

Quantum security analysis of unrestricted isogeny-based group actions

Xavier Bonnetain¹, Max Duparc², Dania Lazzarini³, Luciano Maino⁴, Chloe Martindale^{5,10}, Christophe Petit^{3,4}, Sina Schaeffler^{6,7,10}, André Schrottenloher^{8,10}, and Alessandro Sferlazza^{9,10}

¹ Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

`xavier.bonnetain@inria.fr`

² EPFL, Lausanne, Switzerland

`max.duparc@epfl.ch`

³ Université Libre de Bruxelles, Bruxelles, Belgium

`dania.lazzarini@ulb.be`, `Christophe.Petit@ulb.be`

⁴ University of Birmingham, Birmingham, UK

`l.maino@bham.ac.uk`

⁵ University of Bristol, Bristol, UK

`chloe.martindale@bristol.ac.uk`

⁶ IBM Research Europe, Zurich, Switzerland

⁷ ETH Zurich, Zurich, Switzerland

`sschaeffle@ethz.ch`

⁸ Univ Rennes, Inria, CNRS, IRISA, Rennes, France

`andre.schrottenloher@inria.fr`

⁹ TU Munich, Munich, Germany

`alessandro.sferlazza@tum.de`

¹⁰ TSVP, Okinawa Institute of Science and Technology, Onna, 904-0495, Japan

Abstract. Setting concrete parameters for group action based cryptographic protocols is notoriously difficult due to the incomplete understanding of the concrete efficiency of Kuperberg’s subexponential quantum attack. Estimates for the CSIDH group action do exist in the literature, but they do not easily translate into meaningful estimates for the many recent and far more efficient alternative group actions.

In this paper, we implement the recent qt-Pegasis group action framework as a (simulated) quantum circuit, allowing for concrete estimation of its efficiency and Kuperberg’s attack. Our results suggest that using a 2048-bit base field in qt-Pegasis offers reasonable quantum security, while a 4096-bit base field is required to reach NIST security level 1.

1 Introduction

Isogeny-based cryptography is a promising candidates for post-quantum cryptography, with the signature protocol SQIsign currently considered for standardization by NIST [1]. Notably, isogeny-based cryptography offers natural candidates

Authors listed alphabetically, ams.org/profession/leaders/CultureStatement04.pdf.

for *cryptographic group actions*, allowing to generalize existing cryptographic protocols based on discrete logarithm problems. However, isogeny problems are rather recent in cryptography, so their security remains to be confirmed by sustained cryptanalysis efforts (as shown by the attack on SIKE [18, 51, 61] and recent progress on the endomorphism ring problem [67]). In particular, group action based protocols also suffer from subexponential quantum attacks using Kuperberg’s algorithm [48] and variants.

For practical use, security must be understood not only asymptotically, but as concrete resource estimations as functions of a problem’s main parameters. Given uncertainties around future quantum computer architectures, this is already challenging for simple algorithms. For group action protocols, the numerous parameterization options in Kuperberg’s algorithm and variants and their intensive use of memory further complicate estimations. Additionally, they require group action evaluations, whose efficient quantum implementation is challenging.

The concrete quantum security of (isogeny-based) cryptographic group actions was already studied in [11, 57, 15], focusing on the security of CSIDH [19]. These works showed that CSIDH’s original parameters offered inadequate quantum security and revealed that the group action evaluation subroutine contributed significantly to the overall quantum cost. However, limitations of this prior work restrict its applicability for group action-based protocol designers: they focused on the CSIDH group action framework and its original parameters. However, alternative frameworks exist, and it is now clear that larger parameters are needed. More importantly, new classical algorithms to evaluate group actions [33, 4, 21, 5, 30, 31] have appeared recently, and at least part of this progress is expected to translate into more efficient quantum subroutines.

Contributions. In this paper, we re-assess the quantum resources needed to solve the *vectorization problem* central to all protocols built on top of isogeny-based group actions, focusing in particular on qt-Pegasis [31]. Specifically,

- After simplifying the qt-Pegasis group action evaluation algorithm to make it more amenable to our security analysis, we provide quantum algorithms for all the key subroutines, building circuits for both the norm equation solver (incl. Cornacchia’s algorithm) and the 4-dimensional isogeny chain.
- We provide Qarton implementations [66] of our subroutines, available in the following repository.

github.com/PQarrots/quantum-group-actions

Our implementations are complex quantum circuits: they contain 148 different circuit definitions, including 97 from Qarton’s library, and 51 from our new code.

- We introduce new trade-offs within Kuperberg’s algorithm, and reduce its complexity by tuning the internal list sizes.
- We provide quantum resource estimations for the cryptanalysis of qt-Pegasis [31]. We estimate that a 2048-bit prime offers reasonable quantum security (more than 2^{64} quantum operations and 2^{112} classical operations are required to break it), while a 4096-bit prime is required to reach NIST security level 1.

As our primary aim is to inform design and parameter choices, our estimations rather underestimate than overestimate quantum cryptanalysis costs if necessary.

Technical overview. Let G be a group and X be a set. The group action

$$\tau : G \times X \rightarrow X : (g, x) \rightarrow \tau_g(x)$$

is called a *cryptographic group action* if it satisfies some additional properties [3]. In particular, the associated *vectorization problem* is computationally intractable: given $x_0, x_1 \in X$, it should be hard to compute $s \in G$ such that $\tau_s(x_0) = x_1$. When the group G is abelian, the vectorization problem reduces to the following *hidden shift problem*, which can be solved in quantum subexponential time using Kuperberg’s algorithm [48] and variants.

Problem 1 (Hidden Shift Problem). Let G be an abelian group and X a set. Let f_0, f_1 be two functions $G \rightarrow X$ such that there exists an $s \in G$ verifying $f_1(g) = f_0(g + s)$, for all $g \in G$. Find s .

The reduction is straightforward: define $f_0(g) := \tau_g(x_0)$ and $f_1(g) := \tau_g(x_1)$ so that indeed $f_0(g + s) = \tau_{g+s}(x_0) = \tau_g(\tau_s(x_0)) = \tau_g(x_1) = f_1(g)$, then solve the hidden shift to recover s .

Class group actions on elliptic curves have been proposed as a basis for cryptographic constructions since the works of Couveignes [27], and Rostovtsev and Stolbunov [63]. The approach became much more efficient after the introduction of CSIDH [19], a commutative group action based on supersingular curves defined over $\mathbb{F}_p$. Colò and Kohen generalized CSIDH through the concept of oriented curves [25], which also encompass CSISurf and SCALLOP [17, 33]. The introduction of Kani’s lemma [45] in isogeny-based cryptography led to further considerable efficiency improvements; it also allowed to efficiently evaluate the group action on all elements whereas previously efficient evaluation was (more or less explicitly) restricted to a subset of the group. This led to new cryptographic group action parameters proposed and new group action evaluation algorithms, including [21, 55, 56, 30, 31]. The current fastest *unrestricted* group action is given by qt-Pegasis; here *unrestricted* means that the action can be evaluated for arbitrary group elements. qt-Pegasis uses the same curves as CSIDH but with primes of a different shape. More recently CORAL [8] added further efficiency gains, but only for a subset of group elements, describing a restricted group action. While CORAL uses parameters similar to those used in qt-Pegasis, Houben’s fast restricted group actions [41] requires parameters are more similar to those employed in the original version of CSIDH [19]. Globally, classical algorithms for group action evaluation have improved by several orders of magnitude since the previous quantum cost estimations on CSIDH were made.

Estimating the exact cost of solving the vectorization problem requires assessing the cost of group action evaluation and optimizing Kuperberg’s algorithm for the specific parameters involved.

Our first technical decision is to choose qt-Pegasis as a starting point for the quantum group action evaluation subroutine. Indeed, qt-Pegasis is the fastest

classical algorithm available apart from CORAL and Houben’s work, and we discard these two ideas because Kuperberg’s algorithm requires to evaluate the group action on all elements in superposition (except potentially for a negligible subset). A combination of qt-Pegasis and CORAL, which could make sense classically, is also ruled out: the overall time and circuit size to evaluate all inputs in superposition comes from the slowest and biggest algorithm, so adding CORAL would add complexity for no benefit. Another reason to analyze qt-Pegasis is because it is one of the main building blocks of the PQarrots algorithmic suite [2].

The qt-Pegasis algorithm is made of three steps: a lattice reduction on two-dimensional lattices to explicitly reduce ideals, a norm equation step, and the computation of a chain of 4-dimensional isogenies. We implement a simplified version of this algorithm. While it does restrict the functionality to only part of all possible inputs, the size of the resulting circuit is almost equal to the size of a generic implementation (see Section 4.1). First, we do not implement the lattice reduction step, but explain an algorithm for Gauss-Lagrange reduction in linear space (Section 3.4), improving over [32]. Second, we implement the norm equation solver. It uses a two-layer quantum search based on amplitude amplification [16] to handle the iterations of the classical algorithm, and relies on new quantum circuit implementations like Cornacchia’s algorithm. Third, we implement the chain of 4-dimensional isogenies which relies on optimized modular arithmetic and new quantum circuits for addition on Montgomery curves.

Independently, we provide new optimized tradeoffs for Kuperberg’s second algorithm that are relevant when the isogeny evaluation circuit is expensive.

Further work. While part of our methodology (especially Section 5.1) can be applied to any group action, for concrete estimates we opted for specific parameters used in qt-Pegasis, which a priori favour the attacker. Different choices could have resulted in different security estimations, which we leave as further work.

We note two special features of the group action evaluation problem that are not exploited in our algorithm. First, the qt-Pegasis takes as input an ideal in an arbitrary format, whereas in the context of Kuperberg’s algorithm the adversary will generate those ideals themselves. This could a priori allow them to choose a suitable format to accelerate subsequent computations. Second, in Kuperberg’s algorithm context, the group action must be evaluated on two curves, where the endomorphism ring is known for one of the two. These special features might perhaps lead to further improvements.

We intentionally restricted our scope to the central *vectorization problem* for its greater relevance in protocols, but we note that alternatives computational problems have been suggested in the literature, which may offer distinct levels of security [20, 6, 36].

Finally, the quantum circuit implementations that we developed in this paper are functional, but could be further optimized:

- We relied largely on the arithmetic libraries of Qarion [66], in which operations like multiplications are limited to schoolbook versions. Given that we compute with relatively large numbers (between 500 and 4000 bits), more advanced arithmetic circuits like Karatsuba multiplication could help.

- We had to implement missing arithmetic components like inversion modulo a power of 2 and operations on Montgomery elliptic curves, possibly with under-optimized operations.
- Regarding space, many registers appearing in our circuits are over-sized. While this keeps the quantum circuitry relatively close to the classical code, it is certainly possible to make the implementation more compact.

Given these optimizations, our gate and qubit counts are clearly overestimated. However, we do not expect gains of more than a factor 2^4 in gate count¹¹ and 2 in qubit count, unless one completely changes the underlying logic of the circuits. Given that the complexity of the attack does not depend linearly on the cost of the oracle, reducing the latter by a factor 2^4 has only a moderate impact.

Acknowledgments

Part of this work was conducted while visiting the Okinawa Institute of Science and Technology (OIST) through the Theoretical Sciences Visiting Program (TSVP), the Thematic Program on Isogeny-based Cryptography (TP25IC). Dania Lazzarini is supported by a FRiA grant (40037678) by the National Fund for Scientific Research (F.N.R.S.) of Belgium. Sina Schaeffler is supported by SNSF Consolidator Grant CryptonIs 213766. Christophe Petit is partly supported by EPSRC through grant number EP/V011324/1 and F.R.S.-FNRS through grant number T.0238.25. This work has been partially supported by the French Agence Nationale de la Recherche through the France 2030 program under grant agreements No. ANR-22-PETQ-0007 EPiQ and ANR-22-PETQ-0008 PQ-TLS.

Outline. The paper is organized as follows. Section 2 recalls basic information on isogeny-based group actions, the qt-Pegasis algorithm, and quantum algorithms. Sections 3 and 4 provide the norm equation and isogeny parts of our algorithm respectively, and Section 5 describes our optimizations of Kuperberg’s algorithm.

2 Preliminaries

In this section, we recall some background knowledge. We first recall some basic facts about the class group action in our context (Section 2.1). Then, we recollect the qt-Pegasis algorithm in Section 2.2. We conclude with a summary on the basics of quantum computing, specializing to the case of group actions (Section 2.3).

2.1 The class group action and short norm ideals

Let p be a prime of the form $p = f \cdot 2^{e+3} - 1$, for some odd $f > 0$. We denote n its bitsize. We consider the imaginary quadratic field $K = \mathbb{Q}(\sqrt{-p})$. For $\nu \in \mathbb{Q}(\sqrt{-p})$ we denote by $\bar{\nu}$ its conjugate and we call $n(\nu) = \bar{\nu}\nu$ its norm (a quadratic form).

¹¹ This number is purely imaginary, but close to the progress seen in implementations of elliptic curve point addition, between [40] and [65].

The ring of integers of $\mathbb{Q}(\sqrt{-p})$ corresponds to $\mathbb{Z}[\omega]$, where $\omega := \frac{1+\sqrt{-p}}{2}$. We denote ideals of $\mathbb{Z}[\omega]$ by lower-case fraktur letters. The norm of an ideal $\mathfrak{a}$ is the greatest common divisor $n(\mathfrak{a})$ of the norms of its elements. Another $\mathbb{Z}[\omega]$ -ideal $\mathfrak{b}$ is equivalent to $\mathfrak{a}$ if there is $\alpha \in \mathfrak{a}$ such that $\mathfrak{b} = \mathfrak{a} \frac{\bar{\alpha}}{n(\mathfrak{a})}$. This equivalence relation defines classes of ideals which form the (commutative) ideal class group.

Given a supersingular elliptic curve E , any such ideal $\mathfrak{a}$ corresponds an isogeny denoted $\varphi_{\mathfrak{a}} : E \rightarrow E'$, where (the isomorphism class of) E' is the result of the action of the class $[\mathfrak{a}]$ of $\mathfrak{a}$ on E . For details, see [Appendix A](#). All notations, in particular p, f, n, ω , are used throughout the paper.

2.2 qt-Pegasis

Given an ideal $\mathfrak{a} \subset \mathbb{Z}[\omega]$ and an elliptic curve E oriented by $\mathbb{Z}[\omega]$, the goal of qt-Pegasis [\[31\]](#) is to compute the isogeny $\varphi_{\mathfrak{a}} : E \rightarrow E_{\mathfrak{a}}$ corresponding to $\mathfrak{a}$. First, we compute four ideals $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4$ equivalent to $\mathfrak{a}$ such that $n(\mathfrak{a}_1) + n(\mathfrak{a}_2) + n(\mathfrak{a}_3) + n(\mathfrak{a}_4) = 2^e$, where $N_{1,2} = n(\mathfrak{a}_1) + n(\mathfrak{a}_2)$ and $N_{3,4} = n(\mathfrak{a}_3) + n(\mathfrak{a}_4)$ are both odd. One of the main building blocks to obtain such ideals is *Cornacchia's algorithm* [\[26\]](#) which finds, when it exists, an integer solution (x, y) to the equation $x^2 + dy^2 = m$. In our construction, we only use the case $d = 1$, therefore using it as sum-of-two-squares solver.

Solving the qt-Pegasis norm equation. Given an ideal $\mathfrak{a} \subset \mathbb{Z}[\omega]$, our goal is to find four ideals $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4$ equivalent to $\mathfrak{a}$ such that $n(\mathfrak{a}_1) + n(\mathfrak{a}_2) + n(\mathfrak{a}_3) + n(\mathfrak{a}_4) = 2^e$, where $N_{1,2} = n(\mathfrak{a}_1) + n(\mathfrak{a}_2)$ and $N_{3,4} = n(\mathfrak{a}_3) + n(\mathfrak{a}_4)$ are both odd.

We represent the ideal $\mathfrak{a}$ as (N, α) , where $n(\mathfrak{a}) = N$; we also define $r := n(\alpha)/N$. For the sake of simplicity, we assume that $\mathfrak{a}$ is the smallest ideal in its equivalence class and that α is the smallest element in $\mathfrak{a}$ of norm $N \cdot m$ where m is coprime with N .

The ideal $\mathfrak{a}_i$ can be expressed as $\mathfrak{a} \bar{\alpha}_i / N$, for some $\alpha_i \in \mathfrak{a}$. By rewriting $\alpha_i = b_i N + c_i \alpha$ for some $b_i, c_i \in \mathbb{Z}$, we reduce the task of finding $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4$ such that $n(\mathfrak{a}_1) + n(\mathfrak{a}_2) + n(\mathfrak{a}_3) + n(\mathfrak{a}_4) = 2^e$ to finding $(b_i, c_i) \in \mathbb{Z}^2$, for $i = 1, \dots, 4$, such that

$$N(b_1^2 + b_2^2 + b_3^2 + b_4^2) + r(c_1^2 + c_2^2 + c_3^2 + c_4^2) + \text{tr}(\alpha)(b_1 c_1 + b_2 c_2 + b_3 c_3 + b_4 c_4) = 2^e. \quad (1)$$

Remark 1. In qt-Pegasis [\[31\]](#), [Equation \(1\)](#) is simplified to

$$N(b_1^2 + b_2^2 + b_3^2 + b_4^2) + r(c_3^2 + c_4^2) + \text{tr}(\alpha)(b_3 c_3 + b_4 c_4) = 2^e. \quad (2)$$

Such an equation is firstly solved mod N , where the conditions $c_3 = 1$ and $c_4 = k$ are imposed; this requires finding “short” b_3 and b_4 . Then, the integers b_1 and b_2 are recovered using Cornacchia’s algorithm.

Remark 2. As described in [\[31, Lemma 4 and 5\]](#) the norm equation solving step contains an additional set of rejection tests before the call to Cornacchia’s

Algorithm 1 Modified qt-Pegasis [31, Alg.1] with singular rejections integrated

Input: An invertible ideal $\mathfrak{a} = (N, \alpha) \subset R = \mathbb{Z}[\omega] =: \mathbb{Z}[\omega]$, a number $e \in \mathbb{Z}$.

Output: $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4 \sim \mathfrak{a}$ such that $n(\mathfrak{a}_1) + n(\mathfrak{a}_2) + n(\mathfrak{a}_3) + n(\mathfrak{a}_4) = 2^e n(\mathfrak{a})$.

```

1: Let  $\mathfrak{a}$  be the smallest ideal in  $[\mathfrak{a}]$ 
2: Write  $\mathfrak{a}$  as  $(N, \alpha)$  where  $N = n(\mathfrak{a})$ 
3: Let  $r = n(\alpha)/N$ 
4: Let  $k = k_{\max} = \min\{\sqrt{2^e/r}, \sqrt{N}\}$ 
5: Let rejected = true
6: while rejected do
7:   Compute  $v$  as  $v \equiv (2^e - r(1 + k^2)) \operatorname{tr}(\alpha)^{-1} \pmod{N}$ 
8:   Let  $(b_3, b_4) = (v - \lfloor v/k \rfloor k, \lfloor v/k \rfloor)$ 
9:   Let  $z = \frac{2^e - (1+k^2)r - \operatorname{tr}(\alpha)(b_3+b_4k)}{N} - (b_3^2 + b_4^2)$ 
10:  if  $v \geq k\sqrt{2^e/N - k^2/4}$  or  $z < 0$  then
11:    rejected = false
12:  Set  $\alpha_3 = b_3N + \alpha$ ,  $\alpha_4 = b_4N + k\alpha$ 
13:  if  $z \equiv 2 \pmod{4}$  and the sum of the coordinates of  $\alpha_3 + \alpha_4$  is even then
14:    rejected = false
15:  if  $N \equiv 0 \pmod{2}$  and ( $k \equiv 0 \pmod{2}$  or  $b_3 \equiv r \pmod{2}$  or  $b_4 \equiv r \pmod{2}$ ) then
16:    rejected = false
17:  Set  $\tilde{C}_i, D_i, \tilde{E}_i$  s.t.  $\tilde{C}_1 + \omega\tilde{C}_2 = \overline{\alpha_4}$ ,  $D_1 + \omega D_2 = \alpha_3\overline{\alpha_4}/N$ ,  $\tilde{E}_1 + \omega\tilde{E}_2 = \overline{\alpha_3}$ 
18:  if  $z \pmod{8} \notin \{1, 2, 5\}$  then rejected = false
19:  if  $N \pmod{2} = 1$  and  $z \equiv 1 \pmod{8}$  then  $\triangleright b_2 \equiv 0 \pmod{4}$ 
20:    if None of  $D_1, D_1 + D_2, \tilde{C}_1, \tilde{C}_1 + \tilde{C}_2$  is 2  $\pmod{4}$  then
21:      rejected = false
22:  if  $N \pmod{2} = 1$  and  $z \equiv 5 \pmod{8}$  then  $\triangleright b_2 \equiv 2 \pmod{4}$ 
23:    if ( $D_1$  and  $D_1 + D_2$  not 0  $\pmod{4}$ ) and ( $\tilde{C}_1 - 2\tilde{E}_1$  and  $\tilde{C}_1 + \tilde{C}_2 - 2\tilde{E}_1 - 2\tilde{E}_2$ 
not 2  $\pmod{4}$ ) then rejected = false
24:  if  $N \pmod{2} = 1$  and  $z \equiv 2 \pmod{8}$  then  $\triangleright b_1, b_2 \equiv \pm 1 \pmod{4}$ 
25:    if None of  $N + D_1, N + D_1 + D_2, \tilde{C}_1 - \tilde{E}_1, \tilde{C}_1 + \tilde{C}_2 - \tilde{E}_1 - \tilde{E}_2$  is 2  $\pmod{4}$ 
then rejected = false
26:    if None of  $-N + D_1, -N + D_1 + D_2, \tilde{C}_1 + \tilde{E}_1, \tilde{C}_1 + \tilde{C}_2 + \tilde{E}_1 + \tilde{E}_2$  is 2  $\pmod{4}$ 
then rejected = false
 $\triangleright$  The following line is an addition specific to our quantum implementation,
explained in Section 3.1
27:  if  $z$  (odd case) or  $z/2$  (even case) does not pass a pseudo-primality test then
28:    rejected = false
29:  if not rejected then
30:    Solve  $b_1^2 + b_2^2 = z$  for  $b_1, b_2$  using Cornacchia's algorithm
31:    if Cornacchia's algorithm gives a solution then
32:      if  $z \equiv 2 \pmod{8}$  and  $b_1 \equiv 3 \pmod{4}$  then  $b_1 \leftarrow -b_1$ 
33:      if  $z \equiv 2 \pmod{8}$  and  $b_2 \equiv 3 \pmod{4}$  then  $b_2 \leftarrow -b_2$ 
34:      Let  $\alpha_1 = b_1N$ ,  $\alpha_2 = b_2N$ ,  $\alpha_3 = b_3N + \alpha$ ,  $\alpha_4 = b_4N + k\alpha$ 
35:      return  $\mathfrak{a}_i = \alpha\overline{\alpha_i}/N$  for  $i = 1, \dots, 4$ 
36:  Update  $k$  as  $k - 1$ 
37:  Update  $v$  as  $v + t_1 \pmod{N}$ 
38:  Update  $t_1$  as  $t_1 + t_2 \pmod{N}$ 

```

algorithm, which are shown in [Algorithm 1](#), and its main difference to [\[31, Algorithm 1\]](#). These conditions ensure that, during the construction of the 4-dimensional isogeny as a 2-isogeny chain $f_1, f_2, \dots$, we only encounter cases that simplify the algorithm and that facilitate a constant time implementation. Specifically, $f_1 : E^4 \rightarrow \mathcal{A}$ has to be a gluing isogeny whose codomain is not a product (hence $f_2, \dots, f_e$ can all be computed with generic algorithms; moreover, the codomain of $f_2 : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ must not have a zero dual theta constant. In particular the case of ideals with smallest equivalent of norm $N \equiv 2 \pmod{4}$ is problematic, as detailed in [Section 4.1](#). We additionally introduce a modification in [Line 27](#), due to restrictions in our quantum Cornacchia implementation, as explained in [Section 3](#).

4-dimensional isogeny. Using the ideals $\mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4$ such that $n(\mathfrak{a}_1) + n(\mathfrak{a}_2) + n(\mathfrak{a}_3) + n(\mathfrak{a}_4) = 2^e n(\mathfrak{a})$, where $N_{1,2} = n(\mathfrak{a}_1) + n(\mathfrak{a}_2)$ and $N_{3,4} = n(\mathfrak{a}_3) + n(\mathfrak{a}_4)$ are both odd, we define the endomorphisms $\gamma_1, \gamma_2 \in \mathbb{Z}[\omega]$ as

$$\gamma_1 := \widehat{\varphi}_{\mathfrak{a}_3} \circ \varphi_{\mathfrak{a}_1} + \varphi_{\mathfrak{a}_4} \circ \widehat{\varphi}_{\mathfrak{a}_2}, \quad \gamma_2 := \widehat{\varphi}_{\mathfrak{a}_3} \circ \varphi_{\mathfrak{a}_2} - \varphi_{\mathfrak{a}_4} \circ \widehat{\varphi}_{\mathfrak{a}_1}$$

Then the following two-dimensional isogenies of respective norm $N_{1,2}$ and $N_{3,4}$ induce, by Kani's Lemma [\[45\]](#), the 4 dimensional isogeny Φ of degree 2^e .

$$\begin{aligned} \Phi_1 &:= \begin{pmatrix} \varphi_{\mathfrak{a}_1} & \varphi_{\mathfrak{a}_2} \\ -\varphi_{\bar{\mathfrak{a}}_2} & \varphi_{\bar{\mathfrak{a}}_1} \end{pmatrix}, \quad \Phi_2 := \begin{pmatrix} \varphi_{\bar{\mathfrak{a}}_3} & -\varphi_{\mathfrak{a}_4} \\ \varphi_{\mathfrak{a}_4} & -\varphi_{\bar{\mathfrak{a}}_3} \end{pmatrix} \\ \Phi &:= \begin{pmatrix} \Phi_1 & \widetilde{\Phi_2} \\ -\Psi_1 & \Psi_2 \end{pmatrix} : E^4 \rightarrow E_{\mathfrak{a}} \times E_{\bar{\mathfrak{a}}} \times \mathcal{A}. \end{aligned} \tag{3}$$

$$\ker(\Phi) := \left\langle ([N_{1,2}]P, [N_{1,2}]Q, \gamma_1(P) + \gamma_2(Q), \widehat{\gamma}_1(P) - \widehat{\gamma}_2(Q)) \mid P, Q \in E[2^e] \right\rangle.$$

Using 4 dimensional isogeny formulae, we compute Φ from its kernel and hence recover its codomain $E_{\mathfrak{a}} \times E_{\bar{\mathfrak{a}}} \times \mathcal{A}$. From the codomain, we can extract a Montgomery model for $E_{\mathfrak{a}}$. To maximize efficiency, a 2^e -isogeny computation over $\mathbb{F}_p$ requires two independent points of order 2^{e+2} on E whose x -coordinates are over $\mathbb{F}_p$; such points are guaranteed to exist whenever $2^{e+3} \mid p+1$, which explains the relation between the length of the chain and the available 2-torsion.

2.3 Preliminaries of quantum computing and quantum cryptanalysis of group actions

We write quantum algorithms in the quantum circuit model, like almost all works in quantum cryptanalysis to date, and refer to [\[54\]](#) for details on its definition like qubits, quantum states, the ket notation $|\cdot\rangle$, basic gates, etc. We write our algorithms only as *logical* (error-free) quantum circuits. Unlike works like [\[38\]](#), we do not estimate physical resource overheads, and compare purely on logical resource counts. The main cost metrics of quantum circuits are the qubit count

(space), gate count and depth. We focus on qubit count and Toffoli gate count, like most recent logical implementations of Shor’s algorithm [22, 38, 7, 65]. We stress that we consider quantum algorithms running in the “baseline” quantum circuit model, as an extrapolation of existing and envisioned architectures. This excludes the use of scalable quantum RAM (quantum-accessible classical memory). qRAM is an important component of many quantum cryptanalysis algorithms. In Kuperberg’s algorithm [48], it is not mandatory, but it can be used to reduce the total time complexity. Peikert’s security analysis of CSIDH [57] relies on this assumption, but not the one of [15].

Implementing and validating quantum circuits. A large part of the circuitry used in quantum cryptanalysis is actually *classical*: we require sub-circuits which can be decomposed using only classical gates (e.g., Toffoli). This makes them easy to simulate and validate using classical inputs. Yet, these circuits differ fundamentally from classical implementations since they must be *reversible*.

While some works like [15] make gate count estimates manually, in order to obtain meaningful and modular estimates, it is helpful to build entire implementations, if possible. This is the approach followed hereafter. Kuperberg’s algorithm [48], which is our target, relies on an oracle for computing a group action. We implement the main components of this oracle as quantum circuits. This implementation uses the Qarton library [66], and all explicit gate counts given in the paper come from it. In addition, we authorize our circuits to make *arithmetic failures*, i.e., to fail with some probability on random inputs. It well-known that these failures can be neglected if their probability is sufficiently low compared to the inverse-number of queries, and this exploited in most recent works in quantum cryptanalysis [38, 7, 65].

Quantum search. As part of our oracle algorithm, we need a *two-level quantum search*, which is implemented using quantum amplitude amplification [16]. We consider as search space the integer interval $I := [0; 2^\ell - 1]$, and define a pair of Boolean functions $f_1, f_2 : I \rightarrow \{0, 1\}$. We assume given *phase oracles* for f_1 and f_2 , defined by: $O_{f_1} : |i\rangle \rightarrow (-1)^{f_1(i)} |i\rangle$ and $O_{f_2} : |i\rangle \rightarrow (-1)^{f_2(i)} |i\rangle$. We consider the subspaces:

$$I_1 = \{i \in I, f_1(i) = 1\} \text{ and } I_2 = \{i \in I, f_1(i) = 1 \wedge f_2(i) = 1\} . \quad (4)$$

Let $p_1 = \frac{|I_1|}{|I|}$ and $p_2 = \frac{|I_2|}{|I_1|}$, which we assume known exactly. The exact variant of amplitude amplification (Theorem 4 in [16]) gives us the following algorithms:

- An algorithm $\mathcal{A}_1$ that produces the uniform superposition over I_1 , obtained by applying $\left\lfloor \frac{\pi}{4 \arcsin \sqrt{p_1}} \right\rfloor$ iterations of the operator $-H^{\otimes \ell} O_0 H^{\otimes \ell} O_{f_1}$ to the uniform superposition over I (the starting state), followed by a final “partial” iteration, which uses rotation gates with fixed angles depending on p_1 . Here O_0 is an operator that flips the phase of all non-zero basis vectors.
- An algorithm $\mathcal{A}_2$ that produces the uniform superposition over I_2 , obtained by applying $\left\lfloor \frac{\pi}{4 \arcsin \sqrt{p_2}} \right\rfloor$ iterations of the operator $-\mathcal{A}_1^\dagger O_0 \mathcal{A}_1 O_{f_2}$ to the starting state $\mathcal{A}_1 |0^\ell\rangle$ (e.g., the uniform superposition over I_1).

We use the bound $\arcsin x \geq x$ to simplify the formulas. We neglect the cost of implementing O_0 and the Hadamard gates, which will be asymptotically negligible compared to O_{f_1} and O_{f_2} . We also neglect the cost of implementing the small rotations, which is also negligible at the level of precision that we should require (using [46]). Finally, we make the entire instance of the problem, i.e., p_1, p_2, f_1, f_2, I_1 and I_2 , dependent on some additional quantum input T . In our norm equation solver, T will contain the ideal representative that is input to the solver, and the result of precomputations performed on this ideal. We then control everything using T : the oracles $O_{f_1^T}: |T\rangle |i\rangle \mapsto (-1)^{f_1^T(i)} |I\rangle |i\rangle$, but also the quantum amplitude amplifications, which will apply $\left\lfloor \frac{\pi}{4} \frac{1}{\min_T \sqrt{p_1^T}} \right\rfloor$ *controlled* iterations, and $\left\lfloor \frac{\pi}{4} \frac{1}{\min_T \sqrt{p_2^T}} \right\rfloor$, respectively. These iterations are controlled depending on the actual value of p_1 and p_2 . The correcting rotations of the final iteration also depend on them.

This gives us the following result.

Proposition 1. *Assume given a quantum algorithm that on input T , estimates p_1^T and p_2^T with the requested precision, as well as implementations of the controlled oracles $O_{f_1^T}$ and $O_{f_2^T}$. Then there exists a quantum algorithm that, on input T , constructs the uniform superposition over $I_2^T: |T\rangle |0^t\rangle \mapsto |T\rangle |I_2^T\rangle$.*

This algorithm contains:

$$\left\{ \begin{array}{l} \left(\frac{\pi}{4} \frac{1}{\min_T \sqrt{p_1^T}} \right) \left(\frac{\pi}{4} \frac{1}{\min_T \sqrt{p_2^T}} \right) \text{ calls to } O_{f_1^T} \\ \left(\frac{\pi}{4} \frac{1}{\min_T \sqrt{p_2^T}} \right) \text{ calls to } O_{f_2^T} \end{array} \right.$$

In summary, this two-level search optimizes the search for elements in I_2^T (passing the two tests f_1 and f_2) by first amplifying according to the test f_1 , and then according to the test f_2 . This is a useful strategy if f_2 is much more expensive than f_1 .

3 Quantum algorithms to solve the norm equation

In this section, we explain how we estimate the quantum cost of the first step of the qt-Pegasis group action evaluation: the norm equation solving part.

We first explain precisely what we implemented in [Section 3.1](#), before detailing quantum circuits for the core of the norm equation solver ([Section 3.2](#)) and its central Cornacchia subroutine ([Section 3.3](#)). Finally we add some considerations on quantum versions of the lattice reduction pre-treatment of inputs ([Section 3.4](#)) and on the adaptation of the output format ([Section 3.5](#)) for the following isogeny steps. The results of our implementation are discussed in [Section 3.6](#).

3.1 Detailed norm equation description as implemented

Throughout this section, we assume that the input ideal $\mathfrak{a}$ is minimal (it should have been already reduced as explained in [Section 3.4](#)). In [Section 2.2](#) we introduced the qt-Pegasis norm equation with geometric rejections (introduced in [\[31, Lemma 4,5\]](#), see [Remark 2](#)). These restrictions only allow to handle cases where the input ideal (assumed here and in the following shortest in its class) has norm $N \not\equiv 2 \pmod{4}$. If $N \equiv 2 \pmod{4}$ one has to act first with a single 2-isogeny, so that the norm equation solver gets input which has norm $N \not\equiv 2 \pmod{4}$. See [Section 4.1](#) for details.

Compared to the original qt-Pegasis algorithm, [Algorithm 1](#) keeps the rejection conditions unchanged. Between [Line 18](#) and [26](#), we reject the special cases leading to singularities in the isogeny step (see [Remark 2](#)). We make some changes to facilitate a quantum implementation:

- We do not use the loop structure (the fact that k decreases throughout the loop) to amortize the computation of v . The reason for this is that the While loop at [Step 6](#) will be replaced by a quantum search over k , so the field for v cannot be updated from its previous value like this is done to optimize the classical code.
- We include a pseudo-primality test for z at [Step 27](#). The reason is that our implementation of Cornacchia’s algorithm in the quantum setting is only guaranteed to work if z is a prime (when odd) or twice a prime (when even). Thus, we want to increase the probability that this is the case. A full primality test would dominate the complexity, so we use a pseudo-primality test which looks for small prime factors in z .

Due to the limitations of our Cornacchia implementation, more values of z are rejected than in the classical case. This increases the number of loop iterations compared to the classical implementation of qt-Pegasis but, experimentally, does not restrict the set of ideal classes on which it succeeds.

3.2 Quantum architecture for the norm equation solver

The quantum version of [Algorithm 1](#) has the following structure. We start from an invertible ideal $\mathfrak{a} = (N, \alpha) \subset R = \mathbb{Z}[\omega]$, which is the smallest ideal in the class $[\mathfrak{a}]$.

First, we do a precomputation that corresponds to [Steps 2 to 4](#) in [Algorithm 1](#), i.e., all computations that can be performed outside the loop:

- 1: Let $r := n(\alpha)/N$
- 2: Let $k_{\max} := \min\{\sqrt{2^e/r}, \sqrt{N}\}$
- 3: Let $\text{bound} := k_{\max}^2 \lfloor \sqrt{2^e/N - k^2/4} \rfloor$
- 4: Let $t := \text{tr}(\alpha)^{-1} \bmod N$
- 5: **Return** $k_{\max}, \text{bound}, t, r$

Let $|T\rangle$ be the state including the input to the norm equation solver, as well as the result of the precomputation. On input $|T\rangle$, we do a two-level quantum

search for a correct value of k . A value is *correct* if it satisfies a first constraint detailed below, *and* gives a solution to the norm equation. This search for k spans a set of integers $I = [0; 2^e - 1]$, and the current value of k is defined as $k_{\max} - i$. The size of the search space is 2^e . We are exactly in the situation handled by [Proposition 1](#), and we reuse the same notations.

On input $|T\rangle$, the search produces the uniform superposition of valid values of k . Measuring this superposition would disrupt our algorithm (since it would post-select on the measured k); instead, we will just have to keep this superposition until we have finished the oracle computation.

Definition of the Boolean tests. We first define the two Boolean tests f_1^T and f_2^T . Recall that we will implement them as phase oracles $O_{f_1^T}$ and $O_{f_2^T}$, controlled by the context T :

$$\begin{cases} O_{f_1^T} : |T\rangle |i\rangle \rightarrow (-1)^{f_1(i)} |T\rangle |i\rangle \\ O_{f_2^T} : |T\rangle |i\rangle \rightarrow (-1)^{f_2(i)} |T\rangle |i\rangle \end{cases} \quad (5)$$

The first Boolean test f_1^T corresponds to all computations performed in the main loop of [Algorithm 1](#) before running Cornacchia's algorithm, so Step 36 to 28. We first compute v, b_3, b_4, z , and test some boundary and congruence conditions. Then, we compute the ideals α_3, α_4 and test all cases leading to singularities in the 4d isogeny formulae. These computations are basic arithmetic computations, and the conditions are simply accumulated by writing some qubits, where k remains good if and only if all the condition qubits are 1.

As we mentioned above, we want to increase the probability that z is prime (and that our Cornacchia solver succeeds). To do so, we perform a small pseudoprimality test, which just eliminates composite numbers by testing divisibility by a list of small primes (easier than computing a GCD, and non-dominating anyway). We took a list of size 20 for all parameters, and observe indeed a benefit in the success probability; the difference is minor if we increase the list size further.

The gate count of $O_{f_1^T}$ is thus in $O(n^2)$. The dominating operations are the arithmetic operations for computing $v, z, b_3, b_4, \alpha_3, \alpha_4$, etc. These operations include multiplications, which are performed using schoolbook circuits in quadratic gate count. All numbers involved have a bit-size which is a constant multiple (or fraction) of n .

The second Boolean test f_2^T refines f_1^T . It performs all computations in the main loop of [Algorithm 1](#), including Cornacchia's algorithm. The Cornacchia circuit always returns an output of the form $(x, y, \text{success})$. If $\text{success} = 1$ then $x^2 + y^2 = z$, otherwise (x, y) is not a solution (which may be the case if z is not a prime, and not twice a prime). Our implementation (detailed below) has a gate count in $O(n^3)$. This gate count is much higher than the one of f_1^T , and this justifies to separate the search in two levels. Although $O_{f_2^T}$ will be called much less than $O_{f_1^T}$, the difference in their gate counts is such that $O_{f_2^T}$ will still dominate the total gate count. Thus, it is also relatively less interesting to

separate the first level further into more levels, since the gains would only be on a non-dominating term.

Estimating the probabilities. In order to use the two-level search on input T , we need a quantum algorithm that estimates p_1^T and p_2^T , where p_1^T is the proportion of inputs satisfying the test f_1^T , and p_2^T is the proportion of inputs satisfying the test f_2^T among those already satisfying f_1^T . We justify why this is possible and why the gate count of this step should remain non-dominating, but we note that our implementation does not contain this step. Contrary to [Proposition 1](#), we implemented a search with non-controlled iterations, using a global estimate on p_1^T and p_2^T .

After running experiments with random ideal inputs, we find that the probabilities p_1^T indeed depend on T (i.e., the ideal). But the probability at this step is quite large, and the variance is relatively small. Hence, one is able to use quantum amplitude estimation [\[16\]](#) to estimate precisely the probability of success of this first step, and deduce the corresponding p_1^T . We also conjecture that the probability p_2 is independent from T , though it depends on the ambient prime p , and taking a search space I large enough reduces its variance sufficiently for our needs. We will assume that this is done in the algorithm and neglect the complexity of this step. In our complexity estimates, we will use an average for p_2 and an experimental lower bound for p_1^T , computed over a few ideals.

Result and data. Using [Proposition 1](#), we can construct the uniform superposition of valid k in:

$$\left(\frac{\pi}{4} \frac{1}{\min_T \sqrt{p_1^T}} \right) \left(\frac{\pi}{4} \frac{1}{\min_T \sqrt{p_2^T}} \right) \simeq \frac{\pi^2}{16} \frac{1}{\min_T \sqrt{p_1^T p_2}} \quad (6)$$

calls to an oracle implementing f_1^T , and $\frac{\pi}{4} \frac{1}{\sqrt{p_2}}$ calls to an oracle implementing f_2^T . We have seen that asymptotically, $O_{f_1^T}$ costs $O(n^2)$ gates and $O_{f_2^T}$ costs $O(n^3)$ gates. Since the probability for z to be prime is around $O(\log n/n)$, and the rejection cases have only a constant probability, then the total gate count of the quantum search grows in $O\left(\frac{n^{3.5}}{\sqrt{\log n}}\right)$.

Concrete data. For prime bit sizes between 500 and 4000, we computed p_1, p_2 over a search space of size 2^{22} , taking 20 primes for the pseudo-primality test, and over 20 random instances. We record the range of the probabilities p_1, p_2 , the median of p_1 , and the average of p_2 . We then compute the numbers of iterations in the first level of the search as $\frac{\pi}{4} \frac{1}{\sqrt{\min p_1}}$ and in the second level as $\frac{\pi}{4} \frac{1}{\sqrt{p_2}}$, using the minimum of the obtained p_1 and the average of the obtained p_2 respectively.

Note that the second oracle calls $O_{f_2^T}$ dominate the total gate count, and we perform $\frac{\pi}{4} \frac{1}{\sqrt{p_2}}$ of them (they occur only on the second level of the search), independently of p_1 . Thus, if we were to refine our estimation of p_1 , this would not impact our complexity estimate.

Table 1: Experimental data on p_1 and p_2 . Each individual value was obtained by exhausting a search space of size 2^{22} for k , and the statistics were computed over 20 random ideal inputs. The number of iterations at the first level depends on the minimal value of p_1 , and at the second level it depends on the average of p_2 . It is rounded upwards as an integer (put between parentheses).

Security level	p_1			p_2			Iterations	
	Min	Max	Median	Min	Max	Average	Level 1	Level 2
500	$2^{-10.59}$	$2^{-5.1}$	$2^{-8.66}$	$2^{-4.63}$	$2^{-4.37}$	$2^{-4.47}$	$2^{4.95}(31)$	$2^{1.89}(4)$
1000	$2^{-9.8}$	$2^{-4.54}$	$2^{-8.40}$	$2^{-5.66}$	$2^{-5.4}$	$2^{-5.49}$	$2^{4.55}(23)$	$2^{2.4}(5)$
2000	$2^{-8.04}$	$2^{-4.75}$	$2^{-6.41}$	$2^{-6.62}$	$2^{-6.35}$	$2^{-6.46}$	$2^{3.67}(13)$	$2^{2.88}(7)$
4000	$2^{-10.75}$	$2^{-5.86}$	$2^{-9.84}$	$2^{-9.31}$	$2^{-7.17}$	$2^{-7.57}$	$2^{5.03}(33)$	$2^{3.44}(11)$

3.3 Subroutine: Cornacchia’s algorithm

Our implementations of f_1^T and f_2^T as quantum circuits, although technical, rely mostly on existing building blocks of unsigned, signed and modular integer arithmetic, which were available in the Qarton library [66]. Our implementation of Cornacchia’s algorithm required to implement new arithmetic components.

Our circuit takes as input an integer z which is equal to 1, 2 or 5 modulo 8 (the only cases required in Algorithm 1), and returns a triple $(x, y, \text{success})$ where $x^2 + y^2 = z$ if $\text{success} = 1$, and otherwise $x^2 + y^2 \neq z$. The circuit is guaranteed to return **success** if z is a prime or $z/2$ is a prime. In the following, we note $z' = z$ if $z = 1 \bmod 2$ else $z/2$. We then have $z' \equiv 1 \bmod 4$.

In Algorithm 2 we recall the classical Cornacchia’s algorithm following [52]. It has two steps, which we implement separately:

- Computing a modular square root of -1 modulo z . This is the step which may fail if z' is not a prime, due to our restricted implementation. We cannot *force* the input to be prime, since it would require to implement a primality testing as a quantum circuit, which would cost more than the algorithm we are implementing here.
- Computing the “half-GCD” algorithm. In our implementation, this step always terminates and does not fail.

The constraints in both steps, and the way to solve them, are similar to constant-time implementations like [9], although the details are different. In the first step, we reduce the square root computation to a single exponentiation modulo z . In the second step, like [9], we decompose the Euclidean divisions in the half-GCD into additions and subtractions by simple multiples, but we go further to ensure that the space remains linear in n .

Modular square root. We detail the computation of the modular square root of -1 .

- In the odd case, $z' = z$, let x be a quadratic nonresidue modulo z . Then, $x^{(z'-1)/4} \bmod z$ is a square root of -1 modulo z .

Algorithm 2 Cornacchia's algorithm, classical version

Input: An input $z > 0$ such that $z \bmod 8 \in \{1, 2, 5\}$.

Output: (x, y) solution to $x^2 + y^2 = z$ in $\mathbb{Z}$ or $\perp$ if no solution was found (which implies that z or $z/2$ was not a prime).

- 1: Compute r a square root of $-1 \bmod z$. If the computation fails, use any value $\bmod z$
 - 2: $s \leftarrow z$
 - 3: $r, s \leftarrow s, (r \bmod s)$
 - 4: **while** not $s^2 < m \leq r^2$ **do**
 - 5: $r, s \leftarrow s, (r \bmod s)$
 - 6: $x, y \leftarrow s, \lfloor \sqrt{m - s^2} \rfloor$
 - 7: **if** $x^2 + y^2 = z$ **then**
 - 8: **return** x, y
 - 9: **return** $\perp$
-

Algorithm 3 Reversible Cornacchia's algorithm for quantum implementation

Input: z of $O(n)$ bits such that $z \bmod 8 \in \{1, 2, 5\}$.

Output: $x, y, \text{success}$ such that: if $\text{success} = 1$, then $x^2 + y^2 = z$.

- 1: $r \leftarrow$ modular square root of -1 modulo z $\triangleright$ See “Modular Square Root” paragraph
 - 2: $\text{shift} \leftarrow 0$
 - 3: $\text{stop} \leftarrow 0$ $\triangleright$ Variable that determines if the iterations have actually stopped
 - 4: $u_z \leftarrow \lfloor \sqrt{z} \rfloor$
 - 5: $s' = z \bmod r$
 $\triangleright$ “If” statements are controlled blocks, whose control bits are stored into garbage registers of size $O(n)$ for reversibility.
 - 6: **for** $O(n)$ iterations **do**
 - 7: **if** $s' > r$ and $\text{shift} = 0$ **then**
 - 8: $r, s' \leftarrow s', r$
 - 9: $\text{stop} = \text{stop} + (s' \leq u_z)$
 - 10: **if** $s' > r$ and $\text{shift} > 0$ **then**

$\triangleright$ Moving s back to its original value
 - 11: $\text{shift} = \text{shift} - 1$
 - 12: $s' \gg= 1$ $\triangleright$ This shift corresponds to a division by 2 of s'
 - 13: **if** $2s' \leq r$ and $\text{stop} = 0$ **then**

$\triangleright$ Shifting s' because it's too small compared to r
 - 14: $\text{shift} = \text{shift} + 1$
 - 15: $s' \ll= 1$ $\triangleright$ This shift corresponds to a multiplication by 2 of s'
 - 16: **if** $s' \leq r$ and $r < 2s'$ and $\text{stop} = 0$ **then**
 - 17: $r \leftarrow r - s'$
 - 18: $x \leftarrow \lfloor \sqrt{z - s'^2} \rfloor$ $\triangleright$ Now $s' = s$ contains the result
 - 19: $\text{success} \leftarrow (x^2 + s'^2 == z)$
 - 20: **Return** $x, s, \text{success}$
-

- In the even case, $z = 2z' \equiv 2 \pmod{8}$. Let x a quadratic nonresidue modulo z' . Then, $z' + (x^{(z'-1)/4} \pmod{z'})(1 + z') \pmod{z}$ is a square root of -1 modulo z .

We unify both cases in a single circuit which determines z' , the exponent and the modulus, and computes a single exponentiation (handling both cases at once). An important point is the choice of x , a quadratic nonresidue modulo z . In order to avoid exceptional cases in the modular exponentiation circuit of Qarton, we also require x to be coprime with z .

In our circuit, we precompute a list of 50 large primes, which is fixed and built-in. We test sequentially if they are quadratic nonresidues modulo z' , and keep the first such one. Testing quadratic residuosity is done using Qarton’s implementation of the Jacobi symbol, which is a simplified implementation of an algorithm from [23]. The circuit is only guaranteed to work if z' and the inputs are coprime; this is the case with high probability, since we chose large constant primes for x . It has a gate count in $O(n^2)$ and space $O(n)$.

Since we test only 50 values, the complexity of this step remains in $O(n^2)$. It is also easy to implement and test. However, this introduces a probability of failure in our circuit (roughly 2^{-50}). This not disrupt the hidden shift algorithm, since it makes much less than 2^{50} queries in total (we can also always increase further the number of x in our trials). Next, exponentiation is done using Qarton’s implementation of Fibonacci exponentiation for modular integers, which is described in [58]. It has space $O(n)$ and performs $O(n)$ iterations, where each iteration contains modular multiplications. Since Qarton’s multiplications are in $O(n^2)$ gates, exponentiation has $O(n^3)$ gates.

so we could stop before the half-GCD step. However, because this second step has only a gate count $O(n^2)$, doing so would make the algorithm more complex for no gain.

Half-GCD. The second step is the “half-GCD” computation performed between Step 2 and 5 in Algorithm 2. Our reversible implementation adapts the constant-time implementation proposed in [9] (Algorithm 4). In this algorithm, the iterations are simplified: effectively we are unrolling the $r \bmod s$ operation into a sequence of subtractions of r by shifts of s , i.e., s multiplied by a certain power of 2, depending on the current values of r and s . In order to achieve competitive space and gate counts, we also unroll the *shift* of s , as detailed below. Note that this implementation differs from usual quantum circuits for binary GCD (like [39]), since the binary GCD iterations erase the *least significant bits* first, while we need to erase the most significant bits first, like in the Euclidean algorithm.

The difference between our algorithm and Algorithm 4 in [9] is that the shift of s is not computed separately at each iteration, which would require to remember the value of the shift for each iteration, thus costing space. Instead, we store two registers, one for a current **shift** value, and one for the value $s' = 2^{\text{shift}}s$ (a shifted version of s). Then, at each iteration, we can either:

- Swap r, s' if we are going to a new modular reduction – but only if **shift** = 0, i.e., $s = s'$;

- Reduce `shift` and shift s' to move s' back to s ;
- Increase `shift` and shift s' to move to a bigger multiple;
- Compute $r \leftarrow r - s'$ if the values r, s' are such that $s' \leq r$ and $r < 2s'$, i.e., s' is the correct multiple to subtract.

The authors of [9] show that the number of iterations of their algorithm is linear in n . We can use that to deduce that the number of iterations of ours is linear as well. Indeed, consider a block of iterations alongside which `shift` increases from 0 to a maximum value t , then decreases towards 0 – this block would correspond to a single iteration in [9]. There are at most $3t + 3$ steps in such a block (we shift right, shift left or perform subtractions). Before the block, $s \geq 2^{t-1}r$ by definition of t . After the block, $s < r$. Consequently, these steps reduced s by t bits. By summing the contributions of all these blocks, as well as the additional swaps between r and s , we see that at most $O(n)$ iterations are performed in total. Combining this with the modular square root, results in Algorithm 3, our quantum version of Algorithm 2. Note that the control bits of each iteration are stored into garbage registers; we erase these later by running the reverse of the algorithm. This algorithm is implemented so that it never fails per se, but only returns `success = 0` if no solution was obtained.

3.4 Preliminary step: reduction of ideals

In Section 3.1, Algorithm 1, we supposed that the input ideal $\mathfrak{a}$ is minimal in its class. In this section, we show how to perform the precomputation to obtain such an input ideal. We also show that this part does not impact much the overall gate count; it is therefore not currently included in our implementation.

Finding a smallest representative in a given ideal class $[\mathfrak{a}]$ is a lattice reduction problem in dimension 2, which can be solved by the Gauss–Lagrange algorithm. David, Espitau and Hosoyamada [32] give a quantum algorithm for binary quadratic form reduction – and so, Gauss–Lagrange reduction – in $O(n^2)$ space and $O(n^2 \log n)$ gates for inputs represented on n bits. We do not use this algorithm due to its large space complexity. Instead, we explain how to perform Gauss–Lagrange reduction with $O(n)$ space and $O(n^2)$ gates, and leave a complete implementation for future work.

Let (u, v) a basis of a two-dimensional lattice, equipped with an inner product denoted as $(u \cdot v)$. The Gauss–Lagrange reduction algorithm repeatedly replaces (u, v) by (u, v') where v' is the shortest vector in $v + u\mathbb{Z}$ which can be obtained by $v' = v - \lfloor \frac{(u \cdot v)}{\|u\|^2} \rfloor u$.

Our idea is to perform reductions by powers of 2, in a way similar to the iterations in Cornacchia’s algorithm in subsection 3.3. At each reduction, we compute v' by $v' = v \pm 2^i u$ where i is the integer minimizing $\|v'\|$, until $\|v\| \leq \|u\|$ (then we swap them). The algorithm will now store (u, v, i) and at each iteration, perform one of the following, depending on the current values of u and v :

- Swap u and v ;
- Increase i ;

- Decrease i ;
- Reduce: $v \leftarrow v \pm 2^i u$.

These operations require $O(n)$ gates. The tests that control them depend on the values of u and v , as well as $(u \cdot v)$, $\|u\|^2$ and $\|v\|^2$. We notice that we can update the norms and dot-products at each iteration in $O(n)$ gates, using only additions and subtractions of $O(n)$ -bit integers. So all of the computations in one iteration contain $O(n)$ gates. To ensure reversibility, we only need to store the control bits that determine if each operation was applied, i.e., $O(1)$ bits per iteration.

Finally, we show that after $O(n)$ iterations, the basis is reduced. To show this, we consider a “block” of iterations which increase the shift i from 0 towards a maximum value t , then decrease it, then swap u and v . This block of iterations realizes the projection $v \leftarrow v - \lfloor \frac{u \cdot v}{\|u\|^2} \rfloor u$. It contains at most $3t + 3$ iterations, where we can either decrease or increase the shift, and perform a reduction.

Because the maximal (and first) reduction by a power of 2 is performed with a shift t , we have $\|v \pm 2^t u\|^2 < \|v\|^2$ hence

$$2^{2t} \|u\|^2 - 2^{t+1} (u \cdot v) + \|v\|^2 \leq \|v \pm 2^t u\|^2 < \|v\|^2,$$

and $2^{t-1} \|u\| < \|v\|$. After the block of iterations, $\|v\| \leq \|u\|$, thus the norm of v was reduced by t bits. Summing the contributions of these blocks, $O(n)$ iterations are sufficient for the algorithm. Thus, the described quantum circuit reduces ideals in space $O(n)$ and gate count $O(n^2)$. This does not dominate the $O(n^3)$ gate count from the rest of the algorithm.

3.5 Output adaptation: from norm equation to isogeny computation

In order to use the output of the norm equation for the isogeny computation, some conversions are needed. From the output of [Algorithm 1](#), we compute $A_1, A_2, N_1, \sigma_1, \sigma_2, \sigma_3 \sigma_4 \in \mathbb{Z}/2^{e+2}\mathbb{Z}$ defined by $N_1 = n(\mathfrak{a}_1) + n(\mathfrak{a}_3)$ and $\sigma_1 = C_1 - E_1 \pmod{2^{2+e}}$, $\sigma_2 = \sigma_1 + C_2 - E_2 \pmod{2^{2+e}}$, $\sigma_3 = B_1 + D_1 \pmod{2^{2+e}}$, $\sigma_4 = \sigma_3 + B_2 + D_2 \pmod{2^{2+e}}$ where $A_1 + A_2 \omega = \frac{\alpha_1 \alpha_3}{N}$, $B_1 + B_2 \omega = \frac{\alpha_1 \alpha_2}{N}$, $C_1 + C_2 \omega = \frac{\alpha_1 \bar{\alpha}_4}{N}$, $D_1 + D_2 \omega = \frac{\alpha_3 \bar{\alpha}_4}{N}$, $E_1 + E_2 \omega = \frac{\alpha_2 \bar{\alpha}_3}{N}$. These are sufficient to compute a basis of the kernel of the 4-dimensional isogeny of [Section 4.4](#) as detailed in [\[31, Page 21\]](#), assuming a suitable basis P, Q of the 2^e -torsion. The required modular integer operations for this post-computation are translated literally into their quantum-circuit counterparts. For details, see pseudocode in [Appendix B](#).

3.6 Results

In [Table 2](#) we give the resource counts for the norm equation circuit. Since all sub-circuits of the norm equation iteration have gate count $O(n^2)$, except the modular exponentiation in the square root computation which is in $O(n^3)$, we check that this circuit dominates the total gate count.

We note that our Qarton implementation allowed for precise estimations of circuit metrics, but it has a few limitations. First, we reused most of our

basic arithmetic circuits from the Qarton library, which only has “schoolbook” multipliers. Since we can allow a relatively large ancilla space, we could presumably do better with Karatsuba multiplication. However, we need a modular multiplier with a *variable modulus*, whereas most existing works use a fixed modulus in the context of Shor’s algorithm [44]. Second, as justified in the text, some non-dominating components were omitted in the implementation: the ideal reduction and the dynamic probability estimations in the search.

Table 2: Qubit count and CCX gate count of the norm equation circuit by security level.

Bit size of p	Number of qubits	$\log_2(\text{CCX count})$	Proportion of CCX gates in the $\sqrt{-1}$ computation
500	11,743	36.25	0.90
1000	23,205	39.36	0.95
2000	46,151	42.51	0.98
4000	91,973	45.84	0.98

4 Quantum algorithms for isogeny computation

We now focus our efforts on the second part of the group action computation: its interpolation as a superposition of 4-dimensional isogeny of degree 2^e between product of abelian varieties and, more specifically, the computation of Equation (3) from its kernel. To do so, we will rely on quantum circuits for both finite fields and elliptic curves operation, highlighting in Section 4.2 the difference with those encountered in ECC cryptography. Then, we detail in Section 4.3 circuits for abelian fourfold arithmetic and isogenies, building on the formulae introduced in [29], merging them in Section 4.4 to compute our 4-dimensional isogeny.

4.1 Algorithmic simplification with loss of generality

Our quantum norm equation solver matches the qt-Pegasis paper [31] closely, therefore solving the equation only in cases where the input ideal (assumed shortest in its class) has norm $N \not\equiv 2 \pmod{4}$. The case $N \equiv 2 \pmod{4}$ is managed in [31] by taking a 2-step away using a one-dimensional isogeny, and adapting the ideal to be applied accordingly, so that it now has norm $N \not\equiv 2 \pmod{4}$. Then, this new ideal is applied to the new starting curve.

However, we did not implement this 1-dimensional 2-isogeny step. Implementing it would amount to extending the circuit with a one-dimensional isogeny on top of $O(e \log(e))$ two-dimensional ones (see Section 4.4). Therefore, we expect its impact on the quantum circuit size to be minimal. This simplification means that our implementation is insufficient for running Kuperberg’s algorithm, as that

algorithm only tolerates a negligible amount of failing ideal classes, and there is a non-negligible number of classes whose shortest norm is $2 \bmod 4$. However, as explained above, we believe that generalizing our circuit would not noticeably increase its complexity.

4.2 Operations on finite fields and elliptic curves

The gate count of our entire circuit is dominated by additive operations in $\mathbb{F}_p$. They are used themselves in modular multiplication and squaring, for which we rely on simple double-and-add implementations from the Qarton library [66]. We added new *non-exact* circuits for modular doubling and controlled additions for primes of the form $p = f2^{e+3} - 1$ as used in qt-Pegasis. These circuits are very similar to the ones given in [65] for pseudo-Mersenne primes (of the form $2^e - f$). The idea is to start from exact modular doubling and addition circuits like in [62], which would do the following:

1. Perform the doubling (resp. addition) non-modularly;
2. Perform a comparison of the result with p and subtract p if it is bigger.
3. Perform a check to uncompute the comparison bit. For doubling, we know that p was subtracted if and only if the output is odd. For in-place addition of x into y , we know that p was subtracted if and only if the output is smaller than x .

Then, the first restriction is to perform comparisons only on the most significant bits. This does not take into account the form of the prime. The second restriction is to simplify the subtraction of p . For a prime of the form $f2^{e+3} - 1$, subtracting p can be done as follows: first, we subtract f from the most significant bits; then we increment the least significant bits. The subtraction of f is exact, while the increment is done on a restricted window. We used window sizes of 80 bits to ensure a negligible probability of failure on random inputs across all parameter sizes¹², while bringing the cost significantly down. For large n , the cost of modular doubling becomes negligible, and the cost of (controlled) modular addition is equivalent to uncontrolled modular addition. Thus, n -bit modular products have Toffoli gate count $\simeq 2n^2$ (n controlled additions).

We also implemented point addition on curves in the Montgomery model, which were not available in Qarton. Note that the point addition circuit is more complex than the elliptic curve point addition which typically appears in Shor's algorithm (e.g., in [65]), because the latter is *quantum-classical* (one of the points is a classical constant), while the circuits we use are quantum-quantum. Furthermore, we require them to work with a *variable* curve, whose parameters can be an additional input, while the prime remains fixed. Our out-of-place point addition contains 10 modular multiplications and 2 inverses (forwards and

¹² More precisely, our controlled addition works for uniformly random inputs of the form (x, y) , $(x, p - x)$, $(0, y)$, $(x, 0)$, with the corner cases possibly occurring within some algorithms like it was observed in [65]. Our doubling works for uniformly random inputs, as well as for $x = 0$.

backwards), for a total of $O(n^2)$ gates. In-place point addition doubles this. Scalar multiplication by an n -bit integer is performed in $O(n^3)$ gates and $O(n)$ space using Fibonacci exponentiation, like for modular integers [58].

4.3 The theta model

We needed to define circuits to compute higher dimensional isogenies in the theta model, derived from Mumford theory [53]. We omit most of its underlying mathematics, and direct the reader to [28, 29] for more details. Unlike for norm equation, constant time algorithms for 4-dimensional isogenies, with a special attention to qt-Pegasis, were proposed in [29] and serve as basis for $\mathbb{F}_p$ -arithmetic circuits which, combined with the Qarton library, results in practical quantum circuits for the computation of the 4-dimensional isogeny chain.

The theta model of an abelian fourfold A is an embedding $\theta : A/\pm 1 \hookrightarrow \mathbb{P}^{15}$ induced by a symmetric *level-2 theta structure*. ([29, Definition 3.1]). Points of abelian fourfold are therefore represented as (projective) vector in $\mathbb{F}_p^{16}$. All computations performed in the theta model can be seen as combinations of simple operations on vectors over $\mathbb{F}_p$. The main building blocks are as follows:

- $H(x)$, the Hadamard transform of a vector $x \in \mathbb{F}_p^{16}$ with $H(x)_i = \sum_j (-1)^{\langle i|j \rangle} x_j$.
- $\odot$, the coefficient-wise product of two vectors.
- $S(x) = x \odot x$ the coefficient-wise squaring of x .
- $I(x)$, which denotes the projective inverse of a vector x , defined by $I(x)_i := \prod_{j \neq i} x_j$. By exploiting self-similarities, it can be computed using $3 \cdot (2^4 - 2)$ multiplications over $\mathbb{F}_p$. Specifically:

$$\rho_i^L = \prod_{j < i} x_j, \quad \rho_i^R = \prod_{j > i} x_j, \quad I(x)_i = \rho_i^L \cdot \rho_i^R.$$

Using these fundamental building blocks, we can construct all the routines required to compute higher-dimensional (HD) isogenies chain [Algorithm 4](#). Those subroutines are derived from [29] and detailed in [Appendix E](#).

4.4 Computing the 4-dimensional chain

Isogeny kernel. Given a valid solution as defined in [Section 3.5](#), we now compute the kernel of Φ , as described in [Equation \(3\)](#). We first consider (P, Q) a basis of $E[2^{e+2}]$ defined over $\mathbb{F}_p$ and two 2-torsion points $T_P = \omega(P) - P$ and $T_Q = \omega(Q)$, with ω as defined at the beginning of [Section 2.1](#). Note that (P, Q, T_P, T_Q) can be classically precomputed. Using P, Q, T_P, T_Q , we compute $\{T_1, \dots, T_4\}$ an isotropic basis of our kernel over E^{413} compatible with $\ker(F)$ (meaning that $\ker(F) = [4]\langle T_1, \dots, T_4 \rangle$). This basis is defined as:

$$\begin{aligned} T_1 &= ([N_1]P, \mathbf{0}, [\sigma_3]P + [\sigma_6]T_P, [-\sigma_1]P + [\sigma_5]T_P) \\ T_2 &= (\mathbf{0}, [N_1]P, [\sigma_2]P + [\sigma_5]T_P, [\sigma_2]P - [\sigma_5]T_P) \\ T_4 &= ([1 - \alpha 2^e]Q, \mathbf{0}, [\alpha \sigma_4]Q + [\alpha \sigma_6]T_Q, [-\alpha \sigma_2]Q + [\alpha \sigma_5]T_Q) \\ T_8 &= (\mathbf{0}, [1 - \alpha 2^e]Q, [\alpha \sigma_1]Q + [\alpha \sigma_5]T_Q, [-\alpha \sigma_3]Q + [-\alpha \sigma_6]T_Q) \end{aligned} \tag{7}$$

¹³ More precisely, over $(E \cup E^\top)^4$

with $\alpha = N_1^{-1} \bmod 2^{e+2}$ and all σ_i denoted in [31, equation 12]. We then map the T_i tuples into the theta model; more specifically, into a theta structure compatible with the chain. Following [29], this can be classically computed using Montgomery arithmetic: let $[2^e]P = (a+b : * : a-b)$ and $([2^{e+1}]Q) = (0, 0)$. Then, following [29, Lemma 4.1], for any $X = (X_1, X_2, X_3, X_4)$ with $X_i = (x_i : * : z_i)$,

$$\theta^{E^4}(X) = \mathbf{M}_2 \cdot \left(\begin{pmatrix} a \\ b \end{pmatrix}^{\otimes 4} \odot \left(\overline{H} \cdot \otimes_{i=1}^4 \begin{pmatrix} x_i \\ z_i \end{pmatrix} \right) \right)$$

with $\otimes$ the tensor/Kronecker product, $\odot$ the coefficient wise product, $\overline{H}$ is a bitflip Hadamard transform $\overline{H}((x_i)_i) \rightarrow (\sum_i (-1)^{\langle j|i \rangle} x_i)_{\text{bitflip}(j)}$ and finally with $\mathbf{M}_2$ an invertible matrix dependent on the qt-Pegasis solution. There are exactly 4 possible matrices, all detailed in [29, Appendix A]. $\mathbf{M}_2$ can be expressed as $\sum_{i=1}^4 v_i \mathbf{M}_{2i}$ with $v_i \in \{0, 1\}$ given by $v_i = 1$ if and only if $\sigma_i \equiv 0 \bmod 4$. Note that, apart from $\mathbf{M}_2$, this mapping to the theta model is classical-quantum.

Isogeny chain. Using the T_i points in the theta model and formulae for point doubling and isogeny evaluation, we compute the 4-dimensional isogeny chain as detailed in Algorithm 4. The naive $O(e^2)$ operations (either isogeny evaluations or point doublings, each requiring $O(1)$ multiplications in $\mathbb{F}_p$), can be optimized to $O(e \log e)$ field multiplications (i.e., $O(n^3 \log(n))$ gates) by employing divide-and-conquer optimal strategies [34]. We represent a strategy $S = (s_1, \dots, s_{t-1})$ as a sequence of integers that dictates the optimal sequence of doublings and evaluations. To perform these operations efficiently, we maintain four key variables at step i : the inverted dual theta null point $T = I \circ H(\theta^{B_i}(0))$ used to construct the isogeny evaluations, and the constants U_i, V_i derived from the current null point $\theta^{B_i}(0)$ used to compute point doublings. To navigate the strategy tree, at each step $i = 1, \dots, e$ we construct the following lists:

- $L_{\text{levels}}^{(i)}$: contains integers $[s_0^{(i)} = 0, s_1^{(i)}, \dots, s_{L_i}^{(i)}]$ such that $\sum_{r=1}^L s_r^{(i)} = e - i$. To construct $L_{\text{levels}}^{(i+1)}$, take the previous list dropping $s_{L_i}^{(i)}$, and append new integers “consumed” from the strategy sequence S until the condition $\sum_{r=1}^L s_r^{(i+1)} = e - (i + 1)$ is satisfied. Note that these lists are classical and are in practice just one single list.
- $L_{\text{basis}}^{(i)}$: stores tuples of points $[\mathcal{B}_{i,0}, \dots, \mathcal{B}_{i,L_i}]$, where we define $\mathcal{B}_0 = (T_\ell)_{1 \leq \ell \leq 4}$ and $\mathcal{B}_{i,r} = ([2^{\sum_{r'=1}^r s_{r'}^{(i)}}] f_{i-1} \circ \dots \circ f_1(\mathcal{B}_0))$. Note that $\mathcal{B}_{i,0}$ always consists of points of order $e + 3 - i$, while $\mathcal{B}_{i,L_i}$ has order 8 and is a basis of the kernel of f_i .

To ensure reversibility of the quantum circuit, during the computation of the isogeny chain, all lists $L_{\text{basis}}^{(i)}$ are kept in memory. The time complexity above requires that the length of $L_{\text{levels}}^{(i)}$ is bounded by $O(\log(e))$; with this condition, the isogeny chain requires to keep in memory $O(e \log(e))$ quantum registers containing $\mathbb{F}_p$ -elements, i.e., $O(e^2 \log(e))$ qubits.

Algorithm 4 IsogenyChain

Input: Points $T_1, \dots, T_g$ in theta coordinates adapted to isogeny chain $f = f_e \circ \dots \circ f_1$ such that $\ker(f) = [4]\langle T_1, T_2, T_4, T_8 \rangle$; the initial theta null point $\theta^{B_0}(0)$ over the domain of f ; optimal strategy $S = (s_1, \dots, s_{t-1})$.

Output: The theta null point $\theta^{B_e}(0)$ of the codomain.

- 1: $k \leftarrow 1$
- 2: $L_{\text{levels}}^{(1)} \leftarrow [0]$
- 3: $L_{\text{basis}}^{(1)} \leftarrow [(T_1, T_2, T_4, T_8)]$
- 4: **for** $i = 1$ to e **do**
- 5: $\mathcal{B}_K \leftarrow$ last element of L_{basis}
- 6: $U \leftarrow I(\theta^{B_{i-1}}(0)), \quad V \leftarrow I \circ H \circ S(\theta^{B_{i-1}}(0)).$
- 7: **while** $\sum_{x \in L_{\text{levels}}^{(i)}} x \neq e - i$ **do**
- 8: Append s_k to $L_{\text{levels}}^{(i)}$
- 9: $\mathcal{B}_K \leftarrow [2^{s_k}] \mathcal{B}_K$ ▷ double using U, V and Equation (8)
- 10: Append $\mathcal{B}_K$ to $L_{\text{basis}}^{(i)}$
- 11: $k \leftarrow k + 1$
- 12: ▷ Invariant: at the end of while loop, $[4]\langle \mathcal{B}_K \rangle = \ker(f_i)$
- 13: Compute T from $\mathcal{B}_K$ ▷ Use Equations (10) and (11)
- 14: Pop last elements from $L_{\text{levels}}^{(i)}$ and $L_{\text{basis}}^{(i)}$
- 15: $L_{\text{basis}}^{(i+1)} \leftarrow [f_i(\mathcal{C}) \mid \mathcal{C} \in L_{\text{basis}}^{(i)}]$ ▷ Use Equation (9) with T
- 16: $\theta^{B_i}(0) \leftarrow H \circ I(T).$
- 17: **return** $\theta^{B_e}(0)$

Splitting. Finally, once we have finished our dim 4 chain of isogenies, we retrieve a theta null point over the abelian variety $E_{\mathfrak{a}} \times E_{\overline{\mathfrak{a}}} \times A$ and our goal is to recover $E_{\mathfrak{a}}$. This can be done using linear transformations [29, Sage code: precomp] that we implemented without data-dependent branching.

Finally, we can improve the memory complexity to roughly $O(e \log^2 e)$ qubits at the expense of a higher gate complexity (by a factor $\log e$). The idea is to rewrite the sequence of computations as a sequence of $O(e)$ irreversible steps; the state between each step is only of $O(e \log e)$ qubits. Gidney’s “spooky pebbling” game [37] (using measurement-based uncomputation) generically transforms such a sequence into a reversible quantum circuit which stores only $\log e$ copies of internal states between steps in quantum memory, at most. We refer to [37] for a detailed description of this strategy.

4.5 Results and comparison

As for the norm equation part, we implemented a quantum isogeny chain circuit in Qarton [66] based on the Sagemath implementation of [29]. We first implemented (and tested) the “sequential” version of this circuit, then implemented (and tested) the “pebbled” version using Gidney’s scheme.¹⁴ The required time and space resources for the considered prime sizes are listed in Table 3 and in Table 4

¹⁴ The implementation is available at github.com/PQarrots/quantum-group-actions.

for the “pebbled” version. The asymptotic dependency of the qubit count on e (quadratic in the first case, quasi-linear in the second) is clear. Taking into account the norm equation, the total gate count is the same up to the second decimal in the exponent.

Bit size of p	Number of qubits	$\log_2(\text{CCX count})$
500	99,072,594	42.84
1000	431,447,379	45.91
2000	1,867,110,890	49.00
4000	7,977,429,557	52.10

Table 3: Qubit count and CCX gate count for the isogeny chain circuit, “chain” construction. These numbers were computed using the “balanced” strategy.

Bit size of p	Number of qubits	$\log_2(\text{CCX count})$
500	2,853,795	45.97
1000	8,421,880	49.04
2000	26,114,139	52.14
4000	88,156,290	55.26

Table 4: Qubit count and CCX gate count for the isogeny chain circuit, “pebbled” construction.

Bit-size of p	Toffoli gates	Qubits
512	$2^{49.6}$	$< 40\,000$
1024	$2^{54.8}$	$< 80\,000$
1792	$2^{58.9}$	$< 140\,000$

Table 5: Gate and qubit counts for evaluating the CSIDH class group action, as estimated in [15] (the numbers in [43] differ only by a factor 2).

We can compare these numbers with existing estimations for CSIDH. Bernstein et al. [11] estimate that the CSIDH-512 group action can be evaluated in about $0.7 \cdot 2^{40}$ nonlinear operations (i.e., Toffoli gates). However, their optimization is purely oriented towards gate count, and does not take reversibility into account. Making their circuit generically reversible incurs a memory cost of the same amount of qubits. Bonnetain and Schrottenloher [15] and more recently Huang et al. [43] analyze a quantum circuit implementation using less memory. We recall the results of [15] in Table 5, taking the least number of gates. While our “pebbled” construction uses more memory, it is most certainly non-optimal in space. Indeed, we used a generic implementation of the pebbling game, which does not take into account the differences between steps.

5 Improved trade-offs for Kuperberg’s algorithm

In quantum computing, many algorithms have been proposed for the hidden shift problem (Problem 1), either low in queries with exponential processing, or subexponential in time and queries [35, 47, 59, 48, 24, 14, 12, 57, 15, 60].

We focus on the most tunable and time-efficient variant, Kuperberg’s second algorithm [48] with some improvements from [57]. We provide improved tradeoffs inspired by [15] that allow us to make concrete security estimates. Our optimizations are the best known in the literature for the security levels of interest studied here.

In what follows we assume the hidden shift group is $G = \mathbb{Z}/N\mathbb{Z}$, the general case works the same way. The functions are f_0 and f_1 , and s is the shift. We note $n = \log_2(|G|)$ the asymptotic parameter of the algorithm.

Phase qubits. From quantum oracle access to f_0 and f_1 , one can produce the qubit $|\psi_\ell\rangle = |0\rangle + \exp\left(2i\pi \frac{s\ell}{N}\right) |1\rangle$, with a phase that depends on s and a random, classically known ℓ .

Phase registers. If we make a quantum register from t phase qubits with phases $\ell_0, \dots, \ell_{t-1}$, the corresponding quantum state can be written as $\sum_{j=0}^{2^c-1} \exp\left(2i\pi \frac{sL_j}{N}\right) |j\rangle$, with $L_j = \sum_{i=0}^{t-1} b_i \ell_i$ where $b_0 \dots b_{t-1}$ is the binary representation of j . As above, we have a small quantum state with a phase that depends on s and the classically known list $L = (L_0, \dots, L_{2^t-1})$.

Recovering s . From a phase vector associated with a phase list L whose phases are all smaller than $|L|$ we can recover around $\log_2(|L|)$ bits of s [57].

Collimation. It is possible to merge r phase registers with phase lists of size $\sim 2^t$ and whose phases are all bounded by 2^k into one whose phases are bounded by 2^{k-c} . The expected size of the final list is 2^{rt-c} . We call c the constraint of the merge. The main classical part is to compute all the solutions of an r -list problem, which can be done classically, up to polynomial factors, in $2^{\lceil \frac{r}{2} \rceil t}$ operations and $2^{\lceil \frac{r}{4} \rceil t}$ memory [64]. The quantum part is simple arithmetic plus the final renormalization of the quantum state, which amounts in 2 accesses in quantum superposition to the classically-computed list of solutions. This costs:

- assuming efficient quantum-accessible classical memory access, polynomial quantum time,
- in the quantum circuit model, 2^{rt-c} quantum time, as superposition access can be emulated by sequential equality test+conditional write for each value.

Merging tree. The algorithm can be seen as a tree where the leaves are generated from the quantum queries, and each node combines phase vectors into better ones, until a good enough one is obtained. This leaves many free parameters: the arity of each combination, the initial list size, as well as the constraint put on each merge (or, equivalently, the expected output list size), which means that depending on the cost of a query and the amount of classical and quantum resources available, many tradeoffs are possible.

5.1 Hidden shift exponents

Standard asymptotics. Assuming classical and quantum time are equivalent and queries are very cheap, the best one can do is to use a binary tree that starts with lists of size 2 and an expected list size that doubles at each level, as presented in [48]. This allows to reach $2\sqrt{2\log_2(N)}$ classical and quantum time, which is the best known exponent for this problem.

If we consider that quantum operations are more expensive, or that a query is costly, then it is possible to tweak the algorithm to reach a cost much better than $2\sqrt{2\log_2(N)}$ times the query cost, which we optimize below.

Regular trees. If we assume that all the mergings use the same arity r , between one level and the next one there are r times fewer lists. Thus, they can be larger, which allows to balance the overall cost of each level of the tree. There are two natural ways to balance the tree:

1. By [48], the quantum cost is proportional to the list size (in the circuit model); it will be balanced if after a merge the list size is multiplied by r ,
2. By [48], the classical cost is proportional to the list size to the exponent $\lceil r/2 \rceil$; it will be balanced if after a merge the list size is multiplied by $r^{\lceil r/2 \rceil^{-1}}$.

Both are achievable by matching the constraints: With c bits on constraints on r -list merging with lists of size 2^t , we expect to obtain a list of size 2^{rt-c} on average. Thus, the constraints can be set to $c = (r-1)t - \delta$ bits, with $\delta = \log_2(r)$ or $\frac{2}{r}\log_2(r)$ depending on whether we want the lists to increase by a factor r or $r^{\frac{2}{r}}$. In total, we need $\simeq n$ bits of constraints. Thus, given an initial list size, we can estimate the number of levels in the merging tree, and from that estimate the asymptotic cost. If we note t_0 the initial list size, α the number of levels in the tree and δ as above, we want $\sum_{i=0}^{\alpha-1} (r-1)(t_0 + i\delta) - \delta \simeq n$.

This can be approximated to $\sum_{i=0}^{\alpha-1} (r-1)(t_0 + i\delta) = n$, as δ is small compared to the other terms. Solving this equation leads to $\alpha = -\frac{t_0}{\delta} + \sqrt{\frac{t_0^2}{\delta^2} + \frac{2n}{\delta(r-1)}}$ levels in the tree. The dominant classical cost is the merging at the last level,¹⁵ which by (2) above costs $2^{\frac{r}{2}(t_0 + \alpha\delta)} = 2^{\frac{r}{2}\sqrt{t_0^2 + \frac{2\delta n}{r-1}}}$. The quantum query cost is $t_0 r^\alpha = t_0 2^{\frac{\log(r)}{\delta}(-t_0 + \sqrt{t_0^2 + \frac{2\delta n}{r-1}})}$. The quantum time cost is dominated by the merging of the first level,¹⁶ which by (1) above costs $r^\alpha 2^{t_0} = 2^{t_0(\frac{\log(r)}{\delta} - 1) + \frac{\log(r)}{\delta}\sqrt{t_0^2 + \frac{2\delta n}{r-1}}}$ in the circuit model. Hence, we can derive the following formulae:

- By balancing quantum merging costs:
 - $2^{\frac{r}{2}\sqrt{t_0^2 + \frac{2\log_2(r)}{r-1}n}}$ classical time,
 - $2\sqrt{t_0^2 + \frac{2\log_2(r)}{r-1}n}$ quantum time for the merging part,
 - $t_0 2^{-t_0 + \sqrt{t_0^2 + \frac{2\log_2(r)}{r-1}n}}$ quantum queries.

¹⁵ Depending on δ , either all the levels have the same costs, or the last one dominates

¹⁶ Depending on δ , either all the levels have the same costs, or the first one dominates

- By balancing classical merging costs:
 - $2^{\frac{r}{2}} \sqrt{t_0^2 + \frac{4 \log_2(r)}{r(r-1)} n}$ classical time,
 - $2^{t_0(1-\frac{r}{2}) + \frac{r}{2} \sqrt{t_0^2 + \frac{4 \log_2(r)}{r(r-1)} n}}$ quantum time for merging,
 - $t_0 2^{\frac{r}{2} \left(-t_0 + \sqrt{t_0^2 + \frac{4 \log_2(r)}{r(r-1)} n} \right)}$ quantum queries.

Fundamentally, t_0 dictates the gap between the number of queries and the quantum processing cost, while r dictates the gap between the quantum processing cost and the classical processing cost. From the above formulas, we can derive the following strategies:

- If the quantum processing part costs less than the quantum queries, then the best strategy is to balance classical costs, minimize r and choose t_0 to get the desired classical/quantum trade-off. In particular, assuming qRAM it is optimal to take $r = 2$.
- Otherwise, balancing quantum merging costs is better, and r should be chosen depending on the expected ratio between classical and quantum cost.

In particular, in the qRAM model, the tradeoff space is $2\sqrt{t_0^2 + 2n}$ classical time and $t_0 2^{-t_0 + \sqrt{t_0^2 + 2n}}$ quantum queries, for a gap between the two in $2^{t_0}/t_0$. Assuming an oracle query cost in 2^q , the minimal classical cost is achieved with $t_0 \simeq q + \log_2(q)$, for a classical and quantum time cost in $2\sqrt{(q + \log_2(q))^2 + 2n}$.

5.2 Concrete estimates

Classical and quantum costs. For concrete estimates, we assume that when the quantum computer has to wait for the result of a classical computation that costs $\mathcal{C}$ classical operations, it is equivalent to running $\mathcal{C}/T_c$ quantum operations. This roughly means that classical computing is T_c times faster, but takes into account the parallelism of classical algorithms. As we cannot know this value, we explore a large range of T_c , from $T_c = 1$, which assumes no classical speedup, to $T_c = 2^{48}$. While this is somewhat arbitrary, $T_c = 2^{48}$ allows to estimate what happens when we allow 2^{80} quantum and 2^{128} classical operations, which roughly matches the NIST-1 security level.

We ran a basic optimizer (provided in our code repository) to get the best tradeoffs assuming a given gap between the classical and quantum operation cost, in the quantum circuit model, and using the oracle costs obtained in the previous section. The results are presented in [Table 6](#).

Here is a list of elements not taken into account that could increase the numbers from [Table 6](#):

- It is assumed 2-list-merging costs 1 bit-gate per value in the list. A more refined estimate would be equivalent to an increase of T_c .

Table 6: Concrete quantum cost estimates in the circuit model for varying r and T_c . For $r = 2$, T_c is irrelevant as the classical and quantum processing times match. Costs for $r > 4$ are always worse in this regime. The gate count of a single query is 2^q ; we take the high-memory estimates given in [Section 4.5](#).

p	q	r	$T_c = 1$	$T_c = 2^{10}$	$T_c = 2^{20}$	$T_c = 2^{30}$	$T_c = 2^{40}$	$T_c = 2^{48}$
2048	49	2	2^{71}	-	-	-	-	-
		4	2^{75}	2^{73}	2^{71}	2^{69}	2^{68}	2^{67}
4096	52	2	2^{86}	-	-	-	-	-
		4	2^{93}	2^{90}	2^{87}	2^{84}	2^{82}	2^{81}

- Only the cost of the most expensive level is accounted for. In cases where all the costs are balanced, the corresponding cost is multiplied by the number of levels. For $r = 2$ this impacts classical and quantum time, for $r = 4$ only classical time as the quantum processing time is negligible.
- It is assumed the merging always produces a list with the average number of solutions. Variance effects have been studied in [\[57\]](#), which shows the impact, while asymptotically relevant, is very limited for (reasonable) concrete parameters, and will be below factor 2 for all our concrete parameters.
- Only one run of the algorithm is considered. Such a run recovers only some bits of s . The cost of recovering the remainder depends on the group structure: if the partial information contains a subgroup, following runs can be done in the quotient groups.
- The number of levels as well as the starting list size needs to be an integer.
- The memory footprint of the quantum oracle could be reduced, with an increased cost.

On the other side, a few elements could decrease the cost:

- The actual amount of constraints we need is slightly smaller than n . Moreover, as detailed in [Appendix D](#), it is possible to get some additional constraint by starting the algorithm with subset-sum instead of r -list. Even under some very optimistic assumptions, the impact is small (less than a factor 4) for parameters of interest, and diminishes as n grows.
- An improvement on quantum arithmetic circuits could lead to a better oracle. The improvement would not translate directly to the cost, as for a fixed T_c , to reduce the quantum time we would also need to reduce the classical time.
- We assume there is no large 2-torsion subgroups in the group. This case is strictly easier [\[14\]](#), but there is not reason to expect this here.

Recommended parameters. From the analysis above and the results in [Table 6](#), we conclude that $p = 2048$ is enough to be reasonably resistant against a quantum

computer (i.e., no attack cost less than both 2^{64} quantum operations and 2^{112} classical operation), while $p = 4096$ reaches the NIST-1 security level (no attack costs less than both 2^{81} quantum operations and 2^{128} classical operations).

References

1. Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Giacomo Borin, Isaac Andrés Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Thomas Espitau, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, Riccardo Invernizzi, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Marzio Mula, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez-Henríquez, Sina Schaeffler, Frederik Vercauteren, Alexandre Wallet, Benjamin Wesolowski, and Wessel van Woerden. SQIsign. Technical report, National Institute of Standards and Technology, 2026.
2. Marius A. Aardal, Shahla Atapoor, Karim Baghery, Andrea Basso, Xavier Bonnetain, Giacomo Borin, Daniele Cozzo, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan K. Eriksen, Tako Boris Fouotsa, Arthur Herlédan Le Merdy, Riccardo Invernizzi, Samuel Jaques, Yi-Fu Lai, Dania Lazzarini, Jason T. LeGrow, Chloe Martindale, Luciano Maino, Jonas Meers, Michael Meyer, Sikhar Patranabis, Robi Pedersen, Giacomo Pope, Doreen Riepel, Damien Robert, Ryan Rueger, Sina Schaeffler, André Schrottenloher, and Frederik Vercauteren. PQarrots: Macaw, Kea and Kakapo, January 2026. Preview Writeup, version 0.3, <https://csrc.nist.gov/Projects/threshold-cryptography/tcall-1#PQarrots>.
3. Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 411–439, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.
4. Bill Allombert, Jean-François Biasse, Jonathan Komada Eriksen, Péter Kutas, Chris Leonardi, Aurel Page, Renate Scheidler, and Márton Tot Bagi. PEARL-SCALLOP: Parameter extension applicable in real-life SCALLOP. Cryptology ePrint Archive, Report 2024/1744, 2024.
5. Bill Allombert, Jean-François Biasse, Jonathan Komada Eriksen, Péter Kutas, Chris Leonardi, Aurel Page, Renate Scheidler, and Márton Tot Bagi. Faster SCALLOP from non-prime conductor suborders in medium sized quadratic fields. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part III*, volume 15676 of *Lecture Notes in Computer Science*, pages 333–363, Røros, Norway, May 12–15, 2025. Springer, Cham, Switzerland.
6. Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. CSI-SharK: CSI-FiSh with sharing-friendly keys. In Leonie Simpson and Mir Ali Rezazadeh Bae, editors, *ACISP 23: 28th Australasian Conference on Information Security and Privacy*, volume 13915 of *Lecture Notes in Computer Science*, pages 471–502, Brisbane, QLD, Australia, July 5–7, 2023. Springer, Cham, Switzerland.
7. Ryan Babbush, Adam Zalcman, Craig Gidney, Michael Broughton, Tanuj Khattar, Hartmut Neven, Thiago Bergamaschi, Justin Drake, and Dan Boneh. Securing elliptic curve cryptocurrencies against quantum vulnerabilities: Resource estimates and mitigations. *arXiv preprint arXiv:2603.28846*, 2026.

8. Andrea Basso, Giacomo Borin, Ryan Rueger, and Sina Schaeffler. Coral faster isogeny group action for post-quantum nke. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology – CRYPTO 2026*, pages 506–536, Cham, 2026. Springer Nature Switzerland.
9. 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, Report 2025/2192, 2025.
10. Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany.
11. Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: Optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 409–441, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.
12. Xavier Bonnetain. Improved low-qubit hidden shift algorithms. *CoRR*, abs/1901.11428, 2019.
13. Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.
14. Xavier Bonnetain and María Naya-Plasencia. Hidden shift quantum cryptanalysis and implications. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 560–592, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.
15. Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 493–522, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.
16. Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.
17. Wouter Castryck and Thomas Decru. CSIDH on the surface. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 111–129, Paris, France, April 15–17, 2020. Springer, Cham, Switzerland.
18. Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 423–447, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
19. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.
20. Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear Vélú quantum-resistant

- isogeny action with low exponents. *Journal of Cryptographic Engineering*, 12(3):349–368, September 2022.
21. Mingjie Chen, Antonin Leroux, and Lorenz Panny. SCALLOP-HD: Group action from 2-dimensional isogenies. In Qiang Tang and Vanessa Teague, editors, *PKC 2024: 27th International Conference on Theory and Practice of Public Key Cryptography, Part III*, volume 14603 of *Lecture Notes in Computer Science*, pages 190–216, Sydney, NSW, Australia, April 15–17, 2024. Springer, Cham, Switzerland.
 22. Clémence Chevnard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum factoring. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part II*, volume 16001 of *Lecture Notes in Computer Science*, pages 384–415, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.
 23. Clémence Chevnard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum discrete logarithms on elliptic curves. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026, Part I*, volume 16541 of *Lecture Notes in Computer Science*, pages 371–401, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland.
 24. Andrew M. Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Math. Cryptol.*, 8(1):1–29, 2014.
 25. Leonardo Colò and David Kohel. Orienting supersingular isogeny graphs. Cryptology ePrint Archive, Report 2020/985, 2020.
 26. Giuseppe Cornacchia. Su di un metodo per la risoluzione in numeri interi dell’equazione $\sum_{h=0}^n c_h x^{n-h} y^h = p$. *Giornale di Matematiche di Battaglini*, 46:33–90, 1908.
 27. Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006.
 28. Pierrick Dartois. *Fast computation of higher dimensional isogenies for cryptographic applications*. PhD thesis, Université de Bordeaux, France, 7 2025.
 29. Pierrick Dartois and Max Duparc. Chasing rabbits through hypercubes: Better algorithms for higher dimensional 2-isogeny computations. Cryptology ePrint Archive, Paper 2026/114, 2026.
 30. Pierrick Dartois, Jonathan Komada Eriksen, Tako Boris Fouotsa, Arthur Herlédan Le Merdy, Riccardo Invernizzi, Damien Robert, Ryan Rueger, Frederik Vercauteren, and Benjamin Wesolowski. PEGASIS: Practical effective class group action using 4-dimensional isogenies. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part I*, volume 16000 of *Lecture Notes in Computer Science*, pages 67–99, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.
 31. Pierrick Dartois, Jonathan Komada Eriksen, Riccardo Invernizzi, and Frederik Vercauteren. qt-Pegasis: Simpler and faster effective class group actions. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026, Part IV*, volume 16545 of *Lecture Notes in Computer Science*, pages 636–665, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland.
 32. Nicolas David, Thomas Espitau, and Akinori Hosoyamada. Quantum binary quadratic form reduction. Cryptology ePrint Archive, Report 2022/466, 2022.
 33. Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. SCALLOP: Scaling the CSI-FiSh. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 345–375, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.

34. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Cryptology ePrint Archive*, Report 2011/506, 2011.
35. Mark Ettinger and Peter Høyer. *On Quantum Algorithms for Noncommutative Hidden Subgroups*, page 478–487. Springer Berlin Heidelberg, 1999.
36. Paul Frixons, Valerie Gilchrist, Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Lam L. Pham. Another look at the quantum security of the vectorization problem with shifted inputs. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026, Part I*, volume 16541 of *Lecture Notes in Computer Science*, pages 633–662, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland.
37. Craig Gidney. Spooky pebble games and irreversible uncomputation. <https://algassert.com/post/1905>, aug 2019. Blog post, Algorithmic Assertions.
38. Craig Gidney. How to factor 2048 bit RSA integers with less than a million noisy qubits. *arXiv preprint arXiv:2505.15917*, 2025.
39. Élie Gouzien, Diego Ruiz, Francois-Marie Le Régent, Jérémie Guillaud, and Nicolas Sangouard. Performance analysis of a repetition cat code architecture: Computing 256-bit elliptic curve logarithm in 9 hours with 126 133 cat qubits. *Physical review letters*, 131(4):040602, 2023.
40. Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved quantum circuits for elliptic curve discrete logarithms. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 425–444, Paris, France, April 15–17, 2020. Springer, Cham, Switzerland.
41. Marc Houben. Efficient post-quantum commutative group actions from orientations of large discriminant. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025, Part IV*, volume 16248 of *Lecture Notes in Computer Science*, pages 141–173, Melbourne, VIC, Australia, December 8–12, 2025. Springer, Singapore, Singapore.
42. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.
43. Yan Huang, Yongjie Li, Xiuyu Qiu, Zijian Zhou, Fangguo Zhang, Chao Chen, and Wei Yu. Quantum resource optimization for CSIDH. *Cryptology ePrint Archive*, Paper 2026/1707, 2026.
44. Gregory D Kahanamoku-Meyer and Norman Y Yao. Fast quantum integer multiplication with zero ancillas. *arXiv preprint arXiv:2403.18006*, 2024.
45. Ernst Kani. The number of curves of genus two with elliptic differentials. *Journal für die reine und angewandte Mathematik*, 485:93–122, 1997.
46. Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates. *arXiv preprint arXiv:1206.5236*, 2012.
47. Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35(1):170–188, 2005.
48. Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In Simone Severini and Fernando G. S. L. Brandão, editors, *8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2013, Guelph, Canada, May 21-23, 2013*, LIPIcs, pages 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

49. J. C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. In *24th Annual Symposium on Foundations of Computer Science*, pages 1–10, Tucson, Arizona, November 7–9, 1983. IEEE Computer Society Press.
50. Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th International Workshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22–24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2005.
51. Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 448–471, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
52. François Morain and Jean-Louis Nicolas. On Cornacchia’s algorithm for solving the diophantine equation $u^2 + dv^2 = m$, 1990.
53. David Mumford. On the equations defining abelian varieties. i. 1966.
54. Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.
55. Aurel Page and Damien Robert. Introducing clapoti(s): Evaluating the isogeny class group action in polynomial time. Cryptology ePrint Archive, Paper 2023/1766, 2023.
56. Lorenz Panny, Christophe Petit, and Miha Stopar. KLaPoTi: An asymptotically efficient isogeny group action from 2-dimensional isogenies. Cryptology ePrint Archive, Paper 2024/1844, 2024.
57. Chris Peikert. He gives C-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.
58. Seyoon Ragavan and Vinod Vaikuntanathan. Space-efficient and noise-robust quantum factoring. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VI*, volume 14925 of *Lecture Notes in Computer Science*, pages 107–140, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
59. Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space, 2004.
60. Maxime Rémard, André Schrottenloher, and Jean-Pierre Tillich. Time and query complexity tradeoffs for the dihedral coset problem. In Thomas Johansson and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023*, pages 505–532, College Park, USA, August 16–18, 2023. Springer, Cham, Switzerland.
61. Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 472–503, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
62. Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin E. Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*,

- Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 241–270, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.
63. Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006.
 64. Richard Schroepel and Adi Shamir. A $t=o(2^{n/2})$, $s=o(2^{n/4})$ algorithm for certain np-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.
 65. André Schrottenloher. Optimized point addition circuits for elliptic curve discrete logarithms. *arXiv preprint arXiv:2606.02235*, 2026.
 66. André Schrottenloher and contributors. Qarton. <https://gitlab.inria.fr/capsule/qarton>, 2026. GitLab repository. Accessed: 2026-07-17.
 67. Benjamin Wesolowski. The supersingular isogeny problem in time and memory $p^{1/3+o(1)}$. Cryptology ePrint Archive, Paper 2026/1486, 2026.

A Ideal-isogeny relation and equivalences

For a supersingular elliptic curve E over $\mathbb{F}_p$ we can define an injective ring homomorphism ι mapping $\mathbb{Z}[\omega]$ into the endomorphism ring of E . Given a $\mathbb{Z}[\omega]$ -ideal $\mathfrak{a}$ with norm coprime to p together with a curve E and such a homomorphism ι , we define the isogeny $\varphi_{\mathfrak{a}} : E \rightarrow E_{\mathfrak{a}}$ of degree $n(\mathfrak{a})$ whose kernel is

$$\bigcap_{\alpha \in \mathfrak{a}} \ker(\iota(\alpha)).$$

All equivalent ideals correspond to isogenies with isomorphic codomains. Thus the ideal class group acts on the set of isomorphism classes of supersingular elliptic curves over $\mathbb{F}_p$. This action is free and transitive.

To compute the action of the class group, it is useful to consider representatives of ideal classes of small norm, since the norm of an ideal $\mathfrak{a}$ determines the degree of the corresponding isogeny $\varphi_{\mathfrak{a}}$ and therefore directly affects the complexity of our algorithms.

Recall that Minkowski’s bound guarantees that every ideal class $[\mathfrak{a}] \in \mathcal{C}\ell(R)$ contains an integral ideal $\mathfrak{a}$ satisfying

$$n(\mathfrak{a}) \leq \frac{2}{\pi} \sqrt{|\Delta_R|} = \frac{2}{\pi} \sqrt{p}.$$

Consequently, there always exist ideals $\mathfrak{a}_1, \mathfrak{a}_2 \in [\mathfrak{a}]$ such that $n(\mathfrak{a}_1) n(\mathfrak{a}_2) \leq \frac{16p}{\pi^2}$. In particular, this provides a bound on the size of suitable representatives without requiring us to consider arbitrary ideals in the class.

B Norm equation post-computation

In [Section 3.5](#), we report how the values defining the output of the norm equation algorithm ([Algorithm 1](#)) are transformed to give the modular integers $N_1, \sigma_1, \sigma_2, \sigma_3, \sigma_4$ used in [Equation \(7\)](#). Here we give pseudocode for this computation in [Algorithm 5](#).

Algorithm 5 PostComputation($\mathfrak{a}_1, \dots, \mathfrak{a}_4$)

Input: Integers $e, b_1, b_2 \in \mathbb{Z}$ and ring integers $\delta_1, \delta_2 \in \mathbb{Z}[\omega]$ such that $\mathfrak{a}b_1, \mathfrak{a}b_2, \mathfrak{a}\overline{\delta_1}/N, \mathfrak{a}\overline{\delta_2}/N$ are the outputs of [Algorithm 1](#) with input $(\mathfrak{a}, e)$.

Output: Elements $N_1, \sigma_1, \dots, \sigma_4 \in \mathbb{Z}/2^{e+2}\mathbb{Z}$ used in [Equation \(7\)](#).

- 1: $A_1, A_2 \leftarrow$ coordinates of $b_1\delta_1$ in basis $1, \omega$
 - 2: $B_1 \leftarrow Nb_1b_2$
 - 3: $C_1, C_2 \leftarrow$ coordinates of $b_1\delta_2$ in basis $1, \omega$
 - 4: $D_1, D_2 \leftarrow$ coordinates of $\delta_1\delta_2/N$ in basis $1, \omega$
 - 5: $E_1, E_2 \leftarrow$ coordinates of $b_2\delta_1$ in basis $1, \omega$
 - 6: $N_1 \leftarrow (N \cdot b_1^2 + n(\delta_1)) \bmod 2^{e+2}$ $\triangleright N_1 = (n(\mathfrak{a}b_1) + n(\mathfrak{a}\overline{\delta_1}/N)) \bmod 2^{e+2}$
 - 7: $\sigma_1 \leftarrow (C_1 - E_1) \bmod 2^{e+2}$
 - 8: $\sigma_2 \leftarrow \sigma_1 + (C_2 - E_2) \bmod 2^{e+2}$
 - 9: $\sigma_3 \leftarrow (B_1 - D_1) \bmod 2^{e+2}$
 - 10: $\sigma_1 \leftarrow \sigma_3 + D_2 \bmod 2^{e+2}$
 - 11: **return** $(N_1, \sigma_1, \sigma_2, \sigma_3, \sigma_4)$
-

C Singular rejections

As mentioned in [Remark 2](#), the construction of the 4-dimensional 2^e -isogeny

$$F : E^4 \rightarrow E_{\mathfrak{a}} \times E_{\overline{\mathfrak{a}}} \times \mathcal{A}$$

may run into singularities. In particular, it requires the computation of a chain of 2-isogenies $f_1, f_2, \dots, f_e$, and singularities may occur in the first two steps of this chain, i.e. in the construction of f_1, f_2 , as detailed in [\[31, §4.2\]](#). The authors also describe how to detect these cases at the norm equation solving step, before running Cornacchia's algorithm, as integrated in [Algorithm 1](#). To this end, we now recall the description of the singularity rejections as presented in [\[31\]](#), in order to make clear what leads to these rejection conditions in that form.

The case N odd. Once z has been obtained as in [Algorithm 1](#), there are three possible cases:

- If $z \equiv 0 \pmod{4}$, then $b_1 \equiv b_2 \equiv 0 \pmod{2}$. We reject, since b_1 is required to be odd by [\[31, Proposition 2\]](#).
- If $z \equiv 1 \pmod{4}$, then $b_1 \not\equiv b_2 \pmod{2}$. We choose the signs so that $b_1 \equiv 1 \pmod{4}$ and proceed.
- If $z \equiv 2 \pmod{4}$, then $b_1 \equiv b_2 \equiv 1 \pmod{2}$, and we proceed.

Note that, if $z \equiv 3 \pmod{4}$, the corresponding equations admit no solutions for b_1, b_2 .

The case $N \equiv 2 \pmod{4}$. For $N \equiv 2 \pmod{4}$, the conditions given in [\[31, Lemma 4\]](#) and [\[31, Lemma 5\]](#) for the incomplete gluing of f_1 and f_2 , respectively, are not simultaneously compatible. This case can be instead reduced to the odd case by first applying the 2-isogeny corresponding to the ideal $(2, \alpha)$, and then proceeding with the ideal $(N/2, \alpha)$.

The case $N \equiv 0 \pmod{4}$. When $N \equiv 0 \pmod{4}$, two approaches are possible. One may apply directly the 2-power part of $\mathfrak{a}$, but it's costly. Alternatively, one may handle this case directly by sampling (b_3, b_4) until the norms $n(\mathfrak{a}_3)$ and $n(\mathfrak{a}_4)$ are odd, i.e. until $b_3, b_4 \not\equiv r \pmod{2}$. The construction can then be reduced to the odd N case by exchanging the roles of $\mathfrak{a}_1, \mathfrak{a}_2$ and $\mathfrak{a}_3, \mathfrak{a}_4$.

Let A_i, B_i, C_i, D_i, E_i , as well as the quantities σ_j , be the quantities defined in the pseudocode in [Appendix B](#). To determine the parities of the quantities that are not known before running Cornacchia's algorithm, we introduce

$$\tilde{C}_1 + \omega \tilde{C}_2 = \overline{\alpha_4}, \quad \tilde{E}_1 + \omega \tilde{E}_2 = \overline{\alpha_3}.$$

Avoiding the conditions in [31, Lemma 4]. We first consider the conditions

$$B_1 + D_1 \equiv B_2 + D_2 \equiv 0 \pmod{2}, \quad C_1 - E_1 \equiv C_2 - E_2 \equiv 0 \pmod{2}.$$

By construction, $B_2 = 0$, so the parity of $B_1 = b_1 b_2 N$ is known. Moreover, D_1, D_2 are determined by $D_1 + \omega D_2 = \alpha_3 \overline{\alpha_4} / N$, and therefore known before running Cornacchia's algorithm. It remains to determine the relevant parities of C_i and E_i : despite these values are not known, we have that $C_i = b_1 \tilde{C}_i$ and $E_i = b_2 \tilde{E}_i$. Hence, their parities are determined by the known values of $b_1, b_2 \pmod{2}$ before Cornacchia's algorithm.

Avoiding the conditions in [31, Lemma 5]. We next consider the quantities $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. According to [31, Lemma 5], we must avoid the cases in which any of these quantities is congruent to 2 (mod 4), and the required information on $b_1, b_2 \pmod{4}$ can be obtained from $z \pmod{8}$.

- If $z \equiv 0$ or $4 \pmod{8}$, then b_1 is even, and the instance is rejected by the condition discussed above for odd N .
- If $z \equiv 1 \pmod{8}$, then $b_2 \equiv 0 \pmod{4}$. Consequently, b_2 divides E_1, E_2, B_1 , so we reject the cases in which $D_1, D_1 + D_2, \tilde{C}_1$ or $\tilde{C}_1 + \tilde{C}_2$ is congruent to 2 (mod 4).
- If $z \equiv 5 \pmod{8}$, then $b_2 \equiv 2 \pmod{4}$. In this case, we reject the cases in which D_1 or $D_1 + D_2$ is congruent to 0 (mod 4), and $\tilde{C}_1 - 2\tilde{E}_1$ or $\tilde{C}_1 + \tilde{C}_2 - 2\tilde{E}_1 - 2\tilde{E}_2$ is congruent to 2 (mod 4).
- If $z \equiv 2 \pmod{8}$, then $b_1, b_2 \equiv \pm 1 \pmod{4}$. Since simultaneously changing the signs of b_1 and b_2 leaves all σ_i unchanged, we can restrict the analysis to the two cases $(b_1, b_2) \equiv (1, 1) \pmod{4}$ and $(b_1, b_2) \equiv (1, -1) \pmod{4}$. The corresponding signs can then be adjusted after running Cornacchia's algorithm. In the first case, $B_1 \equiv N \pmod{4}$, $C_i \equiv \tilde{C}_i \pmod{4}$ and $E_i \equiv \tilde{E}_i \pmod{4}$. In the second case, the sign of B_1 and E_i is reversed. The four conditions on the σ_i can therefore be evaluated directly in both cases, and Cornacchia's algorithm is run only if at least one of them avoids a singularity. The sign of b_1 and b_2 has to be updated accordingly.

Finally, if $z \equiv 3, 6$ or $7 \pmod{8}$, the corresponding equations admit no solutions for b_1, b_2 and hence no Cornacchia's computation is required.

D A concrete improvement for Kuperberg

At each level of the tree a r -list problem is solved. While the lists are in general random, this is not the case in the first level: all the initial lists contain values of the form $\sum_{i=0}^{c-1} b_i \ell_i$. That is, for the first level only, solving an r -list problem is equivalent to solve something of the form $\sum_{j=1}^r \sum_{i=0}^{c-1} b_i^j \ell_i^j = C$. This is a subset-sum instance, but in a regime where there is large number of solutions and we want (almost) all of them. While the complexity of the problem is well-studied [64, 49, 50, 42, 10, 13], literature focuses on complexity exponents in the case where 1 solution is wanted. The main exception is [64]. It reduces subset-sum to 4-list, which by definition doesn't help (even for $r = 2$, as this would only reduce the classical memory cost of the first level).

One approach to get estimates would be to adapt the optimization code that compute the exponents (for example, the one from [13]) to this regime, that is, for a fixed γ , the asymptotic exponent when the final list contains $2^{\gamma n}$ solutions. There are polynomial approximations everywhere, including in the number of solutions at the end, and the optimization constraints are non-linear, which makes it extremely hard to get general precise estimates, especially for concrete parameters. Alternatively, as finding 1 solution is easier, the costs in this case would be a lower bound.

In all cases, the known exponents are between 0.283 and 0.5. At the first level we solve classically an r -list instance at a cost $2^{\frac{r}{2}t_0}$, obtain a list of size t_1 , and add $rt_0 - t_1$ bits of constraints. Thus, if instead we apply a subset-sum algorithm, for the same classical cost we can solve a subset-sum of dimension $\frac{t_0}{2^\gamma}$, to obtain a list of size t_1 , we gain more constraints: $(\frac{r}{2^\gamma} - 1)t_0$ bits, instead of $(r - 1)t_0$ bits. If we make the extremely optimistic assumption that assuming breakthrough in subset-sum solving, $\gamma = 0.25$, then this approach can replace the equivalent of $r/(r - 1)$ levels, taking into account the slight overhead in queries. Given that the number of levels is an integer, at best, we can expect to save 1 level, that is, a factor r in queries.

E Detailed subroutines of 4D isogeny chain computation

In this section, we details the key subroutines used in Algorithm 4.

Doubling. Suppose given a theta null point $\theta^A(0)$ such that neither of $\theta^A(0)$ and $H \circ S(\theta^A(0))$ contains zero coefficients¹⁷. Defining $U = I(\theta^A(0))$ and $V = I \circ H \circ S(\theta^A(0))$, we can double points efficiently as:

$$\theta^A([2]P) := U \odot H \left(V \odot (S \circ H \circ S(\theta^A(P))) \right). \quad (8)$$

¹⁷ The isogeny chain constructed in qt-Pegasus fulfills this condition, see [31, §4.2, §4.3 and Appendix B]

Point and codomain evaluation. Evaluating a point P through a 2-isogeny $f : A \rightarrow B$ adapted with the theta model [1, Definition 4.5.8] reduces to:

$$\begin{aligned} \theta^A(P) &\xrightarrow{S} \theta^A(P)^{\odot 2} \xrightarrow{H} H(\theta^A(P)^{\odot 2}) \xrightarrow{\odot I \circ H(\theta^B(0_B))} \\ &\xrightarrow{\odot I \circ H(\theta^B(0_B))} H(\theta^B(f(P))) \xrightarrow{H} \theta^B(f(P)). \end{aligned} \quad (9)$$

However, doing so requires computing $I \circ H(\theta^B(0_B))$. This is the most complex step in the isogeny calculation, though, following the methods of [29, §3], it closely mirrors the projective inversion routine described prior.

Let $T_1^{(8)}, T_2^{(8)}, T_4^{(8)}$ and $T_8^{(8)}$ be eight-torsion points above $\ker(f)$ such that for all $X \in A$, $\theta_j^A(X + [4]T_i^{(8)}) = (-1)^{\langle j|i \rangle} \theta_j^A(X)$. We define the matrix

$$(\pi_{i,j})_{i \in (\mathbb{Z}/2)^4, j \in \{1,2,4,8\}} := \left\{ \left[H \circ S(\theta^A(T_j^{(8)})) \right]_i \mid i \in (\mathbb{Z}/2)^4, j \in \{1,2,4,8\} \right\}. \quad (10)$$

Following [29, §3.1], computing the pointwise inverse of the dual theta null point is equivalent to solving [29, Problem 3.1] over $(\pi_{i,j})$. In the generic case (detailed in [29, §3.4]), the inverse dual theta null point can be computed using Equation (11), where $\eta : [0, 2^g - 1] \xrightarrow{\sim} (\mathbb{Z}/2)^4$ is a Hamiltonian path in the *tesseract graph* with vertex set $(\mathbb{Z}/2)^4$, two vertices connected by an edge if and only if they differ in exactly one component. We can usually fix η to be the so-called Gray Hamiltonian cycle $\eta_0(i) := i \oplus (i \gg 1)$. Denoting $s_j = \eta(j+1) - \eta(j)$:

$$I \circ H(\theta^B(0))_i := \left(\prod_{j=0}^{i-1} \pi_{\eta(j), s_j} \right) \cdot \left(\prod_{j=i+1}^{2^4-1} \pi_{\eta(j), s_{j-1}} \right). \quad (11)$$

Handling unexpected zeros. An important condition in Equation (11) is that the Hamiltonian path η should be such that $P_\eta = (\pi_{\eta(j), s_j}, \pi_{\eta(j+1), s_j})_{0 \leq j \leq 2^g-2}$ contains no zero coordinates, to make inversion possible. For a handful of isogeny steps at the start of the chain, this condition does not hold for all η (see [31, Lemma 6]), so Equation (11) must be preceded by a search step for the right path. As this is limited to the beginning of the chain, its impact on performance is limited.

We look for η iterating over a set Δ consisting of (possibly all) automorphisms of the tesseract graph, as proposed in [29, §3.4]. For each $\sigma \in \Delta$, we check whether $P_{\sigma \circ \eta_0}$ contains any zero, and use $\sigma \circ \eta_0$ in place of η in Equation (11) if all coordinates are nonzero.¹⁸

In view of a quantum implementation suitable for inputs in superposition, one needs to implement the choice of η without data-dependent branching nor data-dependent memory accesses. We adapted in Qarton the implementation of [29],

¹⁸ For the choice of Δ , see [29, §3.4 and Appendix D]. Even though $\sigma \circ \eta_0$ does not range over all possible Hamiltonian paths, the heuristic choices of Δ proposed in the paper have been experimentally verified to ensure we always find a valid path, up to negligible probability.

which is not memory-constant, by essentially implementing a quantum switch-case circuit that evaluates all $\# \Delta$ conditions $c_\sigma : \prod_{j=0}^{2^g-2} \pi_{\sigma\eta_0(j),s_j} \pi_{\sigma\eta_0(j+1),s_j} \stackrel{?}{=} 0$, and for the first true condition c_σ adds $P_{\sigma\circ\eta_0}$ to an auxiliary variable P using $\# \Delta$ conditional additions. The variable P is finally used as input to [Equation \(11\)](#).

First isogenies. The gluing isogeny and the second isogeny of the chain require separate treatment, see [\[29\]](#) for details. It is sufficient to mention that both require a constant number of field multiplications, even though higher than the generic case, and can be implemented using the techniques mentioned above.

F Statement on AI tools

We did not use generative AI tools in the writing of this paper.

During the writing of this code, we used Claude Sonnet 5 to refactor or modify existing functions and circuits (for example, starting from the norm equation circuit in the odd case and adding new code for the even case). This concerns only a few functions and the base functions and classes were always hand-written. The rest of the code, documentation and unit tests were hand-written. In particular, all classical “dummy” functions were hand-written first before creating their circuit implementations.