Ready to SQI? Safety first!

Towards a constant-time implementation of isogeny-based signature SQIsign

David Jacquemin[1], Anisha Mukherjee[1], Péter Kutas[2] and Sujoy Sinha Roy[1]

[1]IAIK, Graz University of Technology, Austria
[2]Eötvös Loránd University and University of Birmingham

Abstract.
NIST has already published the first round of submissions for additional post-quantum signature schemes and the only isogeny-based candidate is SQIsign. It boasts the most compact key and signature sizes among all post-quantum signature schemes. However, its current implementation does not address side-channel resistance. This work is the first to identify a potential side-channel vulnerability in SQIsign. At certain steps within the signing procedure, it relies on Cornacchia's algorithm to represent an integer as a sum of squares of two integers. This algorithm in turn uses a 'half-GCD' (half-greatest common divisor) sub-routine based on Euclid's division algorithm which has often been exploited for side-channel attacks. We show that if the inputs of Cornacchia's algorithm leak, then one can retrieve the signing key in polynomial time. Also, since there is no constant-time implementation for SQIsign, we propose two timing attack-resistant versions of Cornacchia's algorithm. The first version uses a constant-time 'half-GCD' algorithm that runs a fixed number of times for a given upper bound based on the bit-size of the inputs. The second version is based on the two-dimensional lattice reduction algorithm. We show that randomizing the starting basis with an unimodular matrix would make the execution time independent of the input.

Keywords: isogeny-based cryptography, SQIsign, side-channel analysis, isogeny signature, constant-time implementation

1 Introduction

The currently deployed cryptographic primitives largely rely on the mathematical hard problems of integer factorization and discrete logarithms. Peter Shor in 1994 published a breakthrough algorithm [Sho94, Sho97] to factor large integers (which was super-polynomial in time in classical computers) in polynomial time with the use of quantum computing. From then on, the looming threat of large-scale quantum computers has led to immense research in the field of post-quantum cryptography. In 2016 NIST started the procedure to standardize post-quantum Key Encapsulation Mechanisms (KEMs) and digital signature schemes [Nata]. In the initial rounds of evaluation, the main criterion was the theoretical security of the submitted schemes. But moving forward, NIST has broadened its assessment criteria to also include implementation aspects such as performance and resistance to side-channel attacks (SCA). The process is currently in Round 4 and the first winners have been announced in 2022. Three digital signature schemes were chosen for standardization: Dilithium [DKL+18], Falcon and SPHINCS+ [BHK+19]. While all these schemes have their merits, NIST deemed the portfolio of signature schemes not diverse enough and announced a new call for "additional signatures" [Natb] in 2023, exclusively devoted to post-quantum signature schemes. The first round of submissions have been released in June.

Among the diverse range of submissions (code-based, multivariate-based, MPC-based etc.), the only isogeny-based candidate is SQIsign [CSCRSDF+23].

Isogeny-based schemes date back to Couveignes's seminal paper [Cou97] which was then rediscovered by Rostovtsev and Stolbunov [RS06]. Improving on Couveignes's original approach, De Feo and Galbraith constructed SeaSign [DFG19] using the Fiat-Shamir protocol with aborts technique. Then Beullens, Kleinjung and Vercauteren proposed CSi-FiSh [BKV19] which uses Couveignes' original idea combined with a record class group computation. CSi-FiSh achieves respectable performance but it is currently infeasible to instantiate it for higher security levels and it potentially does not reach NIST level I security [Pei20],[BS20]. An alternative option came in the form of SCALLOP [FFK+23] but it is also currently too slow for applications. All these signatures are based on cryptographic group actions (of which CSIDH [CLM+18] and SCALLOP are examples [FFK+23]).

Taking inspiration from SIDH one can also build signature schemes based on the pure isogeny problem which could actually remain unaffected by recent SIDH attacks [CD22, MMP+23, Rob23], such as the work by [DFDGZ23]. This scheme however is based on an identification system with low soundness and thus, has to be repeated many number of times resulting in big signature sizes.

A completely different approach for constructing signatures from isogenies was proposed by Galbraith, Petit and Silva [GPS17](GPS). Their signature relies on the observation that one can compute isogenies between curves of known endomorphism rings. Based on this observation they designed an identification scheme reminiscent of graph isomorphism. For this they rely on a quaternion version of the isogeny path-finding problem which was efficiently solved by Kohel, Lauter, Petit and Tignol, hence dubbed the KLPT algorithm [KLPT14]. The advantage of the GPS signature scheme is that its security is based on the endomorphism ring problem which is the fundamental hard problem in isogeny-based cryptography. However, it has a constrained challenge space as it allows bit-long challenges. In 2020 De Feo, Kohel, Leroux, Petit and Wesolowski proposed SQIsign [DFKL+20] which improves upon the GPS scheme in many ways and provides a much better performance. SQIsign stands out among all its isogeny-based predecessors as well as the rest of its NIST counterparts because it is the most compact post-quantum signature to date.

Within the framework of the NIST standardization process, submissions are clearly required to provide comprehensive assessments of the state of cryptanalysis, specifically pertaining to their security assumptions and their resilience against side-channel attacks. While there is a plethora of side-channel literature in case of most lattice-based candidates ([RRCB20, REB+22, XPR+22, PPM17], to name only a few), side-channel analysis of SQIsign present a stark contrast. Indeed, there exists several isogeny-based side-channel works that target, for instance, point-doubling operations or scalar multiplications on elliptic curves and propose adequate countermeasures [RJB19, PCK+23, SST04, DFEMG+22, CMRS22, ZYD+20, GLK21, MCR19, OAYT19, BBC+21]. However, SQIsign is yet to undergo any rigorous cryptanalysis on its quaternion side and as mentioned in its 'specifications' document [CSCRSDF+23] submitted to NIST, it also lacks a side-channel resistant implementation which is important for practical applications. This implies that several subroutines within SQIsign are non-constant time such as, computing the Hermite Normal Form (HNF) of a matrix or solving diophantine equations (one instance of which we address in this paper) which are fundamental to SQIsign's quaternion arithmetic. This work is the first to undertake one such endeavour on two fronts for SQIsign: on one hand, we analyze its quaternion-based algorithms to look for potential side-channel vulnerabilities, and on the other hand, we also provide constant-time alternatives for some selected algorithms as the first step towards a full constant-time SQIsign implementation. The first part of our work is intended to serve as a reference for any cryptanalyst who wants to target SQIsign but is intimidated by its sheer complexity. Also, the conventional countermeasures in elliptic-curve cryptography often revolve around protecting the scalar multiplication

using several techniques such as masking, blinding or algorithmic modifications. However, conventional countermeasures that are effective in the context of pure elliptic curve (or quaternion) arithmetic would not be directly applicable here. Hence, in the second part of our work we put significant efforts to devise countermeasures that fit the requirements of the scheme without greatly increasing its conceptual or computational complexity.

1.1 Our contributions

SQIsign's success of achieving the most compact key and signature sizes resides in its mathematical foundations. The complex signing procedure makes an immediate in-depth cryptanalysis seem quite daunting. This might urge an adversary to identify points of attack within the 'simpler' building blocks. We discuss one such potential cryptanalytic starting point. In particular, we highlight an instance of Cornacchia's algorithm within SQIsign's signing routine that could make it susceptible to side-channel attacks because it involves Euclid's division algorithm which has been shown to be vulnerable to side-channel attacks [AGS07, AT07, ASS17, AB20]. Moreover, the current implementation of Cornacchia's algorithm in SQIsign is indeed not constant-time.

- For our first contribution, we start by considering a general attack model of a powerful adversary capable of using one or more forms of side-channel attack techniques to exploit the possible side-channel vulnerabilities of the 'leaky' Euclid's algorithm within SQIsign. We then present a detailed polynomial-time key recovery method that reveals the secret signing key to the adversary under the aforementioned hypothesis. We show that the knowledge of Cornacchia's inputs allows the attacker to obtain a feasibly 'small' set of possible intermediate values, which can ultimately reveal the entire signing key (Theorems listed in fig. 1).

  In our next contribution, we only consider a specific timing-attack model and provide constant-time/time-independent solutions for Euclid's and Cornacchia's algorithms in order to realize a constant-time implementation of SQIsign. These alternative algorithms are also of independent interest as they are integral subroutines of many important quaternion functions in SQIsign.

- We propose a constant-time half-GCD algorithm that can replace the current non-constant time version. We give shape to our proposed algorithm by introducing a parameter to control the execution steps and making considerable tweaks to the underlying idea of Euclid's algorithm. We verify our constant-time claim experimentally and report comparison results with its non-constant time counterpart.

- Since solving an equation of the form '$x^2 + y^2 = m$' using Cornacchia's algorithm is equivalent to finding the shortest vector in a suitable two-dimensional lattice, we propose lattice reduction as a replacement for Cornacchia's algorithm. We add an additional basis-randomization step to ensure that the computations within the algorithm no longer correlate with the initial secret-dependent inputs of the algorithm. Using leakage-detection tests, we compare our randomized lattice reduction algorithm with its non-constant time counterparts.

1.2 Organization of the paper

In Sec. 2, we give all the important background information necessary for our paper. The design rationale of SQIsign spans over a vast range of topics related to elliptic curves and quaternion algebra. Many minute details of its construction are not intuitive and can only be understood after an in-depth study of the scheme. We provide explanations only for a handful of carefully selected topics to make our results comprehensible to the reader.

Sec. 3 presents an observation which could lead to a polynomial-time key recovery. We show that if an attacker (using simple timing or power or more advanced side-channel techniques) is able to guess the input variables to Cornacchia, then it can successfully recover the signing key in polynomial time. While we do not present any experimental attack, we propose a theoretical key-recovery procedure which is applicable in any attack scenario that can retrieve the inputs of Cornacchia. Sec. 4 describes constant-time implementations of Cornacchia, as a starting point towards making SQIsign's signature generation resistant to side-channel attacks. We provide performance results in Sec. 5. In Sec. 6 we provide insights into future directions that would motivate greater research towards exploring the potentials of SQIsign and realizing a more practical, side-channel resistant implementation.

2 Preliminaries

In this section we cover certain mathematical and algorithmic topics necessary to present our results. For more details on elliptic curves we urge the reader to refer to [Sil09] and for quaternion algebras we refer to [Voi21].

2.1 Notation

We use the notations $\mathbb{F}_q$, $\mathbb{Z}$ and $\mathbb{Q}$ to denote a finite field of order q, the set of integers and rational numbers respectively. H represents a quaternion algebra over $\mathbb{Q}$. Multiplication between integers, reals or quaternion elements is shown using '$\cdot$', whereas '$*$' represents a matrix multiplication. The dot product between two vectors is given by $(\cdot|\cdot)$ and their norm is denoted by $\|\cdot\|$. The floor function is represented by the symbol $\lfloor\cdot\rfloor$. The p-adic valuation of an integer is denoted by ν_p. Bitwise AND, right and left shift operations are represented by the symbols '&', '$\ll$' and '$\gg$' respectively.

2.2 Supersingular elliptic curves

Let E_1 and E_2 be elliptic curves defined over some finite field $\mathbb{F}_q$. An isogeny is a non-zero rational map that sends the point of infinity of E_1 to the point of infinity of E_2. Alternatively, one can also define isogenies as rational maps which are simultaneously group homomorphisms. An isogeny is a finite map of curves, thus, induces a finite field extension on the function fields of the elliptic curves. An isogeny is called separable if the said field extension is separable. The degree of the isogeny is defined as the degree of the field extension. If an isogeny is separable, then its degree is equal to the size of its kernel. For every isogeny $\phi : E_1 \to E_2$ there exists a dual isogeny $\hat{\phi} : E_2 \to E_1$ which has the same degree as ϕ and $\phi \circ \hat{\phi}$ is the multiplication-by-deg(ϕ) map (this is also true if they are composed in the other order). Two elliptic curves E_1 and E_2 are isomorphic if and only if there exists a degree 1 isogeny between them. To every elliptic curve E defined over $\mathbb{F}_q$ one can associate an element of $\mathbb{F}_q$ called the j-invariant, denoted by $j(E)$. Two elliptic curves defined over $\mathbb{F}_q$ are isomorphic if and only if their j-invariants coincide.

An endomorphism of E is an isogeny from E to itself. In order to obtain a ring structure, the zero map is also considered to be an endomorphism. Endomorphisms thus form a ring under addition and composition. If E is defined over $\mathbb{F}_q$, then its endomorphism ring is never equal to $\mathbb{Z}$ (i.e., only contains scalar multiplications) but in most cases it is still commutative (an order in a quadratic field). If the endomorphism ring of E is commutative, then we call E *ordinary*, otherwise E is *supersingular*. In this paper we will only be looking at supersingular elliptic curves. Supersingular elliptic curves can all be defined over $\mathbb{F}_{p^2}$. One can detect supersingularity efficiently by computing its number of points over $\mathbb{F}_{p^2}$ (as an equivalent definition of supersingularity is that the trace of Frobenius is divisible by p).

2.3 Quaternion algebras and the Deuring correspondence

As mentioned before, the endomorphism ring of a supersingular elliptic curve is a non-commutative ring. In this subsection, we give more details about the endomorphism ring and introduce an important categorical equivalence, called the Deuring correspondence. A rational quaternion algebra H is a central simple algebra of dimension four over $\mathbb{Q}$. It always has a presentation $i^2 = a, j^2 = b, ij = -ij$ where $1, i, j, ij$ constitute a $\mathbb{Q}$-basis and $a, b \in \mathbb{Q}^*$: $H(a, b) = \mathbb{Q} + i\mathbb{Q} + j\mathbb{Q} + k\mathbb{Q}$. An order in a quaternion algebra is a subring that contains 1 and also contains a $\mathbb{Q}$-basis of the algebra. An order is called maximal if it is maximal with respect to inclusion.

Let $B_{p,\infty}$ be the quaternion algebra ramified at p and infinity. In the special case where $p \equiv 3 \pmod 4$ this just amounts to $i^2 = -1$ and $j^2 = -p$. The endomorphism ring of a supersingular curve over $\mathbb{F}_{p^2}$ is a maximal order in $B_{p,\infty}$. Furthermore, every maximal order in $B_{p,\infty}$ is the endomorphism ring of a supersingular elliptic curve. It is also well-understood when non-isomorphic curves have isomorphic endomorphism rings, namely if and only if they are Frobenius conjugates. The Deuring correspondence is a correspondence between isomorphism classes of supersingular elliptic curves over $\mathbb{F}_{p^2}$ and isomorphism classes of maximal orders in $B_{p,\infty}$. The Deuring correspondence is two-to-one in most cases and one-to-one if and only if the curve can be defined over $\mathbb{F}_p$. This relation has considerably more structure than just some map between sets. One can consider supersingular elliptic curves together with isogenies, so we need to define the quaternion analogue of an elliptic curve isogeny.

The analogue of an isogeny with domain E is a left ideal of $\mathrm{End}(E)$. The way to associate a left ideal to an isogeny is as follows. Let $\phi : E \to E_A$ be an isogeny whose kernel is a subgroup G. Then one can take the set of all endomorphisms of E which vanish on G. It is easy to see that this set is a left ideal. In the reverse direction we can take a left ideal I of $\mathrm{End}(E)$ and intersect all the kernels of the endomorphisms in I. This provides us a subgroup which we denote by $E[I]$. Then there is a unique isogeny with kernel $E[I]$ which will be the isogeny associated to I. The norm of the ideal is defined as the greatest common divisors of all the norms in I and is equal to the degree of the associated isogeny.

A natural question is how do we get the "codomain" for a given left ideal I. This is provided by the right order $\mathcal{O}_r(I) = \{\beta \in B_{p,\infty} | I\beta \subseteq I\}$. A connecting ideal between $\mathcal{O}_1$ and $\mathcal{O}_2$ is a left ideal of $\mathcal{O}_1$ whose right order is isomorphic to $\mathcal{O}_2$. An Eichler order is the intersection of two maximal orders. It encodes a particular connecting ideal as $\mathcal{O}_1 \cap \mathcal{O}_2 = \mathbb{Z} + I$ where I is a left ideal of $\mathcal{O}_1$ and a right ideal of $\mathcal{O}_2$. $\mathcal{O}_1 \cap \mathcal{O}_2$ is a $\mathbb{Z}$ sub-lattice of $\mathcal{O}_1$ and $\mathcal{O}_2$ of the same index (sometimes called the level) which is equal to the norm of the ideal I.

2.4 KLPT and SQIsign

In isogeny-based cryptography the natural algorithmic problem is finding isogenies between supersingular elliptic curves. A more specific problem is when the degree, say, d of the isogeny is also specified. When d is small this problem is quite easy, hence, the interesting case is when d is large. In that case however, just writing down the output (essentially polynomials of degree d) might be problematic for a generic d. Therefore, it is rather natural to restrict to degrees which are smooth and quite naturally to the case where $d = l^k$ and l is a small prime (e.g., 2 or 3).

These problems are all thought be hard even for a quantum computer. Due to Deuring correspondence, there is a natural quaternion analogue of this problem. Namely given two maximal orders, find a connecting ideal of norm l^e. Such a connecting ideal should exist for a large enough e. Even though it is the exact analogue of a hard problem, it admits a polynomial time algorithm, discovered by Kohel, Lauter, Petit and Tignol [KLPT14]. We briefly sketch the idea of the KLPT algorithm here.

For simplicity we will assume that $p \equiv 3 \pmod 4$ (the general case is not much harder). The first observation is that if one wants to find a path between two maximal orders, then it is actually enough to connect both maximal orders to one specific order and concatenate the paths. This special order, denoted by $\mathcal{O}_0$ is the endomorphism ring of the elliptic curve $y^2 = x^3 + x$ and contains $1, i, j$ (it is actually generated as a $\mathbb{Z}$-module by $1, i, (1+j)/2$ and $(i+ij)/2$). From now on we assume that we are looking for a connecting ideal between $\mathcal{O}_0$ and some other maximal order $\mathcal{O}$. One finds a connecting ideal I_0 using an algorithm of Kirschmer and Voight [KV10]. Naturally, the norm of this ideal is not likely to be of the form l^e, hence our goal is to find an equivalent left ideal of $\mathcal{O}_0$ such that its norm is l^e. This is basically equivalent to finding some element in I_0 whose norm is $n(I_0) = l^e$. The next important intermediate step is that one finds an equivalent ideal of prime norm N. This is easy as one just scans through elements of I_0 and since primes are relatively dense, one will find a suitable element very efficiently. So our goal is to find $\beta \in I$ such that $n(\beta) = Nl^e$. The next observation is that $\mathcal{O}_0/N\mathcal{O}_0$ is isomorphic to $M_2(\mathbb{Z}/N\mathbb{Z})$ and an explicit isomorphism can be computed efficiently. Take into consideration an endomorphism $\gamma \in \mathcal{O}_0$ of norm Nl^{e_0} for some e_0 (this involves solving a very simple quadratic equation and can be done using RepresentInteger as in alg. 1 [DFKL+20] or its slightly modified version in [DFLLW23]) . Then one can view $\mathcal{O}_0\gamma$ as a left ideal in $\mathcal{O}_0/N\mathcal{O}_0$ which is going to be a proper left ideal. I can also be viewed as left ideal in $\mathcal{O}_0/N\mathcal{O}_0$ and is also a proper left ideal as $n(I) = N$. In $M_2(\mathbb{Z}/N\mathbb{Z})$ every proper left ideal is isomorphic and differs only by a right multiplication which can be computed easily. Thus one can compute $\mu \in \mathcal{O}_0$ such that $(\mathcal{O}_0\gamma)\mu \equiv I \pmod{N\mathcal{O}_0}$. Here everything is only determined modulo $N\mathcal{O}_0$, hence, there is large variety of choices for μ. If one manages to choose a μ whose norm is l^{e_1}, then $\beta = \gamma \cdot \mu$ is going to be an appropriate choice. However, in general, this lifting problem can be hard. The observation of KLPT is that the lifting problem is actually easy if μ is a $\mathbb{Z}$-linear combination of j and ij (we will call this set $j\mathbb{Z}[i]$). This algorithm is called StrongApproximation in alg. 2 [KLPT14]. Luckily μ can be chosen that way. The crucial detail of the KLPT algorithm is the lifting procedure which only works in this specific scenario, as one would need $j\mathbb{Z}[i]$ to be contained in $\mathcal{O}_0$.

One way to interpret KLPT is that if one has two supersingular elliptic curves with known endomorphism rings, then one can compute an isogeny between them (a task that is deemed hard otherwise). This motivates the following sigma protocol which is the high-level idea of SQIsign [DFKL+20]. The public key is a supersingular elliptic curve E and let E_0 be the supersingular elliptic curve, $y^2 = x^3 + x$. Let $\mathrm{End}(E_0) = \mathcal{O}_0$ and let $\mathrm{End}(E) = \mathcal{O}$. Here $\mathcal{O}_0$ and $\mathcal{O}$ are connected by a secret left ideal I_τ of large prime norm. The secret key is the endomorphism ring of E. The main steps of the sigma protocol are the following. The prover computes an isogeny, $\phi : E_0 \to E_1$ and sends E_1 to the verifier. The verifier sends a challenge isogeny, $\psi : E_1 \to E_2$ to the prover. The prover responds with an isogeny between E and E_2 of the degree l^e. The verifier accepts if the isogeny is indeed between those two curves and has the correct degree (of the form l^e).

The degree condition is necessary as otherwise a malicious prover could take an isogeny from E to E_1 as a commitment and then concatenate it with the challenge isogeny. Since one can translate endomorphism rings through isogenies [GPS17], the endomorphism ring of E_2 can be computed by the prover. The naive approach here is to use the KLPT algorithm to provide an isogeny between E and E_2. However, the KLPT isogeny has the peculiar feature that it always goes through E_0. This could reveal the secret key (i.e., the endomorphism ring of E) after one response. SQIsign uses a modified version of KLPT which connects E and E_2 in a direct fashion, not leaking (conjecturally) any information about the endomorphism ring of E. This can be done by deriving a quaternion analogue of commutative isogeny diagrams. One constructs a left $\mathcal{O}_0$-ideal of the correct norm and pushes it forward via I_τ to a left ideal of $\mathcal{O}$. Of course the difficulty here is to ensure that the right order of this left ideal is isomorphic to $\mathrm{End}(E_2)$. This is ensured by using the

Eichler order $\mathcal{O}_0 \cap \mathcal{O}$ and other techniques developed in KLPT.

In order to be able to respond to the verifier one has to translate the aforementioned ideal into an isogeny. There are two issues that arise in practice. First, the norm of the ideal obtained this way is large, hence its kernel will be defined over a large extension field. Second, it is not obvious how one can evaluate elements of $B_{p,\infty}$ as endomorphisms. The first issue is solved by cutting up the ideal into smaller chunks, such that for every chunk the kernel is defined over a small extension field.

The second issue is more complicated, furthermore, there is a difference between the original SQIsign construction [DFKL+20] and the improved version [DFLLW23]. First we recall the original construction. Let $\alpha \in \mathcal{O}$ and we would like to evaluate it on certain points. Now α corresponds to an endomorphism of E which (by abuse of notation) we will also denote by α. The order $\mathcal{O}_0$ has the special property that every element of $\mathcal{O}_0$ can be evaluated in a natural fashion (as i corresponds to the automorphism $(x, y) \mapsto (-x, iy)$ and j corresponds to the Frobenius endomorphism). If α is in the Eichler order $\mathcal{O}_0 \cap \mathcal{O}$, then we can evaluate it in $\mathcal{O}_0$. In order to evaluate it on $\mathcal{O}$ we need to use KLPT to translate it to E.

In the improved version [DFLLW23] they do not need to evaluate all endomorphisms of the Eichler order $\mathcal{O}_0 \cap \mathcal{O}$, just a well-chosen one. This is accomplished with the SpecialEichlerNorm given in alg. 3 (that uses a slightly different version of StrongApproximation called the FullStrongApproximation) which provides a $\beta \in \mathcal{O}$ with a specific norm and some extra property. Now these special elements are used throughout the ideal to isogeny translation algorithm and KLPT is no longer needed. This results in a considerable speed-up, the main reason being that that a much better prime p can be chosen this way.

We have omitted many details from [DFKL+20] and [DFLLW23] but we will recall three algorithms in its entirety here as they will be relevant to our attack: RepresentInteger, StrongApproximation, and SpecialEichlerNorm.

Algorithm 1 $\mathsf{RepresentInteger}_{\mathcal{O}_0}(M)$ [DFKL+20]

Require: $M \in \mathbb{Z}$ such that $M > p$.
Ensure: $\gamma = x + yi + zj + tij \in \mathcal{O}_0$ with $n(\gamma) = M$.
1: Set $m = \left\lfloor \sqrt{\frac{M}{p \cdot (1+q)}} \right\rfloor$ and sample random integers $z, t \in [-m, m]$.
2: Set $M' = M - p \cdot f(z, t)$. ▷ $f(z, t) = z^2 + t^2$
3: **if** Cornacchia$(M') = \perp$ **then**
4: Go back to Step 1.
5: **else**
6: $x, y \leftarrow$ Cornacchia(M').
7: **end if**
8: $\gamma = (x + iy + j(z + it))$.
9: **return** γ.

2.5 Side-channel attacks

Even though the mathematical 'hard' problem at the core of a cryptographic primitive ensures theoretical security, its practical implementation often leaks fatal intermediate information. An adversary can observe and exploit these side-channel leaks. There are a few different classes of such side-channel attacks, some target the physical characteristics of a system while others measure changes in the timing or power consumption associated with the operations being executed. These classes can again be categorized depending on whether they use measurements from just one execution of the algorithm (called

Algorithm 2 StrongApproximation [DFKL+20]

Require: Prime N, l a non quadratic residue modulo N, and $C, D \in \mathbb{Z}$.
Ensure: $\mu = \lambda \cdot \mu_0 + N \cdot \mu_1$, with $\mu_0 = j(C + iD)$, $\mu_1 \in \mathcal{O}_0$ such that $n(\mu) = l^{e_1}$ for some $e_1 \in \mathbb{N}$.
1: Select $e_1 \geq p \cdot N^4$ and adjust the parity so that $\frac{l^e}{p \cdot (C^2 + qD^2)}$ is a quadratic residue mod N. Its square root is denoted by λ.
2: Select z, t such that $l^e - p \cdot f(\lambda C + Nz, \lambda D + Nt) = 0 \mod N^2$.
3: Set $M = \frac{l^e - p \cdot f(\lambda C + Nz, \lambda D + Nt)}{N^2}$.
4: **if** Cornacchia$(M) = \perp$ **then**
5: Go back to Step 2.
6: **else**
7: $x, y \leftarrow$ Cornacchia(M).
8: **end if**
9: **return** $\mu = \lambda j(C + iD) + N(x + iy + j(z + it))$.

Algorithm 3 SpecialEichlerNorm$(\mathcal{O}, I, K)$ [DFLLW23]

Require: $\mathcal{O}$ a maximal order, I a $(\mathcal{O}_0, \mathcal{O})$-ideal and K a left $\mathcal{O}$-ideal of norm l.
Ensure: $\beta \in \mathcal{O} \backslash (\mathbb{Z} + K)$ of norm dividing T^2.
1: Set $L =$ RandomEquivalentPrimeIdeal(I), $N = n(L)$ and compute α such that $L = I\alpha$.
2: Compute $K' = \alpha^{-1} \cdot K \cdot \alpha$.
3: Compute $(C : D) =$ EichlerModConstraint$(L, 1, 1)$.
4: Enumerate all possible solutions of $\mu =$ FullStrongApproximation(N, C, D) until $\mu \notin \mathbb{Z} + K'$. If it fails, go back to Step 1.
5: **return** $\beta = \alpha \cdot \mu \cdot \alpha^{-1}$

single-trace attacks) or more (called multi-trace attacks). Some popular forms of side-channel analysis techniques include Simple or Differential Power Analysis (SPA/DPA), timing-based, memory cache-based, profiling side-channel etc.

As mentioned earlier, attacks on elliptic curve-based cryptosystems often (not always, such as in the case in SIKE [CD22, MMP+23, Rob23]) target point-doubling or scalar multiplications. While SQIsign also has elliptic curve computations and so could have potential elliptic curve-based vulnerabilities worth exploring, what makes it interesting is that there are other possibilities of attack scenarios due to additional quaternion arithmetic. We explore one such scenario: Euclid's GCD algorithm. Many of its versions have been found to be 'leaky' in side-channel literature, [AGS07, AT07, ASS17, AB20]. These works focus on mounting single-trace attacks. Since these attacks target just one execution of the algorithm, they are a more serious threat than multi-trace attacks relying on statistical analysis of several measurements to gain useful secret information. For example, [AGS07] presented a theoretical attack on this algorithm by proposing a model that relates its execution flow with the inputs. More specifically, the authors use a Simple Branch Prediction Analysis (SBPA) attack strategy to exploit timing differences in the execution flow and recover the inputs. This GCD algorithm can also be terminated 'half-way', leading to a sub-quadratic algorithm known as 'half-GCD'. SQIsign uses the half-GCD algorithm within one of its subroutines, the Cornacchia's algorithm to express an integer as a sum of two squares. In this paper, we show how an adversary can guess the signing key of SQIsign if they are able to exploit the side-channel vulnerabilities of Euclid's algorithm. To mitigate timing vulnerabilities of their current non-constant time implementations, we provide constant-time algorithms (the timing complexity is independent of the input).

2.6 Algorithms: Cornacchia and Euclid

Cornacchia's algorithm [Cor08] states that, for two co-prime integers m and d, the solution of the diophantine equation, $x^2 + d \cdot y^2 = m$, in coprime integers x and y (if any) is given by the Euclid's algorithm applied to the pair (x_0, m) where x_0 is any root of $x^2 \equiv -d \pmod{m}$. The algorithm stops when in the successive sequence (r_n) of remainders, it finds an r_k satisfying the relation, $r_k^2 < m \leq r_{k-1}^2$. In practical implementations, this translates to employing the 'half-GCD' algorithm. The solution is then, $\left(x = r_k, y = \sqrt{\frac{m - r_k^2}{d}}\right)$ if y is an integer, otherwise the process is repeated with another modular square root x_0' until all such roots get exhausted. SQIsign implements a version of the Cornacchia's algorithm for $d = 1$. We mention this algorithm in alg. 4 from [CSCRSDF+23].

Algorithm 4 Cornacchia(m) [Cor08]

Require: $m \in \mathbb{Z}$
Ensure: $x, y \in \mathbb{Z}$ s.t. $x^2 + d \cdot y^2 = m$.
1: Compute $u = \sqrt{-d} \pmod{m}$.
2: Compute $r_k = \mathsf{halfgcd}(u, m)$.
3: Check, $\sqrt{(m - r_k^2)/d} \in \mathbb{Z}$.
4: **if** False **then**
5: go to step 1 for a new u.
6: **else**
7: $x = r_k, y = \sqrt{(m - r_k^2)/d}$.
8: **end if**
9: **return** x, y

Algorithm 5 halfgcd(m, u) [Lon23]

Require: $m, u \in \mathbb{Z}$.
Ensure: $a \in \mathbb{Z}$.
1: Set $l = \lfloor \sqrt{m} \rfloor$
2: Set $a, b = m, u$
3: **while** $a > l$ **do**
4: Compute $r = a \pmod{b}$.
5: Set $a, b = b, r$.
6: **end while**
7: **return** a

The sub-routine halfgcd in alg. 4 is just the Euclid's algorithm terminated 'half-way', that is, when the condition of $r_k < \sqrt{m}$ is met, as given in alg. 5. Euclid's algorithm is a sequence of recursive steps used to compute the greatest common divisor (GCD) of two integers. For two integers $a, b \in \mathbb{Z}$ and assuming that $a > b$, it can be visualized as a sequence of linear combinations of successive quotients q_i and remainders r_i such that, $r_{i+1} = r_{i-1} - q_{i+1} \cdot r_i$. If after certain N steps of the algorithm, the remainder $r_N = 0$ then the last non-zero remainder r_{N-1} is the GCD of a and b, that is, $r_{N-1} = gcd(a, b)$.

2.7 Lattices and Lagrange reduction

A lattice L of dimension r is a discrete subgroup of $\mathbb{R}^n$. In other words, $L = \{\sum_{i=1}^{r} c_i \mathbf{v_i} : c_i \in \mathbb{Z},\ i \in \{1, 2, \cdots, r\}\}$. Thus a lattice consists of all $\mathbb{Z}$-linear combinations of linearly independent vectors $\mathbf{v_i} \in \mathbb{R}^n$. The vectors $\mathbf{v_i}$ comprise a 'basis' β of the lattice. A basis β can also be represented by a matrix B of row vectors such as,

$$B = \begin{bmatrix} \mathbf{v_1} \\ \vdots \\ \mathbf{v_r} \end{bmatrix} = \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{r1} & \cdots & v_{rn} \end{bmatrix}.$$

A lattice L usually has an infinite number of bases. If $\{\mathbf{v_1}, \cdots, \mathbf{v_r}\}$ is a basis then another basis would be $\{\mathbf{v_1}, \cdots, \mathbf{v_{i-1}}, \mathbf{v_i} + k \cdot \mathbf{v_j}, \mathbf{v_{i+1}}, \cdots, \mathbf{v_r}\}$, where $i \neq j$ and $k \in \mathbb{Z}$. The volume of the lattice L is given by $\mathrm{V}(L) = \sqrt{\det(G)}$, where $G = B * B^T \in \mathbb{R}^{r \times r}$ is the associated 'Gram matrix'. Transitioning between basis matrices, say B and B' is usually equivalent to multiplying one of the basis matrices with an unimodular matrix M over $\mathbb{Z}$, i.e., $B' = M * B$. The volume of the lattice however, is invariant, $\sqrt{\det(G)} = \sqrt{\det(G')}$. A lattice basis transition is necessary, for example, when one wants to find a "nice" basis for the lattice. A measure for how "nice" the basis is could be found in the 'orthogonality

defect' of a basis. The more orthogonal (lesser defect) the vectors are, the "nicer" is the basis. The orthogonality defect is defined as, $\text{def}(\mathbf{v_1}, \cdots, \mathbf{v_r}) = \frac{\|\mathbf{v_1}\| \cdots \|\mathbf{v_r}\|}{\text{Vol}(L)} \geq 1$.

Lattice basis reduction refers to techniques that are used to transform a given lattice basis into a "nice" lattice basis consisting of vectors that are short and close to orthogonal. This means that algorithms for lattice reduction aim to reduce the orthogonality defect of the starting basis as much as possible. Reduction of two dimensional lattice bases in $\mathbb{R}^2$ was given by Lagrange and Gauss. The LLL algorithm [LLL82] is used for higher dimension lattices. It generalizes the Lagrange-Gauss algorithm and uses the Gram-Schmidt orthogonalization process. Here, we make use of the algorithm by Lagrange-Gauss given in alg. 6 which we will call 'Lagrange Reduction' from hereafter. An ordered basis with $\mathbf{v_1}, \mathbf{v_2} \in \mathbb{R}^2$ is called Lagrange-reduced if $\|\mathbf{v_1}\| \leq \|\mathbf{v_2}\| \leq \|\mathbf{v_2} + k \cdot \mathbf{v_1}\|$, $\forall k \in \mathbb{Z}$. This algorithm is closely related to Euclid's algorithm and we give a brief of the discussion by [Gal12]. For $x, y \in \mathbb{Z}$ the Euclid's algorithm produces a sequence of integers r_i, s_i, t_i satisfying $x \cdot s_i + y \cdot t_i = r_i$ such that $|r_i \cdot t_i| < |x|$ and $|r_i \cdot s_i| < |y|$. The initial values are: $r_{-1} = x, r_0 = y, s_{-1} = 1, s_0 = 0, t_{-1} = 0, t_0 = 1$. Then the lattice L generated by the basis matrix B with the following structure, $B = \begin{bmatrix} 0 & y \\ 1 & x \end{bmatrix} = \begin{bmatrix} s_0 & r_0 \\ s_{-1} & r_{-1} \end{bmatrix}$, contains vectors of the form $(s_i, r_i) = (t_i, s_i) * B$. These vectors are shorter with smaller orthogonality defect than the starting basis vectors of the lattice. The worst-case complexity of Alg. 6 has been studied extensively by [Lag80, Val91] which also discusses the worst-case input types.

Algorithm 6 LagrangeReduction(u, v)

Require: $\mathbf{u}, \mathbf{v} \in \beta(L) \subseteq \mathbb{Z}^2$.
Ensure: $\mathbf{u}, \mathbf{v}$ two minimal basis vectors.
1: $B1 = \|\mathbf{u}\|^2$.
2: Compute $\mu = \lfloor \frac{(\mathbf{u}|\mathbf{v})}{B1} \rfloor$.
3: Compute $\mathbf{v} = \mathbf{v} - \mu \cdot \mathbf{u}$.
4: $B2 = \|\mathbf{v}\|^2$.
5: **while** $B2 < B1$ **do**
6: Swap $\mathbf{u}$ and $\mathbf{v}$.
7: $B1 = B2$
8: Compute $\mu = \lfloor \frac{(\mathbf{u}|\mathbf{v})}{B1} \rfloor$.
9: Compute $\mathbf{v} = \mathbf{v} - \mu \cdot \mathbf{u}$.
10: $B2 = \|\mathbf{v}\|^2$.
11: **end while**
12: **return** $\mathbf{u}, \mathbf{v}$

3 Vulnerabilities of Cornacchia and retrieval of signing key

In order to formulate a theoretical polynomial-time key recovery procedure, the attack model we assume is that of a powerful adversary capable of employing side-channel attack techniques such as SPA, DPA, timing, profiling-based attacks, or even a combination of them. With this assumption in mind, we begin our analysis of the various quaternion-specific algorithms in SQIsign's signing routine. As mentioned previously, there are several non-constant time avenues of the quaternion-specific algorithms that could be a potential target for side-channel attacks. SQIsign represents elements of $B_{p,\infty}$ as a 5-tuple of integers in $\mathbb{Z}^5$ and hence, quaternion arithmetic boils down to linear algebra and integer arithmetic [CSCRSDF+23, Section 2.4]. In the context of linear algebra, computing the HNF of a matrix (or a lattice) is an important operation and is potentially non-constant time. However, we set our focus on integer arithmetic and especially on

Cornacchia's algorithm because of two primary reasons: first, it makes use of Euclid's algorithm which, as mentioned previously in Sec. 2.5, has been shown to be vulnerable under various side-channel attack techniques and so it is a plausible candidate for our hypothetical adversary. The second reason is that both Euclid's (central in SQIsign's integer arithmetic [CSCRSDF+23, Section 2.4.1]) and Cornacchia's algorithm (one of the quaternion building blocks for solving norm equations, [CSCRSDF+23, Section 2.5.1]) are important subroutines of SQIsign and in one instance within the signing routine, Cornacchia's algorithm directly involves the secret endomorphism ring. As a result, if an adversary can exploit side-channel leakages from Cornacchia to obtain its inputs, they can obtain SQIsign's secret signing key (this is shown by the red dotted path in fig. 1), as we demonstrate later in this section.

Remark 1. In this paper we consider both the original version [DFKL+20] and the improved version [DFLLW23] of SQIsign. The version [CSCRSDF+23] submitted to NIST is, for our purposes, essentially the same as [DFLLW23]. Nevertheless changes can be made (e.g., SQISignHD [DLRW23] is a somewhat different construction) and we would like to initiate new methods that might be interesting for future applications as well.

In the context of SQIsign the relevant Cornacchia variant is when one wants to represent an integer m as a sum of two square integers, $x^2 + y^2$. In actual implementation [Lon23], the authors use an 'extended' version of the algorithm which we mention in alg. 7 referred to as $\mathsf{ExtendedCornacchia}(M)$. It ensures that the input m to $\mathsf{Cornacchia}$ is almost always a prime and definitely odd. This primality assurance is implemented using the following two procedures: a check to ensure that M and $h_3 = \prod p_i$ (for p_i congruent to 3 (mod 4) and $3 \leq p_i \leq 83$) are co-prime, and, repeated divisions by 2 and all primes q_j congruent to 1 (mod 4) for $2 \leq q_j \leq 101$ until the highest exponents of all these primes have been removed from M (to represent this repeated division in alg. 7 we use $h_1 = 2^{\nu_2} \cdot 5^{\nu_5} \cdot 13^{\nu_{13}} \cdot \cdots \cdot 101^{\nu_{101}}$ with ν_j denoting the p-adic valuation of q_j w.r.t M). These specific numbers are chosen by the authors of the implementation [Lon23]. The resultant 'unfactored' part m is then expected to be prime.

Algorithm 7 $\mathsf{ExtendedCornacchia}(M)$ [Lon23, DFKL+20]

Require: $M \in \mathbb{Z}$.
Ensure: $x, y \in \mathbb{Z}$ such that $x^2 + y^2 = m$ where m is computed from M.
1: $h_1 = 2^{\nu_2} \cdot 5^{\nu_5} \cdot 13^{\nu_{13}} \cdot \cdots \cdot 101^{\nu_{101}}$
2: $h_3 = 3 \cdot 7 \cdot \cdots \cdot 83$ ▷ Product of all 3 (mod 4) primes until 101
3: **if** $\gcd(M, h_3) == 1$ **then**
4: Compute $m = \frac{M}{h_1}$.
5: Compute $x, y = \mathsf{Cornacchia}(m)$.
6: **return** x, y
7: **end if**

Let us assume that an attacker is able to observe and then analyse the variations in the number of times the division step runs within the $\mathsf{halfgcd}$ (step 2 of alg. 4) sub-routine in alg. 5 through side-channel analysis of Euclid's algorithm such that it reveals m. From experimental observations we found that in large number of instances (∼40%) of the signing algorithm either $M = m$ or, $M = 2 \cdot m$. We assume $m = M$ and show how one can retrieve the output of $\mathsf{StrongApproximation}$ using M in Theorem 1.

Theorem 1. *Suppose we have a way of retrieving outputs and inputs of* $\mathsf{Cornacchia}$*. Then one can obtain the output of* $\mathsf{StrongApproximation}$ *in KLPT if the output size* l^e *is known.*

Proof. Since we have access to $\mathsf{Cornacchia}$ inputs and outputs, we know M and x, y such that $M = x^2 + y^2$. Then one has the equality

$$MN^2 = l^e - p \cdot ((\lambda C + Nz)^2 + (\lambda D + Nt)^2) \tag{1}$$

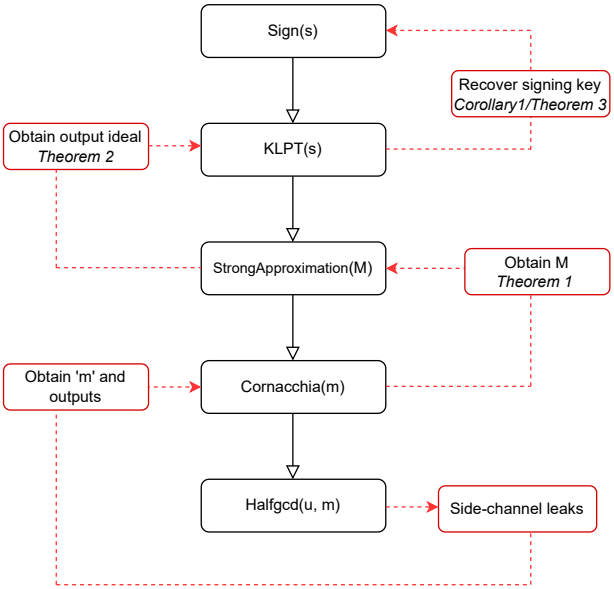

Figure 1: Steps for signing key recovery

and hence, $MN^2 \equiv l^e \pmod{p}$. Since M and l^e are known, we have two choices for N modulo p. In KLPT, $N \approx \sqrt{p}$ which gives us the exact value of N. Indeed, every residue has 2 square roots modulo p, namely some a and the other $p-a$. This implies that the other square root of N^2 will be bigger than $p/2$. Equation 1 implies that we know $(\lambda C+Nz)^2+(\lambda D+Nt)^2$. From this one can solve the equation $(\lambda C+Nz)^2+(\lambda D+Nt)^2 = x_0^2+y_0^2$ and compute all solutions x_0, y_0. Usually there should only be a few solutions here, so by testing we may assume that we have found $\lambda C + Nz$ and $\lambda D + Nt$. The output of StrongApproximation is $\mu = Nx + Nyi + (\lambda C + Nz)j - (\lambda D + Nt)ij$, and thus, we have found μ as we now know every coordinate of μ. □

Proposition 1. *Suppose we have a way of retrieving outputs and inputs of* Cornacchia*. Then we can obtain a small set of valid outputs (amongst which is the actual output) γ of* $\mathsf{RepresentInteger}_{\mathcal{O}_0}$ *in polynomial time if either of the following conditions is satisfied:*

- $n(\gamma)$ *is of the form* Nl^e *where* l, e *are known and* $N < p$.
- $n(\gamma) = M < p\log(p)^c$ *for some constant* c

Proof. First one can obtain M' as the input of Cornacchia in Step 3 of alg. 1. From this we know x, y along with M' such that $x^2 + y^2 = M'$. Now we split our algorithm into two cases depending on which condition is satisfied.

If the first condition is satisfied, then we know that $n(\gamma) = M = Nl^e$ where l^e is known and $N < p$. We also know that $M = M' - p\cdot(z^2+t^2)$ (Step 2, alg. 1) where we know p and M' but don't know z and t. This implies that we know M modulo p. Now as $M = Nl^e$ and l^e is known and coprime to p, we can obtain N modulo p by modular inversion. Since $N < p$, we have obtained N exactly which implies that we have obtained M as well. This in turn implies that $z^2 + t^2$ is known and we can compute z, t with Cornacchia. There are usually not too many solutions to choose from and all the possible solutions can be computed from factoring $z^2 + t^2$ (in practice this is a small number).

If the second condition is satisfied, then we can proceed in a similar fashion. Here we just need to guess M. The condition $M < p\log(p)^c$ implies that m is small thus one has only $O(\log(p)^c)$ choices for z, t. □

Theorem 1 and Proposition 1 are the main building blocks of our attacks. We will provide two key recovery methods, one on the original SQIsign construction [DFKL+20] and one on the newer version [DFLLW23]. First we focus on the original construction.

Theorem 2. *Suppose we have a way of retrieving outputs and inputs of* Cornacchia*. Then one can retrieve the output ideal of the KLPT algorithm.*

Proof. The KLPT algorithm, as mentioned before (and described in [KLPT14]) has three main steps. First computing γ which is the output of the $\mathsf{RepresentInteger}_{\mathcal{O}_0}$ algorithm. Then computing a μ_0 such that $(\mathcal{O}_0\gamma)\mu \equiv I \pmod{N\mathcal{O}_0}$ and then lifting μ_0 to μ whose norm is l^e. Then $\gamma \cdot \mu$ will be an element of I of norm Nl^e which provides an equivalent ideal I J of norm l^e. Theorem 1 implies that we can obtain μ and Proposition 1 implies that we can obtain γ. Putting these together we obtain $\gamma \cdot \mu$ which is exactly needed to get the output of KLPT. □

Remark 2. Proposition 1 might not be strictly necessary in certain contexts if the algorithm $\mathsf{RepresentInteger}_{\mathcal{O}_0}$ is performed deterministically as one can just recompute the algorithm offline. The reason is that 1 also provides N (not just μ), hence if l^e is chosen deterministically one does not need to use the Cornacchia oracle again. Actually, if N is a prime congruent to 1 modulo 4, often one just chooses $e_0 = 0$.

Corollary 1. *Suppose we have a way of retrieving outputs and inputs of* Cornacchia*. Then we can get the signing key in SQIsign [DFKL+20] in polynomial time.*

Proof. We focus on [DFKL+20, Algorithm 9] which provides the ideal-to-isogeny translation. In Step 2, KLPT is used to obtain an equivalent ideal J which is a connecting ideal between $\mathcal{O}_0$ and $\mathcal{O}$ where $\mathcal{O}$ is the endomorphism ring of the public curve. Theorem 2 implies that we can get J which immediately reveals $\mathcal{O}$ as the right order of J is isomorphic to $\mathcal{O}$. □

Theorem 2 is not directly applicable to the new SQIsign version [DFLLW23] as in the ideal-to-isogeny translation algorithm KLPT is no longer used. Instead of computing with entire endomorphism ring, the algorithm SpecialEichlerNorm is called which computes a well chosen endomorphism of the appropriate maximal order. In the next theorem we show how to apply 1 to obtain the signing key.

Theorem 3. *Suppose we have a way of retrieving outputs and inputs of* Cornacchia*. Then we can retrieve the signing key in the improved version of SQIsign [DFLLW23] in polynomial time.*

Proof. We target Step 5 of SpecialEichlerNorm which is a call to StrongApproximation algorithm. Theorem 1 implies that we can obtain μ using our Cornacchia approach. Furthermore, the proof of Theorem 1 implies that we also retrieve $n(L) = N$ as $N < p$. Our goal is to compute generators for the ideal L as then the right order of L will be $\mathcal{O}$ that reveals the endomorphism ring of the public curve (as the first input of SpecialEichlerNorm is the order $\mathcal{O}$). Now $\mu \in \mathbb{Z} + L$, so μ alone is not enough to retrieve L. Since $\mu \in \mathcal{O}_0$ one can write it as a 2×2 matrix with entries from $\mathbb{Z}/N\mathbb{Z}$ via the explicit isomorphism $\mathcal{O}_0/N\mathcal{O}_0 \cong M_2(\mathbb{Z}/N\mathbb{Z})$. Elements in L have the property that when they are written as 2×2 matrices, they are not invertible as their norms are divisible by N. Now $\mu = \lambda + \sigma$ where $\lambda \in \mathbb{Z}$ and $\sigma \in L$ hence one needs to figure out the value of λ. Since $N\mathcal{O}_0$ is contained in L, actually, λ is only determined modulo N. Also, since elements in L correspond to matrices that are not invertible, $\mu - \lambda$ is not an invertible matrix which is equivalent to λ being an eigenvalue of μ. Eigenvalues can be computed efficiently as N is a prime number. In general μ has two eigenvalues which provides two possible choices λ_1 and λ_2 for λ. Elements in L correspond to a set of matrices in $M_2(\mathbb{Z}/N\mathbb{Z})$ whose kernel contains a particular cyclic subgroup of $(\mathbb{Z}/N\mathbb{Z})^2$ (or equivalently a point in $\mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$). Hence for both λ_i's we compute the kernel of $\mu - \lambda_i$ and compute the

corresponding ideal L_i. By computing right orders we get two candidates $\mathcal{O}_1$ and $\mathcal{O}_2$ for $\mathcal{O}$. Then we compute the supersingular elliptic curves corresponding to these orders and choose the one whose j-invariant matches that of the public curve. The last step can be accomplished in polynomial time and is quite practical using the methods of SQIsign or that of [EPSV23]. □

Remark 3. It would be tempting to attack SigningKLPT [CSCRSDF+23, Algorithm 17] using our methods as that is common in both versions as the difference lies in the ideal to isogeny translation algorithm. However, the StrongApproximation algorithms used there will reveal the endomorphism ring corresponding to the pushforward of the secret ideal hence it is not immediate how one would get the signing key from that information. We leave this as an open problem for the future.

4 Timing attack-resistant alternatives

In this section, we only take into consideration a timing-attack based adversary who can exploit the non-constant time property of Cornacchia's algorithm and propose two different alternatives that are either constant-time or time-independent. For the first one, we present a constant-time replacement of the halfgcd algorithm (or, Euclid's algorithm, at the lowest level of the algorithm hierarchy) that we call C-halfgcd. As the second timing-independent alternative, we suggest replacing Cornacchia altogether by a two-dimensional (Randomized) LatticeReduction algorithm (at a higher level of the algorithm hierarchy). Also, proposing constant-time solutions for Euclid's algorithm has a wider and more independent scope because it is one of the fundamental functions that many high-level quaternion-based algorithms depend on, some of which we mention later in this section. We duly note that countermeasures against adversaries using attack techniques like SPA, DPA, etc., make sense only after realizing a complete constant-time implementation. Hence, in this work we focus exclusively on this starting point.

4.1 Constant-time half-GCD

One way of mitigating the attack lies in the use of a constant-time implementation of the halfgcd algorithm. A work by [BY19] proposes certain tweaks such that the iterations within the GCD algorithm no longer depend on the inputs but on auxiliary parameters. It follows an algorithmic flow similar to Stein's [Ste67] binary-GCD algorithm. [BY19] introduces a function called the 'divstep', which is the actual constant-time element in the algorithm. The function, $divstep : \mathbb{Z} \times \mathbb{Z}_2^* \times \mathbb{Z}_2 \to \mathbb{Z} \times \mathbb{Z}_2^* \times \mathbb{Z}_2$ is defined as,

$$divstep(\delta, a, b) = \begin{cases} (1+\delta, b, (b-a)/2), & \text{if } \delta > 0 \text{ and } b \text{ is odd} \\ (1-\delta, b, (b + (b \pmod 2)) \cdot a)/2), & \text{otherwise} \end{cases}$$

where $\mathbb{Z}_2$ is the ring of 2-adic integers (not the finite field) and $\mathbb{Z}_2^*$ is its group of units. In this GCD implementation, the auxiliary parameter δ depends on the maximum of the two input sizes, k, i.e., for two inputs a and b, $|a| < 2^k, |b| < 2^k$. The algorithm is constant-time if k is constant. It then calculates an integer n under a special function of k, iterations(k) and this integer n serves as an upper bound of the number of iterations [BY19, Theorem 11.2]. It ensures that whenever *divstep* is executed n times with initial inputs (a, b), the sequence of intermediate values a_i and b_i are updated in a way such that, $a_n \to \pm gcd(a, b)$ and, $b_n \to 0$.

However this algorithm cannot be trivially ported into a half-GCD algorithm as we demonstrate with the following two examples:

- Let us assume that we need to apply Cornacchia's algorithm to two integers, $M = 7349, u = 2061$. Note that $\lfloor\sqrt{M}\rfloor = 85$. Starting with $a_n = M, b_n = u$, a half-GCD algorithm would terminate as soon as it encounters an $a_i < 85$. In such a scenario, Euclid's algorithm stops at the first such a_i, namely, $a_5 = 82$ such that the square root of $M - a_i$ is also an integer. Hence, $(82, 25)$ are valid solutions of Cornacchia. However, the constant GCD algorithm oscillates between positive and negative values. Although it comes across $b_{24} = 25$, the values of the preceding steps are already $b_{25} = 50$ or $|b_{31}| = 48$. If we were to use Cornacchia's bound-check naively, this algorithm would wrongly terminate at $|b_{31}|$ or b_{25} and Cornacchia would then begin a second run with a new modular square root.

  The constant-time GCD algorithm [BY19] produces the two sequences, $a_{i+1}, b_{i+1} = \text{divstep}(a_i, b_i)$ for $i \in [0, n]$ as shown in fig 2. The steps that the usual Euclid's algorithm follows is given in fig 3.

a	7349	2061	2061	2061	−661	−661	−661	−661	−661	−661	−209	−209	−209	−209	−209	−209
b	2061	−2644	−1322	−661	−1361	−1011	−836	−418	−209	226	113	−48	−24	−12	−6	−3
n	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27

contd..

a	−3	−3	−3	−3	−3	−3	−3	1	1	1	1	1	···	1
b	103	50	25	11	4	2	1	2	1	1	1	0	···	0
n	26	25	24	23	22	21	20	19	18	17	16	15	···	1

Figure 2: Constant-time gcd(7349, 2061)

a	7349	2061	1166	895	271	82	25	7	4	3
b	2061	1166	895	271	82	25	7	4	3	1
n	10	9	8	7	6	5	4	3	2	1

Figure 3: Euclid's gcd(7349, 2061)

- Next, we apply both GCD algorithms to another pair of integers, $(61, 11)$, where 11 is a modular square root of 61. Euclid's algorithm would terminate at the very first iteration, $61 = 11 \cdot 5 + 6$, because $6^2 < 61$. It is easy to see that $(6, 5)$ is also a valid solution of the equation, $x^2 + y^2 = 61$. In contrast however, the constant-time GCD algorithm does not come across either of the values, 6 or 5, as shown in fig 4. In this case again, it would have to re-run the algorithm with another possible modular square root of 61.

a	61	11	11	−7	−7	−7	−7	−7	−1	−1	−1	···	−1
b	11	−25	−7	−9	−8	−4	−2	−1	3	1	0	···	0
n	22	21	20	19	18	17	16	15	14	13	12	···	1

Figure 4: Constant-time gcd(61, 11)

Therefore, taking inspiration from [BY19] and applying the iteration bound of n, we propose a constant-time half-GCD algorithm in 8. We take into account two sequences and make them converge to specific values. Our sequences of intermediate values are n-terms long, with n being calculated based on the size of the input as in [BY19]. We compute the same intermediate values as the usual Euclid's algorithm. In fact, we can make use of Cornacchia's bound check, l (used as an input to alg. 8 and assumed to have

been computed in constant-time) to control the first sequence $(a_i)_{i\in\{0,n\}}$ in our algorithm in such a way that it converges to the half-GCD. The value of the variable 'sign' (step 8, alg. 8) becomes 0 as soon as the algorithm encounters a value less than l. The second sequence $(b_i)_{i\in\{0,n\}}$ then converges to 0, it reaches 0 when the value of a_i at a certain step i is approximately less than half of the initial value, thus freezing the value of the sequence $(a_i)_{i\in\{0,n\}}$ at the half-GCD. It is this sequence that helps us achieve constant time, as past a certain point i in the iterations, the intermediate values no longer change till the end of n iterations.

Algorithm 8 C-halfgcd(M, u, l)

Require: M, u, d such that $u^2 \equiv -d \mod M$ ▷ $d = 1$ in SQIsign
Require: $l = \lfloor \sqrt{M} \rfloor$
Ensure: x, y such that $x^2 + d \cdot y^2 = M$
1: $k = \max(nbit(M), nbit(u))$ ▷ $nbit()$ returns the bit-size of the input
2: $n = \mathsf{iterations}(k)$
3: $a, b = M, u$
4: **while** $n > 0$ **do**
5: $c = \max(a, b)$
6: $k_c = nbit(c)$
7: sign $= \mathrm{msb}(l - c)$ ▷ Considering two's complement representation
8: $b = \min(a, b, c) \cdot \mathrm{sign}$
9: $a = \max(a, b, c) - (\min(a, b, c) \ll \alpha)$ ▷ $\alpha = 0$, or, $nbit(\max) - nbit(\min) - 1$.
10: $n = n - 1$
11: **end while**
12: $x = a,\ y = \sqrt{\frac{M - x^2}{d}}$
13: **return** x, y

Theorem 4. *The algorithm for* C-halfgcd *given in 8 is correct and terminates in constant-time for a given size of the inputs.*

Proof. First, we prove correctness. Without loss of any generality, let, $a > b$ such that, $nbit(a) \geq nbit(b)$ and hence, $k = nbit(a)$ in Step 1. Let the number of iterations of the while loop be n as per the function $\mathsf{iterations}(k)$. When $n = 1$, $c = a$ in Step 5 and $k_c = k$ in Step 6. The variable 'sign' is a boolean value that represents the following conditional statement:

$$\mathrm{sign} = \begin{cases} 0, & \text{if } c < l \\ 1, & \text{else} \end{cases}$$

Considering a two's complement representation, this means checking the most significant bit of the value of $l - c$. In constant time, 'sign' can be evaluated as, sign $= ((l - c)\&(1 \ll k_c)) \gg k_c$. Then, in step 8, if sign $= 1$, then $b = min(a, b, c) = min(a, b) = b$, otherwise $b = 0$. Let us assume that in this iteration, $c > l$ so that sign $= 1$. In Step 9 the value of a gets updated as, $a \leftarrow a - 2^\alpha \cdot b$. Indeed, Step 9 is Euclid's division where we interpret these divisions explicitly as repeated subtractions. Moreover, the rationale behind introducing the variable α in alg. 8, step 9, can be argued as follows: without α, step 9 would include many subtractions which could result in very large worse-case bounds for the algorithm. As a way of optimizing this step, we can merge multiple sequential subtractions by the value of $\min(a, b, c)$ into one subtraction by the value $2^\alpha \cdot \min(a, b, c)$. If the minimum and the maximum value have the same number of bits then we set $\alpha = 0$, else, α is set to one less than the bit difference of the maximum and the minimum values. Next, for $n = 2$, $c = \max(a - 2^\alpha \cdot b, b)$ and so on, such that we get a decreasing sequence of (a_i, b_i). As soon as $c < l$, sign becomes 0 and a_i is not updated anymore for the remaining

length of n. Thus, this becomes equivalent to the scenario where Euclid's algorithm runs until the condition in Step 1 of alg. 5 is met.

The proof that $n \geq 1$ comes directly from [BY19, Theorem 11.2]. The argument for termination is straightforward as the number of times the while loop in Step 4 will run, that is, the value of n, is calculated deterministically in Step 2 as a function of the maximum bit-size k. After every iteration, n is decremented by 1 and the loop terminates when $n = 0$. □

Remark 4. Note that the constant-time GCD algorithm in [BY19] is based on Stein's algorithm [Ste67], but alg. 8 follows the flow of Euclid's algorithm as previously discussed. However, the iteration function of [BY19] works directly for alg. 8 (the number of loops can be reduced even further) because of the following reason: in Step 9 of alg. 8, our algorithm performs the following subtraction $a = \max(a, b, c) - (\min(a, b, c) \ll \alpha)$. Here, $\min \ll \alpha$ is one bit smaller than $\max(a, b, c)$ which means that regardless of the bit-size of the inputs, it will take a maximum of two iterations of n for a to decrease by one bit. Since we want to compute the half-GCD, we can estimate that the maximum number of iterations needed would be around $2 \cdot k/2 = k$, where k is described as in Step 2 of alg. 8. Also, since $\mathsf{iterations}(k) \sim 3 \cdot k$, hence, the execution time of this algorithm can be improved further by imposing tighter bounds, as we experimentally verify in Sec. 5.3.

Applications of C-halfgcd: Recall that C-halfgcd follows the exact algorithmic flow of Euclid's algorithm and hence, simple modifications in alg. 8 can give us a constant-time Euclid's GCD algorithm as shown in alg. 9. First we set the number of iterations similar to alg. 8 (exact comparison is given in the proof of Corollary 2 since the algorithm needs to find the GCD instead of the half-GCD of the two input integers). Also, the boolean variable 'sign' introduced earlier is no longer useful. Moreover, adding additional steps to compute the Bezout's coefficients during GCD calculation would result in a constant-time Extended GCD (XGCD) algorithm. These algorithms are fundamental building blocks of important quaternion-specific algorithms such as StrongApproximation, RepresentInteger, searching for a generator of an ideal, lattice reduction in higher dimensions such as the LLL (Lenstra-Lenstra-Lovász) algorithm, etc. [CSCRSDF+23, Lon23].

Algorithm 9 C-gcd(x, y)

Require: $x \in \mathbb{Z}, y \in \mathbb{Z}$
Ensure: $a = \mathrm{GCD}(x, y)$.
1: $k = \max(nbit(x), nbit(y))$ ▷ $nbit()$ returns the bit-size of the input
2: $n = \mathsf{iterations}(k)$ ▷ $n \approx 3 \cdot k > 2 \cdot k$
3: $a, b = x, y$
4: **while** $n > 0$ **do**
5: $\quad c = \max(a, b)$
6: $\quad b = \min(a, b)$
7: $\quad a = \max(a, b, c) - (\min(a, b, c) \ll \alpha)$ ▷ $\alpha = 0$, or, $nbit(\max) - nbit(\min) - 1$.
8: $\quad n = n - 1$
9: **end while**
10: **return** a

Corollary 2. *The algorithm for* C-gcd *given in 9 is correct and terminates in constant-time for a given size of inputs.*

Proof. In order to prove correctness, observe that the algorithmic flow of C-gcd is exactly the same as that of C-halfgcd except for the fact that the variable 'sign' is no longer being used. From the proof of alg. 8 it is clear that 'sign' was used to ensure that the sequence of

a_i's converges to the half-GCD of the inputs. In the absence of 'sign', alg. 9 will complete a full run of Euclid's algorithm to compute the GCD of the inputs.

The proof of constant-time termination follows from the proof of Theorem 4. Also, since the upper bound of k is already enough for computing the half-GCD of two inputs (Remark 4), an upper bound of $2 \cdot k <$ iterations(k) will suffice for computing the GCD of given inputs.

□

4.2 Lattice reduction

In SQIsign, Cornacchia is used to write a number as a sum of two squares. In general this can be a hard algorithmic problem as it is essentially equivalent to factoring the number. In SQIsign this is circumvented by iterating until the number to be expressed as a sum of two squares is prime. Here we describe an alternative way to write a prime number M as a sum of two squares. First one computes an integer u such that $u^2 \equiv -1 \pmod{M}$. This can be done efficiently (e.g., choosing a random element and raising it to the $(M-1)/4$-th power). Then one can look at the lattice L generated by $(1, u)$ and $(0, M)$. This lattice has two properties: every vector (a, b) in this lattice has the property that $a^2 + b^2 \equiv 0 \pmod{M}$ and, the determinant of the lattice is M. Now, Minkowski's convex body theorem (published in [Min10] but the reader is referred to a more modern introduction in [Cas12]) implies that the shortest non-zero vector in this lattice is shorter than $2M$, hence its length is exactly M. Thus solving $x^2 + y^2 = M$ amounts to finding the shortest vector in this lattice which can be accomplished efficiently with Lagrange reduction (alg. 6). However, the side-channel resistance of this algorithm is not straightforward as we explain in Sec. 5.2. Thus, in order to have a timing attack-resistant algorithm for Cornacchia, we re-randomize the starting basis of L by multiplying it with a uniformly random matrix of determinant 1, with 128-bit uniformly random entries. This process of randomization works similar to masking in side-channel protection. A matrix of determinant 1 (unimodular) is chosen so that the resultant 'randomized matrix' will still represent a basis matrix as has been mentioned in 2.7. The next subsection discusses the methods of generating this matrix in more detail. Due to this randomization trick, the number of executions of the steps from 5-11 in alg. 6 no longer depend directly on the input basis elements but their random scalar combinations. We present it in alg. 10 with the modular square root u passed as as an input to the function. The computation of modular square root u of M in constant-time can be done using the Fermat theorem: $u = v^{(M-1)/2} \pmod{M}$, with $v = M - 1$. This computation is analogous to constant-time exponentiation done using Montgomery ladder [JY02] and has highly regular execution steps in the form of a multiplication always followed by a squaring irrespective of the processed bit.

Algorithm 10 (Randomized) LatticeReduction(M, d, u)

Require: M, d, u three integers with $u^2 \equiv -d \mod M$ ▷ $d = 1$ in SQIsign.
Ensure: x, y such that $x^2 + d \cdot y^2 = 0 \mod M$

1: Set the matrix $L = \begin{bmatrix} 1 & u \\ 0 & M \end{bmatrix}$
2: Generate $S = \begin{bmatrix} r_0 & r_1 \\ r_2 & r_3 \end{bmatrix}$ ▷ A random matrix of determinant 1.
3: Update $L = S * L$ such that $\mathbf{l_0} = L[0]$ and $\mathbf{l_1} = L[1]$.
4: Compute $\mathbf{l_0}, \mathbf{l_1} =$ LagrangeReduction$(\mathbf{l_0}, \mathbf{l_1})$.
5: **return** $x = \mathbf{l_0}[0]$, $y = \mathbf{l_0}[1]$

4.2.1 Generation of the random matrix S

There can be many ways of generating a random unimodular matrix. One simple way is to generate only two random integers r_0 and r_1, each of 128 bits so that $S = \begin{bmatrix} 1 & r_0 \\ r_1 & r_0 \cdot r_1 + 1 \end{bmatrix}$. However, we explain why this does not lead to a truly randomized lattice reduction algorithm. Let M and u be $\sim k$-bits long. Then the inputs of LagrangeReduction (alg. 6) are $\mathbf{l_0} = (1,\ u + r_0 \cdot M)$ and $\mathbf{l_1} = (r_1,\ r_1 \cdot u + M \cdot (r_0 \cdot r_1 + 1))$. Notice that in Step 2, the scalar μ is calculated via a rounding operation. After the first rounding operation the value of μ becomes r_1. Using this value in Step 3 we get, $\mathbf{l_0} = (1,\ u + r_0 \cdot M)$ and $\mathbf{l_1} = (0, M)$. Similarly next, after entering the 'while' loop in Step 5, the value of μ is r_0 due to which $\mathbf{l_1}$ and $\mathbf{l_0}$ get updated to the initial matrix L in alg. 10. Thus with such an S, the randomized lattice reduction *always* takes 2 iterations more than the usual (non-randomized) lattice reduction. This method of generating the random matrix therefore fails to make the execution of the algorithm independent of the inputs.

However, it is also not very feasible to generate 4 distinct 128-bit random integers while ensuring that S is unimodular at every iteration of the lattice reduction algorithm. Hence, we opt for the following method of generating the matrix S: for each call i of the algorithm, instead of 4 distinct 128-bit random integers, we sample just one, say, r_i and set $R_i = \begin{bmatrix} 1 & r_i \\ 0 & 1 \end{bmatrix}$. Additionally, we maintain a list of pre-generated random unimodular matrices (except the identity matrix) of the form, $P_j = \begin{bmatrix} p_{j_0} & p_{j_1} \\ p_{j_2} & p_{j_3} \end{bmatrix}$ with much smaller $(\log_2(p_{j_k}) << 128, k \in \{0,1,2,3\})$ random integers. Then for each i one can pick a j such that every entry of the resultant matrix, $S_{ij} = P_j * R_i * P_j^{-1}$ will have similar bit-sizes (~ 128), thereby preventing irregular-sized components in $\mathbf{l_0}$ and $\mathbf{l_1}$. If $i > j$, then new unimodular matrices can be computed via random product of P_j's, that is, for $t \in \{j+1, \cdots, i\}$, $P_t = P_j * P_{j'}$.

Let $S = \begin{bmatrix} r_0 & r_1 \\ r_2 & r_3 \end{bmatrix}$ be generated using the above-mentioned method. Then the two vectors that are to be Lagrange-reduced are given by $\mathbf{l_0} = (r_0, r_0 \cdot u + r_1 \cdot M)$ and $\mathbf{l_1} = (r_2, r_2 \cdot u + r_3 \cdot M)$, which clearly have more regular-sized entries. Notice now that $\mu = \frac{r_0 \cdot r_2 + (r_0 \cdot u + r_1 \cdot M) \cdot (r_2 \cdot u + r_3 \cdot M)}{r_0^2 + (r_0 \cdot u + r_1 \cdot M)^2}$, so that the value of the numerator is roughly of the same bit-size as that of the denominator. Thus, the rounding operation cannot remove the randomness deterministically after two iterations. The t-test scores verifying the efficacy of this method against the others are provided in Table 2.

5 Results

In this section, we compiled our implementation of all algorithms (alg. 5, 6, 8, 10) in C programming language using gcc-11.4 with optimization flags -O3, hyper-threading support disabled and measured computation time using a single core of an Intel(R) Core(TM) i7-8700 processor running at 3.20GHz on a Desktop computer with Ubuntu 22.04 operating system.

5.1 Constant-time verification

To verify that our implementation is indeed constant-time, we make use of the 'ctgrind' library and the 'Valgrind' analysis tool. 'ctgrind' is a library which when combined with the 'Valgrind' tool allows the detection of data-dependent branches in our program. In Table 1 we provide the mean and standard deviation (sd) in terms of cycle counts for C-halfgcd along with halfgcd. The standard deviation for the C-halfgcd is not null despite

the algorithm being constant time because of how an operating system (OS) behaves, that is, constant-time does not imply a constant run-time.

Additionally, in order to compare using a sample-independent parameter, we take into consideration the Coefficient of Variation corresponding to the three algorithms. The Coefficient of Variation (CV) is a measure of of the extent of variability with respect to the mean and is given by the ratio of the standard deviation to the mean. Since $CV_{C-hgcd} < CV_{hgcd}$, we can say that the execution time of halfgcd varies much more than that of C-halfgcd.

Table 1: Cycle count comparison for 10k runs

Functions	**mean (cc)**	**sd (cc)**	**CV**
C-halfgcd	140,950	8,335	0.059
halfgcd	61,676	5,871	0.095

5.2 t-test to distinguish timing dependencies

Unlike C-halfgcd, the lattice reduction algorithm is not constant-time. The process of randomizing the starting basis simply makes the execution time independent of the secret inputs. Thus, the constant-time verification rationale used in the previous subsection will not be viable here. Since randomization as a countermeasure is similar to the masking techniques used in power side channel, hence, we decided to employ the Test Vector Leakage Assessment (TVLA) method introduced in [GJJR11] to check for leakages. We take into consideration our randomized lattice reduction algorithm under two cases: one where the matrix S in alg. 10 is set to the identity matrix (hence alg. 10 reduces to the usual Lagrange reduction) whereas in the second case S consists of uniformly random integers. We run our t-test experiment for each of the aforementioned cases with 10^5 sample inputs belonging to two different classes: class 1 consisting of a fixed M and class 2 with varying values of M but of a fixed size (according to SQIsign NIST-I parameters [CSCRSDF+23]). We also perform a t-test for the non-constant time half-GCD (alg. 5) as given in Table 2. We execute these algorithms once by enabling the flag -fno-tree-vectorize to prevent the compiler from doing vectorizations and then by disabling it. Clearly, the half-GCD algorithm has a high t-value in both these scenarios. Interestingly, we observe that when the tool is allowed to vectorize, LagrangeReduction has a t-score of 4.5 which means that the null hypothesis cannot be rejected with confidence. The randomized lattice reduction algorithm, however, has a t-score below 4.5 in both cases as shown in Table 2.

Table 2: t-test scores

Mode \ Algorithm	halfgcd	LagrangeReduction	(Randomized)LatticeReduction
× -fno-tree-vectorize	16.6	4.5	2.5
✓-fno-tree-vectorize	15.2	2.5	2.1

5.3 Performance results and choice of countermeasures

Based on the targeted level of the algorithmic hierarchy as shown in fig. 1, we report two distinct comparisons. Starting with the lower level, we compare the performance of C-halfgcd with the non-constant time halfgcd as in alg. 5. The current SQIsign implementation [Lon23] relies on the GMP library for its large-integer arithmetic. Hence, for a fair comparison we also implement a GMP version of C-halfgcd and provide the performance figures in Table 3. Clearly, C-halfgcd has an overhead of roughly 250× when compared to

the usual halfgcd algorithm. As explained in Remark 4, the iteration bound of C-halfgcd can be improved further by taking $n = \mathsf{iterations}(k)/2$ (denoted as C-halfgcd(2)) or even $n \sim k$(denoted as C-halfgcd(3)). We observe that the cycle count improves proportionately with respect to the bound on n.

Then, going up the hierarchical ladder, we give a comparison (again using GMP) of Cornacchia with LagrangeReduction as well as (Randomized) LatticeReduction (assuming that the random unimodular matrices are pre-generated) in Table 4. The run-time of both Cornacchia and LagrangeReduction are similar but the latter exhibits better timing-attack resistance. The (Randomized) LatticeReduction provides the best timing-attack resistance but is 5× slower. Employing a good Random Number Generator (RNG) is crucial for the success of the randomized lattice reduction algorithm.

Table 3: Cycle count comparison of half-GCD algorithms for 10k runs

Functions	C-halfgcd	C-halfgcd(2)	C-halfgcd(3)	halfgcd
mean (cc)	$137,296$	$68,855$	$49,645$	531

Table 4: Cycle count of Cornacchia vs lattice reduction algorithms for 10k runs

Functions	Cornacchia	LagrangeReduction	(Randomized) LatticeReduction
mean (cc)	$1,109$	$1,165$	$5,969$

Therefore, if one aims for for a complete constant-time implementation of SQIsign, one may choose the C-halfgcd algorithm as one can predict and set average/worst-case iteration bounds directly based on a given size of the inputs. If one is intended on a timing-attack resistant implementation without compromising too much on performance, then the (Randomized) LatticeReduction is a viable choice. However, looking at the big picture gives us a very different perspective: when compared to the run-times of computation-heavy algorithms such as IdealToIsogeny, these constant-time alternatives will have almost no performance overhead and can be equally suitable for implementation, as demonstrated for C-halfgcd in Table 5. Also, we present a range of cycle counts for the full signing routines because, as already reiterated throughout the text, there are several other non constant-time functions in SQIsign that greatly affect the run-time.

Table 5: Cycle count of non-constant SQIsign vs with C-halfgcd

Version	Non-constant SQIsign	SQIsign with C-halfgcd
range (cc)	$(8.680644 - 15.195743) \times 10^9$	$(8.738787 - 14.586137) \times 10^9$

6 Conclusion

In this paper, we showed that if inputs and outputs of Cornacchia's algorithm leaks, then one can recover the signing key in SQIsign in polynomial time. This can be a starting point for research on future side-channel attacks on the modified versions of SQIsign [CSCRSDF+23] and to propose efficient countermeasures. We did not demonstrate the experimental method for actually retrieving Cornacchia outputs because the current implementation is not meant to be constant time or resistant to side-channel attacks. Hence, it is a compelling research direction to mount a fine-tuned single trace attack on Cornacchia's algorithm and attack the SigningKLPT algorithm as mentioned in Remark 3.

Additionally, we also proposed two timing-attack resilient alternatives that could replace the non-constant algorithms. The first one is based on lattice reduction techniques in two dimensions. We suggested randomization of the starting lattice basis to eliminate input dependency. The second one is a constant-time half-GCD algorithm. The constant-time

feature of the algorithm comes from setting an upper bound on the iterations based only on the maximum of the input sizes. We also analyzed the performance of our algorithm and verified that it is constant-time using standard tools. A future direction would be to derive tighter bounds of the constant-time half-GCD algorithm that would still ensure correct termination. In fact, coming up with a better bound could significantly improve its performance. The signing procedure of SQIsign is complex and we have not yet proposed a full constant-time implementation but we initiated the first step on the quaternion side, hoping to motivate further cryptanalysis for SQIsign.

References

[AB20] Alejandro Cabrera Aldaya and Billy Bob Brumley. When one vulnerable primitive turns viral: Novel single-trace attacks on ECDSA and RSA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):196–221, 2020.

[AGS07] Onur Aciiçmez, Shay Gueron, and Jean-Pierre Seifert. New branch prediction vulnerabilities in openssl and necessary software countermeasures. In Steven D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*, volume 4887 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2007.

[ASS17] Alejandro Cabrera Aldaya, Alejandro Cabrera Sarmiento, and Santiago Sánchez-Solano. SPA vulnerabilities of the binary extended euclidean algorithm. *J. Cryptogr. Eng.*, 7(4):273–285, 2017.

[AT07] Sarang Aravamuthan and Viswanatha Rao Thumparthy. A parallelization of ECDSA resistant to simple power analysis attacks. In Sanjoy Paul, Henning Schulzrinne, and G. Venkatesh, editors, *Proceedings of the Second International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2007), January 7-12, 2007, Bangalore, India*. IEEE, 2007.

[BBC+21] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):351–387, Aug. 2021.

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2129–2146, New York, NY, USA, 2019. Association for Computing Machinery.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: efficient isogeny based signatures through class group computations. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I*, pages 227–247. Springer, 2019.

[BS20] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of csidh. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic*

Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30, pages 493–522. Springer, 2020.

[BY19] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):340–398, 2019.

[Cas12] John William Scott Cassels. *An introduction to the geometry of numbers*. Springer Science & Business Media, 2012.

[CD22] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh (preliminary version). *Cryptology ePrint Archive*, pages Paper–2022, 2022.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 395–427. Springer, 2018.

[CMRS22] Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. Patient zero and patient six: Zero-value and correlation attacks on CSIDH and SIKE. Cryptology ePrint Archive, Paper 2022/904, 2022. https://eprint.iacr.org/2022/904.

[Cor08] 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, 49:33-90, 1908.

[Cou97] Jean-Marc Couveignes. Hard homogeneous spaces. *Preprint at https://eprint. iacr. org/2006/291*, 1997.

[CSCRSDF+23] Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQISIGN: Algorithm specifications and supporting documentation. National Institute for Standards and Technology, https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures, 2023.

[DFDGZ23] Luca De Feo, Samuel Dobson, Steven D Galbraith, and Lukas Zobernig. SIDH proof of knowledge. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*, pages 310–339. Springer, 2023.

[DFEMG+22] Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Novak Kaluđerovi´c, Natacha Linard de Guertechin, Simon Pontié, and Élise Tasso. SIKE channels: Zero-value side-channel attacks on sike. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):264–289, Jun. 2022.

[DFG19] Luca De Feo and Steven D Galbraith. Seasign: compact isogeny signatures from class group actions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 759–789. Springer, 2019.

[DFKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: compact post-quantum signatures from quaternions and isogenies. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part I 26*, pages 64–93. Springer, 2020.

[DFLLW23] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New algorithms for the deuring correspondence. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 659–690, Cham, 2023. Springer Nature Switzerland.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DLRW23] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQISignHD: New dimensions in cryptography. *Cryptology ePrint Archive*, 2023.

[EPSV23] Jonathan Komada Eriksen, Lorenz Panny, Jana Sotáková, and Mattia Veroni. Deuring for the people: Supersingular elliptic curves with prescribed endomorphism ring in general characteristic. *Cryptology ePrint Archive*, 2023.

[FFK+23] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. SCALLOP: Scaling the CSI-FiSh. In *Public-Key Cryptography–PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part I*, pages 345–375. Springer, 2023.

[Gal12] Steven D. Galbraith. *Lattice basis reduction*, page 347–365. Cambridge University Press, 2012.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance. 2011.

[GLK21] Aymeric Genêt, Natacha LinarddeGuertechin, and Novak Kaluđerović. Full key recovery side-channel attack against ephemeral sike on the cortex-m4. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 228–254, Cham, 2021. Springer International Publishing.

[GPS17] Steven D Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 3–33. Springer, 2017.

[JY02] Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International*

Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.

[KLPT14] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion-isogeny path problem. *LMS Journal of Computation and Mathematics*, 17(A):418–432, 2014.

[KV10] Markus Kirschmer and John Voight. Algorithmic enumeration of ideal classes for quaternion orders. *SIAM Journal on Computing*, 39(5):1714–1747, 2010.

[Lag80] J.C Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1(2):142–186, 1980.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Miklós Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[Lon23] Patrick Longa. sqisign-ec23 (2023). https://github.com/SQISign/sqisign-ec23, 2023.

[MCR19] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of csidh. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 307–325, Cham, 2019. Springer International Publishing.

[Min10] Hermann Minkowski. *Geometrie der zahlen*. BG Teubner, 1910.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, pages 448–471. Springer, 2023.

[Nata] National Institute for Standards and Technology (NIST). Post-quantum crypto standardization (2016), https://csrc.nist.gov/projects/post-quantum-cryptography.

[Natb] National Institute for Standards and Technology (NIST). Post-quantum crypto standardization (2016), https://csrc.nist.gov/projects/pqc-dig-sig/standardization/call-for-proposals.

[OAYT19] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (short paper) a faster constant-time algorithm of CSIDH keeping two points. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security*, pages 23–33, Cham, 2019. Springer International Publishing.

[PCK+23] Dongjun Park, Minsig Choi, Gyusang Kim, Daehyeon Bae, Heeseok Kim, and Seokhie Hong. Stealing keys from hardware wallets: A single trace side-channel attack on elliptic curve scalar multiplication without profiling. *IEEE Access*, 11:44578–44589, 2023.

[Pei20] Chris Peikert. He gives c-sieves on the csidh. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 463–492. Springer, 2020.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 513–533, Cham, 2017. Springer International Publishing.

[REB+22] Prasanna Ravi, Martianus Frederic Ezerman, Shivam Bhasin, Anupam Chattopadhyay, and Sujoy Sinha Roy. Will you cross the threshold for me? generic side-channel assisted chosen-ciphertext attacks on NTRU-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):722–761, 2022.

[RJB19] Prasanna Ravi, Bernhard Jungk, and Shivam Bhasin. Single trace electromagnetic side-channel attacks on fpga implementation of elliptic curve cryptography. In *2019 Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC)*, pages 1–4, 2019.

[Rob23] Damien Robert. Breaking sidh in polynomial time. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, pages 472–503. Springer, 2023.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based PKE and kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):307–335, 2020.

[RS06] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptology ePrint Archive*, 2006:145, 2006.

[Sho94] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

[SST04] Hisayoshi Sato, Daniel Schepers, and Tsuyoshi Takagi. Exact analysis of montgomery multiplication. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 290–304. Springer, 2004.

[Ste67] J Stein. Computational problems associated with racah algebra. *Journal of Computational Physics*, 1(3):397–405, 1967.

[Val91] Brigitte Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12(4):556–572, 1991.

[Voi21] John Voight. *Quaternion algebras*. Springer Nature, 2021.

[XPR+22] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, David Oswald, Wang Yao, and Zhiming Zheng. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. *IEEE Transactions on Computers*, 71(9):2163–2176, 2022.

[ZYD+20] F. Zhang, B. Yang, X. Dong, S. Guilley, Z. Liu, W. He, F. Zhang, and K. Ren. Side-channel analysis and countermeasure design on arm-based quantum-resistant SIKE. *IEEE Transactions on Computers*, 69(11):1681–1693, nov 2020.