcal{O}_0 = \langle 1, i, (i+j)/2, (1+k)/2 \rangle$ (only for $p = 3 \mod 4$). This was already studied in a series of work [ACM25, Cle25] that attempted to characterize every left $\mathcal{O}_0$-ideal and their right order. We do not use their results directly, but these works can be seen as a generalization of the results we use.

As far as we are aware, our work is the first to try to use that structure to improve the concrete arithmetic of these quaternion ideals. The implementation of SQIsign from [AAA+25], which is one of the first attempts at a somewhat generic and efficient quaternion library for the Deuring correspondence, only uses generic algorithms (as we will explain throughout Section 2), and this was also the case for the original implementation from [CRSEL+25]. Moreover, several recent works [BHJ+25, KLKL25, KMJK25] that attempt to improve some aspects of SQIsign's ideal arithmetic (in particular HNF bases computation) do not use this structure. We believe that our algorithms could prove to be much more efficient while minimizing the size of the integers involved and being generally more constant-time friendly.

Regarding computation and evaluation of modular polynomials, there are several other CRT-based methods. Bröker, Lauter, and Sutherland [BLS12] proposed to use isogeny walks on volcanoes of ordinary elliptic curves. Their method was later refined by Sutherland in [Sut13] to evaluate modular polynomials. To this date, the implementations from [BLS12] and [Sut13] are the most efficient for computation and evaluation of modular polynomials which is why we mostly compare our implementation with the one from [Sut13]. There is also a recent method by Kunzweiler and Robert [KR24] which works by lifting efficient isogeny representations between supersingular elliptic curves to deformation of elliptic curves. Their method has the advantage of being provable without any heuristic or assumptions, but it does not appear competitive in practice with the methods from [Sut13] or [CRSEL+25].

1.3. **Acknowledgements.** We thank Sina Schaeffler and Mounir Idrassi for pointing out a few mistakes in a previous version of this paper.

1.4. **Outline of the paper.** In Section 2, we introduce our new efficient quaternion algorithms. Then, in Section 3, we describe a few additional noticeable improvements to the implementation of ModularEvaluationBigCharacteristic from [CRSEL+25] and report the result we obtain with our new implementation.

We cover some of the necessary background on quaternion algebras and the Deuring correspondence in the long version of the paper [Ler26].

2. Quaternion algorithms

Throughout this section, $\mathcal{B}_{p,\infty}$ will be the quaternion algebra ramified at p and ∞. The data structure used for quaternion elements consists of 5 integers: the four numerators of the coefficient of the element in the basis $\langle 1, i, j, k\rangle$ (that may be called the numerator vector) and the common denominator. Similarly, we consider that orders and ideals in $\mathcal{B}_{p,\infty}$ are given as a tuple made of a 4×4 integral matrix giving the numerator of the coefficients of the basis of the order or ideal in the basis $\langle 1, i, j, k\rangle$ (sometimes called the numerator matrix) together with an additional integer which is the common denominator of all the coefficients.

In this section, we introduce various algorithmic improvements to several operations in $\mathcal{B}_{p,\infty}$ required by the Deuring correspondence for concrete computations. Note that most of the time, our algorithms do not work in full generality, but require some specific constraint that can be satisfied almost all the time in our application to ModularEvaluationBigCharacteristic.

In particular, we are going to restrict ourselves to the case where the prime $p = 3 \mod 4$ (as ModularEvaluationBigCharacteristic is a CRT algorithm, it suffices to restrict the set of CRT primes to the primes $p = 3 \mod 4$). In that case, one of the maximal orders in $\mathcal{B}_{p,\infty}$ is $\mathcal{O}_0 := \mathbb{Z}\langle 1, i, \frac{i+j}{2}, \frac{1+k}{2}\rangle$. This maximal order is isomorphic to the endomorphism ring of the curve of j-invariant 1728 that we denote by E_0 in the rest of this paper.

We are going to restrict our algorithms to primitive left $\mathcal{O}_0$-ideals, and in the remainder of this article to improve readability, when talking about an ideal, one may assume that it is in fact a left primitive integral $\mathcal{O}_0$-ideal.

2.1. **Quaternion multiplication.** To the best of our knowledge, improving the multiplication of two elements has never really been addressed in the context of the quaternion algebra $\mathcal{B}_{p,\infty}$, and indeed, both the latest SQIsign implementation and the implementation from [CRSEL+25] uses a naive multiplication method that requires 18 integer multiplications (16 for each products of coefficients plus two additional multiplications by p) when all the coefficients involved are integers. Below, we describe an algorithm to perform the same operation with only 12 multiplications. We focus on the integral case to simplify a bit the exposition. The rational case can be treated with one additional multiplication of the denominators.

For Hamilton quaternions, it is well known that the Karatsuba trick to perform multiplication in $\mathbb{C}$ with only 3 real multiplications can be generalized so that one can go from 16 multiplications down to 8 [DG75]. But in the case of $\mathcal{B}_{p,\infty}$, the need to multiply some of the terms by p makes the direct application of this trick impossible, and it requires a few modifications. We found a method that works with 12 multiplications. We are not sure that this is optimal, but that's already much better than 18. Despite this being a relatively well-known trick, we couldn't find a reference for this in the literature so we report it here as Algorithm 1.

Algorithm 1 Multiplication in $\mathcal{B}_{p,\infty}$

Input: $\alpha = a + ib + jc + kd, \beta = A + iB + jC + kD$, where $a, b, c, d, A, B, C, D \in \mathbb{Z}$
Output: $\alpha\beta \in \mathcal{B}_{p,\infty}$.

$v_1 \leftarrow (a+b)(A+B)$
$t \leftarrow (a-b)(A-B)$
$v_2 \leftarrow (v_1 - t)/2$
$v_1 \leftarrow -(v_1 + t)/2$
$s \leftarrow -(c+d)(C+D)$
$t \leftarrow -(c-d)(C-D)$
$r \leftarrow (s+t)/2$
$s \leftarrow (s-t)/2$
$v \leftarrow (a+b+c+d)(A+B+C+D)$
$u \leftarrow (a-b+c-d)(A-B+C-D)$
$t \leftarrow (u+v)/2 + r + v_1 - 2bD$
$u \leftarrow (v-u)/2 + s - v_2 - 2Bc$
$r \leftarrow v_1 + 2aA + pr$
$s \leftarrow v_2 + p(s + 2cD)$
return $(r + is + jt + ku)$.

2.2. **Ideal basis computation.** To perform any kind of operations over orders and ideals, the first requirement is to have a basis for them, and it is generally handy to use some kind of canonical form for this basis. One possible solution is to use the Hermite Normal Form (HNF), as is done in SQIsign's implementation.

Our goal in this section is to have an algorithm that takes as input a generator γ of I, and computes a basis for $I = \mathcal{O}_0\langle\gamma, N\rangle$. We don't actually care about all the requirements coming from the HNF definition, and will only keep the most important one: that the basis is upper or lower triangular. This seems like a convenient choice because it gives a basis matrix which is quite sparse, and as we will see with Lemma 2.1, left $\mathcal{O}_0$-ideals actually admit a pretty simple basis of this form.

For simplicity, in this section we restrict to the case N is prime and not equal to p. The composite case is partly dealt in the next section where we are going to talk about ideal intersection. We refer the reader to [ACM25] for the prime-power case.

Before exposing our method, let us first describe a generic but slow method to do what we want. This is actually what is done in the implementation from [AAA+25] and what was done in the original implementation from [CRSEL+25].

(1) Multiply the basis of $\mathcal{O}_0$ by γ and N.
(2) Extract a basis for I from these 8 quaternion elements by linear elimination.
(3) Apply a generic algorithm to get a basis of I in the desired form (HNF or something else).

In addition to requiring many more operations than what we propose below, it also makes the size of the integers involved blow-up quite significantly. The recent works [KLKL25, BHJ+25] report the need for integers of several thousand bits for handling ideals with norm whose bit-size is only in the hundreds. On the contrary, our method to get a triangular basis from the generators N, γ requires only a few

operations modulo N, and thus only requires to handle integers smaller than N^2 (and only temporarily to compute multiplications modulo N).

Our method of computation is easily derived from the following structural statement on bases of left $\mathcal{O}_0$-ideals of prime norm. Note that this result can be seen as a special case of the results given in [ACM25, Cle25] in a slightly different setting.

Lemma 2.1. *Let $p = 3 \mod 4$ and N be distinct odd primes. Then, every left $\mathcal{O}_0$-ideal of norm N admits a basis*[1] *in one of the following forms (that we will call henceforth the inert and split form):*

$$M^N_{\mathsf{inert}}(x,y) = \begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ x/2 & y/2 & 1/2 & 0 \\ (2N-y)/2 & x/2 & 0 & 1/2 \end{pmatrix}, \tag{2.1}$$

$$M^N_{\mathsf{split}}(x) = \begin{pmatrix} N & 0 & 0 & 0 \\ x & 1 & 0 & 0 \\ 0 & N/2 & N/2 & 0 \\ 1/2 & (N-x)/2 & (N-x)/2 & 1/2 \end{pmatrix}, \tag{2.2}$$

where $x, y \in \mathbb{N}$ are smaller than $2N$. In the inert case, we have that $x = 0 \mod 2$ and $y = 1 \mod 2$.

There are either 0 or 2 ideals of the split form depending on whether $N = 3 \mod 4$ (inert) or $N = 1 \mod 4$ (split).

Proof. We give below a constructive proof that will underly at the same time Algorithm 2.

We first note that all the coefficients of the elements of $\mathcal{O}_0$ are contained inside $\mathbb{Z}/2$, so the common denominator for all the basis elements we consider is always at most 2. Moreover, since $\mathcal{O}_0 N$ is contained in every ideal of norm N, the coefficients considered can always be defined mod N.

We will start by looking at the split case which only happens if N is split in $\mathbb{Z}[i]$ or equivalently if $N = 1 \mod 4$. In that case, since $\mathbb{Z}[i]$ has class number 1, there exists two elements $a + ib$ and $a - ib$ of norm N for some $a, b \in \mathbb{N}$ (meaning that $a^2 + b^2 = N$). Thus, we have the two distinct principal ideals of norm N: $\mathcal{O}_0(a+ib)$ and $\mathcal{O}_0(a-ib)$, and those are the only two ideals of that form. Since N is prime, it is clear that both a, b are coprime to N. Thus, by multiplying by $b^{-1} \mod N$ and reducing the result mod N, we see that there must be some element of the form $x + i$ in both these ideals.

The other elements of the basis are $N(i+j)/2$ and $-(i+j)(x+i)/2 + N(i+j)/2$. It is pretty clear that both elements are indeed contained in the ideal I. Since the matrix is lower triangular, we see that this is a basis for a dimension 4 lattice, and we just saw that this lattice is contained in an ideal of norm N. To prove that it is a correct basis, it suffices to compute the discriminant of this lattice. One will find that it is equal to $N^2 p$ which is exactly the discriminant of all the left ideals of maximal order of norm N. Thus, this lattice is equal to the full ideal.

Now, on to the inert case. Let $I = \mathcal{O}_0 N + \mathcal{O}_0 \gamma$ for some $\gamma \in \mathcal{O}_0$ of norm divisible by N, and assume that we are not in the split case. Without loss of generality, we

[1] the rows of the matrices are the coefficients of the basis elements of I in the basis $\langle 1, i, j, k \rangle$ of $\mathcal{B}_{p,\infty}$.

can assume that $\gamma = a + ib + j(c + id)$ for $a, b, c, d \in \mathbb{Z}^4$ (since N is odd both γ and 2γ will be suitable generators of I).

Let us show that we can assume (up to replacing γ with another equivalent element) that $c^2 + d^2$ is coprime to N. Indeed, if $c^2 + d^2 = 0 \mod N$, then since $\mathrm{n}(\gamma) = 0 \mod N$, we would also have $a^2 + b^2 = 0 \mod N$. This would mean that both $a + ib$ and $c + id$ are contained in an ideal of norm N in $\mathbb{Z}[i]$. If N is inert in $\mathbb{Z}[i]$ this is simply not possible, and if N is split, there are exactly two such ideals (which are mutually conjugate) and they are both principal. If $(a + ib)$ and $(c + id)$ are contained in the same ideal, this means that we can rewrite $\gamma = \gamma'(x + iy)$ for some $x, y \in \mathbb{Z}$, and so I is contained in $\mathcal{O}_0(x + iy) + \mathcal{O}_0 N$ one of the two ideals in split form, and by equality of norm of the two ideals, they must be equal, which is not possible by assumption.

If $(a + ib)$ and $(c + id)$ are not in the same quadratic ideal, then for any non-zero x, y modulo N the linear combination $x(a + ib) + y(c + id)$ has non-zero norm $\mod N$, and thus we can replace γ by $\gamma' = (x' + jy')\gamma$ for any x', y' such that the norm of $x'^2 + py'^2$ is not zero modulo N (such x', y' always exist for odd N). Since $n(x' + jy')$ is coprime to N and the multiplication is made on the left, it is clear that I is also equal to $O_0\gamma' + \mathcal{O}_0 N$.

Thus, we can assume that $c^2 + d^2$ is invertible $\mod N$, and so if t is the inverse of $2(c^2 + d^2) \mod N$, then $(c + id)t\gamma$ is equal modulo $N\mathcal{O}_0$ to an element of the form $(x + iy + j)/2$ for some $x, y < 2N$. The values of $x, y \mod 2$ are a consequence of the fact that the basis of $\mathcal{O}_0$ is $\langle 1, i, (i + j)/2, (1 + k)/2 \rangle$.

The final basis element is simply the reduction modulo $N\mathcal{O}_0$ of $i(x + iy + j)/2$. Once again, we proved that the lattice generated by this basis is contained in I, and then we can prove equality of the two by equality of the discriminants. □

Remark 2.1. *The structural result presented in Lemma 2.1 is particularly nice because $\mathbb{Z}[i]$ (which is an order of class number 1) is embedded inside $\mathcal{O}_0$. This result can be generalized to any norm N with a bit more work (see [Cle25] for the prime power case, and our Section 2.3 below for the composite case), but the statement is a bit less pleasant already. We could also probably generalize it to other orders and other $\mathcal{B}_{p,\infty}$ with $p = 1 \mod 4$, but the statement would only get messier and messier. For our main application to modular polynomial evaluation, Lemma 2.1 is sufficient. If one wants to apply this to SQIsign, one might need to derive a more generic statement (at least to handle any maximal order $\mathcal{O}$ in $\mathcal{B}_{p,\infty}$).*

Restricting to the inert case. Lemma 2.1 and Algorithm 2 included the split case for completeness, but we don't really need it for our application to ModularEvaluationBigCharacteristic. Indeed, as $\mathbb{Z}[i]$ has class number one, the type of $\mathcal{O}_0$ is the only type of maximal orders including i. And this implies that the ideals in the split case are in fact principal ideals which corresponds to endomorphisms of the curve $j = 1728$. In particular, the right order of these ideals is always isomorphic to $\mathcal{O}_0$. In our applications, where the Deuring correspondence is used to navigate within the supersingular isogeny graph, this essentially means that we can replace those ideals by $\mathcal{O}_0$ (up to re-scaling by the proper endomorphism if needed), and so we need not worry about ideals in the split form. For this reason, in the remainder of this section, we focus on the inert case.

2.3. **Ideal intersection.** Ideal intersection is a very useful operation in the Deuring correspondence as it is equivalent to computing push-forward isogenies.

Algorithm 2 Fast ideal basis computation for left $\mathcal{O}_0$-ideals of prime norm N

Input: an odd prime N, and $\gamma = (a + ib + jc + kd)$ where $a, b, c, d \in \mathbb{Z}$ and $\mathrm{GCD}(\mathrm{n}(\gamma), N^2) = N$
Output: A $\mathbb{Z}$-basis for $I = \mathcal{O}_0\gamma + \mathcal{O}_0 N$

$s \leftarrow c^2 + d^2 \mod N$
if $s = 0$ **then**
  Find non-zero x, y modulo N such that $x^2 + y^2 = 0 \mod N$
  if $\gamma(x + iy) \in N\mathcal{O}_0$ **then**
    $a \leftarrow -xy^{-1} \mod N$
    return $M^N_{\mathsf{split}}(a)$
  else if $\gamma(x - iy) \in N\mathcal{O}_0$ **then**
    $a \leftarrow xy^{-1} \mod N$
    return $M^N_{\mathsf{split}}(a)$
  else
    Find t such that $t^2 + p \mod N \neq 0$
    $\gamma \leftarrow (t + j)\gamma$, and update a, b, c, d accordingly.
    $s \leftarrow c^2 + d^2$
  end if
end if
$t \leftarrow (2s)^{-1} \mod N$
$x \leftarrow 2\left((ac + bd)t \mod N\right)$
$y \leftarrow 1 + 2\left((bc - ad - s)t \mod N\right)$
return $M^N_{\mathsf{inert}}(x, y)$

In the generic setting, lattice intersection is not completely trivial. One standard method directly stems from the following property: the dual of the lattice resulting from the addition of the respective duals of two lattices Λ_1, Λ_2 is equal to the intersection of Λ_1 and Λ_2. This is for instance the method used in the SQIsign implementation and in the original implementation from [CRSEL+25], and it is pretty expensive as it requires several dual lattice computation and one generic lattice addition.

However, in the quaternion algebra setting, we are not dealing with generic lattices, and so we can do much better than that by exploiting the structure stemming from Lemma 2.1. In fact, in this setting, intersection is more or less equivalent to CRT. Indeed, if I_1, I_2 are two left $\mathcal{O}_0$-ideals of coprime norm N_1, N_2, and J is the intersection $I_1 \cap I_2$, then it is easy to see that $I_1 = J + N_1\mathcal{O}_0$ and $I_2 = J + N_2\mathcal{O}_0$. This translates almost directly to a much better algorithm than what we described above.

Let $p_1, p_2 \ldots, p_n$ be distinct primes. Then, Lemma 2.1 can be generalized to $N = p_1 p_2 \ldots p_n$ by considering the reductions modulo each factor, and so there are 2^n cases depending on whether the reduction mod $p_i\mathcal{O}_0$ is in the inert or split case. In the case where all reduction are inert, then the basis can be put into the $M^N_{\mathsf{inert}}(x, y)$ form, and the values of x, y are directly obtained by CRT from the values of each x_i, y_i for each p_i.

When the number of distinct primes is not too big, this method is much faster than the generic intersection method. Moreover, the CRT computation does not require manipulating integers that are much bigger than the final modulus (an

actual upper-bound is $n \times p_1 p_2 \dots p_n$), so this algorithm is also very nice to avoid any blow-up of the required integer size.

2.4. **Right order computation.** Another important component of the algorithms from [Ler23, CRSEL+25] is the need to compute the right order of an $\mathcal{O}_0$-ideal, and as before we need to compute a full basis of the order. The goal of this section is to explain how to do this for ideals whose basis have been put in the inert form described by Lemma 2.1.

One generic way of computing $\mathcal{O}_R(I)$ comes from the formula $\mathcal{O}_R(I) = \overline{I} \cdot I / n(I)$, which requires to perform ideal multiplication. This is a pretty expensive operation if not done carefully, as it requires 16 quaternion multiplications (the product of every pair of basis elements), before performing a linear elimination step to extract a basis.

On the other hand, our method is using a result similar to Lemma 2.1 but for right orders, and it is very easy to compute if the ideal is given in the inert form from Lemma 2.1. Let us write N for the norm of I. Then, with $N\mathcal{O}_R(I) \subset I \subset \mathcal{O}_0$, we can see that the denominator of the coefficient of the elements of $O_R(I)$ in the basis $\langle 1, i, j, k \rangle$ will be $2N$. Since $N\mathcal{O}_0 \subset I \subset \mathcal{O}_R(I)$, we also have that the coefficients of the numerator matrix of the basis of $\mathcal{O}_R(I)$ can be upper-bounded by $2N^2$ as we can always reduce these elements modulo $2N^2$. This also means, that we will be able to compute the matrix using only a few operations modulo $2N^2$.

Note that Lemma 2.2 is not restricted to prime N anymore. It works for any odd, square-free N.

Lemma 2.2. *Let N be an odd integer coprime to p with no square factors, and let I be a left $\mathcal{O}_0$ ideal of norm N. Then, if I admits a basis in inert form, then a basis of $\mathcal{O}_R(I)$ is given by the rows of the matrix[2]:*

$$MO^N_{\mathsf{inert}}(A, a, b, c) = \frac{1}{2N} \begin{pmatrix} N & 0 & 0 & N^2 \\ 0 & 1 & a & b \\ 0 & 0 & 2NA & 2Nc \\ 0 & 0 & 0 & 2N^2/A \end{pmatrix}, \tag{2.3}$$

where $A = \mathrm{GCD}(N, x)$, and $a, b < 2N^2$ and $c < N$.

Proof. As we did for Lemma 2.1, we give a constructive proof that will also underly our algorithm for fast right order computation. Following the same pattern, we just need to prove that the proposed lattice is contained in $\mathcal{O}_R(I)$, and since the discriminant of the lattice associated to the matrix $MO^N_{\mathsf{inert}}(A, a, b, c)$ is p, it must be $\mathcal{O}_R(I)$. Let us write $M^N_{\mathsf{inert}}(x, y)$ for a basis of I in inert form.

We remind that $N\mathcal{O}_0 \subset \mathcal{O}_R(I)$. Thus, as both 1 and $N(1+k)/2$ are contained in $\mathcal{O}_R(I)$, we get that the first basis vector $(1 + kN)/2$ is contained in $\mathcal{O}_R(I)$.

The third element is built as follows: if b_3, b_4 are the third and fourth basis elements of I then $xb_3 - yb_4 = \left((x^2 + y^2) + xj - ky\right)/2$. The value $x^2 + y^2 = 1 \mod 2$, and so by subtracting by $\lfloor (x^2 + y^2)/2 \rfloor$ and $(1 + kN)/2$, before multiplying by 2, we get that the element $(xj - (y + N)k)$ is contained in $\mathcal{O}_R(I)$. Let $A = \mathrm{GCD}(x, N)$. If $\mathrm{GCD}(x/A, N) \neq 1$, we can replace x by $x + 2N$ to ensure that $\mathrm{GCD}(x/A, N) = 1$ as N has no square factors. And then, we get a vector of the form $Aj + ck$ by multiplying by $(x/A)^{-1} \mod N$ and reducing the result mod N.

[2] this gives the coefficients of the basis elements of $\mathcal{O}_R(I)$ in the basis $\langle 1, i, j, k \rangle$ of $\mathcal{B}_{p,\infty}$.

The second vector $(i + aj + bk)/(2N)$ is obtained from

$$(t_0 i + a_0 j + b_0 k)/2N := \overline{b_4} i b_4 / N.$$

A simple computation gives $t_0 = (x^2 + (2N - y)^2 - p)/2$, $a_0 = -(2N - y)$ and $b_0 = x$.

One can verify easily that this is indeed a vector of the right order of I. Since $4N$ divides the norm of $2b_4$ which is equal to $(x^2 + (2N - y)^2 + p)$, the value of t_0 is an integer that is is invertible modulo N and N^2. So we get the desired vector of the form $(i + aj + bk)/2N$ by multiplying by the inverse of t_0 modulo N^2, and then reducing the result $\bmod N\mathcal{O}_0$.

To conclude, we need to prove that $(N/A)k$ is contained in $\mathcal{O}_R(I)$. By definition of $\mathcal{O}_R(I)$, this means that $\alpha(N/A)k \in I$ for all $\alpha \in I$. We will verify this property for every basis element of I. It is obvious for $\alpha = N, Ni$. We are going to make an explicit computation for the third basis element $\alpha = (x + iy + j)/2$ where $x = 2Ax'$ by definition of A and the fact that x is even. Thus, we get $\alpha(N/A)k = Nx'k + \frac{N}{2A}(-jy + ip)$. Since $Nx'k$ is clearly contained in I, it suffices to prove that $\frac{N}{2A}(-jy + pi) \in I$ to prove that $\alpha(N/A)k \in I$. This is what we do next.

As $n(2\alpha) = 0 \mod 4N$, we have that there exists some $\ell \in \mathbb{Z}$ such that $y^2 = 4\ell N - 4A^2x'^2 - p$. Moreover, $y(N/A)\alpha = Nx'y + \frac{N}{2A}(y^2 i + jy)$. Thus, by replacing y^2, and then reordering the terms we get the following sequence of equalities:

$$\begin{aligned} y(N/A)\alpha &= Nx'y + \frac{N}{2A}\left((4\ell N - 4A^2x'^2 - p)\cdot i + jy\right) \\ \frac{N}{2A}(-jy + ip) &= -\frac{N}{A}y\alpha + Nx'y + i\frac{N}{2A}(4\ell N - 4A^2x'^2) \\ \frac{N}{2A}(-jy + ip) &= -\frac{N}{A}y\alpha + N\left(x'y + i\left(\frac{N}{A}2\ell - 2Ax'^2\right)\right) \end{aligned}$$

Since $\frac{N}{A}$ is an integer, $\alpha \in I$ and $N\mathcal{O}_0 \subset I$, the last equality proves that $(N/2A)(-jy + ip) \in I$ and so that $\alpha(N/A)k \in I$. One can make a similar computation for the last basis element of I, and this proves the result.

□

2.5. **Ideal isomorphism.** Finally, the last algorithm we are going to introduce is an algorithm to compute an isomorphism of left ideals. The reasons why we need this specific algorithm are detailed in Section 3.1, but it raises interesting algorithmic questions that could be useful outside of this specific context. For now, let us just focus on the algorithm.

Here is what we want to do exactly: let I_1, I_2 be two $\mathcal{O}_0$-ideals such that $\mathcal{O}_R(I_1) \simeq \mathcal{O}_R(I_2)$, the goal is to determine if I_1 and I_2 are equivalent, and if yes, find the element γ such that $I_2 = I_1\gamma$, or if no, the γ such that $I_2 = I_1\mathfrak{p}_{\mathcal{O}_R(I_1)}\gamma$ where we remind the reader that $\mathfrak{p}_{\mathcal{O}_R(I_1)}$ is the unique two-sided $\mathcal{O}_R(I_1)$-ideal of norm p. We sketch an algorithm to solve that problem in Algorithm 4. The remainder of this section is explaining how that algorithm works.

One obvious way to solve this problem is to compute $\overline{I_1}\cdot I_2$, determine if this ideal is principal, if yes compute the generator $n(I_1)\gamma$ of this ideal, and if not compute the generator $n(I_1)p\gamma$ of $\mathfrak{p}_{\mathcal{O}_R(I_1)}\overline{I_1}\cdot I_2$ and then extract the γ. Thus, this algorithm requires performing 1 or 2 ideal multiplications, testing if an ideal is principal, and computing the generator for a principal ideal. As far as we know, both testing if an

Algorithm 3 Fast right order computation for $\mathcal{O}_0$-ideals of odd norm N in inert form M^N_{inert}

Input: an odd N, two integers $0 \leq x, y < 2N$ such that $I = \mathcal{O}_0(x + iy + j) + \mathcal{O}_0 N$ is an ideal of norm N
Output: a basis for $\mathcal{O}_R(I)$

$A \leftarrow \mathrm{GCD}(x, N)$
if $\mathrm{GCD}(x/A, N) \neq 1$ **then**
  $x \leftarrow x + 2N$
end if
$t \leftarrow (x/A)^{-1} \mod N$
$c \leftarrow -yt \mod N$
$t \leftarrow (x^2 + (2N - y)^2 - p)/2$
$t \leftarrow t^{-1} \mod N^2$
$a \leftarrow -(2N - y)t \mod (2N^2)$
$b \leftarrow xt \mod (2N^2)$
if $t \mod 2 = 0$ **then**
  $a = a + N^2 \mod (2N^2)$
end if
return $MO^N_{\mathsf{inert}}(A, a, b, c)$

ideal is principal, and computing the generator, have essentially the same cost as they require finding the element of smallest norm. If the norm of this element is the same as the ideal, then the ideal is principal, and that element is the generator of the principal ideal. Finding the smallest element requires performing some lattice reduction which is quite expensive.

It seems hard to compute γ without any form of lattice reduction. But our proposed method has two advantages over the one described above:

- It does not require any lattice multiplication (simply some quaternion multiplications).
- The lattice reduction is performed in the Gross lattices of $\mathcal{O}_R(I_1), \mathcal{O}_R(I_2)$ which have dimension 3 instead of 4.

For our application to ModularEvaluationBigCharacteristic, our new algorithm will be particularly interesting because we need to perform the Gross lattice reduction anyway to compute the invariant of the maximal order types, and so the computation of the isomorphism only requires a few additional quaternion multiplications. However, we believe that even in the case where the right order reduction was not previously performed, our proposed algorithm is actually faster as using a smaller dimension for the lattice reduction makes quite a big difference in practice.

Here is the idea of our algorithm: when $I_2 = I_1\gamma$ or $I_2 = I_1\mathfrak{p}_{\mathcal{O}_R(I_1)}\gamma$. The element γ also satisfies $\mathcal{O}_R(I_2) = \gamma^{-1}\mathcal{O}_R(I_1)\gamma$. Thus, to find this element γ, we propose to use the two smallest elements of trace 0 in $\mathcal{O}_R(I_1)$ and $\mathcal{O}_R(I_2)$. If we denote these elements by α_1, β_1 and α_2, β_2, then we look for γ such that $\gamma^{-1}\alpha_1\gamma = \alpha_2$ and $\gamma^{-1}\beta_1\gamma = \beta_2$. This translates to a linear system of 8 equations with 4 unknowns whose kernel has always dimension 1.

Remark 2.2. *There are slight complications when $\mathcal{O}_R(I_1)$ is one of the at most 2 maximal order types whose automorphism group is not trivial, but since we are only preoccupied with an algorithm that works in most cases, we can simply fall back to*

the slower method described at the beginning of this section when encountering one of the rare bad cases.

Remark 2.3. *One might wonder if it would be sufficient to use only α_1, α_2 but it turns out that the kernel of the resulting system has dimension 2 in general. The reason why the kernel has dimension 1 when using $\alpha_1, \beta_1, \alpha_2, \beta_2$ is because a solution γ such that $\gamma^{-1}\alpha_1\gamma = \alpha_2$ and $\gamma^{-1}\beta_1\gamma = \alpha_1$ induces an isomorphism of the quaternion orders $\langle 1, \alpha_1, \beta_1, \alpha_1\beta_1\rangle$ and $\langle 1, \alpha_2, \beta_2, \alpha_2\beta_2\rangle$.*

Solutions to this system can actually be obtained via very simple formulas. This allows us to compute γ from $\alpha_1, \beta_1, \alpha_2, \beta_2$ without resorting to a generic system solving algorithm. This formula comes from the following result:

Lemma 2.3. *Let α_1, α_2 and β_1, β_2 be the elements realizing the first and second minima in the Gross lattices of two isomorphic maximal orders $\mathcal{O}_1, \mathcal{O}_2$ such that $\mathrm{tr}(\alpha_1\beta_1) = \mathrm{tr}(\alpha_2\beta_2)$, then*

$$\gamma = \alpha_2(\beta_1 - \beta_2) + (\beta_1 - \beta_2)\alpha_1$$

satisfies $\alpha_2\gamma = \gamma\alpha_1$ and $\beta_2\gamma = \gamma\beta_1$.

Proof. Since α_1 and α_2 have trace zero and same norm, we have that $\alpha_2^2 = \alpha_1^2 = -n(\alpha_1)$. Thus, one can easily check that indeed $\alpha_2\gamma = \gamma\alpha_1$.

The proof for β_1, β_2 will follow from the fact that

$$\gamma = \beta_2(\alpha_2 - \alpha_1) + (\alpha_2 - \alpha_1)\beta_1$$

which is a consequence of two following facts:

(1) $\overline{\alpha_i\beta_i} = \beta_i\alpha_i$, which comes from the fact all these elements have trace 0.
(2) $\mathrm{tr}(\alpha_1\beta_1) = \mathrm{tr}(\alpha_2\beta_2)$ which is an hypothesis.

Indeed, developing the two formulas we get $\alpha_2\beta_1 - \alpha_2\beta 2 + \beta_1\alpha_1 - \beta_2\alpha_1$ for the left hand side, and $\beta_2\alpha_2 - \beta_2\alpha_1 + \alpha_2\beta_1 - \alpha_1\beta_1$ for the right hand side. Removing $\alpha_2\beta_1 - \beta_2\alpha_1$ from both sides, we need to have the equality $\beta_1\alpha_1 - \alpha_2\beta_2 = \beta_2\alpha_2 - \alpha_1\beta_1$, which can be rewritten as

$$\beta_1\alpha_1 + \alpha_1\beta_1 = \beta_2\alpha_2 + \alpha_2\beta_2$$

or

$$\mathrm{tr}(\alpha_1\beta_1) = \mathrm{tr}(\alpha_2\beta_2)$$

which holds by equality of the traces. □

Remark 2.4. *Most of the time, the solution given by Lemma 2.3 is enough for our purpose. However, there is one unlikely case where the γ given by Lemma 2.3 is not suitable: when this γ is 0. When this occurs, one can remark that $\gamma = \alpha_2(\delta\beta_1 - \beta_2\delta) + (\delta\beta_1 - \beta_2\delta)\alpha_1$ also works for any $\delta \in \langle 1, i, j, k\rangle$. It turns out that all those elements are scalar multiple of one another since the kernel has dimension 1. This seems quite surprising given the fact that δ may range over a four-dimensional space, but it is true. We conjecture that there should be a way to find the value of this scalar from the knowledge of δ, but we weren't able to figure out a simple formula for that. This is not necessary for our application, although it might allow us to speed-up a bit the computation in practice.*

Even if Lemma 2.3 gives (almost always) a valid isomorphism from $\mathcal{O}_R(I_1), \mathcal{O}_R(I_2)$, this is not the end of it as our final goal is to find the isomorphism between our two $\mathcal{O}_0$-ideals I_1, I_2, if it exists. By re-scaling with the appropriate integer, we can get a γ in $\mathcal{O}_0$ which is primitive. Then, any of the four following cases[3] can happen:

(1) $n(\gamma) = n(I_1)n(I_2)$, so $I_1 \sim I_2$ and $I_2 = I_1\gamma/n(I_1)$.
(2) $n(\gamma) = n(I_1)n(I_2)p$ so $I_1 \not\sim I_2$ and $I_2 = I_1\mathfrak{p}_{\mathcal{O}_R(I_1)}\gamma/n(I_1)$.
(3) $n(\gamma) = n(I_1)n(I_2)d$ for some $d > 0$, and $I_1 \not\sim I_2$.
(4) $n(\gamma) = n(I_1)n(I_2)pd$ for some $d > 0$, and $I_1 \sim I_2$.

The first two cases are quite obvious once one knows the following fact: for two primitive ideals I, J in $\mathcal{B}_{p,\infty}$, if $I = J\beta$, then $\beta = \beta'/n(J)$ where $\beta' \in \overline{J}$ and $n(\beta') = n(I)n(J)$ (see for instance [KLPT14, Lemma 5]).

The 3rd and 4th cases are a bit more subtle. In fact, these cases come from the fact that every order contains -1. Thus, the actual isomorphism we're looking for might take α_1, β_1 to $-\alpha_2, -\beta_2$ instead of α_2, β_2 (we can ensure a coherent choice of sign between $\alpha_2\beta_2$ and $\alpha_1\beta_1$ by choosing α_2, β_2 such that $\mathrm{tr}(\alpha_1\beta_1) = \mathrm{tr}(\alpha_2\beta_2)$ but that's all). When this happens, the solution we need can be extracted by multiplying γ by $\delta = 2\alpha_1\beta_1 - \mathrm{tr}\,\alpha_1\beta_1$. Indeed, one can check that $\delta\alpha_1\delta^{-1} = -\alpha_1$, and the same for β_1. The value of d actually comes from the norm of δ which is equal to pd.

We are now ready to give a detailed description of our algorithm to compute the ideal isomorphism. For Algorithm 4 below, we assume that we have a MakePrimitive algorithm that takes a quaternion element $\gamma \in \mathcal{O}_0$ and outputs the primitive $\gamma' \in \mathcal{O}_0$ such that $\gamma = \lambda\gamma'$ for some $\lambda \in \mathbb{Z}$. This algorithm uses some GCD computations to remove the scalar λ.

In Algorithm 4, the default formula to compute γ is actually not the one from Lemma 2.3 but we use $\gamma = (\alpha_2(j\beta_1 - \beta_2 j) + (j\beta_1 - \beta_2 j)\alpha_1)\,\frac{n(I_1)n(I_2)}{p}$ which is also correct as explained in Remark 2.4, and seems to give slightly nicer formulas in practice. To be as efficient as possible to compute γ one needs to develop the expression of each coordinate of γ to minimize the number of operations, but for conciseness of the exposition we give the high-level variant.

3. Application to evaluation of modular polynomials

In this section, we discuss how to improve the implementation of ModularEvaluationBigCharacteristic from [CRSEL+25]. The main part of that improvement is due to the algorithms detailed in Section 2, and the strict control we gain on integer size which allow us to switch to fixed-size integers (see Section 3.2.2). But our new implementation also includes various additional improvements that we describe in this section (although, in a somewhat informal manner in most cases).

3.1. A lower-level description of SpecialSupersingularEvaluation.

To illustrate how the algorithms from Section 2 are used exactly in ModularEvaluationBigCharacteristic, we provide below a more detailed description of its main building block: the SpecialSupersingularEvaluation algorithm (this is to be compared with the one that can be found as [CRSEL+25, Algorithm 1]).

For some level ℓ, big prime characteristic q and small prime characteristic p, the goal of SpecialSupersingularEvaluation is to compute the reduction modulo p of

[3] when the automorphism group of the right orders is trivial

Algorithm 4 Fast ideal isomorphism computation

Input: Two ideals I_1, I_2 such that $\mathcal{O}_R(I_1) \cong \mathcal{O}_R(I_2)$, and the first and second minima α_1, β_1 and α_2, β_2 of $\mathcal{O}_R(I_1)^T$, $\mathcal{O}_R(I_2)^T$ such that $\mathrm{tr}(\alpha_1\beta_1) = \mathrm{tr}(\alpha_2\beta_2)$

Output: A boolean stating whether or not $I_1 \sim I_2$ and the element $\gamma \in \mathcal{O}_0$ such that either $I_2 = I_1\gamma/n(I_1)$ or $I_2 = I_1\mathfrak{p}_{\mathcal{O}_R(I_1)}\gamma/n(I_1)$

$\gamma \leftarrow (\alpha_2(j\beta_1 - \beta_2 j) + (j\beta_1 - \beta_2 j)\alpha_1)\,\frac{n(I_1)n(I_2)}{p}$
if $\gamma = 0$ **then**
  $\gamma \leftarrow (\alpha_2(\beta_1 - \beta_2) + (\beta_1 - \beta_2)\alpha_1)\,n(I_1)n(I_2)$
end if
$\gamma \leftarrow \mathsf{MakePrimitive}(\gamma)$
if $\mathrm{n}(\gamma) = 0 \mod p$ **then**
  if $\mathrm{n}(\gamma)/p = \mathrm{n}(I_1)\,\mathrm{n}(I_2)$ **then**
    Return False, γ
  else
    $\delta \leftarrow 2\alpha_1\beta_1 - \mathrm{tr}(\alpha_1\beta_1)$
    $\gamma \leftarrow \gamma\delta$
    $\gamma \leftarrow \mathsf{MakePrimitive}(\gamma)$
    Return True, γ
  end if
else
  if $\mathrm{n}(\gamma) = \mathrm{n}(I_1)\,\mathrm{n}(I_2)$ **then**
    Return True, γ
  else
    $\delta \leftarrow 2\alpha_1\beta_1 - \mathrm{tr}(\alpha_1\beta_1)$
    $\gamma \leftarrow \gamma\delta$
    $\gamma \leftarrow \mathsf{MakePrimitive}(\gamma)$
    Return False, γ
  end if
end if

$\Phi_\ell(X, j) \mod q$ (seen as an element of $\mathbb{Z}[X]$). The values $(j^i)_{0\le i\le \ell+1} \mod p$ are given in input.

SpecialSupersingularEvaluation works by first collecting a dictionary of all supersingular j-invariants and their corresponding maximal order with the OrdersTojInvariantBigSet algorithm (see [Ler23] for more details). Then, we enumerate enough pairs of ℓ-isogenous j-invariants by computing ideals of norm ℓ and retrieve the j-invariants corresponding to their left and right orders from the dictionary. Concretely, this is done in SpecialSupersingularEvaluation by computing two lists (we use the notations of Algorithm 5):

(1) A list $I_{1,0}, \dots I_{\ell+2,0}$ of ideals belonging to distinct left $\mathcal{O}_0$-ideal classes;
(2) A list $J_1, \dots, J_{\ell+1}$ of all $\mathcal{O}_0$-ideals of norm ℓ.

Then for any $1 \le i \le \ell + 2$, the set of $\mathcal{O}_R(I_{i,0})$-ideals of norm ℓ is given by $J_{i,k} = I_{i,0}^{-1} \cdot (I_{i,0} \cap J_k)$ for all $1 \le k \le \ell + 1$. But since $\mathcal{O}_R(J_{i,k}) = \mathcal{O}_R(I_{i,k})$ with $I_{i,k} = (I_{i,0} \cap J_k)$, it suffices to compute all the $I_{i,k}$ and their right order.

Finally, we reconstruct the desired result from all the pairs of j-invariants. To be able to use the algorithms from Section 2, we are going to manipulate exclusively $\mathcal{O}_0$-ideals.

For this, we will assume that the algorithm OrdersTojInvariantBigSet that performs the precomputation of the j-invariant to maximal order dictionary denoted by $\mathcal{M}$ hereafter also computes for each entry (corresponding to some type of maximal order $\mathcal{T}$ identified by the three successive minima of its Gross lattice):

(1) an ideal I whose left order is $\mathcal{O}_0$ and right order belongs to $\mathcal{T}$;
(2) two elements realizing the two first successive minima in $\mathcal{O}_R(I)$.

We remind the reader that the main idea from [Ler23] for the dictionary $\mathcal{M}$ is to use the norm of the three successive minima of the trace 0 part as an invariant for types of maximal orders. Thus, given three integers n_1, n_2, n_3, we write $\mathcal{M}(n_1, n_2, n_3)$ as either $\perp$ if there is no entry corresponding to n_1, n_2, n_3 or a tuple j, I, α, β.

One of the problems with working with maximal order types is that we cannot distinguish easily between j-invariants and their Galois conjugates. In SpecialSupersingularEvaluation, we use Algorithm 4 to distinguish between the two cases.

3.1.1. *The Weber variant.* A common trick to drastically improve the performance of algorithms computing and handling modular polynomials is to switch from the classical modular polynomial Φ_ℓ to other modular polynomials (related to modular curves of genus 0 other than $X(1)$). One of the most famous (and probably the most efficient) one is related to the Weber modular function $\mathfrak{f}$ which is denoted by $\Phi_\ell^{\mathfrak{f}}$. As argued at the end of [BLS12], the computation of $\Phi_\ell^{\mathfrak{f}}$ is 1728 times faster in practice than Φ_ℓ due to smaller coefficients and the presence of some symmetry over the coefficients.

It is explained in [CRSEL+25] how to extend ModularEvaluationBigCharacteristic to the Weber case, and their implementation in fact mainly deals with the Weber case as it is so much faster. Below, we detail a few practical adjustments that are to be made to SpecialSupersingularEvaluation to handle this case.

The main idea in [CRSEL+25] is to exploit a modular parametrization of Weber invariants from an order 48 level structure. As such, SpecialSupersingularEvaluation can be modified to work with Weber invariants by seeing them as an order 48 level structure, and translating to the corresponding invariant only when needed for the reconstruction at the end.

Concretely this implies the following changes:

(1) Each entry of the dictionary $\mathcal{M}$ also includes the image of the isogeny φ_I on $E_0[48]$, and a list of all Weber invariants and their association to a level structure.
(2) The correct 48-torsion associated to each $j_{i,k}$ needs to be associated with the 48-torsion of $j_{i,0}$ (meaning that we need to compute the image of the ℓ-isogeny associated to each pair $j_{i,0}, j_{i,k}$ on the 48 torsion).
(3) The Weber invariant $f_{i,k}$ needs to be computed from $j_{i,k}$ and associated 48-torsion.

To perform the second step efficiently, we use a common trick of the Deuring correspondence, which justifies the second part of the output of Algorithm 4 (which up to now was unused). Let us take a pair of indices i, k and take the notations used in the main loop of SpecialSupersingularEvaluation. The dictionary $\mathcal{M}$ contains the image of $\varphi_{I_{i,0}}$ on $E_0[48]$, and we need to get the image of $\varphi_{I_{i,k}}$ on $E_0[48]$ to be able to compute the correct Weber invariant of $j_{i,k}$ associated to each Weber invariant of $j_{i,0}$.

Algorithm 5 SpecialSupersingularEvaluation$(p, \ell, x_0, \ldots, x_{\ell+1})$

Input: A prime p, a prime ℓ with $\lceil p/12 \rceil + 1 < \ell$, and $\ell+2$ values $x_0, \ldots, x_{\ell+1} \in \mathbb{F}_p$.
Output: $P(Y) = \sum_{i,j} a_{i,j} x_i Y^j \mod p$, where $\Phi_\ell(X, Y) = \sum_{i,j} a_{i,j} X^i Y^j$.

Compute a set of maximal orders $\mathcal{O}_1, \ldots, \mathcal{O}_m \subset \mathcal{B}_{p,\infty}$ in distinct types and set $\mathfrak{S}$ as the list of these maximal orders.
Compute $\mathcal{M} = \mathsf{OrdersTojInvariantBigSet}(p, \mathfrak{S})$
Select a set $I_{1,0}, \ldots, I_{\ell+2,0}$ of $\mathcal{O}_0$-ideals such that each $\mathcal{O}_R(I_{i,0})$ belongs in a different isomorphism class, and compute their $\mathbb{Z}$-bases with Algorithm 2.
Compute the set $J_1, \ldots, J_{\ell+1}$ of $\mathcal{O}_0$-ideals of norm ℓ and compute their basis with Algorithm 2.
for $i = 1$ to $\ell + 2$ **do**
    Reduce the dimension 3 lattice $\mathcal{O}_R(I_{i,0})^T$ to find n_1, n_2, n_3, the three successive minima of $\mathcal{O}_R(I_{i,0})$.
    $j_{i,0}, *, *, * \leftarrow \mathcal{M}(n_1, n_2, n_3)$.
    for $k = 1$ to $\ell + 1$ **do**
        $I_{i,k} \leftarrow I_{i,0} \cap J_k$ with the CRT approach described in Section 2.3.
        Compute $\mathcal{O}_R(I_{i,k})$ with Algorithm 3
        Reduce the dimension 3 lattice $\mathcal{O}_R(I_{i,k})^T$ to find n_1, n_2, n_3, the three successive minima of $\mathcal{O}_R(I_{i,k})$, and two elements α^*, β^* realizing the first two minima.
        $j, I, \alpha, \beta \leftarrow \mathcal{M}(n_1, n_2, n_3)$.
        Compute a boolean b by calling Algorithm 4 on $I, \alpha, \beta, I_{i,k}, \alpha^*, \beta^*$
        if $b = \mathsf{True}$ **then**
            $j_{i,k} \leftarrow j$
        else
            $j_{i,k} \leftarrow j^p$
        end if
    end for
    $P(j_{i,0}, X) \leftarrow \prod_{k=1}^{\ell+1} (X - j_{i,k})$
    Write $P(j_{i,0}, X) = \sum_{k=0}^{\ell+1} c_{i,k} X^k$
    $y_i \leftarrow \sum_{k=0}^{\ell+1} c_{i,k} x_k$
end for
Interpolate $P(Y) \mod p$ as the polynomial of degree $\ell + 1$ with $P(j_{i,0}) = y_i$ for all $0 \le i \le \ell + 1$.
return $P(Y)$.

We do not know directly this image, but the dictionary gives us an ideal $I \sim I_{i,k}$ together with the image of φ_I on $E_0[48]$. We can recover the image of $\varphi_{I_{I,K}}$ by finding the element $\gamma \in \mathcal{O}_0$ such that $I = I_{i,k}\gamma/n(I_{i,k})$ with Algorithm 4. Then, through the isomorphism from $\mathcal{O}_0$ to $\mathrm{End}(E_0)$, γ can be seen as an endomorphism of E_0 corresponding to the composition $\varphi_{\hat{I}_{i,k}} \circ \varphi_I$. As such, we can see the action of γ on $E_0[48]$ as a matrix $M_\gamma \in M_2(\mathbb{Z}/48\mathbb{Z})$, and so it suffices to apply this matrix on $\varphi_I(E_0[48])$ and multiply by $(n(I))^{-1} \mod 48$ to recover $\varphi_{I_{i,k}}(E_0[48])$.

The knowledge of the matrix M_γ also helps us to perform the final change required by the Weber variant of the algorithm. Indeed, the method described in [CRSEL+25] to go from the level structure to the Weber invariant is quite heavy.

We want to avoid doing it $O(\ell^2)$ times during the main loop of SpecialSupersingularEvaluation. This is where we use the fact that our dictionary already includes the translation from each Weber invariant to the corresponding level structure. We can figure out the correct permutation of the Weber invariants by using the matrix M_γ, thus allowing us to compute the correct Weber invariants with a few operations modulo 48.

3.2. **Other implementation improvements.**

3.2.1. *Dimension 3 reduction.* Outside of polynomial interpolation, the main cost of SpecialSupersingularEvaluation is the lattice reduction part required to compute the three successive minima n_1, n_2, n_3 together with α, β for each $\mathcal{O}_R(I_{i,k})$. Thus, it is quite important to perform this step as fast as possible.

In the implementation from [CRSEL+25], this reduction step was performed with LLL, using the `fplll` library [dt23]. However, we were able to outperform this implementation by using reduction algorithms tailored to dimension 3. In particular, we compared two algorithms:

- Semaev's algorithm [Sem01] (also analyzed by Nguyen and Stehlé in [NS09]),
- A 3-dimensional version of Lagrange's algorithm which consists in applying Lagrange reduction on pairs of basis vectors until no more reduction is possible. See [SSC25, Algorithm 4] for a detailed description.

The first algorithm provably finds the Minkowski-reduced basis and its complexity is in $O(\log^2 n)$ where n is the norm of the biggest vector given in input. The complexity of the second algorithm is in $O(\log^3 n)$, and it does not give a fully-reduced basis.

However, in practice, for the sizes considered for our application, the second solution runs faster than the first one (and both are faster than `fplll`) as we used an optimized implementation of Lagrange's algorithm due to Pornin [Por20]. To correct the fact that the output of the second method is not optimally reduced, we can run Semaev's algorithm at the end. Since the second method outputs a basis that is almost fully reduced, the execution of Semaev's algorithm at the end is very fast and successfully gives the desired result.

3.2.2. *Integer size and the use of floating points.* For practical efficiency, it is very important to use the appropriate size for integers. The previous implementation of [CRSEL+25] used the arbitrary-sized `NTL::ZZ` integers from [S+01] but this is not optimal as we can in fact have a quite precise bound on the maximum size needed thanks to the algorithms from Section 2. We managed to replace `NTL::ZZ` by `long` integers almost everywhere.

To reach $\ell = 11681$, there are a few places where one temporarily needs integers bigger than what 64-bits `long` integers can allow:

(1) During the computation of the Gram matrix of the Gross lattice of maximal orders for the reduction (in particular for the computation of the norm of the second vector given by the basis $MO^N_{\mathsf{inert(A,a,b,c)}}$ from Lemma 2.2).
(2) During the computation of the adjusting coefficients in the dimension 3 CVP sub-algorithm that is part of Semaev algorithm for lattice reduction in dimension 3.
(3) In Algorithm 4, during the initial computation of γ.

Fortunately, it turns out that in all the cases listed above, the only element that goes beyond the 64-bits threshold is the result of some multiplication that needs to be divided by some other integer right after that. In some cases, the division is exact, and in some others, we want the rounding of the result, but in any case, the result that we really need is both an integer and smaller than the `long` threshold. Thus, we were able to overcome the problem by occasionally relying on the C++ `long double` floating point numbers. We didn't make a thorough analysis of the precision needed, but we validated experimentally that `long double` were enough for our purpose for levels $\ell \leq 11681$.

3.2.3. *Finite field and polynomial arithmetic.* All the improvements we have described until now were related to the quaternion part of SpecialSupersingularEvaluation. In this section, we mention several improvements we made to the implementation of [CRSEL+25] regarding the finite field and polynomial arithmetic. Those improvements mostly impact the final interpolation part of SpecialSupersingularEvaluation.

There are no new theoretical ideas in this section, so we just give a brief summary of the changes we made. In practice, those changes are quite significant, as we explain in Section 3.3.

- Better choices of modulus for both $\mathbb{F}_{p^2} = \mathbb{F}_p[i]$ and $\mathbb{F}_{p^{2^k}}$ for faster multiplication. In particular we use, $\mathbb{F}_{p^2} = \mathbb{F}_p[X]/(X^2+1)$ to be able to perform one $\mathbb{F}_{p^2}$ multiplication in 3 multiplications over $\mathbb{F}_p$. We also choose sparse polynomials to define the modulus for higher degree extensions.
- Use Karatsuba multiplication to perform $\mathbb{F}_{p^2}[X]$ multiplication with 3 $\mathbb{F}_p[X]$ multiplications.
- Faster square-root algorithms for $\mathbb{F}_{p^{2^k}}$ using the ideas of [ARH13].
- Improved implementation of the reconstruction of polynomial from its roots using a recursive algorithm.

Additional improvement: CRT batching. ModularEvaluationBigCharacteristic is essentially made of several executions of SpecialSupersingularEvaluation for different primes, followed by a CRT step for the reconstruction of the final result.

Unlike other CRT algorithms (whether it is ModularEvaluationBigLevel from [CRSEL+25] or the CRT algorithm from [Sut13]), there are very few constraints on the choice of the CRT primes p (essentially we only need that $p > \lambda\ell$ to for some constant λ to ensure there are enough supersingular j-invariant or Weber invariants over $\mathbb{F}_{p^2}$). And since the complexity of SpecialSupersingularEvaluation is actually polynomial in p, we want to use the smallest possible primes. All that is to say that we actually use quite small primes. Which might seem like a good thing, but is actually not very beneficial in practice on 64-bit machine as there is not much gain in manipulating integers much smaller than the 64 bits threshold. In particular, the library `NTL::zz_p`, which we use for computations over modular rings, is implemented on `long` integers which are 64 bits in most modern computers.

Thus, to gain time in the interpolation step where most of the `zz_p` operations are done, we first do a small CRT before the interpolation step (so directly on the j-invariants collected throughout the main loop of SpecialSupersingularEvaluation) to batch the interpolation step over a few small primes $p_1, \ldots, p_r$ such that the product $m = p_1 p_2 \ldots p_r$ is smaller than the limit set by NTL for the `zz_p` class (which is slightly below 2^{63}). The arithmetic over $\mathbb{Z}/m\mathbb{Z}$ is less efficient than the arithmetic

over each $\mathbb{F}_{p_i}$ individually, but the underlying operations over `long` integers are not much less efficient for integers smaller than m than for integers smaller than the p_i. For the sizes we consider in our computation, we are able to batch up to 4 or 5 primes into one m which ends up being noticeably faster.

3.3. **Implementation results.** We included all the improvements listed above in the C++ implementation from [CRSEL+25], and ran some benchmarks for the Weber variant of ModularEvaluationBigCharacteristic. We couldn't make the computation on the same platform as [CRSEL+25]. Instead, we ran the benchmarks single-threaded on an Intel core i7 at 2.3GHz with turbo-boost enabled. By running the two implementations on small examples, we observed that the performance on our new set-up is quite similar to the one obtained in [CRSEL+25], so we believe that the comparison is relevant.

TABLE 1. Experimental data: CPU time consumed by runs of the Weber variant of our implementation of ModularEvaluationBigCharacteristic compared to the one of [CRSEL+25] for various levels ℓ, fixing the characteristic $p = 2^{31} - 1$ and the input Weber invariant $w = 2$.

Level ℓ	Ours	[CRSEL+25]	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 min	3.4
1019	46.409 s	3.07 min	3.9
2003	2.05 m	15.13 min	7.4
4003	7.52 m	1.65 h	13.1
6007	18.93 m	5.49 h	17.4
8011	34.40 m	12.13 h	21.15
11681	1.65 h	1.56 d	22.6

Performance analysis. After a bit of profiling, we saw that for $\ell = 11681$, the total running time is divided in 3 roughly equivalent parts:

(1) The execution of OrdersTojInvariantBigSet.
(2) The quaternion operations in the main loop (Steps 9 to 13 in Algorithm 5).
(3) The polynomial reconstruction steps in the main loop (Steps 20 to 22 in Algorithm 5).

The complexity of OrdersTojInvariantBigSet is only linear in ℓ, while the rest is quadratic, but it takes some time to reach the asymptotic regime. For the smaller levels, it constitutes the main cost, and even at $\ell = 11681$, it remains non-negligible. This is due to the very high hidden constants related to the cost of computing isogenies and precomputing all the Weber invariants. We improved that part of the implementation by a factor 2 with a better finite field arithmetic, and some small improvements here and there, which explains the running time ratio of 2 at level 211.

The quaternion part is improved by a factor roughly 30 thanks to the algorithms from Section 2 and the rest of the ideas described in Section 3.2. At $\ell = 11681$, the main computational cost comes from the lattice reduction step. One problem of using a basis of a specific shape (lower triangular in our case) is that it produces vectors quite large (compared to the smallest possible ones) which implies that the lattice reduction time is bigger than it would be if we started with a smaller basis. The second cost after the lattice reduction is Algorithm 4, mainly due to the GCD computations required by the calls to MakePrimitive.

Finally, our implementation also improves the polynomial part by a factor of 20 thanks to the improvements of $\mathbb{F}_{p^2}$ arithmetic described in Section 3.2.3, along with the CRT batching.

As the cost of the quaternion part seems to be roughly equivalent to the one of the polynomial reconstruction which will both eventually dominate the cost of OrdersTojInvariantBigSet, we expect the final factor of improvement to tend toward something like 25 which is indeed what we observe, although $\ell = 11681$ is a bit too small to reach the full asymptotic regime.

We see that our practical improvements allowed us to sensibly bridge the gap between the implementation from [CRSEL+25] of ModularEvaluationBigCharacteristic and the implementation of the CRT algorithm from [Sut13]. But, despite the nice practical boost obtained with our new implementation, it seems that the performances of the implementation from [Sut13] remains a bit better for now as it was reported in [Sut13] that the implementation for $\ell = 11681$ ran in 2 hours for a prime characteristic a lot larger than the one we used (which would essentially double the running time reported in Table 1).

There are several explanations for that. First, the cost of OrdersTojInvariantBigSet is still not negligible at $\ell = 11681$. There is also the fact that due to the linear cost in p in OrdersTojInvariantBigSet, we need to use the smallest possible primes, and so we end up roughly with twice as many CRT primes as in the algorithm of Sutherland [Sut13]. As the primes remain below the 64-bit threshold in [Sut13], there is no real benefit in having smaller primes, and so our implementation loses a factor 2 from this. We mitigate a bit this loss on the polynomial reconstruction part with the CRT batching idea described at the end of Section 3.2.3, but this does not fully compensate for the loss.

The polynomial reconstruction in SupersingularEvaluation is also slowed down by the fact that we work mainly over $\mathbb{F}_{p^2}$ when everything happens over $\mathbb{F}_p$ in [Sut13]. Going from $\mathbb{F}_p$ to $\mathbb{F}_{p^2}$ incurs an additional factor 3 slowdown in the finite field arithmetic.

4. Conclusion, and open problems

We have introduced several new efficient algorithms to handle the quaternion operations required by the Deuring correspondence, and showed how these algorithms can drastically improve the practical performance of a modular polynomial evaluation algorithm from [CRSEL+25].

Open questions and future work. We believe that our algorithms could prove very useful for isogeny-based cryptography and in particular for the SQIsign protocol. The current reference implementation submitted to the NIST competition still relies on very generic algorithms and could be sped-up a lot with some of our ideas. Besides efficiency, we believe that our new algorithms would also help to control

the size of the integers involved and to obtain a constant-time version of SQIsign. This would, however, require some more work as the algorithms we have described here are tailored for our application to modular polynomial evaluation, and do not cover all the operations required in SQIsign.

Finally, despite the numerous improvements we made, the current implementation of ModularEvaluationBigCharacteristic is still not fully optimized yet. In particular, we didn't touch much on the isogeny computation code from [CRSEL+25]. It was explained in [CRSEL+25] that there is some potential of improvement there, and it would also be interesting to speed-up the implementation of ModularEvaluationBigLevel ([CRSEL+25, Algorithm 5]) that runs with a quadratic complexity in ℓ. There may also be some potential of improvement to the computation of the Weber invariants in OrdersTojInvariantBigSet. The method described in [CRSEL+25] requires to compute resultants of really big polynomials, and there may be some ways to do better on that part. It would also be interesting to investigate using a different maximal order basis than the one provided with our structural result to minimize the size of the basis given in input to the lattice reduction which is currently the main cost of the quaternion part of our implementation.

References

[AAA+25] Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac Andrés Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQIsign. Technical report, National Institute of Standards and Technology, 2025.

[ACM25] Laia Amorós, James Clements, and Chloe Martindale. Parametrizing maximal orders along supersingular ℓ-isogeny paths. Cryptology ePrint Archive, Paper 2025/033, 2025.

[ARH13] Gora Adj and Francisco Rodríguez-Henríquez. Square root computation over even extension fields. *IEEE* Transactions on Computers, 63(11):2829–2841, 2013.

[BBC+25] Andrea Basso, Giacomo Borin, Wouter Castryck, Maria Corte-Real Santos, Riccardo Invernizzi, Antonin Leroux, Luciano Maino, Frederik Vercauteren, and Benjamin Wesolowski. Prism: Simple and compact identification and signatures from large prime degree isogenies. In *I*ACR International Conference on Public-Key Cryptography, pages 300–332. Springer, 2025.

[BCRSE+25] Giacomo Borin, Maria Corte-Real Santos, Jonathan Komada Eriksen, Riccardo Invernizzi, Marzio Mula, Sina Schaeffler, and Frederik Vercauteren. Qlapoti: Simple and efficient translation of quaternion ideals to isogenies. In *I*nternational Conference on the Theory and Application of Cryptology and Information Security, pages 174–205. Springer, 2025.

[BDF+24] Andrea Basso, Pierrick Dartois, Luca De Feo, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. Sqisign2d–west: the fast, the small, and the safer. In *I*nternational Conference on the Theory and Application of Cryptology and Information Security, pages 339–370. Springer, 2024.

[BHJ+25] Andrea Basso, Chenfeng He, David Jacquemin, Fatna Kouider, Péter Kutas, Anisha Mukherjee, Sina Schaeffler, and Sujoy Sinha Roy. Constant-time quaternion algorithms for SQIsign. Cryptology ePrint Archive, Paper 2025/2192, 2025.

[BLS12] Reinier Bröker, Kristin Lauter, and Andrew Sutherland. Modular polynomials via isogeny volcanoes. *M*athematics of Computation, 81(278):1201–1231, 2012.

[Cle25] James Clements. Structural results for maximal quaternion orders and connecting ideals of prime power norm in $B_{p,\infty}$. *C*ryptology ePrint Archive, 2025.

[CRSEL+25] Maria Corte-Real Santos, Jonathan Komada Eriksen, Antonin Leroux, Michael Meyer, and Lorenz Panny. Evaluation of modular polynomials from supersingular elliptic curves. *a*rXiv preprint arXiv:2506.15429, 2025.

[DFLLW23] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New algorithms for the deuring correspondence: towards practical and secure SQIsign signatures. In *A*nnual International Conference on the Theory and Applications of Cryptographic Techniques, pages 659–690. Springer, 2023.

[DG75] Hans F De Groote. On the complexity of quaternion multiplication. *I*nformation Processing Letters, 3(6):177–179, 1975.

[DKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQIsign: Compact post-quantum signatures from quaternions and isogenies. In *ASIACRYPT* (1), volume 12491 of *L*ecture Notes in Computer Science, pages 64–93. Springer, 2020.

[DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQIsignHD: new dimensions in cryptography. In *A*nnual International Conference on the Theory and Applications of Cryptographic Techniques, pages 3–32. Springer, 2024.

[dt23] The FPLLL development team. fplll, a lattice reduction library, Version: 5.5.0. Available at https://github.com/fplll/fplll, 2023.

[Gt12] Torbjörn Granlund and the GMP development team. *GNU MP*: The GNU Multiple Precision Arithmetic Library, 5.0.5 edition, 2012. http://gmplib.org/.

[HKK+25] Otto Hanyecz, Alexander Karenin, Elena Kirshanova, Péter Kutas, and Sina Schaeffler. Constant time lattice reduction in dimension 4 with application to SQIsign. *I*ACR Transactions on Cryptographic Hardware and Embedded Systems, 2025(2):511–534, 2025.

[JMKR23] David Jacquemin, Anisha Mukherjee, Péter Kutas, and Sujoy Sinha Roy. Ready to SQI? safety first! towards a constant-time implementation of isogeny-based signature, SQIsign. *C*ryptology ePrint Archive, 2023.

[KLKL25] Won Kim, Jeonghwan Lee, Hyeonhak Kim, and Changmin Lee. Sqisign with fixed-precision integer arithmetic. *C*ryptology ePrint Archive, 2025.

[KLPT14] David Kohel, Kristin E. Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion ℓ-isogeny path problem, 2014.

[KMJK25] Fatna Kouider, Anisha Mukherjee, David Jacquemin, and Péter Kutas. Constant-time integer arithmetic for SQIsign. In *I*nternational Conference on Cryptology in Africa, pages 192–215. Springer, 2025.

[KR24] Sabrina Kunzweiler and Damien Robert. Computing modular polynomials by deformation, 2024.

[Ler23] Antonin Leroux. Computation of Hilbert class polynomials and modular polynomials from supersingular elliptic curves, 2023.

[Ler25] Antonin Leroux. Verifiable random function from the deuring correspondence and higher dimensional isogenies. In *A*nnual International Conference on the Theory and Applications of Cryptographic Techniques, pages 167–194. Springer, 2025.

[Ler26] Antonin Leroux. Efficient quaternion algorithms for the deuring correspondence, and application to the evaluation of modular polynomials. *C*ryptology ePrint Archive, 2026.

[NO24] Kohei Nakagawa and Hiroshi Onuki. QFESTA: Efficient algorithms and parameters for FESTA using quaternion algebras. In *A*nnual International Cryptology Conference, pages 75–106. Springer, 2024.

[NOC+24] Kohei Nakagawa, Hiroshi Onuki, Wouter Castryck, Mingjie Chen, Riccardo Invernizzi, Gioella Lorenzon, and Frederik Vercauteren. SQIsign2D-East: A new signature scheme using 2-dimensional isogenies. In *I*nternational Conference on the Theory and Application of Cryptology and Information Security, pages 272–303. Springer, 2024.

[NS09] Phong Q Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *A*CM Transactions on algorithms (TALG), 5(4):1–48, 2009.

[Por20] Thomas Pornin. Optimized lattice basis reduction in dimension 2, and fast schnorr and eddsa signature verification. *C*ryptology ePrint Archive, 2020.

[S+01] Victor Shoup et al. NTL: A library for doing number theory, 2001.

[Sem01] Igor Semaev. A 3-dimensional lattice reduction algorithm. In *I*nternational Cryptography and Lattices Conference, pages 181–193. Springer, 2001.

[SSC25] Vojtech Suchanek, Marek Sys, and Lukasz Chmielewski. Faster signature verification with 3-dimensional decomposition. *C*ryptology ePrint Archive, 2025.

[Sut13] Andrew Sutherland. On the evaluation of modular polynomials. In *A*NTS X, volume 1 of *T*he Open Book Series, pages 531–555. Mathematical Sciences Publishers, 2013.

DGA-MI, Bruz, France, IRMAR - UMR 6625, Université de Rennes, France,
Email address: antonin.leroux@polytechnique.org