Contemporary Mathematics

Evaluation of Modular Polynomial from Supersingular Elliptic Curves

Maria Corte-Real Santos, Jonathan Komada Eriksen, Antonin Leroux, Michael Meyer, and Lorenz Panny

Abstract. We present several new algorithms to evaluate modular polynomials of level ℓ modulo a prime p on an input j. More precisely, we introduce two new generic algorithms, sharing the following similarities: they are based on a CRT approach; they make use of supersingular curves and the Deuring correspondence; and, their memory requirements are optimal.

The first algorithm combines the ideas behind a hybrid algorithm of Sutherland in 2013 with a recent algorithm to compute modular polynomials using supersingular curves introduced in 2023 by Leroux. The complexity (holding around several plausible heuristic assumptions) of the resulting algorithm matches the $\tilde{O}(\ell^3 \log^3 \ell + \ell \log p)$ time complexity of the best known algorithm by Sutherland, but has an optimal memory requirement.

Our second algorithm is based on a sub-algorithm that can evaluate modular polynomials efficiently on supersingular j-invariants defined over $\mathbb{F}_p$, and achieves heuristic complexity quadratic in both ℓ and $\log j$, and linear in $\log p$. In particular, it is the first generic algorithm with optimal memory requirement to obtain a quadratic complexity in ℓ.

Additionally, we show how to adapt our method to the computation of other types of modular polynomials such as the one stemming from Weber's function.

Finally, we provide an optimised implementation of the two algorithms detailed in this paper, though we emphasise that various modules in our codebase may find applications outside their use in this paper.

1. Introduction

The evaluation of modular polynomials is one of the main subroutines involved in the SEA point counting algorithm [19]. The main application of point counting is to find elliptic curves suitable for cryptography. The asymptotic bottleneck of the computation relating to "Elkies primes" in the SEA algorithm is modular evaluation, and it is thus important to find improvements to the theoretical and practical efficiency of modular evaluation.

More generally, modular polynomials play an important role in the theory of elliptic curves, and so the computation and evaluation of modular polynomials is a central task in algorithmic number theory. Aside from point counting, modular polynomials are related to isogeny computations. While most applications tend to

2020 *Mathematics Subject Classification.* 11Y40.

use the more efficient Vélu formulas [44, 5], we still observe few instances where modular polynomials have been considered. For example, it is used in the CRS key exchange [15, 36], the first isogeny-based protocol, the OSIDH construction [12], the reduction introduced in [2], or the fast implementation of Delfs–Galbraith proposed in [37].

In this work, we introduce several new algorithms for efficiently computing modular polynomials Φ_ℓ of level ℓ, evaluated in one of the variables.

Related work. Almost all known methods to efficiently evaluate modular polynomials first requires their computation, which is the primary reason most literature on the topic focuses on the computation of modular polynomials.

The historical approach to computing modular polynomials is based on the computation of the coefficients of the Fourier expansion of the modular j-function [6, 19, 33, 29]. As this computation works over the integers, it can be applied to compute $\Phi_\ell \bmod p$.

An alternative approach based on the CRT method, which uses supersingular curves and ℓ-isogeny computations, was introduced by Charles and Lauter [11] to work entirely over $\mathbb{F}_p$.

Enge [20] uses interpolation and fast floating-point interpolation to obtain a quasi-linear algorithm over the integers under some heuristics.

One of the main problems behind the computation of modular polynomials over the integers is the large size of their coefficients. In theory, the size of coefficients is less of an issue when dealing with Φ_ℓ as the coefficients are reduced modulo p. However, this is only the case if we can avoid the computation over $\mathbb{Z}$ entirely, which is not easy to obtain in practice.

Bröker, Lauter, and Sutherland (BLS hereafter) [8] obtained the first quasi-linear complexity in both time and space with a careful application of the CRT method using ordinary curves.

More recently, Leroux [31] revisited the CRT approach from supersingular curves with new algorithmic results on the Deuring correspondence to obtain an algorithm with the same complexity as BLS. Leroux claims a better practical efficiency, but provided no experimental proof of this claim.

Robert [35] outlined another CRT method based on supersingular curves by using a p-adic approach in conjunction with the high-dimensional isogeny technique introduced in the context of the cryptanalysis of the SIDH key exchange. This method was later refined and implemented by Kunzweiler and Robert [27]. However, the main advantage of their method is mostly theoretical as it does not require any assumptions (not even GRH!), but it is unlikely to be practical due to a non-optimal high memory requirement. Furthermore, the implementation they provide is in SageMath [43], rather than a low-level implementation.

In 2013, Sutherland [41] provided the first evidence that a method tailored to the evaluation could do better than a computation algorithm. Sutherland showed how to adapt the CRT approach from BLS to obtain two evaluation algorithms with excellent memory requirements. However, these algorithms have essentially the same complexity as the BLS computation algorithm [8].

Contributions. Our main contribution is the introduction of three new algorithms targeted at the task of evaluating modular polynomials. Our work builds upon the previous idea of Leroux [31] to use supersingular curves with the Deuring correspondence for efficient modular polynomial computation. The fact that

his algorithms could be improved to produce a more efficient evaluation algorithm was already mentioned by Leroux, but no precise method was described or even outlined. In this work, we show that there are several interesting approaches to consider in this setting. For a fixed value of ℓ, each of our new algorithms will achieve the best known complexity for some range of primes p (relative to the size of ℓ). Hereafter, we assume that the evaluation is done on an input $j \in \mathbb{F}_p$ that can be seen as an integer $0 \leq j \leq p-1$. All the stated complexities hold under several plausible heuristic assumptions.

(1) ModularEvaluationBigCharacteristic is a generic CRT evaluation algorithm built on top of the OrdersTojInvariantBigSet algorithm from [31] which combines [41, Algorithm 2.2] and ModularComputation from [31]. The complexity is $O\left(\ell^2(\ell \log \ell + \log p) \log^{2+\varepsilon}(\ell \log \ell + \log p)\right)$ and the memory requirement is $O\left((\ell \log \ell + \log p) \log(\ell \log \ell + \log p) + \ell \log p\right)$. The asymptotic complexity is the same as [41, Algorithm 1], but the space requirement is better. This algorithm will achieve the best known space/time complexity in cases where p (i.e., the characteristic) is large.

(2) SupersingularEvaluation works on supersingular j-invariants and has a complexity of $O\left(\ell(\log p^{4+\varepsilon} + \log \ell^{2+\varepsilon} \log^{1+\varepsilon} p) + p^{1/4} \log^{3+\varepsilon} p\right)$, using $O(\ell \log p + p^{1/4} \log^{1+\varepsilon} p)$ space. Despite a limited range of applications (due to the supersingularity constraint), this algorithm achieves the best known complexity when the prime p is small, and it is the only algorithm that is linear in ℓ.

(3) ModularEvaluationBigLevel is a generic CRT evaluation algorithm built on top of SupersingularEvaluation. It has two main steps: a CRT prime collection of complexity $O\left(\ell^2 \log^2 j(\log^{2+\varepsilon}(\ell \log j))\right)$, and a CRT computation of complexity $O(\ell^2 \log j \log^{3+\varepsilon} \ell + \ell^{3/2} \log^{3/2} j \log \ell^{3+\varepsilon} + \log^{2+\varepsilon} \ell \log^{1+\varepsilon} p)$. The global space requirement is $O(\ell \log(pj))$. This algorithm achieves the best known complexity of all generic evaluation algorithm if either one of j or p is small (with respect to ℓ). In the generic case, we will have $j = \Theta(p)$, and the complexity is quadratic in $\log p$, which is worse than the other algorithms. However, note that the quadratic component of the complexity comes from the CRT prime collection, and we note that this step only depends on the input j and can be shared among evaluations of different modular polynomials on the same j. This could prove interesting for applications such as point counting.

Outside of these algorithmic contributions, we also show how to adapt our methods to work for other modular polynomials, such as the one based on the Weber function, by exploiting a modular parametrization of elliptic curves with order 48 level structure underlying the Weber function. This is very convenient in practice as the sizes of the coefficients of modular polynomials associated to the Weber function are smaller by a large constant, which makes them approximately 1728 times faster to compute than standard modular polynomials of the same level.

Finally, in Section 4, we provide an efficient implementation of all our algorithms written in C++ and NTL [38]. Unfortunately, at their current state, the implementations of our two generic algorithms do not seem to outperform the state-of-the-art implementation from [41] for parameters that are within reasonable computational reach. A more detailed analysis of the concrete performance of our implementation can be found in Section 4.2. We emphasise, however, that

many of the algorithms we implement are useful outside their application in this paper. We highlight in particular the following:

(1) An optimised implementation of polynomial interpolation, which outperforms NTL's in-built function for polynomials of large degree.
(2) An implementation of the Deuring correspondence for generic primes, thus providing a low-level optimised implementation of algorithms from [22], which outperforms that implementation by a large constant factor.
(3) An optimised implementation of the computation of the endomorphism ring of a supersingular elliptic curve defined over $\mathbb{F}_p$ for generic primes.

The last two routines are especially useful for the cryptanalysis and construction of isogeny-based cryptography.

Acknowledgements. We thank Drew Sutherland and Sam Frengley for helping us finding the maps that are needed to compute the modular polynomials associated to the Weber function.

This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ISOCRYPT – No. 101020788), by the Research Council KU Leuven grant C14/24/099, and by CyberSecurity Research Flanders with reference number VR20192203. This work was also supported by the European Research Council under grant No. 101116169 (AGATHA CRYPTY) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under SFB 1119 – 236615297.

1.1. Technical overview. Let p, ℓ be two distinct primes, and let $j \in \mathbb{F}_p$. Hereafter, we abuse notation and also write j for the unique integer between 0 and $p-1$ in the class of j modulo p. The goal of all the algorithms introduced in this work is to compute the polynomial $\Phi_\ell(X, j) \in \mathbb{F}_p[X]$.

Our work can be seen as an extension of the ideas introduced by Leroux in [31] to the setting of modular polynomial evaluation, using some ideas outlined by Sutherland in [41]. Note that the complexities of our algorithms rely on heuristic assumptions because the complexity of Leroux's algorithm are only proven assuming various heuristics.

All our algorithms rely on the Deuring correspondence and the OrdersTojInvariantSmallSet and OrdersTojInvariantBigSet algorithms from [31] whose goal is to compute the set of j-invariants corresponding (under the Deuring correspondence) to a set of maximal order types given as input. OrdersTojInvariantSmallSet is generally more efficient, unless the set of j-invariants to be computed is somewhat close to the entire set of supersingular j-invariants, where OrdersTojInvariantBigSet is tailored to be more efficient. These two algorithms were used by Leroux to build two new efficient algorithms to compute modular polynomials modulo p:

(1) SupersingularModularComputation directly applies OrdersTojInvariantSmallSet (or OrdersTojInvariantBigSet, if it is faster) to compute the necessary j-invariants to interpolate $\Phi_\ell(X, Y)$ over $\mathbb{F}_p$.
(2) ModularComputation is a CRT algorithm that applies SupersingularModularComputation on a set of small CRT primes p_i before reconstructing the result mod p. Here, SupersingularModularComputation is always used with OrdersTojInvariantBigSet because the p_i are small compared to ℓ.

The approach by Leroux is amenable to ideas introduced by Sutherland [41] to adapt the BLS modular polynomial computation algorithm to the evaluation setting. Sutherland observed that there are essentially two ways to evaluate Φ_ℓ on $j \in \mathbb{F}_p$. The first and most direct way is to consider j as an integer, compute $\Phi_\ell(X, j)$ over the integers and finally reduce the result modulo p. This leads to [41, Algorithm 2] which has optimal space complexity, but can be quite inefficient when $\log p$ grows and $j = \Theta(p)$. The main problem with that method is that the powers of j are much bigger than j over $\mathbb{Z}$. This increases the size of the coefficients of $\Phi_\ell(X, j) \in \mathbb{Z}[X]$ which has a negative impacts on the performances of this approach.

Sutherland's trick is to realize that we can avoid exponentiating over $\mathbb{Z}$ by exponentiating over $\mathbb{F}_p$ where the powers of j are all in $O(p)$. Thus, he proposes to lift each element $j^k \in \mathbb{F}_p$ to an integer $\widehat{j^k}$ for $1 \leq k \leq \ell + 1$, see $\Phi_\ell(X, j)$ as a multivariate polynomial $P_\ell(X, j_1, \ldots, j_{\ell+1})$ by replacing each j^k by a variable j_k and evaluate this polynomial on the $\widehat{j^k}$ before reducing the result modulo p. This gives [41, Algorithm 1], which obtains the best known complexity but requires more space than the more naive approach. This method has the same complexity as computing the whole modular polynomial, but requires less memory as the full polynomial is never stored in memory.

Sutherland also presents a "hybrid" version of [41, Algorithm 1] (see [41, Section 3.4]). This hybrid algorithm optimises the amount of data stored throughout the computation, and so it is slower but uses an optimal amount of space.

Our first algorithm, ModularEvaluationBigCharacteristic can be seen as an adaptation of Leroux's ModularComputation with the ideas behind Sutherland's hybrid algorithm. Similarly to Sutherland's idea, we lift powers of j to integers $\widehat{j_1}, \ldots, \widehat{j_{\ell+1}}$ before evaluating the polynomial $P_\ell(X, \widehat{j_1}, \ldots, \widehat{j_{\ell+1}})$ with a CRT method. The computation of $P_\ell(X, \widehat{j_1}, \ldots, \widehat{j_{\ell+1}})$ modulo the small CRT primes p_i mainly relies on OrdersTojInvariantBigSet in a manner similar to SupersingularModularComputation. It turns out that this method requires less storage than the version using ordinary curves, and so we can execute the analogue of Sutherland's hybrid version without any negative impact on the complexity.

Thus, just as ModularComputation matches the complexity of BLS, ModularEvaluationBigCharacteristic essentially matches the complexity of [41, Algorithm 1] taking $O\left(\ell^2(\ell \log \ell + \log p) \log^{2+\varepsilon}(\ell \log \ell + \log p)\right)$, but with a better space complexity of $O\left((\ell \log \ell + \log p) \log(\ell \log \ell + \log p) + \ell \log p\right)$ that is quasi-linear in ℓ.

Our second generic evaluation algorithm ModularEvaluationBigLevel applies the naive approach of computing $\Phi_\ell(X, j)$ over $\mathbb{Z}[X]$, but compensates the efficiency loss caused by the large size of the powers of j in $\mathbb{Z}$ by noticing that modulo some well-chosen CRT primes p_i, the computation of $\Phi_\ell(X, j) \bmod p_i$ can be made much more efficient than the computation of $P_\ell(X, \widehat{j_1}, \ldots, \widehat{j_{\ell+1}}) \bmod p_i$ (which is essentially equivalent to the computation of the full $\Phi_\ell(X, Y) \bmod p_i$). The well-chosen primes are those where $j \bmod p_i$ is the j-invariant of a supersingular curve and this efficient algorithm is our third algorithmic contribution: SupersingularEvaluation, whose complexity is linear in ℓ. The main downside of this idea is that the probability that $j \bmod p_i$ is supersingular is in $O(1/\sqrt{p_i})$ which makes the computation of the CRT primes quite costly with a complexity of $O\left(\ell^2 \log^2 j \log^{2+\varepsilon}(\ell \log j)\right)$. In the generic case where $\log j = \theta(\log p)$, this step will be the bottleneck.

Once the CRT primes have been computed, the rest of the computation takes $O\left(\ell^2 \log j \log \ell^{3+\varepsilon} + \ell^{3/2} \log^{3/2} j \log^{2+\varepsilon} \ell + \log^{2+\varepsilon} \log^{1+\varepsilon} p\right)$, and the overall space requirement is $O(\ell \log(pj))$ which is optimal given the size of the output.

SupersingularEvaluation consists of a rather straightforward application of OrdersTojInvariantSmallSet to compute the j-invariants ℓ-isogenous to the supersingular j-invariant given as input. This gives a linear complexity in ℓ, but comes at the cost of computing the endomorphism ring of the input curve, which takes $O(p^{1/4} \log p^{3+\varepsilon})$. The final complexity is

$$O\left(\ell(\log \ell^{2+\varepsilon} \log p^{1+\varepsilon} + \log p^{4+\varepsilon}) + p^{1/4} \log p^{3+\varepsilon}\right).$$

We note that SupersingularEvaluation is interesting in its own right, as it achieves the best known complexity to evaluate Φ_ℓ when the prime p is small. However, the constraint of having a supersingular j-invariant as input is quite limiting for practical applications.

Other modular functions. We also show how to adapt our method to compute modular polynomials associated to other modular functions such as the Weber function. It was already suggested by Leroux in [31] that, similarly to the BLS algorithm, the supersingular approach could be adapted to work for other modular functions. However, unlike the prediction by Leroux, who qualified the task as "not too daunting", concretely implementing this comes with a few technical obstacles. In particular, the approach taken in BLS cannot be made to work here. Our solution uses a level structure parametrization of Weber invariants and elimination theory to solve the issue.

Organisation of the article. The rest of this paper is organised as follows: in Section 2, we introduce some background on isogenies, quaternion algebras and the Deuring correspondence. The main technical contributions are introduced in Section 3 where we present our new algorithms. Finally, in Section 4, we give some details on our implementation.

2. Background material

Throughout the paper, we consider logarithm in base 2 that we simply write $\log(\cdot)$, and we use $\mathrm{llog}(\cdot)$ for $\log(\log(\cdot))$.

Background on the main topics covered in this article can be found in:

(1) Silverman [24] for elliptic curves and isogenies;
(2) Voight [45] for quaternion algebras
(3) the thesis of Leroux [30] for the algorithmic aspects of the Deuring correspondence.

In short, the Deuring corrrespondence makes a link between elliptic curves and isogenies over $\mathbb{F}_p$ on one side, and orders and ideals of the quaternion algebra ramified at p and ∞ on the other side. This result thus provides a way to navigate efficiently the supersingular isogeny graph by using some simple operations over some lattices of dimension 4. This is what we call the effective Deuring correspondence.

Effective Deuring correspondence. More precisely, one important algorithm for us is a heuristic polynomial-time algorithm introduced in [18] which computes the j-invariant corresponding to a maximal order type given as input.

It makes use of a smooth isogeny connecting the desired j-invariant to a special j-invariant (for instance $j = 1728$ or $j = 0$). This smooth isogeny is found using the Deuring correspondence with the KLPT algorithm [25]. The smooth isogeny found with KLPT is usually quite large (the heuristic estimates suggest that one should be able to find an isogeny of degree $\tilde{O}(p^3)$ with that algorithm), and so the concrete computation of this isogeny, for a generic prime p, can be quite slow. The asymptotic cost of this computation is $O(\log^{4+\varepsilon} p)$, but the constants hidden in that complexity are dauntingly large for generic primes. Nevertheless, in [22] it was shown that this algorithm can be made practical, even for generic primes, by applying several practical improvements to the original algorithm from [18] Henceforth, we call this algorithm SingleOrderTojInvariant().

Remark 2.1. There seems to be a typo in [31] where the cost of this algorithm is estimated to be $O(\log p^{5+\varepsilon})$. However, if we look at [23, Lemma 4] we see that the cost should be of $O(\log^{4+\varepsilon} p)$ (after a precomputation of $\log^{6+\varepsilon} p$ that can be amortized across all isogeny computations).

Efficient Deuring correspondence for several orders. In [31], Leroux looked at the problem of the effective Deuring correspondence for a set of order types. For a small number of orders (relative to the prime characteristic p), the best method seems to be simply to apply SingleOrderTojInvariant() to each order in the set. Leroux called this algorithm OrdersTojInvariantSmallSet, and we will keep that notation throughout the paper. If S is the size of the set of orders to be computed, the complexity of this algorithm is $O(S \log^{4+\varepsilon} \log p)$.

However, when the set contains a lot of order types, Leroux introduced a much faster algorithm called OrdersTojInvariantBigSet (the algorithm really becomes interesting for sets of size $O(p)$). The idea of this algorithm is to go through the entire set of supersingular curves through quaternions and collect the desired j-invariants along the way in order to minimize the number of isogeny computations required.

The asymptotic complexity of this algorithm is $O(S \log p^{2+\varepsilon} + p \log p^{1+\varepsilon})$, and it also has better hidden constants than OrdersTojInvariantSmallSet due to the fact that it involves the computation of isogenies of degree $O(p)$ instead of $O(p^3)$.

Application of the Deuring correspondence to the computation of modular polynomials. The principle that "quaternion operations" are generally more efficient than "elliptic curve operations" is at the heart of the recent algorithm ModularComputation of Leroux [31] for computing modular polynomials. Indeed, the main part of this computation is to collect pairs of ℓ-isogenous j-invariants. By working with supersingular j-invariants, Leroux showed how to exploit the effective Deuring correspondence to realize that collection with a minimal amount of "elliptic curve" operations by working mostly with quaternions. The idea is that the set of maximal order types corresponding to the j-invariants to be collected can be generated purely from quaternion operations. Then, it suffices to apply the algorithms we mention above to collect all the necessary j-invariants.

3. Evaluation of modular polynomials

In this section, we introduce our main theoretical contributions, namely the three new algorithms ModularEvaluationBigCharacteristic, SupersingularEvaluation, and ModularEvaluationBigLevel (respectively introduced in Sections 3.1 to 3.3). In Section 3.4, we provide a concrete comparison with several other algorithms from

the literature. Finally, in Section 3.5, we explain how to adapt our method to evaluate modular polynomials associated with the Weber invariant.

3.1. A first CRT approach for big characteristic. Our first CRT approach is inspired by the "hybrid" algorithm described by Sutherland but where [41, Algorithm 2.2] is replaced with an algorithm SpecialSupersingularEvaluation described below. This algorithm combines the ideas from SupersingularModularComputation from [31], with the trick introduced by Sutherland to perform the evaluation with a minimal space requirement. The part of our algorithm that constructs the final output by CRT is identical to the one in [41, Algorithm 2]. We therefore do not describe these steps precisely, and instead refer the reader to [40, Section 6] for more details.

Algorithm 1 SpecialSupersingularEvaluation$(p, \ell, x_0, \ldots, x_\ell)$

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

1: Compute the set of maximal order types $\mathcal{O}_1, \ldots, \mathcal{O}_m \subset \mathcal{B}_{p,\infty}$ and set $\mathfrak{S}$ as the list of these maximal order types
2: Compute $J =$ OrdersTojInvariantBigSet$(p, \mathfrak{S})$
3: Select a set $\mathcal{O}_{1,0}, \ldots, \mathcal{O}_{\ell+2,0}$ in $\mathfrak{S}$
4: **for** $i = 1$ to $\ell + 2$ **do**
5: Find $j_{i,0}$ as the j-invariant in J corresponding to $\mathcal{O}_{i,0}$
6: Compute $I_{i,1}, \ldots, I_{i,\ell+1}$, the $\ell + 1$ $\mathcal{O}_{i,0}$-ideals of norm ℓ
7: **for** $k = 1$ to $\ell + 1$ **do**
8: Select $j_{i,k}$ as the j-invariant in J corresponding to $\mathcal{O}_R(I_{i,k})$
9: **end for**
10: $P(j_{i,0}, Y) \leftarrow \prod_{k=1}^{\ell+1} (Y - j_{i,k})$
11: Write $P(j_{i,0}, X) = \sum_{k=1}^{\ell+1} c_{i,k-1} X^{k-1}$
12: $y_i \leftarrow \sum_{k=1}^{\ell+1} c_{i,k-1} x_{k-1}$
13: **end for**
14: Interpolate $P(Y) \bmod p$ as the polynomial of degree $\ell + 1$ with $P(j_{i,0}) = y_i$.
15: **return** $P(Y)$.

Proposition 3.1. SpecialSupersingularEvaluation *is correct, and can be executed in* $O(\ell^2 \log^{2+\varepsilon} \ell \log^{1+\varepsilon} p + p \log^{2+\varepsilon} p)$*, and requires* $O(\ell \log p + p \log p)$ *space.*

Proof. All maximal orders in $\mathcal{B}_{p,\infty}$ can be enumerated in $O(p \log p)$ with the techniques described for SupersingularModularComputation from [31]. By [31, Theorem 1], the running time of OrdersTojInvariantBigSet is $O(p \log^{2+\varepsilon} p)$. Thus, all the steps before the main loop can be run in $O(p \log^{2+\varepsilon} p)$.

Computing an ideal of norm ℓ takes $O(\log(\ell p))$. Thus, the asympotic cost of the loop is dominated by the cost of the polynomial reconstruction at the end which takes $O(\ell \log^{2+\varepsilon} \ell \log^{1+\varepsilon} p)$ with the fast interpolation algorithms. Since the loop is repeated ℓ times, the final cost is $O(\ell^2 \log^{2+\varepsilon} \ell \log^{1+\varepsilon} p)$.

The sets $\mathfrak{S}$ and J use $O(p \log p)$ space to store, and the space requirement of OrdersTojInvariantBigSet is $O(p \log p)$. The amount of space for each iteration of the loop is $O(\ell \log p)$ to store the j-invariants and the polynomials $P(j_{i,0}, Y)$.

After each iteration of the loop, we need to store one value in $\mathbb{F}_p$, so the total is $O(\ell \log p)$. □

Remark 3.2. The algorithm outlined as Algorithm 1 is correct up to some small detail: order types correspond to j-invariants up to Galois conjugacy. Since we work over $\mathbb{F}_{p^2}$, this means that there can be up to two distinct j-invariants with the same endomorphism ring (j and j^p). In practice, this means that our algorithm needs a small modification to know which one of the two is the correct among $j_{i,k}$ and $j_{i,k}^p$. This can be done by modifying OrdersTojInvariantBigSet to store not only couples of order types and j-invariants, but also ideals whose corresponding isogeny's codomain is the computed j-invariant. Then, we can check if the j-invariant corresponding to $I_{i,k}$ is $j_{i,k}$ or $j_{i,k}^p$ by checking if $I_{i,k}$ is equivalent or not to the ideal stored along with the type of $\mathcal{O}_R(I_{i,k})$.

Algorithm 2 ModularEvaluationBigCharacteristic(p, j, ℓ)

Input: A prime p, $j \in \mathbb{F}_p$, a prime ℓ.
Output: $\Phi_\ell(j, Y)$.

Let $(\overline{j_i})$ be the integers in $[0, p-1]$ equal to $j^i \bmod p$ for $1 \le i \le \ell+1$
Set $B = 2^{6\ell \log \ell + 18\ell + \log p + \log(\ell+2)}$.
Set $\mathcal{P}_\ell = \{\}$, $P = 1$, $q = 12(\ell+2)+1$.
while $P < B$ **do**
  if q is prime **then**
    $P \leftarrow q \cdot P$, $\mathcal{P}_\ell \leftarrow \mathcal{P}_\ell \cup \{q\}$
  end if
  $q \leftarrow q + 2$
end while
Perform the pre-computations for the explicit CRT mod p using $\mathcal{P}_\ell$
for $q \in \mathcal{P}_\ell$ **do**
  Set j_q to be the vector $(\overline{j_i})$ reduced mod$q$.
  $P_q(Y) \leftarrow$ SpecialSupersingularEvaluation(q, ℓ, j_q)
  Update the CRT sums for each coefficient of $P_q(Y)$
end for
Perform the post-computation for the explicit CRT to obtain $P(Y) \in \mathbb{F}_p[X]$
return $P(Y)$

Proposition 3.3. *The expected running time of* ModularEvaluationBigCharacteristic *is* $O(\ell^2(\ell \log \ell + \log p) \log^{2+\varepsilon}(\ell \log \ell + \log p))$ *and requires* $O((\ell \log \ell + \log p) \log(\ell \log \ell + \log p) + \ell \log p)$ *memory.*

Proof. With the same reasoning as in the proof of the expected running time of ModularComputation in [31], we can take $\#\ \mathcal{P}_\ell = O(\log B / \mathrm{llog} B)$ and $\max_{q \in \mathcal{P}_\ell} q = O(\log B)$. We obtain the result by combining $\log B = O(\ell \log \ell + \log p)$ with Proposition 3.1, and the cost of the CRT operations given in [41]. □

3.2. The supersingular case. Our algorithm to evaluate the modular polynomial on the j-invariant j of a supersingular curve is simple to describe. It mainly consists in the application of the OrdersTojInvariantSmallSet (or OrdersTojInvariantBigSet) algorithms from [31] to collect the j-invariants required to reconstruct

$\Phi_\ell(j, X)$ from its roots. However, the input of these algorithms are the maximal orders isomorphic to the endomorphism ring of the curve ℓ-isogenous to j. The simplest way to compute these maximal orders is to know the endomorphism ring associated to j. Hence, the first building block we need is an algorithm to compute the endomorphism ring of a supersingular curve defined over $\mathbb{F}_p$.

Computing the endomorphism ring of a supersingular elliptic curve over $\mathbb{F}_p$. We introduce an algorithm EndomorphismRing of heuristic complexity $\widetilde{O}(p^{1/4})$. This algorithm relies on a subroutine that computes isogenies between two given supersingular curves over $\mathbb{F}_p$, which can morally be traced back to [17, Algorithm 1]. In this paper, it is simply refined to be efficient in practice and allow the computation of the ideal corresponding to the computed isogeny (via the Deuring correspondence). We then simply apply this algorithm between the target curve E and some starting curve E_0 of known endomorphism ring.

The main property behind this algorithm is that the set of supersingular curves E defined over $\mathbb{F}_p$ with a given $\mathbb{F}_p$-endomorphism ring $R := \mathrm{End}(E) \cap \mathbb{Q}(\pi)$, where $\pi\colon E \to E$ is the p-power Frobenius endomorphism, admits (very few exceptional cases exempt) a free and transitive group action by $\mathrm{Cl}(R)$. In the supersingular setting, the only two choices are $R = \mathbb{Z}[\pi] \cong \mathbb{Z}[\sqrt{-p}]$ and $R = \mathbb{Z}[\frac{1+\pi}{2}] \cong \mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$. Thus, the set of supersingular curves over $\mathbb{F}_p$ with a given $\mathbb{F}_p$-endomorphism ring can be obtained by applying the action of a set of elements spanning $\mathrm{Cl}(R)$ on any given starting curve $E_0/\mathbb{F}_p$ with the correct choice of $\mathbb{F}_p$-endomorphism ring ($\mathbb{Z}[\pi]$ or $\mathbb{Z}[\frac{1+\pi}{2}]$, corresponding to the surface or floor of the 2-isogeny volcano). Heuristically, we can obtain such a set by considering the combination of small powers of $O(\log p)$ ideal classes in $\mathrm{Cl}(R)$. This is formalized as Heuristic 1.

This heuristic result is convenient in practice because computing a policyclic representation as in [40, Algorithm 2.2] requires a number of operations linear in the class number, whereas constructing the set used in the heuristic method can be done in polylogarithmic time.

HEURISTIC 1. *Write* $R \in \left\{\mathbb{Z}[\pi], \mathbb{Z}[\frac{1+\pi}{2}]\right\}$. *There exists a constant* C *such that for any prime* p, *any pair of elements* $[\mathfrak{g}_1], [\mathfrak{g}_2]$ *in* $\mathrm{Cl}(R)$, *any set of ideals* $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$ *with pairwise distinct odd prime norms in* R, *and set of exponents* $e_1, \ldots, e_n$ *such that* $\prod_{i=1}(4e_i + 1) > Cp^{1/2} \log p$, *there exists* $\mathfrak{a} = \prod_{i=1}^n \mathfrak{l}_i^{b_i}$ *with each* $b_i \in [-2e_i, 2e_i]$ *such that* $[\mathfrak{g}_1] = [\mathfrak{a}][\mathfrak{g}_2]$ *in* $\mathrm{Cl}(R)$.

Proposition 3.4. *Assuming Heuristic 1,* EndomorphismRing *always returns a maximal order isomorphic to the endomorphism ring of the input curve. The expected running time of* EndomorphismRing *is* $O(p^{1/4} \log p^{3+\varepsilon})$, *and the expected memory cost is* $O(p^{1/4} \log p^{2+\varepsilon})$. *The size of the output is* $O(\log p)$.

PROOF. Under *GRH*, the smallest value of q such that $\sqrt{-q}$ is contained in $\mathcal{B}_{p,\infty}$ is in $O(\log^2 p)$ by a result of Ankeny [1]. Thus, with the CM method, a curve E_0 can be found using $O(\mathsf{poly}(\log(p)))$ binary operations.

By Heuristic 1 and [10, Theorem 7], all supersingular j-invariants defined over $\mathbb{F}_p$ will be spanned by $\left(\prod_{i=1}^n \mathfrak{l}_i^{b_i}\right) * j_0$, for $b_i \in [-2e_i, 2e_i]$ with overwhelming probability. Thus, we know there will be a collision in J and J_0.

Under GRH, if we select the norms $\ell_1, \ldots, \ell_n$ of the ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$ as the smallest primes that are split in $\mathbb{Z}\sqrt{-n}$, then we have $\mathfrak{l}_i = O(\log p^{1+\varepsilon})$ for all i, and we have $n = O(\log p)$.

Algorithm 3 EndomorphismRing(p, j)

Input: A prime p, and the j-invariant of a supersingular curve over $\mathbb{F}_p$, denoted j.
Output: $\perp$, or maximal order $\mathcal{O} \in \mathcal{B}_{p,\infty}$ isomorphic to $\mathrm{End}(E)$ where $j(E) = j$.

Let $\mathcal{O}_0$ be one of the maximal orders of $\mathcal{B}_{p,\infty}$ given in [25, Lemma 2,3,4]
Compute E_0 a supersingular curve defined over $\mathbb{F}_p$ of known endomorphism ring whose endomorphism ring is isomorphic to $\mathcal{O}_0$
Compute a set of ideals $\mathfrak{l}_1, \dots, \mathfrak{l}_n$ in $\mathfrak{O}$ of odd prime norm $\ell_1, \dots, \ell_n$ in $\mathfrak{O}$ and exponents $e_1, \dots, e_n < 10$ such that $\prod_{i=1}^{n}(4e_i + 1) > Cp^{1/2}\log p$
Set $J_0 = \{(j_0, \mathfrak{O})\}$
Set $J = \{(j, \mathfrak{O})\}$
for $i = 1$ to n **do**
  for $e \in [-e_i, e_i]$ and $(j_1, \mathfrak{a}_1), (j_2, \mathfrak{a}_2) \in J_0 \times J$ **do**
    $J_0 = J_0 \cup \{(\mathfrak{l}_i^b \star j_1, \mathfrak{l}_i^b \mathfrak{a}_1)\}$
    $J = J \cup \{(\mathfrak{l}_i^b \star j_2, \mathfrak{l}_i^b \mathfrak{a}_2)\}$
  end for
  if there is a collision $(j', \mathfrak{a}), (j', \mathfrak{b}) \in J_0 \times J$ **then**
    return $\mathcal{O} = \mathcal{O}_R(\mathcal{O}_0 \mathfrak{a}\mathfrak{b}^{-1})$
  end if
end for
return $\perp$

The computation of isogenies of degree ℓ_i can be done in $O(\ell_i M_{\mathbb{P}}(\ell_i) M_{\mathbb{Z}}(\log p))$, as Vélu's formulas can be computed in $O(\ell_i)$ operations over the field of definition of the kernel. The degree of the field of definition of the ℓ_i-torsion over $\mathbb{F}_p$ is $O(\ell_i)$ (see [31, Lemma 2] for instance). And so arithmetic operations over that field can be performed in $O(M_{\mathbb{P}}(\ell_i) M_{\mathbb{Z}}(\log p))$.

The kernel of the isogenies realizing the action of $\mathfrak{l}_i$ can be computed as the eigenvalue of the Frobenius morphism. These points can be computed by evaluating the Frobenius on a basis of the ℓ_i-torsion. The overall cost of this operation is $O(\sqrt{\ell_i} M_{\mathbb{P}}(\ell_i) M_{\mathbb{Z}}(\log p))$.

With $\ell_i = O(\log p)$, we get the final bound of $O(\log p^{3+\varepsilon})$ on the cost of each group action computation.

Under GRH, we have $h(\mathfrak{O}) = O(\sqrt{p}\,\mathrm{llog}(p))$, and so, by the birthday paradox, the number of j-invariants that needs to be computed in the sets J and J_0 before a collision can be found is in $O(p^{1/4}\log^{\varepsilon} p)$. Thus, the expected running time of EndomorphismRing is $O(p^{1/4}\log^{3+\varepsilon} p)$.

The order $\mathcal{O}_0$ can be computed with coefficients over $\mathbb{Q}$ of size $O(\log p)$ (see [25], for instance). The computation of $\mathcal{O}$ can be done in $O(\log p)$ as the coefficients of the ideal $\mathcal{O}\mathfrak{a}\mathfrak{b}^{-1}$ are in $O(p)$. The maximal order $\mathcal{O}$ can be given by 16 coefficients over $\mathbb{Q}$. Since $\mathcal{O}$ it is the right order of an ideal of norm in $O(p)$, the size of the coefficients is $O(\log p)$. The j-invariants and the ideals take $O(\log p)$ to store so the total memory cost is $O(p^{1/4}\log^{1+\varepsilon} p)$.

The output is the right order of the ideal $\mathcal{O}_0\mathfrak{a}\mathfrak{b}^{-1}$. This ideal has norm $O(\log p)$ and so the coefficients of its right order over the basis $1, i, j, k$ of $\mathcal{B}_{p,\infty}$ can be upper-bounded by $O(\log p)$. Indeed, it can be easily verified that $\mathcal{O}n(\mathfrak{a}\mathfrak{b}) \subset \mathcal{O}_0$, and so we can express a basis of $\mathcal{O}$ as elements of $\mathcal{O}_0$ divided by $n(\mathfrak{a}\mathfrak{b})$. This gives the desired upper-bound on the size of $\mathcal{O}$. □

The supersingular evaluation algorithm. Computing $\Phi_\ell(j, X)$ mainly consists in computing the j-invariants that are ℓ-isogenous to j. When j is supersingular, we run this operation without any ℓ-isogeny computation by computing the endomorphism ring of j and using the Deuring correspondence to compute the ℓ-isogenous j-invariants.

Algorithm 4 SupersingularEvaluation(p, j, ℓ)

Input: A prime p, the j-invariant of a supersingular curve over $\mathbb{F}_p$, a prime ℓ.
Output: $\Phi_\ell(j, Y)$.

1: Compute E the curve whose j-invariant is j
2: Compute $\mathcal{O} =$ EndomorphismRing(p, j)
3: Compute $I_1, \cdots, I_{\ell+1}$ the $\ell + 1$ $\mathcal{O}$-ideals of norm ℓ
4: Set $\mathfrak{S} = \{\mathcal{O}_R(I_i) | i \in \{1, \ldots, \ell + 1\}\}$
5: Compute $J =$ OrdersTojInvariantSmallSet$(p, \mathfrak{S})$ (or use OrdersTojInvariantBigSet$(p, \mathfrak{S})$ if this is more efficient)
6: **return** $\prod_{j_i \in J}(Y - j_i)$

Proposition 3.5. SupersingularEvaluation *can be executed in*

$$O\left(\ell(\log \ell^{2+\varepsilon} \log p^{1+\varepsilon} + \log p^{2+\varepsilon}) + p \log p^{1+\varepsilon}\right)$$

binary operations and requires $O((\ell + p) \log p)$ *space when* OrdersTojInvariantBigSet *is used in Step 5 (assuming GRH,*[**31**, Claim 1] *and Heuristic 1). When using* OrdersTojInvariantSmallSet*, it can be executed in*

$$O\left(\ell(\log p^{4+\varepsilon} + \log \ell^{2+\varepsilon} \log^{1+\varepsilon} p) + p^{1/4} \log^{3+\varepsilon} p\right)$$

binary operations, requiring $O(\ell \log p + p^{1/4} \log^{1+\varepsilon} p)$ *space (under GRH, Heuristic 1 and the heuristics from* [**25**]*).*

Proof. The correctness follows directly from the correctness of the sub-algorithms.

The computation of an $\mathcal{O}$-ideal of norm ℓ can be done in $O(C + \log \ell)$ where C is a bound on the size of the coefficients of $\mathcal{O}$. Thus, by Proposition 3.4, the complexity of computing the $O(\ell)$ ideals is $O(\ell(\log \ell + \log p))$. The computation of the rights orders has the same complexity.

Then, the result follows from [**31**, Theorem 1] for OrdersTojInvariantBigSet, the complexity of $O(\ell \log^{4+\varepsilon} p)$ for OrdersTojInvariantSmallSet for a set of size $O(\ell)$, Proposition 3.5 for EndomorphismRing and the complexity of the algorithm to compute a polynomial from its roots by building a product tree.

The memory requirements follow from the same results. □

3.3. A second CRT algorithm for big level. We obtain our second generic algorithm to evaluate modular polynomials by applying SupersingularEvaluation on a set of well-chosen small primes and then reconstructing the desired result using the CRT method. The only constraint on the choice of the CRT primes is that we need to find primes p_i where the reduction modulo p_i of the j-invariant to be evaluated is supersingular.

Algorithm 5 ModularEvaluationBigLevel(p, j, ℓ)

Input: A prime p, $j \in \mathbb{F}_p$, a prime ℓ.
Output: $\Phi_\ell(j, Y) \in \mathbb{F}_p[X]$.

Let $\bar{j}$ be the integer in $[0, p-1]$ equal to $j \mod p$
Set $B = 2^{6\ell \log \ell + 18\ell + (\ell+1)\log j + \log(\ell+2)}$
$\mathcal{P}_\ell(j) \leftarrow \{\}$, $P \leftarrow 1$
$\Delta \leftarrow \lceil \log B \rceil$
Compute $\mathcal{P}_\Delta$ as the set of primes smaller than Δ with Eratosthenes sieve
$q_\delta \leftarrow \max \mathcal{P}_\Delta$, $n \leftarrow q_\Delta + 2$
while $P < B$ **do**
  $S \leftarrow [n, n+1, \ldots, n+\Delta]$
  Remove all multiples of the elements of $\mathcal{P}_\Delta$ from S
  for $q \in S$ **do**
    Let $j_q = \bar{j} \mod q$ and E_q be the elliptic curve over $\mathbb{F}_q$ of j-invariant equal to j_q
    if E_q is supersingular **then**
      $P \leftarrow q \cdot P$, $\mathcal{P}_\ell(j) \leftarrow \mathcal{P}_\ell(j) \cup \{q\}$
    end if
  end for
  $n \leftarrow n + \Delta + 1$
  if $q_\Delta^2 \leq n$ **then**
    $S = [q_\Delta + 2, q_\Delta + 3, \ldots, 2q_\Delta + 1]$
    Remove all multiples of the elements of $\mathcal{P}_\Delta$ from S
    $\mathcal{P}_\Delta \leftarrow \mathcal{P}_\Delta \bigcup S$
    $q_\Delta \leftarrow \max \mathcal{P}_\Delta$
  end if
end while
Perform the pre-computations for the explicit CRT mod p using $\mathcal{P}_\ell$
for $q \in \mathcal{P}_\ell$ **do**
  $P_q(Y) \leftarrow$ SupersingularEvaluation(q, j_q, ℓ)
  Update the CRT sums for each coefficient of $P_q(Y)$
end for
Perform the post-computation for the explicit CRT to obtain $P(Y) \in \mathbb{F}_p[X]$
return $P(Y)$

Proposition 3.6. ModularEvaluationBigLevel *is correct and, under the Lang–Trotter Conjecture* [28]*, Heuristic 1, and the heuristic from* [25]*, the computation of* $\mathcal{P}_\ell(j)$ *can be done in*

$$O\left(\ell^2 \log^2 j (\log^{1+\varepsilon}(\ell \log j))\right)$$

and the rest of the computation in

$$O\left(\ell^2 \log j \log^{3+\varepsilon} \ell + \ell^{3/2} \log^{3/2} j \log \ell^{3+\varepsilon} + \ell \log^{2+\varepsilon} \log^{1+\varepsilon} p\right).$$

The memory complexity is $O(\ell \log(pj))$ *for each step.*

Proof. According to the Lang–Trotter conjecture, there are $O(\sqrt{x}/\log x)$ primes q smaller than x such that the reduction of any given $j \in \mathbb{Q}$ modulo q is supersingular. To ensure that their product is bigger than B, we can take $x = O(\log^2 B)$, and there will be $O(\log B / \mathrm{llog} B)$ different primes.

With $\log B = O(\ell \log j)$, we obtain $\#\mathcal{P}_\ell(j) = O(\ell \log j / \log(\ell \log j))$ and so $\max_{q \in \mathcal{P}_\ell(j)} q = O(\ell^2 \log j^2)$. The set $\mathcal{P}_\ell(j)$ takes $O(\ell \log j)$ memory to store.

The algorithm we propose to use to compute $\mathcal{P}_\ell(j)$ is a simple variation of the segmented sieve of Bays and Hudson [4] that can enumerate through all primes smaller than a given x in $O(x \mathrm{llog} x)$ complexity while using $O(\sqrt{x})$ space when $\Delta = O(\sqrt{x})$, and combine it with supersingularity tests on the fly (to avoid storing a list of all the primes smaller than x). Supersingularity testing over $\mathbb{F}_q$ can be performed in $O(\log^{2+\varepsilon} q)$ [42], and it is only performed on prime numbers. Thus, we end up with a total complexity of $O\left(\ell^2 \log^2 j (\log^{1+\varepsilon}(\ell \log j))\right)$ and a space complexity of $O(\ell \log j)$ to compute $\mathcal{P}_\ell(j)$.

For each q, by Proposition 3.1 and the fact that $q = O(\ell^2 \log j)$, the complexity of the execution of SupersingularEvaluation with OrdersTojInvariantSmallSet (which will be faster than using OrdersTojInvariantSmallSetbecause $p = \Theta(\ell^2 \log^2 j)$) is

$$O\left(\ell(\log^{4+\varepsilon}(\ell \log j) + \log^{2+\varepsilon} \ell \log^{1+\varepsilon}(\ell \log j)) + \ell^{1/2} \log j^{1/2} \log(\ell \log j)^{3+\varepsilon}\right).$$

We deduce that the global cost of all the executions of SupersingularEvaluation is

$$O\left(\ell^2 \log j \log^{3+\varepsilon}(\ell \log j) + \ell^{3/2} \log^{3/2} j \log^{2+\varepsilon}(\ell \log p)\right).$$

Finally, the cost of the CRT computation is $O(\log^{2+\varepsilon} \log^{1+\varepsilon} p)$ as shown in [8]. This concludes the proof of the final complexity result.

The prime search can be done with $O(\ell \log j)$ memory (to store the set $\mathcal{P}_\ell$). Each execution of SupersingularEvaluation requires a memory of

$$O(\ell \log(\ell \log j)) + \ell^{1/2} \log^{1/2} j \log^{1+\varepsilon} \log(\ell \log j).$$

The only thing to store between each CRT prime computation are the CRT data whose size is $O(\ell \log p)$. This proves the result on the memory requirement. □

Remark 3.7. We note that unlike most other CRT algorithms, in the generic case where $j = \Theta(p)$, the asymptotically dominant step of ModularEvaluationBigLevel is the selection of the primes.

Sharing the computation of the CRT primes. The set of primes $\mathcal{P}_\ell(j)$ only depends on the value of j. Thus, if we want to evaluate modular polynomials of different levels at the same j-invariant (like in the SEA algorithm), the cost of the selection of the primes can be done once and for all for the biggest levels, and then the computation for each level can simply use a subset of these primes. This means that for enough levels, the bottleneck will not necessarily be the computation of the primes anymore. In particular, in point counting for elliptic curves over a finite field of prime characteristic of bitsize n where one needs to evaluate j on $O(n)$ modular polynomials of level $O(n)$, the cost of the prime selection is amortized over the different levels.

3.4. Comparison between the existing methods. The best algorithms from the literature are Sutherland's algorithms from [41] and Robert's algorithm from [35]. We are going to compare our new algorithms with them. To establish what is the best asymptotic algorithm, we fix a value of ℓ and vary the value of p (and j when it is relevant). We will see that the best algorithm will change as p grows.

In [41], Sutherland introduces three different CRT algorithms to compute modular polynomials. The main one has a complexity of

$$O\left(\ell^2(\ell\log\ell+\log p)\log^{2+\varepsilon}(\ell\log\ell+\log p)\right),$$

with a $O(\ell\log p+\ell^2\log(\ell\log\ell+\log p))$ space requirement. The complexity is essentially the same as the cost of computing the entire modular polynomial with a CRT algorithm, but the memory requirement is smaller. Despite this, the space complexity is quadratic in ℓ, which is not optimal since the size of the output is $O(\ell\log p)$. The second algorithm due to Sutherland achieves a $O(\ell^{3+\varepsilon}\log p)$ complexity with an optimal space requirement. The third is a hybrid version of the first two and has a complexity of $O(\ell^3\log^{6+\varepsilon}\ell)$. It also requires an optimal amount of space.

More recently in [35], Robert briefly outlined an algorithm with time complexity equal to $O(\ell^2\log^{\alpha+\varepsilon}\ell\log p)$ where α is at best equal to $1+2u$ (according to [35, Proposition 3.1]) for $u\in\{1,2,4\}$ such that ℓ minus a product of small primes powers is equal to a sum of u squares. Heuristically, by adjusting the product of small primes, one can expect to find a case where $u=2$. Thus, at best, the complexity of Robert's algorithm should be $O(\ell^2\log^{5+\varepsilon}\ell\log p)$. However, note that Robert's paper only gives an outline of the algorithm and of its complexity analysis. Without clear statements, it is hard to assess the real complexity of the algorithm. Indeed, from the statements in [35], it is not completely clear to us if his asymptotic estimate can be used in the generic case. If not, then the correct value of α is actually $3+2u=7$. The space complexity of Robert's algorithm is not clearly stated either, but it does not seem to be optimal.

Thus, for small values of p or j, our algorithm ModularEvaluationBigLevel appears to have the best known asymptotic complexity. In particular, if $j=O(p^\varepsilon)$ for $\varepsilon<1$, then our algorithm has the best complexity of all existing algorithms.

If $j=\Theta(p)$, as p grows, the quadratic factor in $\log j$ in our complexity will mean that ModularEvaluationBigLevel will be outperformed by Robert's algorithm. The asymptotic complexity tells us that the breaking point should occur when $\log p\approx\log\ell^{1+2u}$. Although, note that Robert's algorithm probably requires more memory, therefore there may be some practical case where our algorithm would still outperform Robert's. Also note that if we remove the cost of the CRT prime computation (that can be shared among the computation for several levels as already argued), then we expect our algorithm to outperform Robert's for a much larger range of primes (up to $\log p\approx\ell$).

When $\ell=O(\log p)$, the best complexity will be obtained by Sutherland's first CRT algorithm and our ModularEvaluationBigCharacteristic algorithm, but ModularEvaluationBigCharacteristic has a better space complexity than Sutherland's algorithm (which is quadratic in ℓ).

Note that the proof of the complexity of Robert's algorithm does not require any assumptions, whereas our method and Sutherland's are only heuristic.

In Section 4.2, we will compare the practical performance of our C++ implementation with the one from Sutherland.

3.5. Other modular functions. Modular polynomials can be generalized to modular functions other than the j-function. Considering other kinds of modular polynomials is a well-known trick to make computations more efficient (see for instance [8]). Indeed, these alternate modular polynomials may have smaller

coefficients. One of the most interesting examples appears to be the modular polynomial associated to the Weber function $\mathfrak{f}$. Over $\mathbb{C}(j)$, this function is a root of the polynomial

$$\Psi(X, j) = (X^{24} - 16)^3 - jX^{24} \tag{3.1}$$

and the logarithmic height bound of the associated modular polynomial $\Phi_\ell^{\mathfrak{f}}$ is expected to be 72 times smaller than the one of Φ_ℓ. Moreover, the coefficient of X^aY^b is non-zero if and only if $\ell a + b = \ell + 1 \mod 24$. These two facts imply that the complexity of computing $\Phi_\ell^{\mathfrak{f}}$ should be roughly $72 \times 42 = 1728$ faster than Φ_ℓ. By using the simple algebraic relation between j and $\mathfrak{f}$ given by Eq. (3.1), it quickly becomes more efficient to use $\Phi_\ell^{\mathfrak{f}}$ rather than Φ_ℓ.

Over $\mathbb{F}_q$, every j-invariant is associated to several $\mathfrak{f}$-invariants that are the roots of $\Psi(X, j)$ given in Eq. (3.1). In fact, this polynomial comes from the covering map of $X(1) \cong \mathbb{P}^1$ by X_H, a modular curve of level 48 isomorphic to the map labelled 48.72.0.d.1 in the (Beta version of the) LMFDB [32]. The polynomial $\Phi_\ell^{\mathfrak{f}}$ can be constructed from $\mathfrak{f}$-invariants in a manner analogous to Φ_ℓ from j-invariants. When ℓ is coprime to 48, it makes sense to talk about ℓ-isogenous $\mathfrak{f}$-invariants, and evaluating $\Phi_\ell^{\mathfrak{f}}(f_0, X)$ can be done by computing all the ℓ-isogenous $\mathfrak{f}$-invariants.

This idea was described in BLS [8], and Leroux already stated in [31] that the BLS method could be extended to this setting analogously. The goal of this section is to explain how this can be realized concretely. We find that the BLS method cannot be directly applied, and instead new ideas are needed. The main obstacle of using $\mathfrak{f}$-invariants instead of j-invariants is the multiplicity of the polynomial Ψ, which implies that there are several $\mathfrak{f}$-invariants corresponding to the same j-invariant (in the worst case, 72). Therefore, to find the ℓ-isogenous $\mathfrak{f}$-invariants, the usual method of projecting from X_H down to $X(1)$, finding an ℓ-isogenous j-invariant, and then lifting again to the correct $\mathfrak{f}$-invariant does not work. Indeed, it is hard to know which lift of j to consider.

The method used in [8]. Bröker, Lauter, and Sutherland overcome this obstacle by working purely over X_H. Indeed, by using the class group action of a quadratic imaginary order of conductor ℓ on $X(1)$, they enumerate the set of neighbours in the ℓ-isogeny graph. This action can then be extended to X_H. More explicitly, by using the modular polynomial of level ℓ_i, one can compute the action of ideals of small norm ℓ_i. When working over X_H, we can simply use the modular polynomial $\Phi_{\ell_i}^{\mathfrak{f}}$ instead, and the roots will directly give the ℓ_i-isogenous $\mathfrak{f}$-invariants. We furthermore remark that this method works as the BLS method only needs to consider points of $X_H(\mathbb{F}_p)$.

To adapt this idea to our setting of supersingular curves defined over $\mathbb{F}_{p^2}$, the main problem is that the required j-invariants are not computed with the help of modular polynomials, but rather as the codomain of certain isogenies. As there are no efficient isogeny formulæ for isogenies of degree ℓ between elements of $X_H(\mathbb{F}_{p^2})$, we cannot directly derive the ℓ-isogenous $\mathfrak{f}$-invariants. Our idea is to use a different interpretation of the curve X_H. Indeed, modular curves are known to parametrize elliptic curves enriched with level structure. Thus, if we have an explicit way to associate the elements of X_H with curves and associated level structure of order 48 (meaning that we can compute the value of the $\mathfrak{f}$-invariant only from the curve and the level structure), then it suffices to push the level structure through the isogenies

of degree ℓ to be able to recover the ℓ-isogenous $\mathfrak{f}$-invariant. This will work for ℓ coprime to 48, hence for every prime ≥ 5.

The parametrization associated to X_H. The level of X_H is $48 = 3 \times 16$, so we instead consider the level structure of order 3 and 16 separately.

Let us first consider the level 3 part. For this, we look at Eq. (3.1) and observe that if f is a root of $\Psi(X, j)$ for a given j, then f^8 is a root of $\Psi'(X, j^{1/3}) = (X^3 - 16) - Xj^{1/3}$. This is convenient because the j-function is well-known to be the cube of another modular function of level 3, often denoted by γ_2. In terms of modular curves, this can be interpreted as the cover map

$$X_{ns}^+(3) \to X(1),\ t \mapsto t^3.$$

There is a classical formula to compute the three possible γ_2-invariants above a given j-invariant of a curve E and the x-coordinates of $E[3]$ (see for instance in [9, Section 6.6, Example 1]). If E is the curve $y^2 = x^3 + Ax + B$ and x_1, x_2, x_3, x_4 are the x-coordinates of the non-trivial points of $E[3]$, then the roots of $X^3 - j(E)$ are given by

$$\frac{-48A}{2A - 3(x_1x_2 + x_3x_4)},\quad \frac{-48A}{2A - 3(x_1x_3 + x_2x_4)},\quad \frac{-48A}{2A - 3(x_1x_4 + x_3x_2)}.$$

In summary, given an ordering for the x-coordinates of the non-trivial points of $E[3]$, we can compute the three possible γ_2-invariants associated to E. This covers the part of level 3.

We now consider the level 16 part. Here, we look for the solutions of $\Psi''(X, j) = (X^8 - 16)^3 - jX^8$, which will correspond to $\mathfrak{f}^3$. The map $t \mapsto \frac{(t^8-16)^3}{t^8}$ corresponds to the cover of $X(1)$ by the modular curve $X_{H'}$ of level 16 labelled as 16.24.0.p.1 in the new beta version of LMFDB. To the best of our knowledge, there are no known formulas to recover the $\mathfrak{f}^3$-invariant from a level structure of order 16. Such formulas certainly exist, but it is unclear to us how can they be computed efficiently. Instead, we propose to use the much better studied modular curve $X_s(16)$ above $X_{H'}$. In particular, $X_s(16)$ parametrizes curves together with two subgroups of order 16 which intersect trivially. In this way, given a curve $E(\mathbb{F}_q)$ and two subgroups $G_1, G_2 \subset E[16](\mathbb{F}_q)$, we can recover the corresponding element of $X_s(16)(\mathbb{F}_q)$ by looking at the common preimage of $j(E)$ through the two maps $X_s(16) \to X(1)$ factoring through the two covers

$$\begin{aligned}\psi_1 &\colon X_s(16) \to X_0(16), \quad E, G_1, G_2 \mapsto E, G_1,\\ \psi_2 &\colon X_s(16) \to X_0(16), \quad E, G_1, G_2 \mapsto E, G_2;\end{aligned}$$

see Figure 1.

Remark 3.8. The method that we outlined above works most of the time, but there are some problems when there exist 16-isogenies between the same pair of curves (as the map $X_0(16) \to X(1) \times X(1)$ is not injective anymore and so our GCD polynomial will have several roots). In that case, it is still possible to identify the element of $X_0(16)$ corresponding to the given subgroup of order 16 by decomposing the 16-isogeny associated to this subgroup as 4 isogenies of degree 2, finding the 4 $X_0(2)$ points corresponding to each of these 2-isogenies, and finally identifying the correct root of the GCD with the different existing maps from $X_0(16)$ to $X_0(2)$.

We do this as follows:

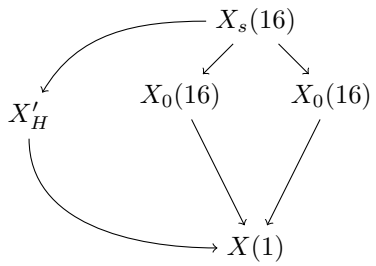

FIGURE 1. Modular covers of $X(1)$ by $X_s(16)$

(1) Compute the points of $X_0(16)$ lying above the two pairs (j, j_1) and (j, j_2) of j-invariants, i.e., for $k = 1, 2$, compute x_k such that $\rho(x_k) = (j, j_k)$ where $\rho : X_0(16) \to X(1) \times X(1)$ is given by

$$\rho(X) := \left(\frac{G_0(16)(X)}{H_0(16)(X)}, \frac{I_0(16)(X)}{J_0(16)(X)}\right),$$

for $G_0(16)(X), H_0(16)(X), I_0(16)(X), J_0(16)(X) \in \mathbb{Z}[X]$. If it fails because there are multiple solutions, use the method outlined in Remark 3.8

(2) Recover the point (x, y) on $X_s(16)$ above the two points of $X_0(16)$ by finding a solution to the system of equations consisting of

$$F_s(16)(X, Y) = 0 \qquad \frac{G_s(16)(X)}{H_s(16)(X, Y)} = (x_1) \qquad \frac{I_s(16)(X)}{J_s(16)(X, Y)} = (x_2),$$

for $F_s, G_s, H_s, I_s, J_s \in \mathbb{Z}[X, Y]$. Where F is the equation of $X_s(16)$, and $\frac{G_s}{H_s}, \frac{I_s}{J_s}$ give the two projection maps $X_s(16) \to X_0(16)$.

(3) Project this point (x, y) on $X_s(16)$ to X'_H via $\varphi\colon X_s(16) \to X'_H$, defined as

$$\varphi(X, Y) := \frac{-16X^6Y^{12} - X^6Y^4 + 64X^2Y^{16} + 20X^2Y^8 + X^2}{32Y^{15} + 4Y^7}.$$

Then, $\varphi(x, y) = \mathfrak{f}^3$.

The formulas for all the maps involved in the computation described above can be found in the full version of this paper. Some of these maps can be found on the LMFDB, and the other maps were computed using MAGMA [7].

To develop an efficient algorithm that follows the procedure above, we use the method detailed in [13, §5]. We first define polynomials

$$\begin{aligned} f_1(X, Y) &:= F_s(16)(X, Y), \\ f_2(X, Y) &:= G_s(16)(X, Y) - \alpha_1(j, j_1)H_s(16)(X, Y), \\ f_3(X, Y) &:= I_s(16)(X, Y) - \alpha_2(j, j_2)J_s(16)(X, Y), \end{aligned}$$

where, for $k = 1, 2$, $\alpha_k \in \mathbb{Z}[j, j_k]$ is defined as

$$\alpha_k(j, j_k) := \gcd\big(G_0(16)(X) - jH_0(16)(X), I_0(16)(X) - j_kJ_0(16)(X)\big).$$

For $i, j \in \{1, 2, 3\}$, we also define polynomials

$$R_{i,j}(Y) := \operatorname{res}_X(f_i(X, Y), f_j(X, Y)) \in \mathbb{Z}[j, j_1, j_2][Y].$$

By the elimination property of the resultant (e.g., see [16, §3.6, Lemma 1]), the specialisations $(R_{i,j})_{[E,G_1,G_2]}(Y)$ given by evaluating the coefficients of each $R_{i,j}$ at $j := j(E)$, $j_1 := j(E/G_1)$ and $j_2 := j(E/G_2)$, vanish at the Y-coordinate of any common solution to the specialised polynomials $(f_j)_{[E,G_1,G_2]}(X,Y)$.

However, these resultants (generically) have factors which correspond to spurious (unwanted) solutions. Therefore, we instead consider polynomials where these spurious solutions have been removed, namely:

$$P_{1,2}(Y) := \frac{16}{Y^{108}(16Y^8+1)^7} \cdot R_{1,2}(Y), \text{ and}$$
$$P_{2,3}(Y) := \frac{16}{Y^{77}(16Y^8+1)^7} \cdot R_{2,3}(Y).$$

Note that now $P_{1,2}$ and $P_{2,3}$ are coprime.

If there exist $x, y \in \mathbb{F}_p$ such that $(f_i)_{[E,G_1,G_2]}(x,y) = 0$ for each $i = 1,2,3$ then the degree of

$$g(Y) := \gcd\big((P_{1,2})_{[E,G_1,G_2]}(Y), (P_{2,3})_{[E,G_1,G_2]}(Y)\big)$$

is 1. Conversely, if $y \in \mathbb{F}_p$ is a root of $g(Y)$, then there exist $x, x' \in \mathbb{F}_p$ such that

$$(f_1)_{[E,G_1,G_2]}(x,y) = (f_2)_{[E,G_1,G_2]}(x,y) = 0, \text{ and}$$
$$(f_2)_{[E,G_1,G_2]}(x',y) = (f_3)_{[E,G_1,G_2]}(x',y) = 0.$$

We assume $x = x'$ (otherwise, we throw an error and restart the procedure; note this will only happen with negligible probability).

Therefore, from a gcd computation, we can extract $y \in \mathbb{F}_p$, and then recover x via a square root computation. Then, (x,y) is a point on $X_s(16)$, and we have $\mathfrak{f}^3 = \varphi(x,y)$, where $\varphi : X_s(16) \to X'_H$.

In practice, rather than computing the polynomials $P_{1,2}$ and $P_{2,3}$ on the fly, we precompute and store them. Then, given E, G_1, G_2 defined over $\overline{\mathbb{F}}_p$, we evaluate the polynomials at corresponding j, j_1, j_2, reduce them modulo p, and compute their gcd. We summarise the discussion above in Algorithm 6. Here, Roots denotes a function that returns the roots of a polynomial over a finite field.

Algorithm 6 GetWeberCube(p, j, j_1, j_2)

Input: A prime p, and j-invariants j, j_1, j_2 of $E, E/G_1, E/G_2$ respectively.
Output: The cube $\mathfrak{f}^3$ of the $\mathfrak{f}$-invariant associated to E.

1: Evaluate the coefficients of $P_{1,2}(Y)$ at j, j_1, j_2 and reduce modulo p to obtain $h_1(Y) \in \mathbb{F}_p[Y]$
2: Evaluate the coefficients of $P_{2,3}$ at j, j_1, j_2 and reduce modulo p to obtain $h_2(Y) \in \mathbb{F}_p[Y]$
3: $c_0 Y + c_1 \leftarrow \gcd(h_1, h_2)$
4: $y \leftarrow -c_1 \cdot c_0^{-1}$
5: `rts` $\leftarrow$ Roots$(f_1(X,y))$
6: **for** r in `rts` **do**
7: **if** $f_3(r,y) = 0$ **then**
8: **return** $\varphi(r,y)$
9: **end if**
10: **end for**
11: **return** $\perp$

In conclusion, an ordering x_1, x_2, x_3, x_4 of the x-coordinates of $E[3]$ and two subgroups of order 16 with trivial intersection in $E[16]$, we obtain two polynomials in X^8 and X^3. It then suffices to take the gcd to find the value of the Weber invariant associated to this level structure of order 48 on E: if t is the output of $\mathsf{GetWeberCube}(p, j, j_1, j_2)$, we compute

$$\gcd(X^3 - t, X^8 - \gamma_2)$$

and obtain $\mathfrak{f}$-invariant associated to E.

To find the ℓ-isogenous $\mathfrak{f}$-invariants corresponding to some ℓ-isogeny φ it suffices to evaluate the level structure in φ and repeat the procedure depicted above. Since our algorithms relies on computing alternative paths through the Deuring correspondence (in particular, we never explicitly compute any ℓ-isogenies), obtaining this evaluation requires the use of techniques which at this point are standard in isogeny-based cryptography. We refer to Section 4.1 for details.

4. Implementation results

We have implemented the algorithms ModularEvaluationBigLevel and ModularEvaluationBigCharacteristic in C++/NTL. The aim of this section is to give an overview of the code-base, before we present our implementation results.

4.1. Implementation details. We first describe the different parts of our library, which is available at:

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

NTL functionalities. Our library uses NTL to perform basic arithmetic over finite fields $\mathbb{F}_{p^k}$ (for $k \geq 1$), and can be compiled both using classes `NTL::zz_p` and `NTL::zz_pE` or `NTL::ZZ_p` and `NTL::ZZ_pE` if needed, integer arithmetic using class `NTL::ZZ`, and polynomial arithmetic using classes `NTL::zz_pX`, `NTL::zz_pEX` or `NTL::ZZ_pX`, `NTL::ZZ_pEX` if needed. We furthermore use NTL vectors and matrices constructed from these classes.

Finding CRT primes. We precomputed a list of all primes up to 2^{25}, and use a segmented version of the sieve of Eratosthenes [4] to find larger primes if required (with an upper bound of 2^{50}). Based on the implementation of [14], it takes an interval $[L, R]$ and finds all contained primes. Sieving different intervals is independent, so the segmented sieve parallelizes perfectly.

Instead of storing all primes and testing supersingularity of the corresponding elliptic curves later, we run supersingularity tests on the fly after identifying all primes in a sieving interval $[L, R]$. Since the overwhelming majority of curves we test is ordinary, our implementation benefits most from a test that discards those curves quickly. Following [3], the best choice in our context is Sutherland's supersingularity test based on determining whether the 2-isogeny graph has a volcano structure (ordinary case) or not (supersingular case) [42]. We implement Sutherland's test including the optimisations proposed in [3].

Field arithmetic. We require arithmetic in $\mathbb{F}_{p^{2k}}$ for small k. For all basic operations, we rely on NTL. Further, we precompute a matrix corresponding to the Frobenius action. Additionally, due to how the isogeny computations work, we also

need a fixed, effective embedding $\iota_k : \mathbb{F}_{p^2} \hookrightarrow \mathbb{F}_{p^{2k}}$ for all k used. This embedding is given by a fixed image $\iota_k(\omega)$, where ω is a fixed generator of $\mathbb{F}_{p^2}$ of trace 0.

For an element $a + b\omega \in \mathbb{F}_{p^2}$ (represented in NTL as the tuple (a, b)), we can easily lift this to $\mathbb{F}_{p^{2k}}$ by computing $a + \iota_k(\omega)b$ in $\mathbb{F}_{p^{2k}}$. To coerce elements into $\mathbb{F}_{p^2}$, we proceed as follows. Given an element $\alpha \in \mathbb{F}_{p^{2k}}$ (represented in NTL as the tuple $(a_1, \ldots, a_{2k})$), which we know is in the image of ι_k, i.e., $\alpha = \iota_k(a + \omega b)$:

(1) Recover a from the trace as $a = \frac{\mathrm{tr}\,\alpha}{2k}$.
(2) Recover b as $a_j/\iota_k(\omega)_j$, where $j > 1$ is any non-zero index of the representation $(\iota_k(\omega)_1, \ldots, \iota_k(\omega)_{2k})$ of the element $\iota_k(\omega)$.

We extended these lifting and coersion operations to the polynomial rings $\mathbb{F}_{p^{2k}}[X]$, by lifting and coercing each coefficient, respectively.

Elliptic curves and isogenies. We implement standard elliptic curve operations using Weierstraß curves. One of the main computational tasks required by the elliptic curve part of our library is the computation of smooth degree isogenies given by kernel generators of smooth order defined over small extensions $\mathbb{F}_{p^{2k}}$ of $\mathbb{F}_{p^2}$. This setting is similar to that in [22], and thus the strategy employed is the same. Namely, we use supersingular curves E with $\pi_E^2 = [-p]$, where π_E denotes the p-Frobenius endomorphism on E. As a result, all the isogenies will be defined over $\mathbb{F}_{p^2}$, even if the kernel generators are not. Due to this, we can compute our isogenies by recovering the kernel polynomial using [22, Algorithm 4], before employing Kohel's formula to recover the isogeny [26, Chapter 2.4].

Computing the endomorphism ring of a given supersingular elliptic curve E defined over $\mathbb{F}_p$ is done by brute-force searching an isogeny to a curve with known endomorphism ring. Concretely, we start by determining the $\mathbb{F}_p$-endomorphism ring R of E; the two cases $\mathbb{Z}[\pi] \cong \mathbb{Z}[\sqrt{-p}]$ and $\mathbb{Z}[\frac{1+\pi}{2}] \cong \mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$ are readily distinguished by testing whether the rank of $E(\mathbb{F}_p)[2]$ equals 1 or 3 respectively. This boils down to checking whether the cubic polynomial $f(x)$ in the short Weierstraß equation $y^2 = f(x)$ defining E splits over $\mathbb{F}_p$ or not. Then, using a suitable starting curve E_0 with the same $\mathbb{F}_p$-endomorphism ring and known $\mathbb{F}_{p^2}$-endomorphism ring, there must exist an ideal class $[\mathfrak{a}] \in \mathrm{Cl}(R)$ such that $[\mathfrak{a}] * E_0 = E$, which we recover using a straightforward meet-in-the-middle brute-force search using a (heuristic) generating set of small generators for $\mathrm{Cl}(R)$. In practice, this means we do not set explicitly the constant C from Algorithm 3, we just increase the exponents until a solution is found.

Lattices in quaternion algebras. Our library identifies the quaternion algebra $B = H(-q, -p)$ simply by the pair $p, q \in \mathbb{Z}$, and the format of its elements consists of a reference to B together with a 5-tuple of integers $t, x, y, z, d \in \mathbb{Z}$ representing the quaternion $(t+xi+yj+zk)/d$. Arithmetic in this representation is straightforward. We further represent a quaternion lattice I by a basis matrix in $\mathbb{Q}^{4\times 4}$, which is in turn represented as a matrix in $\mathbb{Z}^{4\times 4}$ together with a denominator in $\mathbb{Z}$. All basic functionality for quaternion lattices, such as computing the intersection, product, or sum of two lattices, computing left or right orders, or finding canonical or reduced lattice bases, all essentially rely on the linear-algebra routines provided by NTL.

Our library also includes an implementation of a variant of the KLPT algorithm [25], which works for all primes where there exists an embedding $K \hookrightarrow B_{p,\infty}$, where K is an imaginary quadratic field of class number 1. Note that this is a very mild assumption on the prime p (a quick heuristic estimate is that only 1 in every 2^9

primes fail this requirement). Our implementation uses the standard improvement due to Petit and Smith [34] to reduce the output size. However, our implementation also differs in another way. The output size (with the improvement from [34]) of KLPT is typically stated as being in $\tilde{O}(p^3)$. However, this is based on the heuristic that the shortest prime-normed equivalent ideal is of norm $O(p^{1/2}\log(p))$. While this heuristic almost never fails in large characteristic, we are working in small enough characteristic to regularly encounter ideals where this fails. Hence, we change the size of the solution we search for on the fly to be in $\tilde{O}(p^2 N_I^2)$, where N_I denotes the norm of the shortest prime-normed equivalent ideal.[1]

Orders to j-invariants. In OrdersTojInvariantSmallSet, we repeatedly apply the same procedure as in [22] to translate orders to j-invariants. The only difference is that our implementation has precomputed elliptic curves $E/\mathbb{Q}$ with CM by an order of class number 1, together with the isogeny ι defining the complex multiplication. Thus, for the first step described in [22], we can take the starting curve $E_0/\mathbb{F}_p$, to be a supersingular reduction of one of our precomputed curves. Note that the requirement on p that one of the precomputed curves has supersingular reduction coincides with the requirement we described for KLPT (except for a constant number of primes p, where the precomputed curves might have bad reduction). Further, note that for $p \equiv 1 \pmod{12}$, the isomorphism $\mathcal{O}_0 \cong \mathrm{End}(E_0)$ cannot be determined solely by the (reduction of the) endomorphism ι; see [22, Section 3.1] for details, and how to resolve this ambiguity.

ModularEvaluationBigCharacteristic also requires the OrdersTojInvariantBigSet algorithm introduced in [31] to compute efficiently all the supersingular j-invariants and the corresponding maximal order. Our implementation follows quite closely the algorithm detailed by Leroux (see [31, Algorithm 1]). Maximal orders are represented by the 3 successive minima (that can be computed with LLL) of the trace-0 sublattice.

Computing Weber invariants. To compute the Weber invariant of a curve with a given level structure, we implement the ideas discussed in Section 3.5. In particular, we implement subroutines to evaluate multivariate polynomials of degree 2 and 3 (NTL only supports univariate polynomials), and we use built-in NTL functions for gcd computations and root finding. Furthermore, the algorithm requires computing the gcd of two precomputed resultants, which are stored in `getresultants.cpp`. The memory cost of the hardcoded resultants is negligible in the context of the full algorithm, and the efficiency gain is significant.

To obtain the ℓ-isogenous Weber invariants in ModularEvaluationBigLevel, we also need to evaluate the level structure at the ℓ-isogenies. Since no ℓ-isogenies are ever explicitly computed, this is done by evaluating the level structure on a well-chosen endomorphism, before pushing it through the alternative isogeny path we have computed. Explicitly, this endomorphism equals the composition of the ℓ-isogeny with the dual of the alternative path, and can be found by multiplying the corresponding ideals.

Furthermore, OrdersTojInvariantBigSet needs to be adapted to work with Weber invariants instead of j-invariants. For that, we represent Weber invariants as a j-invariant together with some basis of the 48-torsion, and with some coefficients

[1] Note that we do have the guarantee that $N_I \in \tilde{O}(p)$, hence the size of the output size is still upper bounded by $\tilde{O}(p^4)$.

to represent the level structure in the basis. By exploiting the symmetries coming from the Weber function, we obtain an efficient algorithm to recover the full list of Weber invariants.

OrdersTojInvariantBigSet also needs to be adapted to work with Weber invariants instead of j-invariants, and so we need to modify OrdersTojInvariantBigSet to compute all supersingular $\mathfrak{f}$-invariants as well. For that, we will use our level structure parametrization of $\mathfrak{f}$-invariants. That way, by using a basis of the 48-torsion, each Weber invariant can be fully derived from a set of small integers (giving the coefficients of the points of the level structure in the basis). Then, it suffices to propagate the 48-torsion basis throughout the isogenies involved in OrdersTojInvariantBigSet. For each supersingular j-invariant, we will then need to compute all $\mathfrak{f}$-invariants above it from the basis. The symmetries coming from the Weber function allow us to be much more efficient to recover the full list of Weber invariants than one application of the algorithm we described in Section 3.5 for each of the 72 Weber invariants.

The CRT method. For the implementation of the CRT method in our library, we follow Sutherland's software `classpoly` [39] based on [21, 40]. In particular, our CRT functions in `crt.cpp` take heavy inspiration from `crt.c` in Sutherland's software. The main difference is that our algorithms use C++/NTL rather than C/GMP. We therefore defer a detailed explanation on the implementation of the CRT method to these previous works.

Polynomial interpolation. In our library, we implement univariate polynomial interpolation following Algorithm 10.11 in [46, §10]. In particular, we implement two main functions:

- An algorithm to interpolate a univariate polynomial $f(X) \in \mathbb{F}_{p^{2k}}[X]$ of degree d at $d+1$ points $(x_0, f(x_0)), \dots, (x_d, f(x_d))$.
- An algorithm to interpolate a univariate polynomial $f(X) \in \mathbb{F}_{p^{2k}}[X]$ of degree d from its roots $(x_1, \dots, x_d)$. This is a special case of the first algorithm where $f(x_i) = 0$, but here we optimise for this case.

We remark that these algorithms outperform the in-built NTL interpolation function `interpolate` and the NTL function `BuildFromRoots`, respectively, for polynomials of relatively large degrees (which is our use case in this paper).

4.2. Results. In this section, we present the performance of our implementation of the algorithms ModularEvaluationBigLevel and ModularEvaluationBigCharacteristic, and compare it to the state-of-the-art [41].

Performance analysis. All code was compiled and run on a Linux server with two AMD EPYC 9754 CPUs running at 2.25 GHz (with dynamic frequency scaling and simultaneous multithreading disabled), 1 TB RAM, and using `g++` version 15.1.1 with flags `-std=c++17 -O3 -march=native -DNDEBUG`.

We ran experiments for the Weber variant of both algorithms ModularEvaluationBigLevel and ModularEvaluationBigCharacteristic (i.e., we compute the Weber modular polynomial evaluated at a Weber invariant). To understand the performance of these algorithms for varying level ℓ, we fix the characteristic $p = 2^{31} - 1$, the Weber invariant $w = 2$, and run both algorithms for the same set of growing levels ℓ. Since our implementation is heavily multithreaded, to enable a meaningful comparison with data collected on other hardware, we report the CPU *core* time

(i.e., in single-core equivalents) rather than the wall-clock time consumed by each run of each algorithm. Our results are summarized in Table 1.

We see that despite a better asympotic complexity in ℓ, we were not able to reach a point where ModularEvaluationBigLevel is faster than ModularEvaluationBigCharacteristic. This is the due to the fact that the hidden constants in the asymptotic complexity of the SingleOrderTojInvariant algorithm are very large (due to the computation of a smooth isogeny of degree $\approx p^3$). Nonetheless, the current data is enough to witness that the scaling of ModularEvaluationBigLevel and ModularEvaluationBigCharacteristic are indeed quadratic and cubic in ℓ respectively.

TABLE 1. Experimental data: CPU core time consumed by runs of ModularEvaluationBigLevel and ModularEvaluationBigCharacteristic for various levels ℓ, fixing the characteristic $p = 2^{31} - 1$ and the input Weber invariant $w = 2$.

Level ℓ	ModularEvaluationBigLevel	ModularEvaluationBigCharacteristic	Ratio
211	7.68 min	18.191 s	25.3
419	24.01 min	40.702 s	35.4
607	51.34 min	1.24 min	41.3
811	1.46 h	2.00 min	43.7
1019	2.28 h	3.07 min	44.5
2003	9.12 h	15.13 min	36.2
3011	21.54 h	46.53 min	27.8
4003	1.63 d	1.65 h	23.7
5003	2.66 d	3.31 h	19.3
6007	3.89 d	5.49 h	17.0
7019	5.32 d	8.42 h	15.2
8011	7.00 d	12.13 h	13.8
9007	9.13 d	17.93 h	12.2
10007	11.39 d	1.00 d	11.3
11681	15.78 d	1.56 d	10.1

Extrapolating the results presented in Table 1, we observe that our current implementation falls short of a record computation (using a similar computational power as in [41]). Take, for instance, the large characteristic setting. In the SEA point-counting record described in [41, Section 5] with $\log(p) \approx 16646$, the total time to compute evaluated modular polynomials using Weber invariants for level $\ell = 11681$ is reported to be around 2 CPU hours on a single thread.

For the same level ℓ, but a much smaller characteristic p, our implementation of ModularEvaluationBigCharacteristic took around 40 CPU hours. The dependency on $\log p$ in the asymptotic complexity comes from the $\log p$ term in the height bound. When taking $\log(p) \approx 16646$ for $\ell = 11681$, one sees that the height bound is essentially doubled (compared to $\log p \approx 30$). Thus, we can estimate that the same computation as Sutherland would have taken around 80 CPU hours with our

implementation, which is 1 or 2 orders of magnitude slower. Since the computations were run on different hardware, it is difficult to be more precise than that.

Possible improvements. There are still some places where significant improvements could be made. We identify the following possible improvements to our software (in order of expected impact):

(1) Currently, the quaternion operations are the bottleneck in ModularEvaluationBigCharacteristic (taking up to 80% percent of the global run time) despite being asymptotically negligible compared to interpolation. One explanation is that the hidden constants in these operations can be quite big (one quaternion multiplication is already 17 integer multiplications), and most operations on orders and ideals require to apply operations on bases of four quaternion elements. Another reason is that our implementation uses `NTL::ZZ` which are integers of arbitrary size when the integers involved in all our quaternion computation can in fact be bounded. Thus, switching to fixed-sized integers should bring a noticeable improvement. Moreover, some of the operations on ideals use generic algorithms that could be improved with a tailored implementation.
(2) Using x-only arithmetic for elliptic curve operations, and in particular in isogeny computations which are by far the bottleneck in ModularEvaluationBigLevel due to the high constants involved in the complexity of computing the isogenies in OrdersTojInvariantSmallSet.
(3) Precomputing the action of each basis element of the endomorphism ring on the relevant ℓ^e-torsion bases of the starting curve would also greatly improve the cost of computing the isogeny kernels in OrdersTojInvariantSmallSet and thus positively impact the performances of ModularEvaluationBigLevel.

We are hopeful that the concrete efficiency gains from these improvements could allow to significantly improve our implementation to the point where ModularEvaluationBigCharacteristic could match or even beat the performances of the software from [41].

Despite the better asymptotic scaling, it seems unlikely that the improvements listed above would be enough for ModularEvaluationBigLevel to beat the other two algorithms for a realistic value of ℓ. To reach that point, one would likely need a deeper algorithmic improvement allowing us to decrease the degree of the isogenies used in SingleOrderTojInvariant.

We remark, however, that all the changes listed above require a significant implementation effort, and are thus out of scope for this paper.

5. Conclusion

We have introduced several new algorithms to evaluate modular polynomials using supersingular curves and the Deuring correspondence. These algorithms all improve the asymptotic time or space complexity of previously known algorithms for some range of inputs.

We implemented our algorithms in C++, exhibiting the practicality of the algorithms presented. We further suggest potential improvements that could lead to important practical speed-ups of our implementation, but they require an extensive optimization effort and are thus left for future work.

References

1. Nesmith Cornett Ankeny, *The least quadratic non residue*, Annals of Mathematics (1952), 65–72.
2. Sarah Arpin, James Clements, Pierrick Dartois, Jonathan Komada Eriksen, Péter Kutas, and Benjamin Wesolowski, *Finding orientations of supersingular elliptic curves and quaternion orders*, Des. Codes Cryptogr. **92** (2024), no. 11, 3447–3493.
3. Gustavo Banegas, Valerie Gilchrist, and Benjamin Smith, *Efficient supersingularity testing over $\mathbb{F}_p$ and CSIDH key validation*, Mathematical Cryptology **2** (2022), no. 1, 21–35.
4. Carter Bays and Richard H. Hudson, *The segmented sieve of Eratosthenes and primes in arithmetic progressions to 1012*, BIT Numerical Mathematics **17** (1977), no. 2, 121–127.
5. Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith, *Faster computation of isogenies of large prime degree*, ANTS (2020).
6. Ian F. Blake, János A Csirik, Michael Rubinstein, and Gadiel Seroussi, *On the computation of modular polynomials for elliptic curves*, HP Laboratories Technical Report (1999).
7. Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. **24** (1997), no. 3-4, 235–265, Computational algebra and number theory (London, 1993). https://www.math.ru.nl/~bosma/pubs/JSC1997Magma.pdf. MR MR1484478
8. Reinier Bröker, Kristin Lauter, and Andrew Sutherland, *Modular polynomials via isogeny volcanoes*, Mathematics of Computation **81** (2012), no. 278, 1201–1231.
9. Reinier Martijn Bröker, *Constructing elliptic curves of prescribed order*, Ph.D. thesis, Leiden University, 2006.
10. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes, *CSIDH: an efficient post-quantum commutative group action*, ASIACRYPT 2018, Springer, 2018, pp. 395–427.
11. Denis Charles and Kristin Lauter, *Computing modular polynomials*, LMS Journal of Computation and Mathematics **8** (2005), 195–204.
12. Leonardo Colò and David Kohel, *Orienting supersingular isogeny graphs*, Number-Theoretic Methods in Cryptology 2019 (2019).
13. Maria Corte-Real Santos, Craig Costello, and Sam Frengley, *An algorithm for efficient detection of (N, N)-splittings and its application to the isogeny problem in dimension 2*, PKC 2024. Part III, Lecture Notes in Computer Science, vol. 14603, Springer, 2024, pp. 157–189. MR 4763489
14. Craig Costello, Michael Meyer, and Michael Naehrig, *Sieving for twin smooth integers with solutions to the Prouhet-Tarry-Escott problem*, EUROCRYPT 2021, Lecture Notes in Computer Science, vol. 12696, Springer, 2021, pp. 272–301.
15. Jean Marc Couveignes, *Hard homogeneous spaces*, IACR Cryptology ePrint Archive, 2006.
16. David Cox, John Little, and Donal O'Shea, *Ideals, varieties, and algorithms*, 3 ed., Undergraduate Texts in Mathematics, Springer, 2007. MR 2290010
17. Christina Delfs and Steven D. Galbraith, *Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$*, Designs, Codes and Cryptography **78** (2016), 425–440.
18. Kirsten Eisenträger, Sean Hallgren, Kristin Lauter, Travis Morrison, and Christophe Petit, *Supersingular isogeny graphs and endomorphism rings: Reductions and solutions*, EUROCRYPT 2018, Springer, 2018, pp. 329–368.
19. Noam D. Elkies et al., *Elliptic and modular curves over finite fields and related computational issues*, AMS/IP Studies in Advanced Mathematics **7** (1998), 21–76.
20. Andreas Enge, *The complexity of class polynomial computation via floating point approximations*, Mathematics of Computation **78** (2009), no. 266, 1089–1107.
21. Andreas Enge and Andrew V. Sutherland, *Class invariants by the CRT method*, International Algorithmic Number Theory Symposium, Springer, 2010, pp. 142–156.
22. Jonathan Komada Eriksen, Lorenz Panny, Jana Sotáková, and Mattia Veroni, *Deuring for the people: Supersingular elliptic curves with prescribed endomorphism ring in general characteristic*, LuCaNT 2023, 2023.
23. Steven D. Galbraith, Christophe Petit, and Javier Silva, *Identification protocols and signature schemes based on supersingular isogeny problems*, ASIACRYPT 2017, 2017.
24. Joseph H. Silverman, *The arithmetic of elliptic curves*, 2 ed., Graduate Texts in Mathematics, vol. 106, Springer, 2009.

25. David Kohel, Kristin E. Lauter, Christophe Petit, and Jean-Pierre Tignol, *On the quaternion ℓ-isogeny path problem*, 2014.
26. David R. Kohel, *Endomorphism rings of elliptic curves over finite fields*, Ph.D. thesis, University of California at Berkeley, 1996.
27. Sabrina Kunzweiler and Damien Robert, *Computing modular polynomials by deformation*, 2024.
28. Serge Lang and Hale Trotter, *Frobenius distributions in GL_2-extensions: Distribution of Frobenius automorphisms in GL_2-extensions of the rational numbers*, Lecture Notes in Mathematics, vol. 504, Springer, 2006.
29. Frank Lehmann, Markus Maurer, Volker Müller, and Victor Shoup, *Counting the number of points on elliptic curves over finite fields of characteristic greater than three*, International Algorithmic Number Theory Symposium, Springer, 1994, pp. 60–70.
30. Antonin Leroux, *Quaternion algebra and isogeny-based cryptography*, Ph.D. thesis, Ecole doctorale de l'Institut Polytechnique de Paris, 2022.
31. Antonin Leroux, *Computation of Hilbert class polynomials and modular polynomials from supersingular elliptic curves*, 2023.
32. The LMFDB Collaboration, *The L-functions and modular forms database*, https://www.lmfdb.org, 2024, [Online; accessed 7 June 2024].
33. François Morain, *Calcul du nombre de points sur une courbe elliptique dans un corps fini: aspects algorithmiques*, Journal de théorie des nombres de Bordeaux **7** (1995), no. 1, 255–282.
34. Christophe Petit and Spike Smith, *An improvement to the quaternion analogue of the ℓ-isogeny problem*, Presentation at MathCrypt (2018).
35. Damien Robert, *Some applications of higher dimensional isogenies to elliptic curves (overview of results)*, IACR Cryptology ePrint Archive, 2022.
36. Alexander Rostovtsev and Anton Stolbunov, *Public-key cryptosystem based on isogenies*, IACR Cryptology ePrint Archive, 2006.
37. Maria Corte-Real Santos, Craig Costello, and Jia Shi, *Accelerating the Delfs-Galbraith algorithm with fast subfield root detection*, CRYPTO (3), Lecture Notes in Computer Science, vol. 13509, Springer, 2022, pp. 285–314.
38. Victor Shoup et al., *NTL: A library for doing number theory*, 2001.
39. A. V. Sutherland, `classpoly`, Software package, version 1.0.3, https://math.mit.edu/~drew/classpoly.html, accessed 10 January 2025.
40. Andrew Sutherland, *Computing Hilbert class polynomials with the Chinese remainder theorem*, Mathematics of Computation **80** (2011), no. 273, 501–538.
41. ———, *On the evaluation of modular polynomials*, ANTS X, vol. 1, The Open Book Series, no. 1, Mathematical Sciences Publishers, 2013, pp. 531–555.
42. Andrew V. Sutherland, *Identifying supersingular elliptic curves*, LMS Journal of Computation and Mathematics **15** (2012), 317–325.
43. The Sage Developers, *SageMath, the Sage Mathematics Software System (Version 10.0)*, 2024, `https://www.sagemath.org`.
44. Jacques Vélu, *Isogénies entre courbes elliptiques*, Comptes-Rendus de l'Académie des Sciences, Série I **273** (1971), 238–241.
45. John Voight, *Quaternion algebras*, Graduate Texts in Mathematics, Springer, 2018.
46. Joachim von zur Gathen and Jürgen Gerhard, *Modern computer algebra*, Cambridge University Press, New York, 1999.

ENS de Lyon, CNRS, UMPA, UMR 5669, Lyon, France
Email address: maria.corte_real_santos@ens-lyon.fr

COSIC, KU Leuven, Belgium
Email address: jeriksen@esat.kuleuven.be

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

University of Regensburg, Germany
Email address: michael@random-oracles.org

Technische Universität München, Germany
Email address: lorenz@yx7.cc