Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Algorithms for Quaternion Algebras in SQIsign

Master Thesis

Sina Schaeffler

September 16, 2023

Advisors: Prof. Dr. Kenny Paterson, Dr. Luca De Feo

Applied Cryptography Group
Institute of Information Security
Department of Computer Science, ETH Zürich

Abstract

SQIsign is a post-quantum signature scheme based on isogenies and quaternion algebras. It was recently submitted to the NIST call for additional post-quantum signatures. For this, a new implementation in C was written, which only depends on the multi-precision integer library GMP. Thus, for the first time, all necessary operations on objects in the quaternion algebra were implemented in a unified way specifically for SQIsign, without relying on external computer algebra systems.

Through the work on the basic quaternion algebra functions for this implementation, a collection of algorithms for these operations has been made. This collection, improved by simpler or more efficient solutions found since the submission, is presented in this work. None of the algorithms or formulas are completely new. However, this is the first unified presentation of solutions for all basic quaternion algebra operations required by the KLPT and ideal-to-isogeny algorithms of SQIsign.

The operations which are presented include lattice equality, addition, multiplication, intersection, dual and colon lattice, and operations on ideals such as the computation of left or right orders and connecting ideals between maximal orders. Most of these have been implemented, and sometimes several variants are compared.

Contents

Chapter 1

Introduction

Even if the development of quantum computers has not yet reached a state where they break current public-key cryptography, there is research on signatures and public key encryption methods called post-quantum schemes. These are supposed to be executed on classical computers to preserve secrecy and authenticity of data even if large quantum computers become practical.

Current families of public-key schemes are based on the assumed hardness of the RSA problem or the discrete logarithm problem in the group of points of an elliptic curve, which are solved efficiently by Shor's algorithm on quantum computers. As a consequence, new hardness assumptions for different problems are used in post-quantum cryptography. Schemes are then classified into families which are based on similar assumptions. Isogeny-based cryptography is one of these families. Its most famous members include the key exchanges SIDH and CSIDH and the signature scheme SQIsign.

1.1 SQIsign

This post-quantum signature scheme was invented in 2020 by Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit and Benjamin Wesolowski in [14]. SQIsign has extremely small signature and public key sizes, which is new among post-quantum schemes.

The Deuring correspondence

In order to achieve this, it uses the Deuring correspondence, which links isogenies between supersingular elliptic curves to ideals in a specific quaternion algebra.

Quaternion algebras, which are defined in Section 2.3, are non-commutative four-dimensional $\mathbb{Q}$-algebras. In a quaternion algebra, ideals are lattices

which are closed under multiplication, and orders are lattices which are subrings. Every ideal in a quaternion algebra is an ideal in the usual sense for some orders in the algebra. This means that it is an additive subgroup of these orders and closed under multiplication by order elements at left or at right, but not necessarily both. More precisely, ideals are sait to "link" two orders of the algebra, since they are an ideal for multiplication at left for the first one and at right for the second one.

This is similar to the way in which isogenies (defined in Section 2.2), which are a special kind of application between elliptic curves, link two curves. More precisely, supersingular elliptic curves over a fixed field have endomorphism rings which are isomorphic to orders in a quaternion algebra. This algebra depends only on the field. Furthermore, isogenies between two supersingular elliptic curves correspond to ideals in the algebra which connect orders isomorphic to the endomorphism rings of the curves. These are just two examples of the surprising links between objects in this algebra and supersingular elliptic curves described by the Deuring correspondence.

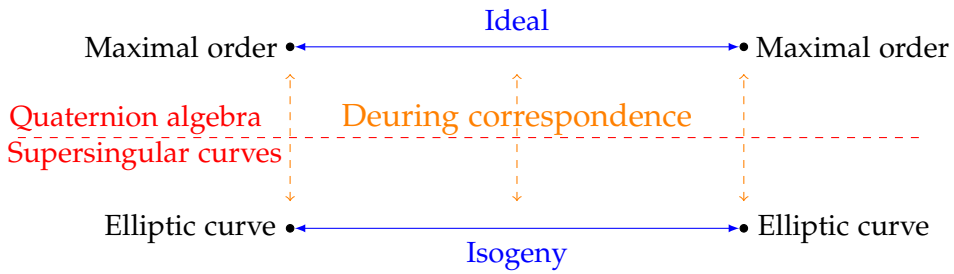

Figure 1.1: Illustration of the Deuring correspondence between isogenies of supersingular elliptic curves and ideals and orders in a quaternion algebra

Another aspect of this correspondence it that it is computable. Given an ideal, a corresponding isogeny can be computed efficiently, as described in Section 2.4 of this text, or, in more detail, in articles on this subject [13]. In the other direction computing the endomorphism ring of a supersingular curve is assumed to be hard if no additional knowledge is given. This also means that computing an ideal corresponding to a given isogeny is in general difficult. Furthermore, the fundamental assumption of isogeny-based cryptography, which is that computing an isogeny between two supersingular elliptic curves is in general hard, depends on the hardness of this computation, because an ideal connecting two maximal orders in a quaternion algebra is efficiently computable, as explained in Section 7.4. Another efficient, but more complex algorithm called KLPT (invented in [21]) even computes an ideal whose norm is power of a prime ℓ, which is the quaternion algebra analog of the ℓ-isogeny path problem, since the norm of an ideal equals the

degree of the corresponding isogenies. The isogeny path problem and the endomorphism ring problem where even shown to be equivalent in [12, 36].

Use of the Deuring correspondence in SQIsign

SQIsign exploits this correspondence in two ways: First, connection computations are done in the quaternion algebra, and the results are converted to isogenies and elliptic curves. Second, all public keys and signatures are elliptic curves and isogenies, which makes key recovery and signature forging hard, since the given information is conjectured to be insufficient for accessing the quaternion algebra world. More details on the scheme are given in Section 3.2.

Even if isogeny-based cryptography exists since more than fifteen years (even twenty-five if counting an unpublished note by Couveignes [7]), and the quaternion algebra algorithm KLPT has been used in cryptography since more than five years, constructive use of the Deuring correspondence in cryptography is rather new. Its first use in a signature scheme was the GPS signature [18] which was not implemented, so that SQIsign is the first practical scheme which needs an implementation of algorithms for the Deuring correspondence and thus of quaternion algebras.

NIST candidate and current state

Three years after the first paper on it, SQIsign was recently submitted to the NIST call for additional post-quantum signature schemes [4].

Its security relies not only on the hardness of computing a non-trivial endomorphism of a supersingular elliptic curve, but also on the assumption that a specialized variant of the KLPT algorithm used for signing does not leak information on its secret inputs. The very recent SQIsign variant SQIsignHD [8] uses very different isogeny representations and higher-dimensional isogenies, in the hope of achieving better security proofs and performance.

While SQIsign is still very slow, as signing takes about a second and verification tens of milliseconds, it has extremely short signature and public key sizes, which are at 177 and 64 bytes respectively (for NIST-1 security, equivalent to AES-128). These are smaller than modern RSA signatures and keys (which need at least 256 bytes at similar pre-quantum security), and impressively small when compared to other post-quantum schemes such as the signature Falcon [17], which uses signatures of 666 and public keys of 897 bytes for NIST-1 security. The small keys and signatures are the main advantage of SQIsign over other submitted schemes, while its slowness and complexity are its main drawback.

There are only a handful of implementations of SQIsign. Some of them are improvements of the original C implementation [14, 9, 23]. Magma [14] and Sagemath [6] implementations also exist. As far as I know, all implementations except the one for the NIST submission [4] rely on external algebra systems.

The NIST submission, to which I contributed, provides an implementation in C which only relies on the multi-precision integer library GMP, and a specification which fixes parameter sets. Both are expected to help future research, the specification by a more precise description of the scheme, and the implementation by a better understanding of the operations it requires.

1.2 Implementing quaternion algebras for SQIsign

As described more precisely in Chapter 3, at the heart of SQIsign are two families of algorithms on quaternion algebras: The translations to and from the isogeny world, and several variants of the KLPT algorithm for finding equivalent ideals of a certain norm, which are needed to facilitate the translation. Both of these require many basic operations on quaternion algebra objects.

While the translation and KLPT algorithms are rather new and studied in several papers related to the scheme [14, 9, 13], the underlying basic operations on quaternion algebra elements, lattices, orders and ideals (defined in 2.3) are mostly classical, well-known formulas which are for a large part implemented in general algebra libraries such as PARI/GP. So far, SQIsign has not attracted any particular attention to most of them.

These basic operations include addition, multiplication, norm, trace and conjugate computations on algebra elements, lattice addition, multiplication, intersection, equality, membership and colon lattice computations, as well as the addition, multiplication, creation, norm and generator computations for ideals of maximal orders in a specific quaternion algebra. Furthermore, right orders of ideals as well as isomorphisms between ideals and the computation of a connecting ideal between two maximal orders are required.

NIST submission reference implementation

Within the implementation submitted to NIST, we made the choice to create a dedicated quaternion algebra module which provides all these basic operations on quaternion algebras to other modules which implement KLPT and the translation algorithms. The quaternion algebra module, whose implementation was mostly part of this work (for more details see Chapter 9),

only relies on a module for integers of arbitrary size, which itself uses the multi-precion integer library GMP.

On the isogeny side, we have two modules, of which one provides $\mathbb{F}_{p^2}$ arithmetic, and the other one elliptic curves and isogenies. They are used by the translation algorithms, but also for the verification, which does not require quaternion algebra operations, as explained more precisely in Chapter 3. The dependencies between the modules in this implementation are shown in Figure 1.2.

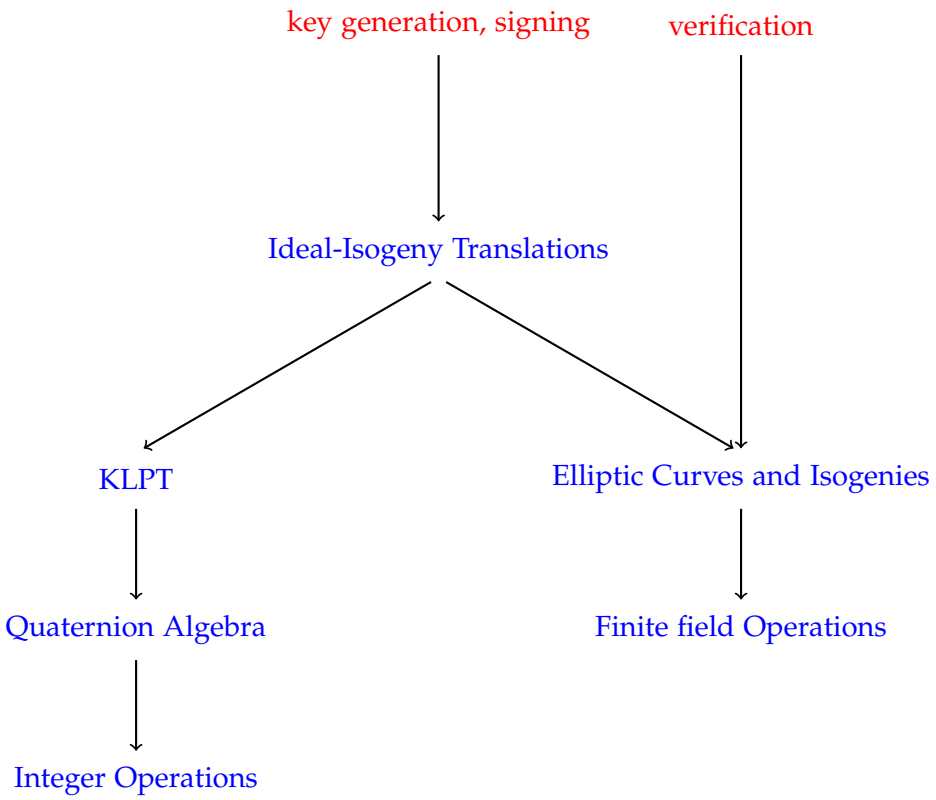

Figure 1.2: Modules and dependencies for the NIST implementation of SQIsign

The progess this implementation provides compared to its predecessors is not only in the choice of parameters, which were determined for this submission, but also that it only depends on a multi-precision integer library. It is therefore the first SQIsign implementation which does not rely on an external library for quaternion algebra operations.

Comparison to prior implementations

More precisely, all prior C implementations I know of used the algebra library PARI/GP. While facilitating implementations, its use has several disadvantages.

First, PARI is not intended for cryptographic use. Quaternion algebra computations in SQIsign are mostly done on secret inputs, which means that particularly high standards of security are needed. These cannot be met as long as this external library is used, and gaining control and knowledge on the algorithms which were so far borrowed from PARI was one of the major reasons for developing an independent quaternion algebra module.

Furthermore, PARI is a large and very general library. Its size means that it is likely to contain bugs, which, if they impact us, are hard to fix. Its generality (PARI has no quaternion-algebra specific code, but supports central simple $\mathbb{Q}$-algebras in general) means that the algorithms were chosen for their generality, and not for their speed or simplicity in the particular case of the quaternion algebra used in SQIsign. As a consequence, reimplementing PARI's functions in our special case allows for quaternion algebra-specific optimizations. A few of these are given in this text, for example in the colon lattice computation in Section 7.4 which replaces PARI's more general and slower approach. Since quaternion algebra operations are not the slowest operations in SQIsign, these improvements might not give impressive speedups of the entire protocol, but this is obviously not a reason to use excessively general and slower methods, especially if they are also less readable.

In addition, PARI is highly optimized, but not always according to our needs. These optimizations sometimes make its code rather unreadable and hard to understand, while not necessarily improving performance in our case (see Section 5.1 for an example). Furthermore, even without these optimizations, moving from PARI to a custom, C- and GMP-based implementations improves readability, since PARI uses a very complex internal memory management system and dynamic typing.

In conclusion, reimplementing the functions which prior implementations borrowed from PARI allows us to improve readability and performance, and gives a starting point for future work towards a secure implementation.

1.3 Algorithms for quaternion algebras

This text focuses on the basic quaternion algebra operations as listed in 1.2. To provide context, their description is preceded by a high-level description of SQIsign and closed by an outlook on KLPT and a comparison to the reference implementation.

Motivation, goals and content

During the implementation of the quaternion algebra module, a first exhaustive list of all basic quaternion algebra operations required by SQIsign was made. In the corresponding specification [4], the section on basic operations briefly lists them. However, that text was not supposed to get into details on this aspect. Therefore, a more precise description of the required basic quaternion algebra operations still needed to be done, in order to gain a better understanding of the problems to solve, the chosen solutions and possible variants. This text attempts to address this.

In order to do so, we give an introduction to quaternion algebras, which mostly consists in the definition of the objects and operations we need to compute. Proofs are mostly omitted, since they can be found in textbooks on quaternion algebras such as Voight's book [35]. SQIsign and, in more detail, the KLPT algorithm, are briefly described, to give an idea of the requirements. A list of quaternion algebra operations, ordered by the objects they operate on, is provided. For each operation, at least one, sometimes several solution methods, which are either formulas or simple algorithms in pseudocode, are explained. None of them are new, but the lists provided in Chapters 6 and 7 are, to my knowledge, the first systematic overview of basic quaternion algebra functions specifically for SQIsign which is more detailed than the specification [4]. Here again, proofs are generally left to Voight's book. All of the given methods only use previously explained operations, in order to facilitate implementation.

The described solution methods are selected following two criteria: simplicity and efficiency. The most important of them is simplicity. The described solutions should be simple enough to easily understand, reimplement, verify or adapt them. This is important, since understanding solutions for implementing these operations is necessary for future work on secure SQIsign implementations. Efficiency is a secondary goal, but can still be used to differentiate if several simple solutions exist. Most often in these cases, all methods we tried out, as well as, in rare cases, methods we did not try, are described and compared with respect to execution speed.

The representations, algorithms and formulas used are often, but not always, similar to the ones used in the reference implementation. Sometimes we found simpler or more efficient expressions after NIST's June 2023 deadline, in which case these are explained instead of or in addition to the originally implemented methods. For a detailed description of the differences, see Section 9.1.

As a summary, the goal of this text is not to introduce new, particularly fast or even safe algorithms for quaternion algebra operations. It is rather supposed to gather the simplest existing solutions which are still reasonably

efficient, to present them in a coherent, unified way suitable for implementation, and to combine this presentation with a few insights gained from our implementation submitted to NIST and its improvements since. Stronger optimizations (which might reduce readability) or security considerations could be built upon this in future, but are not the topic of this text.

Outline

In Chapter 2, we first fix a few notations and give basic notions on isogenies and quaternion algebras. The isogeny part just covers the necessary for the high-level description of SQIsign in Chapter 3, while the definitions on quaternions are more precise and used throughout the rest of the document. Chapter 4 fixes the representation of quaternion ideals and lattices used in the following sections. Chapter 5 gives the definition of the Hermite Normal Form, LLL-reduction and a few considerations on matrix computations for our representation of objects in our quaternion algebra. Chapters 6 and 7 are the heart of this work and give an exhaustive list of algorithms and formulas on lattices (Chapter 6) and ideals (Chapter 7) required by the KLPT and translation algorithms in SQIsign, as described above. Chapter 8 gives an outlook on a basic version of the KLPT algorithm, and details a few of its subfunctions which, within the reference implementation for NIST, were also situated in the quaternion algebra module. Chapter 9 focuses on this implementation and its differences compared to the description in this text. It also provides an analysis of benchmarks for a few algorithms where several variants were implemented. In Appendix A.1, a brief list states which algorithms for ideals also work when the ideal's left order is not maximal, while Appendix A.2 contains the benchmarking tables explained in Section 9.2.

Chapter 2

Preliminaries

This chapter introduces the notations and notions needed to understand the remainder of the text. After a few words on notations, we list some concepts on elliptic curves and isogenies needed for the introduction to SQIsign in Chapter 3. The following part contains all necessary definitions on quaternion algebras and objects within these. Those are also required by the chapters on computations in a quaternion algebra. The last section presents the Deuring correspondence which links both of the previously introduced worlds, including some of its computational aspects. This again is necessary for understanding SQIsign, while most other chapters can be read without it. However, the parts on isogenies and the Deuring correspondence explain the choice of the quaternion algebra used in all later chapters.

2.1 Notations

The algebraic closure of a field $\mathbb{K}$ is denoted by $\overline{\mathbb{K}}$. $\mathbb{F}_q$ denotes the finite field with q elements (where q is a power of a prime number).

In the following, $\gcd(\cdot)$ applied to lists of integers, matrices of integers, vectors of integers or a mixture of any of these elements means the greatest common divisors (gcd) of all integers in these objects. A gcd is always a positive integer.

For $x \in \mathbb{R}$, the absolute value is denoted by $|x|$, and $\lfloor x \rfloor$ denotes the unique integer such that $x - 1 < \lfloor x \rfloor \leq x$.

For a matrix M, M^T denotes M transpose. If M is square, $\det(M)$ denotes its determinant and $\mathrm{adj}(M)$ its adjugate. If M is invertible, M^{-1} denotes its inverse. For a scalar $a \in \mathbb{Q}$, $a.M$ means that a is multiplied to all entries of the matrix M, while for $a \neq 0$, $\frac{M}{a}$ means that all entries of M are divided by a. For a vector v, $a.v$ and $\frac{v}{a}$ are used similarly. The concatenation $L||M$ of two n-row matrices with m_L and m_M columns respectively means their

concatenation to a n-row, $m_L + m_M$-column matrix. The notation $||$ is also used for the concatenation of more than two n-row matrices.

For M a matrix with n rows and m columns and i, j such that $1 \leq i \leq n$ and $1 \leq j \leq m$, $M_{i,j}$ denotes the coefficient in M at row i in column j, and M_j the vector which is equal to the j^{th} column of M.

The cardinality of a set is taken using #.

The symbol $:=$ is used in formulas to indicate that the function to its left can be computed using the expression to its right.

In the pseudocode of two algorithms in Section 8.4 a return statement does not terminate the execution. This is announced in the description of the enumeration algorithms. In all other cases, a return statement stops the execution immediately.

2.2 Elliptic curves and isogenies

SQIsign is an isogeny-based scheme, so a few notions on elliptic curves, isogenies and supersingular elliptic curves are necessary for understanding it.

The following is a collection of elements used in SQIsign, and as such similar to the introduction to isogenies in Antonin Leroux's thesis and the SQIsign specification [22, 4]. Most of the concepts are explained in more detail in Silverman's book [31], or in courses on isogenies [30, 33].

Elliptic curves

An elliptic curve is a smooth projective curve of genus one with a distinguished point. However, since defining the terms of this definition is too long for this text, we describe here only Weierstrass curves on a field of characteristic different from 2 or 3, and only consider their affine representation.

Such a curve is given by an equation of the form $y^2 = x^3 + ax + b$, where a and b are elements of a field $\mathbb{K}$ such that $4a^3 + 27b^2 \neq 0$. This last point is important, since it otherwise describes a curve with singular points which an elliptic curve cannot have. An elliptic curve is defined on a field $\mathbb{K}$ if the coefficients a, b of its equation are in $\mathbb{K}$. Therefore, a curve which is defined on $\mathbb{K}$ is also defined over all of its extension fields.

Given an elliptic curve E and a field $\mathbb{K}$ on which it is defined, we define its set of $\mathbb{K}$-rational points $E(\mathbb{K})$. This is the set of solutions in $\mathbb{K}$ to its Weierstrass equation, to which is added a special element called point at infinity and denoted by ∞. Therefore, all rational points except ∞ can be written as (x, y) where $x \in \mathbb{K}$ and $y \in \mathbb{K}$. This representation is unique.

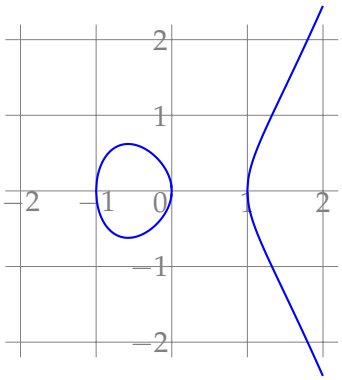

Figure 2.1: Solutions of the equation $y^2 = x^3 - x$ over $\mathbb{R}$

Elliptic curves are particularly interesting for cryptographers because they have a group law with explicit formulas which make it easy to compute. It is commutative and written as $+$. Its neutral element is ∞, and for a point P of E its inverse is denoted by $-P$ and defined to be the unique point such that $P + (-P) = \infty$.

Definition 2.1 *Group law*

Given an elliptic curve E of Weierstrass equation $y^2 = x^3 + ax + b$ which is defined on a field $\mathbb{K}$, the group law $+$ operates on its points as follows:

$$(P,Q) \longmapsto \begin{cases} \text{if } P = \infty : & Q \\ \text{if } Q = \infty : & P \\ \text{if } P = Q = (x,y), y \neq 0 : & f(P) \\ \text{if } P = (x,y), Q = (r,s), x = r, s = -y : & \infty \\ \text{if } P = (x,y), Q = (r,s), r \neq x : & g(P,Q) \end{cases}$$

where $f(P) = f((x,y)) = (\lambda^2 - 2x,\ -\lambda^3 + 2x\lambda - \frac{-x^3+ax+2b}{2y})$ with $\lambda = \frac{3x^2+a}{2y}$ and $g(P,Q) = g((x,y),(r,s)) = (\mu^2 - r - x,\ -\mu^3 + (r+x)\mu + \frac{ry-ry}{x-r})$ with $\mu = \frac{y-s}{x-r}$.

The case list is exhaustive, because the curve equation assures that $x = r$ implies $y = \pm s$.

The formulas of the group law are essentially fractions of polynomials and only depend on a and b. As a consequence, if the elliptic curve is considered over an extension field of $\mathbb{K}$, the group law stays essentially the same, and it still operates in the same way on points which already exist when considering E over $\mathbb{K}$. Therefore, the group law of E is a property of the curve E, not only of $E(\mathbb{K})$.

The iteration of the group law of an elliptic curve is called *scalar multiplication*. It depends only on $+$, and therefore on the curve.

Definition 2.2 *Scalar multiplication*

Let E an elliptic curve defined over a field $\mathbb{K}$.

*The map **scalar multiplication by** $m \in \mathbb{Z}$ is written as $[m]$, and for P a point of E it is given by:*

$$[m]P \quad = \quad \begin{cases} \text{if } m = 0 : \infty \\ \text{if } m > 0 : \sum_{i=1}^{m} P \\ \text{if } m < 0 : \sum_{i=1}^{-m} (-P) \end{cases}$$

For $m \neq 0$, the scalar multiplication by m is an example of an isogeny, or even more precisely, an endomorphism. These notions are defined in the next paragraph.

Isogenies

Isogenies are maps between elliptic curves with specific properties, which are used in the construction of isogeny-based cryptographic schemes. The following description of isogenies and their properties mostly follows Andrew Sutherland's lecture notes [33].

Rational maps on a field $\mathbb{K}$ are maps which have an expression as fractions of polynomials over $\mathbb{K}$. *Isogenies* are a special kind of rational maps between elliptic curves which are also group homomorphisms.

Definition 2.3 *Isogeny*

Let E, E' elliptic curves defined on a field $\mathbb{K}$.

Let $\phi : E \longrightarrow E'$ be such that:

- *ϕ is a rational map mapping $E(\overline{\mathbb{K}})$ to $E'(\overline{\mathbb{K}})$*
- *ϕ is not constant*
- *ϕ maps the point at infinity of E to that of E'*

*then ϕ is called an **isogeny** from E to E' and denoted by $\phi : E \longrightarrow E'$. If it is defined over $\mathbb{K}$ as a rational map, it is called a $\mathbb{K}$-rational isogeny.*

As two curves defined over a field $\mathbb{K}$ are also defined on its algebraic extensions, isogenies between them can be considered for any of these extensions.

Since rational maps on $\mathbb{K}$ are maps which have an expression as fractions of polynomials over $\mathbb{K}$, any isogeny can be written as a pair of fractions

of polynomials of $\mathbb{K}[X, Y]$. To evaluate the isogeny on a $\mathbb{K}$-rational point (x, y), the two fractions are evaluated on x and y, one of them providing the first, the other one the second coordinate of the resulting point. There is even a way to obtain a standard representation of isogenies by fractions of polynomials over $\mathbb{K}$.

Definition 2.4 *Standard representation by polynomials*

Let E, E' elliptic curves defined on a field $\mathbb{K}$, and ϕ a $\mathbb{K}$-rational isogeny from E to E'.

Then there are four polynomials $P, Q, R, S \in \mathbb{K}[X]$ such that P and Q are coprime, R and S are coprime, and for any $\overline{\mathbb{K}}$-rational point of E represented as (x, y) with $x, y \in \overline{\mathbb{K}}$, its image by ϕ is the point of E' represented by $\left(\frac{P(x)}{Q(x)}, y\frac{R(x)}{S(x)}\right)$ if $Q(x) \neq 0$ and $S(x) \neq 0$ and the point at infinity of E' otherwise.

These four polynomials are unique up to multiplication by an invertible element of $\mathbb{K}$.

Isogenies have many interesting properties, for example they are group morphisms for the group law of the elliptic curves they link, and surjective when the curves are considered over the algebraic closure. The composition $\psi \circ \phi : E \longrightarrow E''$ of two isogenies $\phi : E \longrightarrow E'$, $\psi : E' \longrightarrow E''$ is an isogeny.

Definition 2.5 *Degree, kernel, separability*

Let ϕ an isogeny of a curve E over $\mathbb{K}$ of standard form $(x, y) \longmapsto \left(\frac{P(x)}{Q(x)}, y\frac{R(x)}{S(x)}\right)$ for $P, Q, R, S \in \mathbb{K}[X]$.

*Its **degree** is $deg(\phi) = max(deg(P), deg(Q))$.*

*Its **kernel** is $\ker(\phi) = \{P \in E(\overline{\mathbb{K}}) : \phi(P) = \infty\}$.*

*If the derivative $(\frac{P}{Q})'$ of the fraction $\frac{P}{Q}$ is not zero, ϕ is called **separable**.*

The kernel of an isogeny on E is always a finite subgroup of $E(\overline{\mathbb{K}})$. Its order divides the degree of the isogeny. The degree of the composition of two isogenies is the product of their degrees.

Separable isogenies have a few additional properties. The composition of separable isogenies is separable. The degree of a separable isogeny is equal to the order of its kernel. On a field of non-zero characteristic p, it is not a multiple of p. Furthermore, a separable isogeny is entirely determined by its kernel. That means that given a finite subgroup of the group $E(\overline{\mathbb{K}})$ for E an elliptic curve over $\mathbb{K}$, there is (up to composition by an invertible isogeny) an unique image curve E' over $\overline{\mathbb{K}}$ and a unique rational map ϕ such that $\phi : E \longrightarrow E'$ is a separable isogeny over $\overline{\mathbb{K}}$. This isogeny can be computed, as explained in Section 3.1. As groups generated by a single element are called cyclic, an isogeny with cyclic kernel is called *cyclic*.

Two curves linked by an isogeny are called *isogenous*. Even though isogenies are in general not invertible, every isogeny between two curves has a dual, which is an isogeny linking the same curves in opposite order.

Definition 2.6 *Dual*

Let $\phi : E \longrightarrow E'$ an isogeny.

There is a unique isogeny $\hat{\phi} : E' \longrightarrow E$ of degree $deg(\phi)$ such that $\hat{\phi} \circ \phi = [deg(\phi)]$. $\hat{\phi}$ then also verifies $\phi \circ \hat{\phi} = [deg(\phi)]$.

The dual of a composition of isogenies is the composition of their duals in inverse order.

Since isogenies are group morphisms between elliptic curves (even if not every group morphism is an isogeny), some vocabulary from morphisms is also used for isogenies.

Definition 2.7 *Endomorphism, isomorphism*

*An **isomorphism** is a bijective isogeny whose inverse is also an isogeny. An **endomorphism** is an isogeny from a curve to itself.*

Being isomorphic (that is, linked by an isomorphism) defines an equivalence relation over the elliptic curves which are defined over the same field (or extension fields of it). The equivalence classes for being isomorphic over an algebraic closure are characterized by their *j-invariant* which is the same for two elliptic curves over the same field if and only if they are isomorphic.

Definition 2.8 *j-invariant*

Let E the elliptic curve over $\mathbb{K}$ given by the equation $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{K}$ and $4a^3 + 27b^2 \neq 0$.

*The **j-invariant** of E is $j(E) = 1728\frac{4a^3}{4a^3+27b^2}$*

As we only care about elliptic curves up to isomorphism, most often in the following the word *elliptic curve* is used for an isomorphism class of elliptic curves over an algebraic closure of their base field.

Also, isogenies which are obtained from each other by composition with an isomorphism are called *equivalent*. This also defines an equivalence relation. In particular, each equivalence class of isogenies links two isomorphism classes of elliptic curves. However, there are sometimes several distinct equivalence classes of isogenies linking the same isomorphism classes of elliptic curves. The degrees of two equivalent isogenies are equal.

Definition 2.9 *Endomorphism ring*

*The set of endomorphisms of a curve together with the constant mapping the whole curve to its point at infinity form a ring for point-wise addition and composition. This is called the **endomorphism ring** and denoted by* $\mathrm{End}(E)$.

Supersingular curves

Given the definition of an endomorphism and the endomorphism ring, we can finally define supersingular elliptic curves. This is the kind of elliptic curve we will work with in Chapter 3.

Definition 2.10 *Supersingular elliptic curve*

A supersingular elliptic curve is an elliptic curve E over a field of non-zero characteristic p such that its endomorphism ring is isomorphic to an order in a quaternion algebra (defined in Section 2.3). Equivalently, the kernel of $[p]$ on E is $\{\infty\}$.

Supersingular curves only exist on fields of non-zero characteristic, since the endomorphism ring is isomorphic to $\mathbb{Z}$ or an order in a quadratic imaginary field (and not a quaternion algebra) otherwise.

Proposition 2.11 *Supersingular curves are isogenous*

All supersingular elliptic curves over a field $\mathbb{K}$ are isogenous over the algebraic closure $\overline{\mathbb{K}}$ of $\mathbb{K}$.

Furthermore, every supersingular elliptic curve defined over $\mathbb{K}$ in characteristic $p > 0$ is isomorphic to a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. This is very useful in computations which only need to be done up to isomorphism, and used by SQIsign which only works with supersingular elliptic curves over $\mathbb{F}_{p^2}$ for a fixed large prime p.

Working with isogenies and endomorphisms of these curves requires some notions on orders and ideals in a quaternion algebra, which are introduced in the next paragraph.

2.3 Quaternion algebras

The notions introduced here are presented in more details and generality in Voight's book on quaternion algebras [35]. Other useful references on the quaternion algebra notions used in SQIsign are Antonin Leroux's thesis [22], the first paper on SQIsign [14] and the specification [4].

Quaternion algebras

As quaternion algebras behave slightly differently in characteristic two, we do not treat that case. Therefore all fields in this part are supposed to be of characteristic different from 2.

Definition 2.12 *Quaternion algebra*

Let $\mathbb{K}$ a field.

A 4-dimensional $\mathbb{K}$-algebra which has a basis $(1, i, j, ij)$ and $a, b \in \mathbb{K} \setminus \{0\}$ such that:

- $i^2 = a$
- $j^2 = b$
- $ij = -ji$

is called a quaternion algebra. Since all quaternion algebras over $\mathbb{K}$ with $i^2 = a$ and $j^2 = b$ are isomorphic, they are called the quaternion algebra $\mathbb{H}(a, b)$ over $\mathbb{K}$.

Since $ij = -ji \neq 0$, a quaternion algebra is non-commutative.

Every quaternion algebra has a standard involution which defines its reduced norm and trace.

Definition 2.13 *Standard involution, reduced norm and trace*

Let $\mathbb{K}$ a field and $\mathbb{H}(a, b)$ a quaternion algebra over $\mathbb{K}$ of standard basis $(1, i, j, ij)$.

Let $x = x_1 + x_i i + x_j j + x_{ij} ij \in \mathbb{H}(a, b)$ with $x_1, x_i, x_j, x_{ij} \in \mathbb{K}$.

The standard involution of x is $\overline{x} = x_1 - x_i i - x_j j - x_{ij} ij$.

Its reduced norm is $N(x) = x\overline{x}$, and its reduced trace is $tr(x) = x + \overline{x}$. The reduced trace is a linear map and the reduced norm is a quadratic form.

For SQIsign, we work within the quaternion algebra $\mathbb{H}(-1, -p)$ over the field $\mathbb{Q}$, where $p \equiv 3 \mod 4$ is the characteristic of the finite field over which we consider our elliptic curves, and therefore prime. Since $-p$ and -1 are both negative, $\mathbb{H}(-1, -p)$ is a definite quaternion algebra. In this case, the reduced norm is a positive definite quadratic form and all non-zero elements $x \in \mathbb{H}(-1, -p)$ have a multiplicative inverse which is $\frac{1}{N(x)}\overline{x}$.

	1	i	j	ij
1	1	i	j	ij
i	i	-1	ij	$-j$
j	j	$-ij$	$-p$	pi
ij	ij	j	$-pi$	$-p$

Figure 2.2: Multiplication table of the standard basis $(1, i, j, ij)$ of $\mathbb{H}(-1, -p)$

For the remainder of this text, we note $\mathcal{A} = \mathbb{H}(-1, -p)$ this quaternion algebra, and all considered objects and operations which are in a quaternion algebra are in $\mathcal{A}$. Furthermore, the letters i and j will only be used to denote the elements of the basis $(1, i, j, ij)$ such that $i^2 = -1$ and $j^2 = -p$ of $\mathcal{A}$.

Lattices in an algebra

Definition 2.14 *Lattice*

Let V a $\mathbb{Q}$-vector space of finite dimension n.

Let $B = (b_1, ..., b_n)$ a basis of V.

Then $L = \{x \in V \mid \exists (a_1, ...a_n) \in \mathbb{Z} : x = \sum_{k=1}^{n} a_k b_k\}$ is a lattice of V.

This definition of a lattice is very simple, but sufficient for our case, even though there exists a more general definition for arbitrary dimensions and other rings than $\mathbb{Z}$. It is important to notice that we require all lattices to have the same dimension as the vector space.

Since lattices are $\mathbb{Z}$-linear combinations of a basis, they are groups for addition.

Definition 2.15 *Definitions for a lattice*

Let L a lattice in a finite-dimensional $\mathbb{Q}$-vector space V.

*The **dual** of L is defined with respect to a symmetric nondegenerate bilinear form s on V as the set $L_s^\# = \{b \in V \mid \forall a \in L,\ s(a,b) \in \mathbb{Z}\}$. $L_s^\#$ is a lattice in V.*

*The **conjugate** of L is $\overline{L} = \{\overline{x} \mid x \in L\}$. $\overline{L}$ is a lattice.*

*The **volume** of L is the absolute value of the determinant of any basis of L.*

*Elements of L which are not multiples of another element of L by any positive integer larger than one are called **primitive elements** of L. For $\alpha = n.\beta$ with $\alpha, \beta \in L$, $n \in \mathbb{N} \setminus \{0\}$ and β primitive in L, we call β the **primitive part** of α.*

There are also operations on more than one lattice.

Definition 2.16 *Operations on lattices*

Let L_1 and L_2 two lattices in the same $\mathbb{Q}$-vector space V.

*The **sum** of L_1 and L_2 is $\{a + b \mid a \in L_1, b \in L_2\}$, and their **intersection** is their intersection as sets. Both of these are again lattices of V.*

*If $L_1 \subset L_2$, the **index** $[L_2 : L_1]$ of L_1 in L_2 is the integer $\#(L_2/L_1)$ (the number of elements of L_2 quotiented by L_1). This is also the quotient of the volumes of L_2 and L_1.*

Since we work in a fixed quaternion algebra, all lattices we consider are lattices of $\mathcal{A}$. Lattices in an algebra are often called *fractional ideals* in the literature, because they are obtained by dividing integral ideals (defined later) by an integer. In order to avoid confusion with integral ideals, the term *fractional ideal* is not used in this text. The equivalent term *lattice* is used instead. It is understood that from this point all lattices are in $\mathcal{A}$, except if explicitly stated otherwise.

The multiplication in the algebra allows us to define additional operations on lattices in $\mathcal{A}$, such as the product and the colon lattice of two lattices.

Definition 2.17 *Lattice product*

*The **product** of two lattices L_1 and L_2 is the lattice L_1L_2 generated by $\{ab \,|\, a \in L_1, b \in L_2\}$, where the product used is the product of the quaternion algebra $\mathcal{A}$. Therefore, the lattice product is not commutative.*

It is also possible to multiply lattices with algebra elements. The product of a lattice L by $x \in \mathcal{A}$ on the left is $xL = \{xb \,|\, b \in L\}$. The right product of L by x is $Lx = \{bx \,|\, b \in L\}$. The sets resulting from these lattice products are lattices (except if $x = 0$).

Definition 2.18 *Colon lattice*

*The **right colon lattice** from a lattice L_1 to a lattice L_2 is $(L_2 : L_1)_\mathrm{R} = \{x \in \mathcal{A} \,|\, L_1x \subset L_2\}$. Similarly, the **left colon lattice** from L_1 to L_2 is defined by $(L_2 : L_1)_\mathrm{L} = \{x \in \mathcal{A} \,|\, xL_1 \subset L_2\}$.*

The colon lattice is frequently called *colon ideal* in the literature, including Voight's book [35]. As we do not use the term fractional ideal and reserve the term ideal for the objects defined below, the less confusing term *colon lattice* is preferred in this text. Furthermore, the SQIsign specification, the PARI/GP library and the reference implementation call the same object *transporter*. This term is not used here.

Orders and ideals

Definition 2.19 *Order*

*An **order** is a lattice which is also a subring of $\mathcal{A}$. An order is called **maximal** if it is not strictly included in any other order.*

Examples of orders are the lattices generated by $(1, i, j, ij)$ and $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$. The later is a maximal order. Another way of finding orders is to obtain them from lattices.

Definition 2.20 *Left and right order of a lattice*

*Let L a lattice of $\mathcal{A}$. The sets $\mathcal{O}_\mathrm{L}(L) = \{x \in \mathcal{A} \,|\, Lx \subset L\}$ and $\mathcal{O}_\mathrm{R}(L) = \{x \in \mathcal{A} \,|\, xL \subset L\}$ are orders of $\mathcal{A}$ and called the **left order** and the **right order** of L.*

The right order of a lattice L is the right colon lattice $(L : L)_\mathrm{R}$ and its left order is $(L : L)_\mathrm{L}$.

Definition 2.21 *Ideal*

*A lattice I such that $I \subset \mathcal{O}_\mathrm{L}(I)$, or equivalently $I \subset \mathcal{O}_\mathrm{R}(I)$, or, again equivalently, $I^2 = II \subset I$ is called an **integral ideal** or simply an **ideal**.*

*An integral ideal I only contains elements of integer norm. Its **norm** is denoted by $N(I)$ and defined as the greatest common divisor of the norms of its elements.*

The *sum* and *intersection* of two ideals with the same left order are their sum and intersection as lattices. The product of two ideals I, J such that $\mathcal{O}_{\mathrm{R}}(I) = \mathcal{O}_{\mathrm{L}}(J)$ is their product IJ as lattices. Intersection, sum and product of ideals are ideals. The conjugate $\overline{I}$ of an ideal I is its conjugate as lattice. It is an ideal of the same norm with $\mathcal{O}_{\mathrm{L}}(\overline{I}) = \mathcal{O}_{\mathrm{R}}(I)$ and $\mathcal{O}_{\mathrm{R}}(\overline{I}) = \mathcal{O}_{\mathrm{L}}(I)$.

An integral ideal is a left ideal of its left order and a right ideal of its right order in the usual sense of ideals in rings, since it is included in both and orders are subrings and therefore rings. But it is also a left (respectively right) ideal for all suborders of $\mathcal{O}_{\mathrm{L}}(I)$ (respectively $\mathcal{O}_{\mathrm{R}}(I)$) in which it is included. Therefore, given an order O, a *left O-ideal* is an ideal I such that $I \subset O \subset \mathcal{O}_{\mathrm{L}}(I)$. Right O-ideals are defined analogously.

A left O-ideal I is called *cyclic* if for all primes q, $I \not\subset qO$. With this denomination, we follow Leroux's thesis [22] and remain coherent with the denominations for isogenies. What we call cyclic ideals is normally known as primitive ideals.

A *principal* left ideal is generated by an element α of its left order, as $I = \mathcal{O}_{\mathrm{L}}(I)\alpha$. α is called its generator.

All ideals I of $\mathcal{A}$ are generated as $I = \mathcal{O}_{\mathrm{L}}(I)x + \mathcal{O}_{\mathrm{L}}(I)n$ by pairs n, x where n is their norm and x an element of their left order. Such an element x is called a *generator* of I even if the ideal is not principal. An ideal has a primitive generator if and only if it is cyclic.

Definition 2.22 *Equivalences*

Two orders O and O' are equivalent if there is $\alpha \in \mathcal{A} \setminus \{0\}$ (so α is invertible) such that $\alpha O = O'\alpha$.

Two left ideals I and J of an order O are isomorphic if there is some $\beta \in \mathcal{A} \setminus \{0\}$ such that $I = J\beta$. Being isomorphic is an equivalence relation (since all non-zero elements are invertible), and isomorphic ideals are therefore called equivalent.

If two left O-ideals are isomorphic, their right orders are equivalent.

Definition 2.23 *Connecting ideals*

*An ideal I is said to **connect** $\mathcal{O}_{\mathrm{L}}(I)$ and $\mathcal{O}_{\mathrm{R}}(I)$ if these are maximal orders. I is then called a **connecting ideal** of $\mathcal{O}_{\mathrm{L}}(I)$ and $\mathcal{O}_{\mathrm{R}}(I)$.*

All pairs of maximal orders in $\mathcal{A}$ have a connecting ideal, and a connecting ideal is not unique. For example, all its multiples by elements of $\mathbb{Q}$ are still connecting ideals of the same orders.

An ideal I which has $\mathcal{O}_{\mathrm{L}}(I)$ or $\mathcal{O}_{\mathrm{R}}(I)$ maximal has both of them maximal. Ideals of maximal orders have an *inverse*. For an ideal I, this is the lattice $I^{-1} = \frac{1}{N(I)}\overline{I}$. It verifies $II^{-1} = \mathcal{O}_{\mathrm{L}}(I)$ and $I^{-1}I = \mathcal{O}_{\mathrm{R}}(I)$.

In the later sections of this text, all ideals are ideals of maximal orders.

2.4 Linking both: The Deuring Correspondence

We have already seen that the endomorphism ring of a supersingular elliptic curve over a field of characteristics $p = 3 \mod 4$ is isomorphic to an order of the quaternion algebra $\mathcal{A} = \mathbb{H}(-1, -p)$. This is only one point of a much larger correspondence between equivalence classes of ideals and orders in $\mathcal{A}$ and equivalence classes of isogenies and supersingular elliptic curves over a field of characteristic p.

SQIsign exploits this correspondence in order to do many computations on ideals in $\mathcal{A}$, even though all public information consists in elliptic curves and isogenies. The Table 2.3 summarizes the equivalences on which SQIsign relies most. It is important to notice that everything in this table is valid only up to isomorphism.

Supersingular elliptic curves	**Ideals and orders of** $\mathcal{A} = \mathbb{H}(-1, -p)$
endomorphism ring	maximal order
endomorphism	principal ideal of a maximal order
isogenies between two curves	ideals between two maximal orders
composition of isogenies	ideal product
isogeny degree	ideal norm
dual isogeny	conjugate ideal

Figure 2.3: The Deuring correspondence for curves over $\mathbb{F}$ of characteristic p

Most arguments and examples in the following paragraphs are taken from the usual texts on SQIsign [22, 14]. Some additional information on computations and hardness assumptions can be found in articles on the Deuring correspondence [13, 36].

An example

As explained in the original SQIsign paper and Leroux's thesis [14, 22], the maximal order O_0 of basis $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ in $\mathcal{A} = \mathbb{H}(-1-p)$ corresponds exactly to the endomorphism rings of supersingular elliptic curves with j-invariant 1728 over fields of characteristic p. One such curve, denoted by E_0, is given by the equation $y^2 = x^3 + x$.

In order to write an explicit mapping of endomorphisms of E_0 to elements of O_0, we first need to identify a generating subset of $\mathrm{End}(E_0)$. Consider the identity 1, the Frobenius endomorphism π which maps a $\overline{\mathbb{F}_p}$-rational point given by (x, y) to (x^p, y^p) and the endomorphism ι mapping (x, y) to $(-x, \sqrt{-1}y)$ where $\sqrt{-1}$ denotes a square root of -1 in $\overline{\mathbb{F}_p}$. As isogenies are pairs of fractions of polynomials, they can be added, composed and divided by 2 to obtain $(1, \iota, \frac{\pi+\iota}{2}, \frac{1+i\circ\pi}{2})$, which is a basis of $\mathrm{End}(E_0)$.

The $\mathbb{Z}$-linear application mapping $(1, \iota, \frac{\pi+\iota}{2}, \frac{1+j\circ\pi}{2})$ element by element to the basis $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ of $\mathcal{O}_0$ is an isomorphism of rings between $\mathrm{End}(E_0)$ and O_0.

Since this explicit map is helpful for computations around the Deuring correspondence, the curve E_0 and the corresponding order O_0 play an important role in some later parts of this text. We therefore keep the notations E_0 and O_0 for them.

Two worlds separated by computation

Even though the world of isogenies and the world of quaternion ideals are linked by this correspondence, they are computationally very different. One world requires to work with polynomials, kernels of isogenies and their evaluation, and the other one mostly relies on integer linear algebra in dimension 4. More importantly, there are some problems whose difficulty is very different in both worlds.

The *isogeny path problem* for example is the following: Given the Weierstrass equations of two curves, find an isogeny between them. This problem is believed to be hard, as no efficient solution is known. Variants, like finding isogenies of a given degree or of a degree which is a power of some given prime (the latter is called ℓ-isogeny path problem, where ℓ denotes the prime) are also considered hard, and various cryptographic schemes are built on these assumptions.

The equivalent problem in the ideal world, which is finding an ideal linking two maximal orders in $\mathcal{A}$, is not hard, and algorithms for it are known (as the one in Section 7.4 for example). It is more complicated, but, if it exists and with some restrictions, also possible to find an ideal of a given norm between two orders. This is the main purpose of the algorithm KLPT from 2014 [21] which is explained in Section 8.

Passing from one world to the other

This difference in computational difficulty of the same problem in the quaternion world and in the isogeny world makes clear that the translation from one world to the other is not always easy.

It is in general asumed to be hard to compute the endomorphism ring of an elliptic curve if only its equation is given. Exceptions such as the curve E_0 from the example above exist, but there is no efficient solution for the general case of this *endomorphism ring problem*. Benjamin Wesolowski even showed that it is equivalent to the ℓ-isogeny path problem under the Generalized Riemann Hypothesis [36]. Therefore, passing from the isogeny to the ideal world is hard, as long as only curve equations are known.

However, if an isogeny from a curve with known endomorphism ring to another curve is known, the endomorphism ring of the image curve can be obtained using this knowledge, as described in Section 3.1.

In the other direction, given a maximal order O, it is possible to compute an elliptic curve with endomorphism ring isomorphic to O in polynomial time, even if it is not always fast, as [13] points out. Since the specification [4] of SQIsign fixes the prime p to values which simplify this computation, it is feasible in reasonable time in our case.

The correspondence between ideals and isogenies can be detailed as follows: Given ϕ an isomorphism from the endomorphism ring $\mathrm{End}(E)$ of a supersingular elliptic curve E to the corresponding order O in $\mathcal{A}$, any isogeny of E corresponding to a left O-ideal I has a kernel equal to $E[I]$, which is the intersection of the kernels of the endomorphisms in the image $\phi^{-1}(I)$. For translating an isogeny into an ideal, it is therefore necessary to know the endomorphism ring of the domain curve. Algorithms for computing the translation in both directions are described in detail in [13], and summarized in Section 3.1 in the next chapter.

Chapter 3

Introduction to SQIsign

The Deuring correspondence is at the heart of SQIsign, a signature scheme where most private computations are done in $\mathcal{A}$ and all public ones on isogenies and elliptic curves. It exploits the easily computable cases identified in Section 2.4 to translate between both worlds, while relying of the hardness on the more difficult ones for its security.

In order to give an idea of the functioning of SQIsign, we first introduce a few important algorithmic building blocks, without going too much into details. Then, we describe the scheme. This allows us to finally group the different objects and algorithms used by SQIsign into a handful of modules, and to understand the role of quaternion algebra operations in its implementation.

This chapter is a summary of the descriptions of SQIsign in previous documents such as [14, 22, 4]. It mostly follows the specification [4], but without going as much into details.

3.1 Building blocks

In this section, we introduce some algorithmic building blocks of SQIsign. Most of these are needed for the translations between the quaternion and the isogeny world. We first summarize how isogenies are represented and evaluated. Then the KLPT algorithm for finding equivalent ideals of given norm is shortly introduced. Finally, we explain at high level how these two blocks are used for changing between both worlds. In addition, an idea of how random ideals and isogenies are constructed is given in the last paragraph.

Representing and evaluating isogenies

Since an isogeny is a rational function between two elliptic curves, one simple way of representing it is to give two fractions of polynomials, a domain and an image curve. Computations and evaluation using this representation are fast.

Separable isogenies are also uniquely represented by the domain curve and a subgroup of it which is the kernel of the isogeny. To evaluate a separable isogeny given by its kernel, Vélu's formula [34] and its variants are used. Since these computations are linear in the degree of the isogeny (or in square root of the degree for $\sqrt{\text{élu}}$ [2]), this computation is in general impossible for isogenies with large degree. However, it is possible to factorize a separable isogeny ϕ of composite degree n into a composition of separable isogenies such that n is the product of their degrees. Then Vélu's formula can be applied for each of the prime degree isogenies in the decomposition. If ϕ is of smooth degree, this is fast enough. Therefore, separable isogenies can be computed from their kernel if and only if their degree is smooth.

Isogenies can also be given as ideals in a quaternion algebra. Depending on the representation of the endomorphism ring, this often means that isogenies are represented by $\mathbb{Q}$-linear combinations of endomorphisms as described in [13]. As explained in Section 2.4, this corresponds to giving a kernel of the isogeny, and can therefore only be evaluated if the ideal is of smooth norm (which corresponds to an isogeny of smooth degree).

We often use cyclic isogenies, because their kernel is generated by one element (instead of two).

KLPT for equivalent ideals

The algorithm KLPT was invented in 2014 by Kohel, Lauter, Petit and Tignol in [21]. It takes as input an ideal I in a quaternion algebra and an integer n, and computes an ideal J which is isomorphic to I and has norm n, as long as n is large enough.

More precisely, KLPT uses the special structure of the order O_0 of basis $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ which contains the suborder $\mathbb{Z}[i] + j\mathbb{Z}[i]$, in order to find an element α of I of norm $nN(I)$. The conjugate of α is then multiplied to I and divided by $N(I)$ in order to obtain an isomorphic ideal of norm n (since $N(\frac{1}{N(I)} I\overline{\alpha}) = \frac{1}{N(I)^2} N(I\overline{\alpha}) = \frac{N(I)N(\overline{\alpha})}{N(I)^2} = n$). The search for such an element is described in more detail in Chapter 8. A variant of this method is used to find an equivalent ideal whose norm is a power of some prime number, where the exponent is not necessarily specified at the beginning.

The original KLPT algorithm always produces ideals connecting to O_0. It was modified by the inventors of SQIsign in order to not immediately re-

veal connections from O_0 to the left or right order of its input. Details of this adaptation are not explained here, but can be found in the papers on SQIsign [14, 9].

Passing between worlds

The algorithms which realize the correspondence from Section 2.4 by translating isogenies to ideals and ideals to isogenies are central in SQIsign. They are quite complex and need to be very optimized since they are the slowest part of current implementations. Since they are not at the center of this work, only a few notes are given here in order to explain some parameter choices and requirements.

Given a left O_0-ideal I of norm not divisible by p, the elements of a basis B of I are elements of O_0 and therefore generators of principal ideals. Since O_0 corresponds to E_0 and an explicit correspondence of generators of O_0 to endomorphisms of E_0 is known, the elements of B are represented using these generators and then translated into endomorphisms of E_0. Then the intersection $E_0[I]$ of their kernels is computed. The separable isogeny of kernel $E_0[I]$ is one of the equivalent isogenies corresponding to I, and it is computed from its kernel $E_0[I]$ using Vélu's formula. As explained above, this is efficient if the degree of the isogeny (which equals the norm of I) is smooth.

Sometimes a left O_0-ideal I is given and only the image curve of an isogeny corresponding to I is needed, but not the isogeny itself. In this case, an ideal J equivalent to I and with smooth norm is computed in a first step using KLPT. Then J is translated into an isogeny ϕ_J and its image curve is computed.

Variants of these algorithms which do not only work on left O_0-ideals exist. There are many possible optimizations, which are described for example in the papers on SQIsign [14, 9] or on the Deuring correspondence [13].

In the other direction, given a cyclic, separable isogeny of domain E_0 represented by a generator P of its kernel, it is possible to compute the corresponding left O_0-ideal by evaluating two orthogonal endomorphism on P. The right order of the resulting ideal is isomorphic to the endomorphism ring of the image curve of the isogeny.

Detailed descriptions of these algorithms are given by [13] in a quite general setting. The papers on SQIsign and the specification [14, 9, 4] add many optimizations which are sometimes specific to our case.

Randomly sampled ideals and isogenies

When a random isogeny of a given degree D is needed, there are two ways of generating it. Either it is created as an isogeny, or as an ideal.

If the ideal world is not accessible, a separable and cyclic isogeny of smooth degree D is chosen by selecting a point of order dividing D on its domain curve, which is then used as generator of the kernel of the isogeny. To randomly select such a point K, it is sufficient to choose a uniformly random integer h between 1 and D (both included) and to compute K as $P + [h]Q$ where P, Q is a (deterministically chosen) basis of the subgroup of order D of the domain curve.

Randomly sampling a cyclic ideal of norm n in a maximal order O of $\mathcal{A}$ is done by selecting a random primitive order element x in O such that $\gcd(N(x), n) = n$. The ideal generated by x and n is then used as random ideal. Selecting a random order element of norm a multiple of n is very similar to some internal steps of the KLPT algorithm. For example, in the case where $O = O_0$ the algorithm SolveNormEquation from Section 8.1 can be used. If the norm of the ideal is not fixed in advance, it has to be randomly chosen among suitable values in order to then use the above method.

The specification [4] explains the random choice of objects required for SQIsign in more detail.

3.2 Description of SQIsign

SQIsign is a signature scheme obtained from a sigma protocol via a Fiat-Shamir transformation [14, 4].

A public-coin sigma protocol for identification consists in the exchange of three messages between two participants. The *prover* who has the secret key, tries to convince the *verifier* that he knows the secret key, without revealing any other information on it. For this, he first computes a *commitment* message using his secret information and public parameters. The verifier then sends a random value within some protocol-dependent space as *challenge*. He does not keep any information secret. The prover finally computes his *response* message as a function of the parameters, his secret values and the challenge. Upon reception of the response, the verifier can check whether the total *transcript* formed by commitment, challenge and response convinces him. For the protocol to be correct, he should accept if the prover behaved as specified. For its security, the verifier should generally not accept if the prover does not know the secret key, and not gain any information on the secret key except whether the prover does or does not know it.

The Fiat-Shamir transformation (invented in [15], applied to SQIsign in [14]) replaces the interaction with the verifier by computing the challenge as a

hash of the commitment and the message to sign. The signature then contains the commitment and the response.

Setup and keys

Following the specification [4], the public parameters are two large, smooth, positive and coprime integers D_{com} and D_{chall}, as well as the curve $E_0 : y^2 = x^3 + x$ over the finite field $\mathbb{F}_{p^2}$ and its endomorphism ring which is isomorphic to the order $\mathcal{O}_0 \subset \mathcal{A}$ via the explicit isomorphism given as example in Section 2.4.

The private key is a random left $\mathcal{O}_0$-ideal I_κ with randomly chosen norm D_κ of the same bit length as $p^{1/4}$, sampled as described in Section 3.1. An equivalent ideal of smooth norm and the corresponding isogeny κ of domain E_0 as well as its image curve E_κ are computed using KLPT and the ideal-to-isogeny translation. E_κ is the public key.

Commitment-challenge-response

The specification [4] explains in great detail the computation of commitment, challenge and response in SQIsign, which are, with slight variations, also presented in [14, 9]. This presentation summarizes the specification.

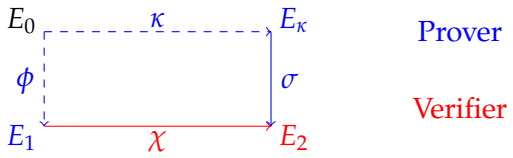

Figure 3.1: SQIsign in the isogeny world. Dashed arrows are secret

Commitment: The prover chooses a random left $\mathcal{O}_0$-ideal I_ϕ of norm equal to D_{com}. Then he computes the corresponding separable isogeny ϕ of domain E_0 and the image curve E_1. E_1 is the commitment, and ϕ is kept secret.

Challenge: The verifier selects a random cyclic separable isogeny χ of domain E_1 and degree D_{chall}, which he sends as challenge message.

Response: The prover computes the composition of the challenge isogeny χ and its secret ϕ. This gives a separable cyclic isogeny $\chi \circ \phi : E_0 \longrightarrow E_2$ of smooth degree $D_{chall}D_{com}$. He converts it into a left $\mathcal{O}_0$-ideal I, which he multiplies with the conjugate of his secret key $\overline{I_\kappa}$, which is a right $\mathcal{O}_0$-ideal. He obtains $\overline{I_\kappa}I$. Then he uses a specialized version of KLPT to get an equivalent ideal J with a power of 2 as norm D. For this computation, I_κ is needed in order to provide a link to $\mathcal{O}_0$. J is conjectured by [14, 9] to not reveal any information on ϕ or I_κ. Since the response isogeny σ should be

such that $\hat{\chi} \circ \sigma$ is cyclic, J is replaced by another equivalent ideal if it does not fulfill this condition (this is tested by checking if there is a primitive generator of the non-principal ideal $J\overline{I_\chi}$). The final J is converted to a cyclic separable isogeny σ which is used as response.

Verification: The verifier checks whether the response σ is a cyclic separable isogeny from the public key E_κ to the image curve E_2 of the challenge isogeny.

Signature and verification

In order to obtain SQIsign, the above identification protocol is transformed into a signature scheme by the Fiat-Shamir transformation [4, 14].

When signing, there is no interaction with the verifier. As a consequence, instead of randomly choosing the challenge isogeny, a hash of the commitment curve and the message is computed. This results in an integer between 1 and D_{chall}, which is then used to deterministically compute a cyclic separable isogeny χ of degree D_{chall} in the same way than random isogenies are obtained from random integers in Section 3.1.

As fixed in the specification [4], the signature then consists in generators of the kernels of the response isogeny σ and the dual of the challenge isogeny $\hat{\chi}$. The later one is used instead of E_1 because a generator of the kernel of $\hat{\chi}$ is a short representation of E_1 given the other available information.

During signature verification, it is necessary to check whether the signature encodes two separable isogenies σ and $\hat{\chi}$ of domains E_κ and the image curve of σ respectively and with the expected degrees. Since $\hat{\chi} \circ \sigma$ has E_1 as image curve and the public key E_κ as domain, the commitment E_1 is obtained as image curve of $\hat{\chi} \circ \sigma$. Then the challenge isogeny χ is computed as hash of E_1 and the message. This is used to verify that $\hat{\chi}$ and χ are duals of each other.

Security of SQIsign

A prerequisite for the security of SQIsign, as for any scheme, is that is correctly used. For example, the prover or signer must keep secret the ideals and isogenies from E_0 to the public key E_κ and to the curves E_1 and E_2 involved in a signature. If he gave out for example the commitment ideal or isogeny, an attacker could compose it with the public challenge and response isogenies to obtain a connection from E_0 to the public key curve E_κ. This allows them to efficiently compute the corresponding ideal as described in Sections 2.4 and 3.1 in order to use it as secret key. Almost the same key recovery attack is possible given a connection from E_0 to E_2 or E_κ.

Similarly, revealing the endomorphism ring of any of the three curves E_κ, E_1 and E_2 involved in a signature gives an attacker access to the quaternion algebra world which allows them to efficiently compute an equivalent secret key.

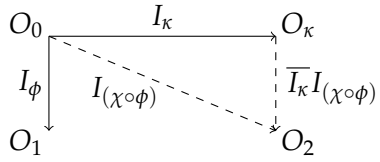

Figure 3.2: Quaternion ideals and orders used in SQIsign's signing procedure. All of them are secret except O_0

If used correctly, the security of SQIsign relies on two assumptions.

The first one is the assumed hardness of the computation of a non-trivial endomorphism of smooth degree of a supersingular elliptic curve [4]. So far, the most efficient algorithms for solving this problem have a complexity at least quasi-linear in $\sqrt{p}$ for classical computers (Delfs-Galbraith [10]) and in $p^{\frac{1}{4}}$ for quantum computers (Biasse-Jao-Sankar [3]). Quasi-linear complexity in x means that their asymptotic complexity $C(x)$ towards large values of an integer parameter x is bounded above and below by terms of the form $\lambda x \log(x)^k$ where some $k > 1$ and $\lambda > 0$. Even if it is not proven that finding a single non-trivial endomorphism is as hard as finding all of them, these algorithms are the same as those used for computing the full endomorphism ring. They therefore also solve the isogeny path problem, since these two problems where shown to be equivalent by [12, 36].

The second one is that the response isogenies obtained from the output of the KLPT variant used for the response computation (often called Signing-KLPT as in [14, 9, 4]) have a distribution which is computationally indistinguishable from random cyclic isogenies of same degree and domain, if the secret key is unknown. For the original Signing-KLPT from [14], this assumption was shown to be false by [9] and a corrected version is used today. However, no attack exploiting a distinguisher for this weakness is known so far.

As this work focuses on the quaternion algebra algorithms of the scheme and not on its security, the above presentation is very short and not very detailed. A more precise wording of the involved assumptions, more details on security notions and justified parameter choices (which guarantee that all known attacks on SQIsign have a complexity not better than quasi-linear in $\sqrt{p}$ for classical computers and in $p^{\frac{1}{4}}$ for quantum computers) are given in the specification [4] submitted to NIST. Explanations of the security of

SQIsignare given in the articles [14, 9].

3.3 Modules and dependencies in SQIsign

Since we have seen above how SQIsign's key generation, signing and verification procedures work, we can now list the objects and basic operations this scheme requires and classify them into modules as needed for practical implementations.

The quaternion algebra operations used for key generation and signing are mostly the generation of a random ideal of fixed norm, product, conjugation, computation of equivalent ideals and ideal-to-isogeny and isogeny-to-ideal translations.

Since the computation of an equivalent ideal using KLPT requires many basic operations on quaternion algebra elements, orders and ideals, and both translations require the computation of equivalent ideals on the quaternion algebra side, these three main parts can be seen (and implemented) as three different modules. Then KLPT depends on the basic quaternion module and the ideal-isogeny translations depend on both of them. Furthermore, all of these require operations on large integer numbers and fractions, and are used by the signing and key generation algorithms of the signature.

Obviously, the ideal-to-isogeny translation also depends on some isogeny and elliptic curve computation module. The same module is used for the verification. The elliptic curve and isogeny computations require arithmetic in $\mathbb{F}_{p^2}$ provided by another module.

This results in the module dependency graph in Figure 1.2 on page 5, which corresponds closely to the implementation made for the NIST submission [4].

The rest of this work focuses on objects and algorithms for the quaternion algebra module. This module contains mostly general-purpose quaternion algebra operations, which might also be useful outside of SQIsign, if operations in a definite quaternion algebra over $\mathbb{Q}$ are needed. However, since this description is intended for SQIsign, we explain them for the quaternion algebra $\mathcal{A} = \mathbb{H}(-1, -p)$. Furthermore, since KLPT is also an algorithm operating exclusively on quaternion algebra elements, a basic variant of it is described in Section 8, to give an idea of what the KLPT module requires from the quaternion algebra module it relies on.

Chapter 4

Representing algebra, ideals and orders

For describing the computation of most of the operations on elements, lattices, orders and ideals from Section 2.3, a representation of these objects is necessary. This section fixes and explains one which is used in the remainder of the text. It is similar, but not identical to the representation chosen in the implementation from [4]. The differences are detailed in Section 9.1.

4.1 The algebra and its elements

As stated above, all objects and computations within a quaternion algebra we use are relative to $\mathcal{A} = \mathbb{H}(-1, -p)$ for a fixed prime $p \equiv 3 \mod 4$. This algebra and the prime p which defines it are supposed to be universally known. All algorithms presented in this work receive p as an implicit parameter.

Algebra elements

Quaternion algebra elements are represented as vectors of four rationals. For this to be meaningful, it is necessary to define the basis in which they are represented. The standard basis $(1, i, j, ij)$ of the algebra is a natural choice and is used in the remainder of this text. This choice allows to easily compute most operations on algebra elements as follows:

The **sum** of two algebra elements is computed by vector addition in $\mathbb{Q}^4$.

Their **product** is done using the multiplication Table 2.2.

The **conjugate** $\overline{\alpha}$ of $\alpha \in \mathcal{A}$ is computed by negating its 3 last coordinates.

The **reduced norm** of $\alpha \in \mathcal{A}$ is the product of α with its conjugate.

The **reduced trace** of $\alpha \in \mathcal{A}$ is the sum of α and its conjugate.

Vectors of four rationals can be represented as vectors of four integers on a common denominator (using five integers in total), or as vectors of four rationals with each of them having its own denominator (using therefore eight integers). For our SQIsign reference implementation we chose the first possibility. However, the choice of any of these representations is compatible with the rest of this text.

4.2 Lattices, ideals, orders

Ideals and orders are lattices, so a representation for lattices of $\mathcal{A}$ is needed in order to represent them.

Lattice representation

Lattice are represented by a basis. Since all lattices we consider are of full rank and in dimension 4, we represent them by a rational invertible 4×4 matrix whose columns correspond to quaternion algebra elements in the basis $(1, i, j, ij)$ which form a basis of the lattice. This rational matrix is represented as a 4×4 integer matrix with a single integer as common denominator.

Furthermore, in order to efficiently obtain a basis from any finite set of generators, the Hermite Normal Form (HNF, described in Section 5.1) of the integer matrix is used. This form defines a unique representation of a lattice, if the fraction of the matrix on the common denominator is reduced, which means that the smallest possible positive denominator is used.

As an example, the representation of the lattice generated by $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ as HNF on the lowest positive denominator is

$$\left(\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} /2 \right)$$

In the remainder of the text, we assume that all lattices are in this normal form. A lattice in this form is written as M/d for M the integer square matrix in HNF and d the denominator.

Orders

An order is a lattice which is also a subring of our algebra. Therefore, it is represented as a lattice.

Ideals

As stated above, ideals are lattices which are ideals of an order of $\mathcal{A}$. We make the choice of representing them as left ideals. This is coherent with isogenies and endomorphisms, which usually are noted together with their domain.

An ideal is represented by a pair (L, O) where L is the ideal seen as a lattice and O its left order. This notation (L, O) is kept during the rest of this text.

All considered ideals are ideals of maximal orders. The left orders of all ideals in this text are therefore maximal. Appendix A.1 precises which of the presented algorithms and formulas remain valid for ideals in $\mathcal{A}$ whose left order is not necessarily maximal.

Chapter 5

Matrix algorithms

Since lattices are represented as matrices of integers as explained in Section 4.2, some algorithms for these are needed.

We first introduce the Hermite Normal Form used in our lattice representation. Then we make a few considerations on inversion and determinant computation on integer matrices. Finally, we present the LLL-reduction of a rational matrix and the guarantees it gives.

5.1 Hermite Normal Form

The Hermite Normal Form (HNF) has many slighly different definitions. There are variants on rows and on columns, and some resulting in lower or upper triangular matrices. Since we only consider matrices of maximal rank, the corresponding case of the definition used in Cohen's book [5] is given here.

Definition 5.1 *Hermite Normal Form*

A matrix M in HNF which has n rows, $m \geq n$ columns, rank n and integer coefficients respects the following conditions:

- *The first $m - n$ columns of M are zero, and the n last form an upper triangular matrix: $\forall i \leq n, \forall j < m - n + i, M_{i,j} = 0$*
- *The diagonal elements are positive: $\forall i \leq n, M_{i,m-n+i} > 0$*
- *The diagonal elements are the largest of their row, and all coefficients are positive or zero: $\forall i \leq n, \forall j > m - n + i, 0 \leq M_{i,j} < M_{i,i}$*

Every integer matrix M has a unique matrix in HNF whose columns have the same $\mathbb{Z}$-span as M. This matrix in HNF can be obtained using algorithms described for example in Section 2.4.2 of Cohen's book [5]. The computation is efficient, except that the large size of intermediary coefficients (which

grow during the execution) slows it down. In our 4-dimensional case, this problem has however less impact than the solution Cohen suggests for it, as explained in Section 9.2.

In the rest of this work, we denote HNF a function which takes a 4-row integer matrix of rank 4 and returns its Hermite Normal Form as an upper triangular 4×4 integer matrix (therefore omitting any initial zero columns). Furthermore, when mentioning a matrix in HNF, we generally assume it to be square.

5.2 Inverse

Matrix inverse computation is a very well-studied topic. However, many algorithms (such as Gaussian elimination) do involve rationals and output rational matrices. Since we mostly operate on integer matrices with a common denominator, it is preferable to have an algorithm computing the inverse of an integer matrix in this form.

Inverse and determinant of integer matrices

An expression of the inverse of a matrix M is $M^{-1} = \frac{1}{\det(M)}.\mathrm{adj}(M)$. The adjugate of an integer matrix has integer coefficients and it is computed without any operations on rationals. When implemented naively, it is not very efficient. In dimension 4, about $12 \times 16 = 192$ integer multiplications are used for the computation of the adjugate (by computing all sixteen 3×3 minors of the matrix), and a few more for the determinant. This is usable, but it can be slightly improved.

Using some optimizations described in [11] to avoid recomputations of some 2×2 minors, only about 78 integer multiplications per inversion are required. In addition, the denominator of the output is still guaranteed to be the determinant of the input matrix, which is helpful if inverse and determinant are needed.

When considering only multiplications, other algorithms might be even more efficient. A candidate is for example Banachiewicz's method [1], which also uses only integer coefficients if the determinants stay as denominators outside of the 2×2 block matrices it operates on. We have not studied this yet, nor tried it out.

Specialized methods for triangular matrices

Since most of our matrices are in HNF and therefore triangular, it is often not necessary to use general matrix inversion.

On triangular matrices, the determinant is the product of the diagonal elements. Also, all except three of the 3×3 minors are triangular or zero, which accelerates the computation of the adjugate. Taking advantage of this and avoiding recomputations of products which appear several times, the adjugate and determinant of a 4×4 integer triangular matrix are computed with only 21 multiplications of integers. Similarly, specialized algorithms for product and sum of triangular matrices are helpful for optimizing lattice operations.

5.3 LLL reduction

Another useful form of a matrix representing a lattice is the LLL-reduced form, due to Lenstra, Lenstra and Lovász [20]. This form has the advantage of containing vectors of small norm which are not too far from being orthogonal to each other.

For its definition, we need to introduce Gram-Schmidt orthogonalization (without normalization) first.

Definition 5.2 *Gram-Schmidt orthogonalization*

Let M a invertible matrix in dimension n with rational coefficients.

The Gram-Schmidt orthogonalization of M for a positive definite symmetric bilinear form s is M^* defined by induction over $0 \leq k \leq n$ as $M_k^* = M_k - \sum_{l=1}^{k} \frac{s(M_k, M_l^*)}{s(M_l^*, M_l^*)} M_l^*$ where the empty sum is read as the zero column.

Since this only performs orthogonalization, a matrix with integer or rational coefficients has a Gram-Schmidt orthogonalization with rational coefficients (assuming that s takes values in $\mathbb{Q}$).

Definition 5.3 *LLL-reduced matrix*

For a invertible matrix M in dimension n with rational coefficients, being LLL-reduced with respect to a parameter $\gamma \in]\frac{1}{4}, 1[$ and a symmetric positive definite bilinear form b of associated quadratic form N_b (such that $\forall x \in \mathbb{Q}^n$, $N_b(x) = b(x, x)$) means that:

- *$\forall k \leq n$ and $\forall l < k$, $\left| \frac{b(M_k, M_l^*)}{b(M_l^*, M_l^*)} \right| \leq \frac{1}{2}$*
- *$\forall k$ such that $1 < k \leq n$, $N_b(M_k^* - \frac{b(M_k, M_{k-1}^*)}{b(M_{k-1}^*, M_{k-1}^*)} M_{k-1}^*) \geq \gamma N_b(M_{k-1}^*)$*

Where M^* denotes the Gram-Schmidt orthogonalization of M with respect to b.

It is possible to compute a LLL-reduced matrix (for some γ in $]\frac{1}{4}, 1[$ and bilinear form b) of same $\mathbb{Z}$-span for any invertible square rational matrix by using the Lenstra-Lenstra-Lovász algorithm (LLL) with parameter γ and

bilinear form b (denoted by $\mathsf{LLL}_{\gamma,b}$). Since γ only has a small impact on performance and quality of the result, we may not always specify it. In this case any value in $]\frac{1}{4}, 1[$ can be used. This algorithm is given in many variants in Section 2.6 of Cohen's book [5]. If performance is not an issue, the integer version is sufficient for our purpose.

Even if both output a basis for a given lattice which has a special form, LLL-reduction is in many aspects very different from HNF computation. For example, even for an input matrix with integer coefficients, the output of LLL is likely to contain fractions. Furthermore, there are possibly several LLL-reduced matrices which have the same $\mathbb{Z}$-span, so there is no unicity of an LLL-reduced form. Therefore LLL is not used for normalization, but for obtaining a basis containing short vectors.

Proposition 5.4 *The vector M_1 of an $n \times n$ matrix M which is LLL-reduced for a quadratic form N_b and $\gamma = \frac{3}{4}$ is such that for v a shortest vector in the lattice generated by M, $N_b(M_1) \leq 2^{n-1} N_b(v)$.*

Exactly computing a shortest vector is usually done by applying LLL or similar reductions and then enumerating vectors shorter than its output [25]. Since enumeration is slow and LLL often gives a shortest vector directly, we sometimes use LLL to efficiently obtain a short vector in a lattice, for example in the isomorphism computation in Section 7.4.

Other reduction algorithms for obtaining short bases of lattices exist. Some of them even guarantee a shortest basis in dimension 4, as shown in [28]. We did not implement these nor study them in detail so far.

Chapter 6

Lattice operations

The functions computing addition, multiplication, intersection, equality, membership, dual, index and colon lattice of lattices are introduced in this section. Most of them are stated as formulas, which rely on algorithms and formulas explained before them, including those on matrices described in Section 5.

As described in Section 4.2, lattices are represented by a basis. More precisely, the representation of a lattice is an integer matrix and a denominator, which is the common denominator of all elements in the matrix. Furthermore, all lattices input to or output by the functions below are in a normalized form, which is the reduced fraction of the Hermite Normal Form (HNF) of the integer matrix and a positive denominator. A lattice in this form is often written M/d where M is the integer matrix under HNF and d the positive denominator.

The main references for this section are the algorithms from Cohen's book [5] and Micciancio's course notes [25].

6.1 Normalization, equality, membership, coordinates and primitive part

Checking for containment of an algebra element in a lattice and equality between two lattices are problems which benefit from our normalized lattice representation. For lattice elements, the computation of the primitive part and of coordinates in the lattice basis are also simplified by it.

Normalization

As described in Section 4.2 we use the reduced fraction of an integer matrix in HNF on a common positive denominator as normal representation of a

lattice. So given an integer $4 \times n$ matrix M, which, together with a denominator d, represents $n \geq 4$ algebra elements generating a lattice, the normal form of the lattice is obtained as

$$\mathsf{normalize}(M, d) := \left(\frac{\mathsf{HNF}(M)}{(\gcd(\mathsf{HNF}(M), d)} \Big/ \frac{|d|}{\gcd(\mathsf{HNF}(M), d)} \right)$$

In the following, normalization takes as input any kind of representation of a generating set of a lattice, and the decomposition into an integer matrix and a common denominator is sometimes left to the reader.

Equality

Two lattices are equal if and only if their normal forms are equal.

$$(M_1/d_1 =_{Lat} M_2/d_2) := ((M_1 = M_2) \text{ and } (d_1 = d_2))$$

Membership and coordinates

An algebra element is contained in a lattice if and only if its coordinates in any basis of this lattice are integers.

The coordinates of $a \in \mathcal{A}$ in a lattice M/d are given by a solution x of the linear system $Mx = d.a$. Since our input lattice M/d has a basis M which is in HNF and therefore a triangular matrix, the computation is simple. Pseudocode for it is given in Algorithm 1. The computation is stopped immediately if a non-integer coordinate is found.

By its success or failure to provide integer coordinates, this algorithm also reveals whether an element is in the lattice.

$$\mathsf{memberOfLattice}(a, M/d) := \begin{cases} \text{if } \mathsf{coordinatesInLattice}(a, M/d) \text{ fails:} & \text{False} \\ \text{else :} & \text{True} \end{cases}$$

Primitive elements and parts

An element α of a lattice L is primitive if $\forall n \in \mathbb{Z}$, $\frac{1}{n}\alpha \notin L$. This is equivalent to its coordinates in a lattice basis having a greatest common divisor (gcd) of 1, which gives a primitivity test.

$$\mathsf{isPrimitive}(\alpha, M/d) := (\gcd(\mathsf{coordinatesInLattice}(\alpha, M/d)) = 1)$$

Algorithm 1 coordinatesInLattice$(a, M/d)$

Require: M/d lattice (so M is an upper triangular integer matrix), a algebra element
Ensure: output x such that $Mx = d.a$ or FAIL

```
y = d.a
if y ∉ Z^4 then
    FAIL
else
    for k from 4 to 1 do
        z_k = y_k − Σ_{l=k+1}^{4} M_{k,l} x_l
        if M_{k,k} divides z_k then
            x_k = z_k / M_{k,k}
        else
            FAIL
        end if
    end for
end if
return(x)
```

Any lattice element α can be written as $n\beta$ for a unique $n \in \mathbb{N} \setminus \{0\}$ and β primitive. This β is called the primitive part of α, and computed by using that $\gcd(B^{-1}\alpha)\beta = \alpha$ for B a basis of the lattice.

$$\mathsf{primitivePart}(\alpha, (M/d)) := \frac{1}{d}.M\frac{\mathsf{coordinatesInLattice}(\alpha, (M/d))}{\gcd(\mathsf{coordinatesInLattice}(\alpha, (M/d)))}$$

6.2 Addition and multiplication

Representations of the sum and product of lattices are obtained by reducing larger sets of generators to a basis via the normal form computation.

Addition

The sum of two lattices L_1, L_2 is the set $\{a + b \mid a \in L_1, b \in L_2\}$. Since all lattices contain zero, it is equal to the lattice generated by the union $L_1 \cup L_2$, and sometimes called the union of L_1 and L_2. As a consequence, taking the concatenation of bases of both input lattices directly gives a set of generators of the resulting lattice, which only has to be normalized in order to obtain a lattice representation.

$$(M_1/d_1) +_{Lat} (M_2/d_2) := \mathsf{normalize}((d_2.M_1)||(d_1.M_2), d_1d_2)$$

Multiplication with an algebra element

The right multiplication of a lattice L by an algebra element a is defined as the lattice generated by $\{ba \mid b \in L\}$. By bilinearity of the product, this is equal to the lattice generated by $\{ba \mid b \in B\}$ where B is a basis of L. Normalization is then applied to the resulting lattice basis in order to obtain a representation.

In the formula below, M_k is the algebra element given by the k^{th} column of M. The algebra elements resulting from products $M_k a$ are then read as matrices with four rows and one column. The input to normalization is therefore a 4×4 matrix with rational coefficients.

$$(M/d)a := \mathsf{normalize}(\frac{1}{d}.(M_1 a || M_2 a || M_3 V_a || M_4 a))$$

The left multiplication of a lattice L by an element a is defined as $\{ab \mid b \in L\}$ and computed similarly.

Multiplication of two lattices

The product of two lattices L_1 and L_2 is the lattice generated by the set $\{ab \mid a \in L_1,\, b \in L_2\}$. Since the product is bilinear, it is sufficient to compute $\{ab \mid a \in B_1, b \in B_2\}$ for B_1 and B_2 bases of L_1 and L_2 respectively. There are essentially two ways to compute and normalize this union.

If the concatenation of two lattices L_1 and L_2 (including the adjustment of denominators, as explicitly described in the addition formula) is written $L_1||L_2$, and l_1, l_2, l_3, l_4 denote the algebra elements given by the representation of L_2, the product is equal to the lattice generated by $L_1 l_1 || L_1 l_2 || L_1 l_3 || L_1 l_4$.

$$L_1 L_2 := \mathsf{normalize}(L_1 l_1 || L_1 l_2 || L_1 l_3 || L_1 l_4)$$

Since the set of generators defining the lattice product of L_1 and L_2 is identical to the union of the sets $\{ab \mid a \in L_1\}$ for all algebra elements b in L_2, successive additions of two lattices can be used instead of the concatenation above.

$$L_1 L_2 = L_1(l_1, l_2, l_3, l_4) := (L_1 l_1 +_{Lat} L_1 l_2) +_{Lat} (L_1 l_3 +_{Lat} L_1 l_4)$$

The first variant computes only one normalization on a family of 16 elements, while the second one computes three normalizations on 8 elements. Experimental comparison as described in 9.2 shows that the first version is more efficient. The lattice multiplication operation by any of the above methods is also denoted by $\times_{Lat}$.

6.3 Sublattice and index

When working with orders and left ideals of these orders, lattices are often included in one another. Therefore, inclusion tests and index computations are useful.

Sublattice

Given two lattices L_1 and L_2, inclusion of L_1 in L_2 is tested by verifying that their union is equal to L_2.

$$\mathsf{sublattice}(L_1, L_2) := (L_2 =_{Lat} (L_1 +_{Lat} L_2))$$

Index

Given a lattice L_2 and a sublattice L_1 of L_2, the index $[L_2 : L_1]$ of L_1 in L_2 is the ratio of their volumes, which are absolute values of the determinants of their bases. This is independent of the choice of the basis elements for both lattices. Under HNF the determinant is always positive, which makes taking absolute values useless.

$$\mathsf{index}((M_1/d_1), (M_2/d_2)) := \frac{d_2^4 \det(M_1)}{d_1^4 \det(M_2)}$$

The determinants are simply the product of diagonal elements, since the matrices in our lattice representations are in HNF and therefore triangular.

6.4 Dual, intersection and colon lattice

The computation of dual lattices allows us to obtain the intersection of two lattices and their colon lattices by simple formulas.

Dual

The dual of a lattice L is defined with respect to a symmetric nondegenerate bilinear form s as the set $L_s^\# = \{x \in \mathcal{A} \mid \forall y \in L, s(x, y) \in \mathbb{Z}\}$. This set is a lattice.

The gram matrix of a bilinear form s in a basis (b_1, b_2, b_3, b_4) is the matrix G such that $G_{l,k} = s(b_l, b_k)$. Following Micciancio's course notes [25], the gram matrix G_s of the bilinear form s in the standard basis $(1, i, j, ij)$ of $\mathcal{A}$ allows us to compute the dual of a lattice with respect to s using the formula $L_s^\# = (B^T G_s)^{-1}$ where B is a basis of the lattice L.

$$\mathsf{dual}_s(M/d) := \mathsf{normalize}(d.\mathsf{adj}(M^T G_s), \det(M^T G_s))$$

There are two symmetric nondegenerate bilinear forms which are particularly interesting. The first one is the usual dot product of $\mathbb{Q}^4$ whose gram matrix is the identity. We denote the dual for it $\mathsf{dual}_{\mathrm{Id}}$. It is used for all dual computations where the bilinear form does not matter, and could be replaced by any other symmetric nondegenerate bilinear form.

The second one is defined as the reduced trace of the product of the elements in the algebra. Its gram matrix in the basis $(1, i, j, ij)$ of $\mathcal{A}$ is the diagonal matrix given by $(2, -2, -2p, -2p)$ for p the prime defining the algebra. The dual computation for this bilinear form is denoted by $\mathsf{dual}_{\mathrm{tr}}$.

Intersection

The intersection of two lattices is their intersection as sets, which also is a lattice. It is computed using dual lattices, as explained by Micciancio [25].

$$L_1 \cap_{Lat} L_2 := \mathsf{dual}_{\mathrm{Id}}(\mathsf{dual}_{\mathrm{Id}}(L_1) +_{Lat} \mathsf{dual}_{\mathrm{Id}}(L_2))$$

The intersection does not depend on the algebra, so that the dual for any symmetric nondegenerate bilinear form can be used. The most efficient choice is therefore the usual scalar product of $\mathbb{Q}^4$ whose Gram matrix is the identity, since this removes a matrix multiplication in the dual computation. It is also possible, but less efficient, to use any other dual.

Colon lattice

The right colon lattice $(L_2 : L_1)_{\mathrm{R}}$ from a lattice L_1 to a lattice L_2 is defined as $(L_2 : L_1)_{\mathrm{R}} = \{x \in \mathcal{A} \mid L_1 x \subset L_2\}$. As proven in the chapter on duality in Voight's book on quaternion algebras [35] it is equal to the dual of the product of the dual of L_2 by L_1, where the dual must be taken with respect to the symmetric nondegenerate bilinear form given by $\langle a \mid b \rangle = \mathrm{tr}(ab)$ for a and b in $\mathcal{A}$.

$$\mathsf{colon}_{\mathsf{R}}(L_1, L_2) := \mathsf{dual}_{\mathrm{tr}}((\mathsf{dual}_{\mathrm{tr}}(L_2)L_1))$$

$\mathsf{colon}_{\mathsf{R}}(L_1, L_2)$ computes the right colon lattice from L_1 to L_2 which is denoted by $(L_2 : L_1)_{\mathrm{R}}$. If needed, the left colon lattice defined by $(L_2 : L_1)_{\mathrm{L}} = \{x \in \mathcal{A} \mid xL_1 \subset L_2\}$ is computed similarly.

This method for computing colon lattices is different from the one described in SQIsign's specification [4]. It is more readable and takes less time according to our benchmarks. An even faster method specifically for lattices which are ideals of the same maximal order is presented in Section 7.4.

Chapter 7

Ideal operations

This Chapter covers the creation of ideals from algebra elements, the obtention of their norm and an additional generator, and the computation of addition, multiplication and intersection of ideals. Algorithms for getting right orders of ideals, ideal isomorphisms and connecting ideals between maximal orders are also explained.

As mentioned in Section 4.2, all ideals in this text are ideals of maximal orders and represented by two lattices. The first one describes the ideal as a lattice L. The second one is its left order O. This representation is written (L, O).

Since ideals and orders are lattices, most algorithms on ideals rely on those on lattices from Chapter 6. The ideas and formulas used in this chapter are mostly taken from the code of the original C implementation of SQIsign [14] or from formulas in Voight's book [35].

7.1 Ideal creation

For a fixed order, only few lattices are ideals. Therefore, a first problem is to find left ideals of a given order containing some specific order element or having a certain norm.

Principal ideal

A principal ideal I of left order O and generator $\alpha \in O$ is defined as $I = O\alpha$. Once it is verified that $\alpha \in O$, the principal ideal of O generated by α is obtained by multiplying the lattice O with the algebra element α.

$$\mathsf{principalIdeal}(O, \alpha) := \begin{cases} \text{if } \mathsf{memberOfLattice}(\alpha, O): & (O\alpha, O) \\ \text{else :} & \mathsf{FAIL} \end{cases}$$

Ideal from order element and integer

The ideal I of left order O generated by $n \in \mathbb{Z}$ and an element $\alpha \in O$ is $I = On + O\alpha$. It is the sum of the principal ideals generated by n and α. Using the ideal addition described in Section 7.3, the resulting ideal is computed as a sum. To create a cyclic ideal, a primitive element α of O must be used.

$$\mathsf{createIdeal}(O, n, \alpha) := \mathsf{principalIdeal}(n, O) +_{Id} \mathsf{principalIdeal}(\alpha, O)$$

A more efficient method for obtaining the principal ideal On from O and n is to multiply the matrix of O by $|n|$ and divide denominator and matrix by their greatest common divisor to recover a standard form.

If α is primitive, the norm of the resulting cyclic ideal is the greatest common divisor of n and $N(\beta)$, where β is the primitive part of α.

7.2 Get norm and generator

In the preceding paragraph, a left O-ideal was created as $I = On + O\alpha$ for an integer n and a element $\alpha \in O$. All ideals in $\mathcal{A}$ can be written as $I = ON(I) + O\alpha$ for an element α of their left order O. Therefore, the computation of $N(I)$ and a generator α is useful.

Norm

The norm of an ideal of a maximal order is the greatest common divisor of the norms of its elements. It is computed as the square root of its index as a sublattice of its left order.

$$\mathsf{norm}_{Id}((L, O)) := \sqrt{\mathsf{index}(L, O)}$$

Generator

Given I a cyclic left O-ideal, we search a primitive $\alpha \in O$ such that $I = ON(I) + O\alpha$.

There are two methods for finding such an element. The first is to randomly draw elements of I until finding a primitive generator and the second and deterministic one replaces the random selection in the first method by an enumeration. In both cases, the test $\gcd(N(I)^2, N(\alpha)) = N(I)$ is used to verify that α is a generator of I.

The deterministic method is described in Algorithm 2, where Enumerate(L) means to enumerate elements of the lattice L. In practice, enumeration ordered by norm ℓ_1 works well.

Algorithm 2 generator$((L, O))$

Require: (L, O) cyclic ideal
Ensure: FAIL or output α such that α is primitive in O and $I = N(I)O + O\alpha$

```
for α = Enumerate(L) do
  if isPrimitive(α, O) then
    if N(α) ∈ ℤ then
      if gcd(N((L,O))^2, N(α)) = N((L,O)) then
        return(α)
      end if
    end if
  end if
end for
FAIL
```

For non-cyclic ideals, a generator is also searched by enumerating until finding one. However, a primitive generator does not exist in this case.

7.3 Equals, add, multiply, intersect

The equality, addition, multiplication and intersection operations for ideals are exactly the same as their lattice equivalents, except that most of them require an additional check for compatibility or equality of the left orders. If the inputs are not compatible, no result is given and the computation fails.

Equality of two ideals

Two ideals are equal if and only if they are equal as lattices. This implies that they have the same left order.

$$((L_1, O_1) =_{Id} (L_2, O_2)) := (L_1 =_{Lat} L_2)$$

Addition

Addition of ideals is exactly lattice addition if the two ideals have the same left order. The left order of the sum is then the left order of both of the input ideals. Otherwise, the sum is not defined.

$$(L_1, O_1) +_{Id} (L_2, O_2) := \begin{cases} \text{if } (O_1 =_{Lat} O_2): & ((L_1 +_{Lat} L_2), O_1) \\ \text{else :} & \text{FAIL} \end{cases}$$

Multiplication with an algebra element

Right multiplication of an ideal I with an algebra element α can always be computed, but its output is not always an integral ideal. This is tested by verifying if the product of the norms of I and α is an integer.

$$(L,O)\alpha := \begin{cases} \text{if } N(\alpha)\mathsf{norm}_{Id}((L,O)) \in \mathbb{Z}: & (L\alpha,O) \\ \text{else :} & \mathsf{FAIL} \end{cases}$$

Multiplication of two ideals

The multiplication IJ of two ideals I and J (in this order) is defined if and only if the left ideal of J equals the right ideal of I. This is analogous to isogeny composition, which is only possible if the domain and image curves are compatible. The right and left order computations are explained in Section 7.4.

$$(L_1,O_1)(L_2,O_2) := \begin{cases} \text{if } \mathsf{order_R}((L_1,O_1)) =_{Lat} \mathsf{order_L}((L_2,O_2)): & (L_1L_2,O_1) \\ \text{else :} & \mathsf{FAIL} \end{cases}$$

Intersection of two ideals

The intersection of two ideals with the same left order is their intersection as lattices. The intersection then has as left order their common left order. The intersection of ideals of different left orders is not defined.

$$(L_1,O_1) \cap_{Id} (L_2,O_2) := \begin{cases} \text{if } (O_1 =_{Lat} O_2): & ((L_1 \cap_{Lat} L_2),O_1) \\ \text{else :} & \mathsf{FAIL} \end{cases}$$

7.4 Isomorphism, right order, connecting ideal

Besides the preceding operations, a few functions which link ideals and orders are needed. These are the computation of the right order for an ideal, of an isomorphism between two ideals and of a connecting ideal for two given maximal orders. In order to obtain some of these more efficiently, a specialized version of the colon lattice computation for two left ideals of the same maximal order is given first.

Colon lattice for ideals

Since ideals of maximal orders have an easily computable inverse, the right colon lattice $(I : J)_{\mathrm{R}} = \{b \in \mathcal{A} \mid Jb \subset I\}$ of two left ideals I and J of the

same maximal order O can be obtained as $J^{-1}I$. This is more efficient than the general colon lattice computation from Section 6.4, because it avoids the dual computation and uses less normalizations. More precisely, the inverse computation requires to compute the conjugate of a lattice. We denote by M_1, M_2, M_3, M_4 the quaternion algebra elements corresponding to the columns of the integer matrix M of the lattice representation.

$$\mathsf{conj}((M/d)) := \mathsf{normalize}((\overline{M}_1, \overline{M}_2, \overline{M}_3, \overline{M}_4)/d)$$

The colon lattice is then computed using the formula $(I : J)_{\mathrm{R}} = J^{-1}I$, which is given in Chapter 17.3 of Voight's book [35] and used in the Sagemath implementation of SQIsign [6] for the computation of ideal isomorphisms.

$$\mathsf{colon}_{\mathsf{R}}((L_1, O_1), (M_2/d_2, O_2)) := \begin{cases} \text{if } (O_1 =_{Lat} O_2): & (\frac{1}{\mathsf{norm}_{Id}((L_1,O_1))}\mathsf{conj}(L_1)L_2) \\ \text{else :} & \mathsf{FAIL} \end{cases}$$

The computation of the left colon lattice is simplified similarly under the same conditions.

Isomorphism

An isomorphism from an ideal I to an ideal J is represented by an invertible algebra element $\alpha \in \mathcal{A}$ such that $I\alpha = J$. We present here a function to determine such an α (if it exists) which is efficient but not proven to always succeed. Therefore, it can fail to find an isomorphism even between isomorphic ideals.

Algorithm 3 $\mathsf{isom}_{Id}((L_1, O_1), (L_2, O_2))$

Require: (L_1, O_1) and (L_2, O_2) ideals
Ensure: algebra element α is such that $(L_1, O_1)\alpha = (L_2, O_2)$ or FAIL otherwise

1: $((M/d) = \mathsf{colon}_{\mathsf{R}}((L_1, O_1), (L_2, O_2))$
2: $R = \frac{M}{d}$
3: $L = \mathsf{LLL}_N(R)$
4: $\alpha = L_1$
5: **if** $N((L_1, O_1)\alpha) = N((L_2, O_2))$ **then**
6: return(α)
7: **else**
8: FAIL
9: **end if**

For this we compute the right colon lattice from I to J and LLL-reduce it with respect to the reduced norm N, in order to find a short vector α in the colon lattice. We then hope that α is so short that $I\alpha$ is not just included in J (this is the case for all colon lattice elements), but equal to J. The result is Algorithm 3.

If efficiency is not important, the lattice version of $\mathsf{colon_R}$ can be used instead of the one specialized on ideals of maximal orders. Furthermore, other lattice reduction algorithms could be used instead of LLL.

Right order

The right order of an ideal I is the set $\{\alpha \in \mathcal{A} \mid I\alpha \subset I\}$ which is an order of $\mathcal{A}$ and therefore a lattice. Given that the definition of the right colon lattice $(L_2 : L_1)_\mathrm{R}$ from a lattices L_1 to a lattice L_2 is $\{\alpha \in \mathcal{A} \mid L_1\alpha \subset L_2\}$, the right order of (L, O) is exactly the right colon lattice of L and L.

$$\mathsf{order_R}((L,O)) := \mathsf{colon_R}((L,O),(L,O))$$

The left order of a given ideal is computed similarly using the left colon lattice, and we denote the corresponding function by $\mathsf{order_L}$.

The general lattice version of the colon lattice computation can be used instead of the ideal specific one, even though this is slower according to our experiments presented in Section 9.2.

$$\mathsf{order_R}((L,O)) := \mathsf{colon_R}(L,L)$$

Connecting ideal

An ideal connecting two maximal orders O_L and O_R is an integral ideal I such that $O_1 = \mathcal{O}_\mathrm{L}(I)$ and $O_2 = \mathcal{O}_\mathrm{R}(I)$. This is not unique.

Since for all $n \in \mathbb{Q}$, $O_1 n O_1 O_2 \subset nO_1O_2$ and $nO_1O_2O_2 \subset nO_1O_2$, nO_1O_2 is a connecting ideal of O_1 and O_2 as soon as it is a (non-zero) integral ideal.

$$\mathsf{connecting}(O_1,O_2) := (\mathsf{getScalar}(O1,O2), O_1O_2, O_1)$$

An example of a scalar achieving this is the integer $[O_1 : O_1 \cap O_2]$. It has the additional advantage of making the resulting ideal cyclic (as explained by Antonin Leroux in a message exchange) and is used in the implementation of [9].

$$\mathsf{getScalar}(O_1,O_2) := \mathsf{index}(O_1 \cap_{Lat} O_2, O_1)$$

Another one, without this property and which is possibly larger, but easier to compute, is $(d_1 d_2)^4$ where d_1 and d_2 are the denominators of O_1 and O_2 respectively. The denominator d of the lattice $(M/d) = O_1 O_2$ works also and directly given by our normalized lattice representation.

$$\mathsf{getScalar}((M_1/d_1),(M_2/d_2)) := d_1^4 d_2^4$$

$$\mathsf{getScalar}(O_1, O_2) := \mathsf{denominator}(O_1 O_2)$$

For efficiency, it is better to multiply the the matrix of $O_1 O_2$ by the positive scalar and to simplify the fraction of the resulting matrix and the denominator of $O_1 O_2$. This gives directly a normalized lattice representation, and is faster than using the scalar as an algebra element in a lattice-element multiplication. If the scalar is computed as denominator of $O_1 O_2$, simply setting the denominator to 1 is even more efficient.

Chapter 8

KLPT and its subfunctions

Even though the KLPT algorithms in SQIsign are slightly outside of the scope of this work, some of their subfunctions, presented in Sections 8.2, 8.3 and 8.4, are located within the quaternion algebra module in the NIST implementation. Furthermore, since KLPT is one of the two modules relying on the quaternion algebra operations presented in the previous chapters, a basic understanding of it gives an idea of their possible applications.

Therefore, a simplified version of KLPT is described in Section 8.1. This is followed by a more detailed presentation of the subfuctions which were left in the quaternion algera module even if they are very specific to KLPT or not strictly speaking operations on objects in $\mathcal{A}$. Descriptions of the KLPT variants used in SQIsign are given in its specification [4] and in papers on the scheme [14, 9].

KLPT and its variants

The KLPT algorithm was originally invented in 2014 by Kohel, Lauter, Petit and Tignol [21]. It operates on left ideals of the maximal order O_0 of basis $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ which contains the suborder $\mathbb{Z}[i] + jZ[i]$. In its original version, it takes as input a left O_0-ideal and outputs an equivalent left O_0-ideal whose norm is power of a fixed small prime ℓ. The algorithm solves in probabilistic polynomial time the ℓ-ideal path problem, whose isogeny equivalent through the Deuring correspondence is assumed to be unsolvable in polynomial time.

This original algorithm has several variants, many of which were developed specifically for SQIsign [14, 9]. One element that distinguishes them is whether the norm of the resulting equivalent ideal is required to be a precise number, a divisor of a given number or a power of a given prime. Another source of diversity is whether the algorithm is supposed to work with left O_0-ideals or with left ideals of any maximal order. Finally, there is a version

specifically built for the requirements of the SQIsign signing algorithm. The specification [4] the original SQIsign paper [14] and Leroux's thesis [22] list and explain some of these versions and variants. Since we only give an introduction to this subject, we present only the case of left O_0-ideals and norms which are expected to be an exact number (even if solutions for this do not always exist). To do so, we follow the description in Leroux's thesis [22] in a special case.

8.1 The inner workings of KLPT

KLPT uses the structure of the order O_0 already presented in Section 2.4. This is an order of basis $(1, i, \frac{i+j}{2}, \frac{1+ij}{2})$ and therefore contains the suborder of basis $(1, i, j, ij)$ which is equal to $\mathbb{Z}[i] + j\mathbb{Z}[i]$.

We start by an overview on the different steps of KLPT, to get an idea why it works. Later subalgorithms for the first three steps will be explained.

Overview

Let I be a left O_0-ideal. The goal is to find an ideal J which is equivalent to I whose norm equals the target norm n. We assume that I is of prime norm (as explained below, this can always be reached), and n such that $nN(I)$ is divisible by d, d_2 such that $d > p$ and $d_2 > pN(I)^4$ and $dd_2 = nN(I)$.

KLPT then proceeds as follows:

1. Compute $\gamma \in \mathcal{O}_0$ of norm d dividing $N(I)n$

   (SolveNormEquation)

2. Find $C, D \in \mathbb{Z}$ not both zero such that $\mu_0 = j(C + iD)$ verifies $\gamma\mu_0 \in I$

   (LinearCombinations)

3. Find $\mu_1 \in O_0$ and $\lambda \in \mathbb{Z}$ such that $\mu = \lambda\mu_0 + N(I)\mu_1$ has norm $d_2 = \frac{nN(I)}{d}$

   (StrongApproximation)

4. Set $\alpha = \gamma\mu$ (this gives $\alpha \in I$ and $N(\alpha) = nN(I)$)
5. Output $J = \frac{1}{N(I)} I\overline{\alpha}$

First verify that α is in I. We have $\alpha = \gamma\mu = \lambda\gamma\mu_0 + \gamma\mu_1 N(I)$ through step 4. and 3. (by using the commutativity of scalars with algebra elements). In this expression, $\gamma\mu_1 N(I)$ is in I since $N(I)$ is in I and γ and μ_1 are in the order O_0 of which I is a left ideal. Also, $\gamma\mu_0$ is in I thanks to step 2. So their integer linear combination $\alpha = \lambda\gamma\mu_0 + \gamma N(I)\mu_1$ is also in I.

Furthermore, $\alpha = \gamma\mu$ has norm $nN(I)$, since γ has norm d (step 1) and μ has norm $\frac{N(I)n}{d}$ (step 3.), and the algebra norm is multiplicative.

Since $I\overline{\alpha} \subset I\overline{I}$ and $I\overline{I} = N(I)O_0$ the resulting J verifies $J = \frac{1}{N(I)}I\overline{\alpha} \subset O_0$ and is therefore a left O_0-ideal. Its norm is $N(J) = N(\frac{I\overline{\alpha}}{N(I)}) = \frac{N(I\overline{\alpha})}{N(I)^2} = \frac{N(I)N(\overline{\alpha})}{N(I)^2} = \frac{N(I\overline{\alpha})}{N(I)^2} = \frac{N(I)nN(I)}{N(I)^2} = n$. Therefore the right multiplication by $\frac{1}{N(I)}\overline{\alpha}$ is an isomorphism of ideals and J is a left O_0-ideal equivalent to I.

A small complication not shown in the above description is that $\frac{d_2}{p(C^2+D^2)}$ must be a square modulo $N(I)$ in order to solve step 3. If this is not the case, the algorithm can be restarted at step 1., which is randomized and might allow to overcome the issue. However, it is not guaranteed the algorithm as described here will find a solution. For some other variants and restrictions of inputs it is possible to justify that KLPT succeeds often enough.

Ideal with prime norm

Many steps of KLPT use the norm of the input ideal I and are simpler if it is prime. Otherwise, an equivalent ideal of prime norm can be obtained by choosing random elements in I until finding some $\alpha \in I$ such that $\frac{N(\alpha)}{N(I)}$ is prime, and then using $J = \frac{1}{N(I)}I\overline{\alpha}$ instead of I in the KLPT algorithm. Since I and J are equivalent, any resulting ideal which is equivalent to J is also equivalent to I, so this substitution does not cause any problems, as long as the norm of the J is small enough to still run KLPT for n (which requires that n has a factorization d, d_2 such that $d_2 > pN(J)^4$ and $d > p$). In the description of the following steps, we therefore assume that $N(I)$ is prime.

Solving norm equations

Any integer solution of the equation $x_1^2 + x_i^2 + px_j^2 + px_{ij}^2 = d$ of unknowns x_1, x_i, x_j, x_{ij} yields an element of O_0 with norm $d > p$. Sadly, solving multivariate quadratic equations over the integers is in general a hard problem.

However, for the special form $x^2 + y^2 = a$ (of unknowns x, y) an efficient algorithm due to Cornacchia finds in some cases a solution. The existence of a solution and the efficiency of its computation depend on the value of the parameter a. This algorithm is explained in Section 8.2.

The key idea of the algorithm used to find an element of O_0 of norm n is therefore to randomly sample two of the coefficients and then use Cornacchia's algorithm to obtain the remaining ones, and to repeat such attempts until succeeding.

More precisely, x_j is randomly chosen between the two square roots of $\frac{d}{p}$ and x_{ij} between the square roots of $\frac{d-x_j^2}{p}$. Then $d' = d - p(x_j^2 + x_{ij}^2)$ is computed,

and since x_j and x_{ij} were chosen in appropriate intervals, if $d > p$, $d' \geq 0$. Then Cornacchia's algorithm is used to solve $x_1^2 + x_i^2 = d'$. If this succeeds, the problem is solved, and otherwise the algorithm restarts at the choice of x_j and x_{ij}. This algorithm is not guaranteed to find a solution for every $d > p$, so it might fail, in which case another large enough d can be tried. Pseudocode for this algorithms is given in Algorithm 4.

Algorithm 4 SolveNormEquation(d)

Require: d integer larger than p
Ensure: $x \in O_0$ such that $N(x) = d$
$m = 4d$
while not found **do**
  $b_j = \sqrt{\frac{m}{p}}$
  x_j randomly sampled integer in $[-b_j, b_j]$
  $b_{ij} = \sqrt{\frac{m - x_1^2}{p}}$
  x_{ij} randomly sampled integer in $[-b_{ij}, b_{ij}]$
  $m' = m - p(x_j^2 + x_{ij}^2)$
  if CornacchiaExtended(m') does not fail **then**
    $x_1, x_2 =$ CornacchiaExtended(m')
    $x = x_1 + x_i i + x_j j + x_{ij} ij$
    return(x)
  end if
end while

Linear combinations

Now that we have $\gamma \in O_0$ and a left O_0-ideal I, the next step is to find $C, D \in \mathbb{Z}$ such that $\gamma j(C + iD) \in I$.

This means to find a linear combination of γj,γij and a basis of I which equals zero. If a basis of I which contains $N(I)$ is used, it is enough to find a linear combination of the three other basis vectors, γj and γ which is zero modulo $N(I)$. In addition, it is required that at least one of the coefficients C and D of γj and γij in the linear combination is invertible modulo $N(I)$, which excludes trivial solutions.

Solving modular systems can be done with the algorithms from section 8.3. Computing modulo $N(I)$ has the advantage of giving rather small values for C and D.

Strong approximation

Finally, given C, D output by the linear combination step above and $\mu_0 = j(C + Di)$ their combination, we search for $\lambda \in \mathbb{Z}$ and $\mu_1 \in O_0$ such that $\mu = \lambda\mu_0 + N(I)\mu_1$ has norm equal to $d_2 = \frac{nN(I)}{d}$. As announced above, we assume $N(I)$ to be prime. Furthermore d_2 must be such that $d_2 > pN(I)^3$ and $\frac{d_2}{p(C^2+D^2)}$ is a square modulo $N(I)$.

Again, we search for μ_1 with integer coefficients in order to guarantee that it is included in O_0. This means, we search for integer coefficients x_1, x_i, x_j, x_{ij} of μ_1 and $\lambda \in \mathbb{Z}$ such that $N(\mu) = d_2$.

Therefore, we need to solve the following equation, for integer unknowns $\lambda, x_1, x_i, x_j, x_{ij}$:

$$N(I)^2x_1^2 + N(I)^2x_i^2 + p((\lambda C + N(I)x_j)^2 + (\lambda D + N(I)x_{ij})^2) = d_2 \tag{8.1}$$

By considering Equation 8.1 modulo $N(I)$, Equation 8.2 is obtained.

$$\lambda^2 p(C^2 + D^2) \equiv d_2 \mod N(I) \tag{8.2}$$

This gives λ modulo $N(I)$, since all other elements of this equation are known. Choose one (small) such value for λ.

As next step, consider the Equation 8.1 modulo $N(I)^2$ given in Equation 8.3.

$$p\lambda^2(C^2 + D^2) + 2p\lambda N(I)(Cx_j + Dx_{ij}) \equiv d_2 \mod N(I)^2 \tag{8.3}$$

This is linear in x_j and x_{ij}. Given any solution x_j, x_{ij}, the remaining unknowns x_1, x_i of Equation 8.1 verify Equation 8.4.

$$x_1^2 + x_i^2 = \frac{d_2 - p((\lambda C + N(I)x_j)^2 + (\lambda D + N(I)x_{ij})^2)}{N(I)^2} \tag{8.4}$$

This equation is of the form to which Cornacchia's algorithm (as described in Section 8.2, more specifically the extension described in Algorithm 7) finds solutions if there are any. Therefore, we can try to apply Cornacchia to the equation resulting of a choice of x_j, x_{ij} solving Equation 8.1 modulo $N(I)^2$, until finding one where Cornacchia manages to find corresponding x_1, x_i.

The remaining question is how to choose the solutions modulo $N(I)^2$. In the initial version of the algorithm, random solutions were used [21], but

since the first paper, SQIsign used an improved version from an unpublished paper presented in [29]. The main idea of this improvement is that the solutions x_j, x_{ij} to the linear equation modulo $N(I)$ form a 2-dimensional lattice (translated by some vector s). Therefore, in order to get small solutions, vectors close to s in this lattice are enumerated. This gives smaller solutions and therefore lets us use KLPT for smaller target norms than in the initial version. The details of this enumerations are described in 8.4.

Algorithm 5 StrongApproximation(N, C, D, d_2, m)

Require: N norm of the let O_0 ideal I to which we search an equivalent, C, D as output by LinearCombinations, d_2 the desired norm of the output element, m maximal number of tries

Ensure: $\mu \in \mathcal{A}$ where $\mu\gamma \in I$ (for γ from SolveNormEquation) and $N(\mu) = d_2$

1: **if** sqrtMod$(d_2(p(C^2 + D^2))^{-1}, N)$ does not FAIL **then**
2: $\quad \lambda =$ sqrtMod$(d_2(p(C^2 + D^2))^{-1}, N)$
3: **else**
4: $\quad$ FAIL
5: **end if**
6: v a solution of 8.1 modulo N^2
7: L matrix whose columns are a basis of the 2-dimensional lattice of the solutions of Equation 8.1 modulo N^2, shifted by v
8: $f =$ False
9: Set bound B for enumeration to a reasonable value
10: run EnumerateCloseVectors$(L, 1, v, m, B)$ until the call CornacchiaExtended$(\frac{d_2 - p((\lambda C+Nx)^2+(\lambda D+Ny)^2)}{N^2})$ does not fail for an enumerated vector x, y
11: **if** previous step fails **then**
12: $\quad$ FAIL
13: **end if**
14: x_j, x_{ij} the values returned by the enumeration
15: $x_1, x_i =$ CornacchiaExtended$(\frac{d_2 - p((\lambda C+Nx_j)^2+(\lambda D+N(I)x_{ij})^2)}{N^2})$
16: $\mu = x_1 + x_i i + x_j j + x_{ij} ij + j(C + Di)$
17: return(μ)

8.2 Cornacchia's algorithm

Cornacchia's algorithm solves equations of integer unknowns x and y of the form $x^2 + ny^2 = m$, with n and m coprime integers. These equations do not always have a solution, but if solutions exist, the algorithm finds one, and if it fails, there certainly is no solution.

Cornacchia for primes

In order to compute a solution efficiently, the algorithm requires a square root modulo m of $-n$. This is in general hard to compute (it is assumed to be equivalent to factoring m), except if m is prime. An efficient algorithm for computing a square root modulo a prime is explained for example in Cohen's book [5]. In practice, Cornacchia's algorithm is therefore only used on equations of the form $x^2 + ny^2 = p$ with p prime and n not multiple of p. Pseudocode for Cornacchia's algorithm (following the description from [27]) is given in Algorithm 6, using a function $\mathsf{sqrtMod}(x, d)$ which computes a square root of an integer x modulo another integer d and fails if this does not exist.

Algorithm 6 $\mathsf{Cornacchia}(n, m)$

Require: m, n coprime integers
Ensure: x, y integers such that $x^2 + ny^2 = m$ if this exists, FAIL otherwise

$d = \mathsf{sqrtMod}(-n, m)$
if $d =$ FAIL **then**
    FAIL
end if
$r_1, r_2 = m, d$
while $r_2^2 > m$ **do**
    $r_1, r_2 = r_2, r_1 \mod r_2$
end while
$x, y = r_2, \sqrt{\frac{m - r_2}{n}}$
if $y \notin \mathbb{Z}$ **then**
    FAIL
end if
return(x, y)

Extended Cornacchia

In cases where m is not prime but a factorization of it is known, it is still possible to use Cornacchia's algorithm for each of the prime factors of m, and if solutions for all resulting equations were found, they can be combined to a solution for a composite m. This sometimes allows to find a solution for composite m without computing modular square roots for non-prime integers.

We only explain the recombination in the case where $n = 1$. In this case, it is easy to verify if solutions exist: A solution exists if and only if all odd prime factors of m are $1 \mod 4$. If this is verified, we denote the prime factors of m by p_k with exponents e_k and an integer solution of the equation $x^2 + y^2 = p_k$

by x_k, y_k. i denotes a square root of -1 in $\mathbb{C}$ or in $\mathcal{A}$, and for $a, b \in \mathbb{Q}$, $\mathrm{Re}(a+ib) = a$ and $\mathrm{Im}(a+ib) = b$. Then the product $P = \prod_k (x_k + iy_k)^{e_k}$ is such that $\mathrm{Re}(P)^2 + \mathrm{Im}(P)^2 = m$ and $\mathrm{Re}(P)$ and $\mathrm{Im}(P)^2$ are integers. It does not matter whether the product is seen as a product in $\mathbb{C}$ or in $\mathcal{A}$, since the elements of $\mathbb{C}$ with rational real and imaginary part form a sub-algebra of $\mathcal{A}$. Pseudocode is given in Algorithm 7, where $\mathsf{factor}(x)$ outputs a list of pairs (q, e) of prime factors of x with their exponents. This decomposition is either computed using trial division or any other efficient factorization algorithm (details are given by Cohen [5] for example). If available, an already known list of prime factors is used instead.

Algorithm 7 $\mathsf{CornacchiaExtended}(m)$

Require: m integer
Ensure: x, y integers such that $x^2 + y^2 = m$ if this exists, FAIL otherwise
1: $l = \mathsf{factor}(m)$
2: **for** (q, e) in l **do**
3: **if** $q = 3 \mod 4$ **then**
4: FAIL
5: **end if**
6: **end for**
7: **for** (q, e) in l **do**
8: $a, b = \mathsf{Cornacchia}(1, q)$
9: $\pi = \pi \times (a + ib)^e$
10: **end for**
11: $x, y = \mathrm{Re}(\pi), \mathrm{Im}(\pi)$
12: return(x, y)

8.3 Modular matrix kernel computations

As part of KLPT, it is necessary to find a quaternion algebra element μ_0 of the form $j(C + Di)$ for C, D integers with at least one of them invertible modulo $N(I)$ and such that $\gamma\mu_0$ is in the ideal I for a given $\gamma \in O_0$. The equation $\gamma(Cj + Dji) = \alpha$ for some $\alpha \in I$ only needs to be solved modulo $N(I)$, since $N(I) \in I$.

There are two different approaches to this: Either C and D such that $C\gamma j + D\gamma ij$ is a linear combination of a basis of I are searched by linear algebra modulo $N(I)$, or they are searched over the integers, by adding columns with coefficients which are multiples of $N(I)$ to the system in order to take into account that solutions modulo $N(I)$ are sufficient. In both cases, C and D are obtained from a vector in the kernel of a rectangular matrix formed by γi, γji and a basis of I. Furthermore, all elements in the matrix must

be put on a common denominator, since the modular linear operations are done on integers.

We present first the approach using kernel computation modulo $N(I)$ as in the reference implementation [4], then the variant using integer linear algebra, as it is used by [6]. We did not investigate which of the two is more efficient, as we have not implemented the approch from [6] yet.

Modular kernel computation

Computing the kernel of a matrix modulo a prime is simply the computation of the kernel of a matrix over a field. Algorithms for it can be found for example in Section 2.3 of Cohen's book [5].

For a non-prime modulus the situation is slightly more complex, since there might be non-invertible coefficients. The standard method for computing a kernel in this case is the Howell Normal Form computation, as described in [32].

Let M be a matrix with more columns than rows considered modulo $n \in \mathbb{N}$. Denote by r its number of non-zero columns and if the k^{th} column is non-zero, denote by l_k the index of its first non-zero element. M is in Howell Normal Form if and only if

- Its first r columns are non-zero
- $l_1 < l_2 < ... < l_r$
- $M_{l_k,k}$ divides n for all $1 \leq k \leq r$
- For $1 \leq h < k \leq r$, $=\leq M_{l_k,h} < M_{l_k,k}$
- For any vector in the span of M having zeros at its $l_k - 1$ first coefficients for some $1 \leq k \leq r$, it is linear combination of the $m - k + 1$ last columns of M (Howell property)

This is very similar to the definition of a HNF for matrices of not necessarily full rank, except for the condition that diagonal elements must divide n, and the last condition. The Howell Normal Form is the analogue of the Hermite Normal Form for matrices modulo an integer. Each matrix A modulo n has a unique matrix M in Howell Normal Form modulo n whose columns have the same span than A. This matrix can be computed using an algorithm described in [32].

In order to compute the kernel modulo n of a matrix M, the equivalent matrix in Howell Normal Form H is computed first. It is lower triangular and respects the Howell property, so its kernel can be computed iteratively row by row.

The kernel of M can be obtained from the kernel of H by applying the inverse of a transformation matrix U such that $H = MU$ at left to the kernel of H. Then it is sufficient to take a vector from the resulting kernel where at least one of the coefficients corresponding to γj and γji is invertible modulo n to get C and D.

Using kernel computation over the integers

As done in the Sagemath implementation [6], an alternative to the modular kernel computation is to compute the kernel of a slightly larger matrix over the rationals.

In order to allow for solutions modulo $N(I)$ despite computing the kernel without modular operations, it is useful to add columns representing $N(I)\gamma$ and $N(I)\gamma i$ to the matrix before putting it on the same denominator and starting the kernel computation.

The kernel then is computed over the rationals, using again the algorithms from Section 2.3 of Cohen's book [5]. A vector in the resulting lattice which has integer, small (below $N(I)$) coordinates for the columns corresponding to γj and γji, of which at least one invertible modulo $N(I)$, is computed (using integer linear combinations of columns having non-zero coefficients on the rows corresponding to γj and γji). These two coefficients are used as values for C and D respectively.

8.4 Close vector enumeration in dimension 2

The improved variant of the StrongApproximation algorithm presented in Section 8.1 needs to enumerate vectors in a specific lattice in dimension 2 close to a given target vector. In StrongApproximation, the enumerated vectors are tested for a condition (if an equation depending on them can be solved), and enumeration is stopped as soon as an output verifies this condition. In the following we describe how to enumerate lattice vectors for this task using a short vector enumeration algorithm due to Fincke and Pohst.

Close vector enumeration in a lattice is a classical problem, even if it is less well-studied than the closely related short vector enumeration according to a survey from 2017 [24], and less frequently implemented. Algorithms solving it are mostly used for determining a closest vector, and optimized algorithms specifically adapted to higher dimensions exist, as listed in [26]. Since our lattice is in dimension two, these variants are however not expected to be particularly efficient. We therefore use a very simple algorithm due to Fincke and Pohst [16] and described in Cohen's book [5] as Algorithm 2.7.5. This is fundamentally a short vector enumeration algorithm, so that it needs some adaptation for the close vector problem in our case.

In the next steps, we describe first the exact problem we need to solve and the simple short vector enumeration algorithm from Fincke and Pohst we use for it. Then a first approach which enumerates close vectors as sums of one fixed close vector and enumerated short vectors is presented, followed by a different version which describes our close vector problem as a short vector problem in another lattice. Both of these approaches were implemented. However, their comparison is difficult, as detailed in Section 9.2.

There might be other approaches for transforming close vector problems into short vector problems using embedding techniques, but we did not find enough explanations on their use in enumeration, so we implemented the rather straightforward ideas described in the following.

Precise problem and inputs

Close vector enumeration takes as arguments a lattice basis L and the target vector t. In our case, both of them have only integer coefficients.

We did not go into this detail before, but it is technically possible to run KLPT for orders different of O_0 as long as they have a suborder of the form $\mathbb{Z}[\omega] + j\mathbb{Z}[\omega]$ where ω^2 is a small negative rational $-q$. In the case of O_0, the suborder has basis $(1, i, j, ij)$ and $\omega = i$, so $q = 1$. Changing the suborder has slightly affects all subalgorithms of KLPT, most often it only means that an additional factor q appears. In order to give an idea of the impact of this change on the enumeration algorithms, we will explicit q in the following. They therefore take an additional parameter q, whose value is 1 if KLPT is run using O_0.

The norm used for measuring closeness of lattice elements to the target vector t is the quadratic form $N_q(v) = x^2 + qy^2$ where x, y are the coordinates of a vector v. Furthermore, the Fincke-Pohst algorithm we use for enumeration needs a bound b as input. Vectors are only enumerated until the norm of their difference to t reaches B.

Short vector enumeration

Using these inputs except the target vector t, we now present the short vector enumeration algorithm due to Fincke and Pohst which we use in both of our strategies for close vector enumeration. This algorithm allows to enumerate elements of a lattice whose image by a positive quadratic form is below a bound B.

The quadratic form, which corresponds to the norm on the lattice for which short vectors should be enumerated must first be put into a special form, which is an intermediary step in the computation of a Cholesky Decomposition as described by algorithm 2.7.6 in Cohen's book [5]. The goal is to write

a quadratic form $Q(x_1, ..., x_n)$ in dimension n as a list of $f_{k,l}$ for $1 \leq k \leq l \leq n$ such that $Q(x_1, ...x_n) = \sum_{k=1}^{n} f_{k,k}(x_k + \sum_{l=k+1}^{n} f_{k,l}x_l)^2$ for all $(x_1, ..., x_n)$. Cohen only describes this algorithm for positive definite quadratic forms. For quadratic forms which are only positive (as our distance when doing close vector enumeration later), a test for zeros on the diagonal has to be added. For the details, see the Algorithm 8 below.

Algorithm 8 PrepareQuadraticForm(Q)

Require: Q symmetric matrix of dimension n representing a positive quadratic form

Ensure: A matrix f in dimension n with entries $f_{k,l}$ such that such that $Q(x_1, ...x_n) = \sum_{k=1}^{n} q_{k,k}(x_k + \sum_{l=k+1}^{n} f_{k,l}x_l)^2$ for all $(x_1, ..., x_n) \in \mathbb{Q}$

1: **for** (k, l) such that $1 \leq k \leq l \leq n$ **do**
2: $f_{k,l} := Q_{k,l}$
3: **end for**
4: **for** k such that $1 \leq k \leq n$ **do**
5: **if** $f_{l,k} \neq 0$ **then**
6: **for** l such that $k < l \leq n$ **do**
7: $f_{l,k} := f_{k,l}$
8: $f_{k,l} := \frac{f_{k,l}}{f_{k,k}}$
9: **end for**
10: **for** (g, h) such that $k < g \leq h \leq n$ **do**
11: $f_{g,h} := f_{g,h} - f_{g,k}f_{k,h}$
12: **end for**
13: **else**
14: **for** l such that $k < l \leq n$ **do**
15: $f_{k,l} := 0$
16: $f_{l,k} := 0$
17: **end for**
18: **end if**
19: **end for**
20: **return** f

Once this special form f is computed, the short vector enumeration by Fincke and Pohst [16] as described by algorithm 2.7.5 from Cohen's book [5] consists in the computation of lower bounds (used as initialization) and upper bounds on possible values for each variable, between which enumeration is done by incrementing the variables, which represent coordinates in the lattice. For each coordinate x_i, the bounds depend on the current values of all coordinates x_j with $j > i$. Since we only need short vector enumeration

in dimension two and for a quadratic form given as $N_q(v) = x^2 + qy^2$ for x, y coordinates of v, Algorithm 9 is simplified specifically for this case. In its pseudocode, the algorithm is assumed to continue execution after return statements.

Algorithm 9 EnumerateShortVectors(L, q, m, B)

Require: q positive integer defining N_q and the associated bilinear form B_q, $L = (b_0, b_1)$ a lattice, m maximal number of tries, B positive integer
Ensure: A list of vectors $v \in L$ such that $N_q(v) \leq B$

$Q := \begin{pmatrix} N_q(b_0) & B_q(b_0, b_1) \\ B_q(b_0, b_1) & N_q(b_1) \end{pmatrix}$
$f :=$ PrepareQuadraticForm(Q)
$B_y := \lfloor \sqrt{\frac{B}{f_{2,2}}} \rfloor$
$y := -B_y - 1$
while $(y < B_y)$ and $(i < m)$ **do**
  $i = i + 1$
  $y := y + 1$
  $B_x := \lfloor \sqrt{\frac{B - f_{2,2}y^2}{f_{1,1}}} - f_{1,2}y \rfloor$
  $x := -\lfloor (\sqrt{\frac{B - f_{2,2}y^2}{f_{1,1}}} + f_{1,2}y \rfloor - 1$
  while $(x < B_x)$ and $(i < m)$ **do**
    $i := i + 1$
    $x := x + 1$
    return $L \begin{pmatrix} x \\ y \end{pmatrix}$
  end while
end while

Adding short vectors to a very close vector

The first way to apply a short vector enumeration algorithm to our problem consists in computing one vector in the lattice which is very close to the target (using for example Babai's nearest plane algorithm) and then enumerating short vectors in the lattice and adding them to the close vector, in order to obtain other close vectors. In order to make enumeration more efficient, the reduced lattice basis computed for finding the inital close vector is also used for the enumeration.

However, this approach has the disadvantage that even if lattice element s and the difference d of the initial close element and the target are both below the bound for N_q, their sum $s + d$ which is the difference of the new close vector to the target, is not necessarily of small enough norm N_q. Therefore, it is necessary to test for each s whether $N_q(s + d)$ respects the bound B

before outputting s.

If short vector enumeration below a given bound is done using the Fincke-Pohst algorithm as it is described in Cohen's book [5] and given in pseudocode in Algorithm 9, the first vectors encountered during the enumeration have comparatively large size, since the coordinates are initially set to values which are as small as possible to still have an image by the quadratic form N_q which is below the bound, and then incremented. Therefore, it is possible that many short vectors s computed in the beginning of the enumeration are such that $N_q(s+d)$ is too large.

To reduce this risk, it is possible to perform short vector enumeration on a bound which is lower that the bound for closeness. However, if this enumeration bound is too low, it is possible that no short enough vector is found, even though several close enough vectors exist. As a consequence, the bound for short vector enumeration allows to trade success probability against efficiency.

Direct close vector enumeration

In order to not need this decision, it is also possible to implement close vector enumeration directly and exactly, so that only vectors v in the lattice which do respect $N_q(t-v) < B$ (for some bound B) are enumerated. As I did not find any explicit description of how to adapt Fincke-Pohst for close vector enumeration even though the existence of such an adaptation seems to be assumed by [26], I suggest the following.

The function $f(x,y) = N_q(t - Lv)$ is a positive quadratic polynomial in the coordinates x and y of v, where L a basis of the lattice, v the vector searched for and t the target. Applying preparation step PrepareQuadraticForm in dimension 4 to the positive quadratic form $N_q(\begin{pmatrix} c \\ d \end{pmatrix} - L \begin{pmatrix} a \\ b \end{pmatrix})$ of four variables a, b, c, d results in a matrix f whose lower half is zero, since this quadratic form is not positive definite.

Getting inspiration from Fincke-Pohst in dimension 4, the fixed values of the coordinates of the target are substituted for c and d in the obtained expression of the quadratic form, and bounds are computed as when running the inner two loops of a 4-dimensional Fincke-Pohst version. A slight difference to that is that the $f_{i,i}$ are zero for the last two coordinates, so that their fixed value only impact the bounds through the values of $f_{1,i}$ and $f_{2,i}$. The pseudocode given in Algorithm 10 below assumes that a return statement does not interrupt its execution.

This algorithm also benefits from a reduced basis, since that reduces the number of enumerated y for which no x forming a close enough vector of coordinates (x,y) exists. Given the large numbers involved in the scheme

Algorithm 10 EnumerateCloseVectors(L, q, t, m, B)

Require: q positive integer defining N_q and the associated bilinear form B_q, $L = (b_0, b_1)$ a lattice, t target vector, m maximal number of tries, B positive integer

Ensure: A list of vectors $v \in L$ such that $N_q(t - v) \leq B$

$Q :=$ Symmetric matrix representing the quadratic form $N_q(\begin{pmatrix} c \\ d \end{pmatrix} - L \begin{pmatrix} a \\ b \end{pmatrix})$ of variables a, b, c, d in this order
$f :=$ PrepareQuadraticForm(Q)
$r, s :=$ coordinates of t
$B_y := \lfloor \sqrt{\frac{B}{f_{2,2}}} - f_{2,3}r - f_{2,4}s \rfloor$
$y := -\lfloor \sqrt{\frac{B}{f_{2,2}}} + f_{2,3}r + f_{2,4}s \rfloor - 1$
while $(y < B_y)$ and $(i < m)$ **do**
  $i = i + 1$
  $y := y + 1$
  $B_x := \lfloor \sqrt{\frac{B - f_{2,2}(y + f_{2,3}r + f_{2,4}s)^2}{f_{1,1}}} - f_{1,2}y - f_{1,3}r - f_{1,4}s \rfloor$
  $x := -\lfloor \sqrt{\frac{B - f_{2,2}(y + f_{2,3}r + f_{2,4}s)^2}{f_{1,1}}} + f_{1,2}y + f_{1,3}r + f_{1,4}s \rfloor - 1$
  while $(x < B_x)$ and $(i < m)$ **do**
    $i := i + 1$
    $x := x + 1$
    return $L \begin{pmatrix} x \\ y \end{pmatrix}$
  end while
end while

and the small dimension of the lattice, reducing the basis L before applying the computation of the coefficients of f and enumerating strongly improves performance in sufficiently many cases to make this optimization relevant, if not even necessary, for use in SQIsign.

Chapter 9

Implementation and experiments

The algorithms described in this text except in Section 8.1 almost exactly match the functions in the quaternion algebra module of the reference implementation of SQIsign submitted to NIST in June 2023. This module is situated in the dependency graph of the implementation as depicted in Figure 1.2, and its conception and implementation were part of this work. More precisely, I wrote most of the interface and the implementation of the quaternion algebra module except the Howell Normal Form and its usage for kernel computations modulo powers of two (implemented by Luca De Feo) and LLL reduction in dimension 4 (made by Basil Hess), on which I never worked. In addition, Antonin Leroux helped with precise specifications for the implementation of the KLPT subfunctions from Section 8, and added a third variant of Cornacchia's algorithm to the module. Furthermore, several algorithms on ideals and the right colon lattice were implemented by Luca De Feo following the implementation from [9] or PARI/GP.

Since the submission, I improved some of these algorithms, such as the colon lattice, connecting ideal and lattice multiplication computations. I also implemented a second version of the close vector enumeration in dimension two. Some experiments with specialized functions for triangular matrices and HNF computation modulo the determinant of the result were also made.

In the following, the differences between the methods described in Chapters 4 and 7 and the implementation in the quaternion algebra module of the SQIsign implementation submitted to NIST are described. Finally, the different improvements made since the submission are explained and benchmarks for them are given.

9.1 Comparison to the reference implementation of SQIsign

The quaternion algebra module provides almost all functionalities described in Chapters 5 to 7 as well as Cornacchia's algorithm, close vector enumeration in dimension two and modular kernel computations for KLPT, while KLPT itself was implemented in all needed versions in a separate module. Therefore, the quaternion algebra module implements solutions to most of the problems for which formulas or algorithms are given in this text.

Since the submission, I improved and rewrote some ideal and lattice operations, for example the colon lattice and connecting ideal computations. These therefore differ largely between this text, which explains the improved variants, and the submitted code. Smaller modifications, refactoring and elimination of redundant steps have touched almost all ideal functions since the submission. Since this text mostly uses the simplest methods I found, it deviates slightly from the code at the submission date for some of them. A more precise description of these changes including their performance impact can be found in Section 9.2.

Other differences to the implementation appear when this text deliberately describes algorithms in a slightly less specialized form than the one used in the reference implementation. As an example, the implementation searches for a generator of an ideal which has a norm coprime to some given integer, while the generator search algorithm in Chapter 7 does not have this coprimality condition. The close vector enumeration algorithms are also touched by this, since their implementation takes the accepting condition from the StrongApproximation as a parameter and terminates as soon as it is satisfied on an enumerated vector.

In addition to these changes at the level of individual functions, there are more fundamental differences in the representation of lattices and ideals which are detailed below.

Lattice representation and normalization

In the reference implementation, lattices are also quadratic integer matrices with a common denominator, as in described in Section 4.2. Furthermore, the matrices most often are in HNF. However, there are some exceptions, since lattices are often not renormalized within a function. For example, the dual computations in the intersection formula or the lattice variant of the right colon lattice computation are made by an internal dual function which outputs a matrix which is not necessarily in HNF. The functions using this dual only put its output in HNF if it is a result they return, and even this normalization is done only by functions which are not internal to the module. Since most lattice operations except the dual naturally return results

in HNF, this optimization is only used on dual and conjugate or inverse computations.

Furthermore, normalization in the reference implementation means only that the integer matrix is in HNF, while in this text we also assumed that the denominator takes the smallest possible positive value, in order to obtain unicity. This has little impact, except that lattice equality test of the implementation has to reduce the denominator before comparing two lattices.

Ideal representation and norm computations

In this text, the simplest possible representation of left ideals is used, which consists in a lattice for the ideal and a lattice representing its left order. The implementation differs from it, because its ideal representation also contains the norm, which is precomputed whenever an ideal is created or returned as result of an operation. This obviously influences the efficiency of all algorithms operating on ideals, since norms of output ideals have to be computed every time, and norms of input ideals are directly available when needed (for example in the ideal variant of the colon lattice computation).

In order to compute the norm of ideals output by the algorithms in Chapter 7, a few tricks can be used. For example, the norm of a principal ideal is the norm of its generator, the norm of the sum of two principal ideals is the greatest common divisor of their norms and the norm of a product of an ideal by an algebra element is the product of their norms. Therefore, only in very few functions the computation described in the norm_{Id} formula in Section 7.2 is needed for obtaining the norm of the resulting ideal.

9.2 Benchmarks of implementation variants

Since the large majority of SQIsign's signing and key generation runtime is spent on the isogeny operations of the translation between ideals and isogenies, it is difficult to get a realistic estimation of the performance of the functions in the quaternion algebra module during the execution of SQIsign. Each of them has a runtime close to negligible and is therefore not correctly measured by profilers when running key generation or signing.

In order to compare the efficiency of different variants of these functions, we therefore run them in isolation and on randomly generated inputs. For the benchmarks in Figures A.3, A.4 and A.5, the size of these inputs was chosen to be similar to the maximal integer size observed outside of HNF computations when running SQIsign for a given security level. The quaternion algebra was fixed using one of the primes chosen as parameters in the specification [4]. This parameter choices might not be perfect. In particular, fixing the size of input integers close to the maximum of observed

inputs in the signature is not necessarily realistic, and smaller inputs should be considered also. In order to allow this kind of comparison, Figures A.1 and A.2 use smaller integer sizes within the same algebra than Figure A.3. The integer sizes have been chosen arbitrarily with no other purpose than to cover smaller integers. Furthermore, when generating inputs which are required to be in HNF, the input size only refers to coordinates of a basis before applying normalization.

Another issue with the given benchmarks is that the implementation of some optimizations required changes in other parts of the code. Additionally, while implementing them, sometimes small but completely unrelated parts of the code were also modified in order to fix mistakes or to make them more readable. These changes were rare and minor; however, this means that between two modifications even functions not impacted by the main modification might have slightly changed.

Finally, the SQIsign reference implementation and therefore also the code used for benchmarks does not use exactly the same representation of ideals and lattices as defined in Section 4.2. The detail of the differences is described in Section 9.1. These choices are likely to influence the relative performances of the tested algorithms and variants.

Despite all shortcomings, the numbers given in Figures A.1 to A.5 help to understand the efficiency comparisons in Sections 6 and 7.

All benchmarks were run on an Intel i7-10510U CPU at 1.80GHz, on a release build of the SQIsign reference implementation from [4], modified with implementations of the given functions. The tables show iterative improvements, so every line in a table also contains the modifications made in previous lines.

Below we explain the changes between the lines in the benchmarking tables given in the Appendix A.2 and a few possible reasons of the observed speed differences.

Clean multiplication

The clean multiplication step is mainly an attempt to make the code more readable by defining a function for lattice-element multiplication and making use of all existing multiplication functions where appropriate. It has no real impact on performance, and is listed here mostly as a more realistic baseline, since all improvements were built on top of that version rather than the older and less readable NIST version.

New colon lattice

The modification of the colon lattice computation from the rather complex and more generic variant of the reference implementation described in the specification [4] to the formula from Chapter 6 gives huge performance gains in the colon lattice computation and the right order depending on it.

New connecting ideal

An optimization of the connecting ideal computation, which from computing and using the scalar factor $[O_1 : O_1 \cap O_2]$ to just clearing the denominator of the result, unsurprisingly accelerates the connecting ideal computation. However, this might come at some expense, since the returned ideal has larger norm. Since the KLPT module has no benchmarking code so far, the impact of this has not been measured yet.

Specialized functions for triangular matrices

The use of specialized functions for computations on triangular matrices impacts almost all lattice functions since lattices in normal form are represented by triangular matrices. The improvements are between small and not noticeable.

Larger HNF and new lattice multiplication

The HNF computation is modified to allow for arbitrary-sized inputs (instead of 4×8 matrices as in the reference implementation), and this is immediately applied to all HNF computations. As a consequence, lattice multiplication is now done with one HNF on sixteen generators instead of three lattice additions. Multiplication gains speed due to this change, while other lattice functions are almost not impacted, and if so, then rather negatively.

Colon lattice specifically for ideals

The specialized ideal variant of the colon lattice computation which is presented in 7.4 makes the right order computation even faster. The direct benchmark of a colon lattice computation is still done on the variant for general lattices in $\mathcal{A}$ presented in 6, in order to keep the same benchmarking function so that measures comparable to the previous lines in the tables in Appendix A.2.

HNF modulo the determinant

Using computations modulo the determinant of the output lattice (as described in Algorithm 2.4.8 of Cohen's book [5]) for HNF computations,

which is a measure against the extreme growth of the intermediary coefficients in the naive implementation used previously, induces a slowdown in most lattice functions. This is most visible in the intersection, which uses two HNF computations. The only measures where it appears to have no negative effects are on colon lattice computations for very large inputs (above NIST-1 parameters), while the negative impact is largest for small input sizes.

The slowdown is surprising, since the new HNF computation is expected to be much faster than the naive one according to Cohen [5]. Its inefficiency could be due to two causes:

First, our modular operations are rather slow, since the use of the GMP library means that we have to do a separate call to a modular reduction function after every operation, and then we need some more computations on its output in order to obtain a result in the interval $]-d/2, d/2]$, as required by the algorithm from Cohen's book.

Second, our algebra has very small dimension. This might limit the coefficient growth of the naive implementation, and therefore also the possible advantage of the modular version. Experimentally, we observed a factor 26 at most between the size in bits of the largest integer coefficient of the input matrix and the largest intermediary coefficient during executions of SQIsign key generation and signing, for all three NIST levels. Since the determinant of a triangular matrix of dimension 4 is also about four times larger than its coefficients, the gain in coefficient size obtained by computing modulo the determinant is small. In our experiments, the largest integers observed using the HNF modulo determinant still were about 10 times larger the largest coefficients of the input matrix. In conclusion, the additional execution time due to modulo operations appears in our experiments to be more important than the speed gain due to smaller intermediary coefficients, except at extremely large integer sizes.

Close vector enumeration

This change does not appear in the benchmark tables in Appendix A.2, since it impacts none of the benchmarked lattice and ideal algorithms.

The two variants of close vector enumeration described in Section 8.4 were both implemented, even though the direct method was only completed after the submission. The implementation is different from the pseudocode given here, since it takes the acceptance condition as an argument and calls it whenever a result is found. This makes realistic benchmarks hard, since the condition takes some time to evaluate, which masks the possible speed differences in enumeration.

In both cases, the lattice reduction before any other operation is absolutely necessary for realistic integer sizes of a few hundreds or thousands of bits. We observed reasonable inputs where first applying a reduction allowed the direct enumeration to find a result for the test condition $x + y \equiv 2 \mod 3$ condition within 3 incrementation steps, while without the reduction over a million of y values were enumerated before a single x value could be computed.

Chapter 10

Conclusion

With the chapters on matrices, lattices, ideals and the short outlook on KLPT we covered all concepts, normal forms, formulas and algorithms needed for the quaternion algebra module of SQIsign. The precise formulations might not always match the reference implementation we submitted to NIST, but all its algorithms have at least an alternative or variant listed and explained, and a comparison to the implementation is given. When several reasonable strategies for the same problem were found, they are listed, and if more than one was implemented, also compared in terms of efficiency.

Given that only a small fraction of the key generation and signing time of SQIsign is actually spent within the quaternion algebra module, optimizing its functions does not yield great speedups, and might not be necessary. However, the rather simple and well-understood formulas and algorithms in this work provide a basis for future research on constant-time implementations of SQIsign, either via constant-time adaptations of these and the higher-level KLPT and translation algorithms, or via masking techniques. Today, no such implementation exists, even though there is research on timing attacks against the scheme [19].

Furthermore, the content of this work might also be helpful for implementations, optimizations and constant-time considerations on SQIsign variants such as SQIsignHD [8] or other cryptographic schemes using ideals in quaternion algebras.

Appendix A

Appendix

A.1 Algorithms from Section 7 and ideals of non-maximal orders

Since Section 4.2 we only considered ideals of maximal orders. These have some properties (such as invertibility) which do not always hold for ideals of non-maximal orders. Therefore, some algorithms from Section 7 do not generalize to the general case of left ideals of orders in $\mathcal{A}$, while other do. We give here a list for clarification.

For finding out which formulas or algorithms work in which other cases, it is helpful to consult Voight's book [35] and Antonin Leroux's thesis [22]. The information compiled below was also found there.

Ideal creation, norm and generator

The generation of principal ideals is the same for ideals of non-maximal orders. The algorithm primitiveToIdeal outputs a left O-ideal I when a non-maximal order O (and a generator contained in O) is used as input. The only point which is different is that an ideal I generated by principalIdeal for a non-maximal order O might also be a principal ideal for an order O' containing O. Therefore, $\mathcal{O}_{\mathrm{L}}(I)$ is not necessarily equal to O.

Ideals of non-maximal orders can also be written as $I = N(I)O + O\alpha$. Therefore, all of their ideals can be generated by the algorithm in createIdeal. Here again, the obtained ideals might be left ideals not only of the input order O, but of some larger order in which O is included.

The formula we use in norm_{Id} which states that the norm of a left O-ideal is $\sqrt{[O : I]}$, does not necessarily hold if O is not maximal.

Enumeration for finding a generator as in generator can also be used for ideals of non-maximal orders.

Ideal addition, multiplication and intersection

The given formulas for ideal addition and intersection work for left O-ideals even if O is not maximal. For them to be applied, it is sufficient that the two ideals are left ideals of the same order O, but O is not required to be their left order. The resulting ideal will be a left ideal of this order O, but again might have a larger left order.

Multiplication at right by any quaternion algebra element is still possible, as long as the resulting lattice has integer norm, which might be harder to verify than in the case of maximal orders.

Multiplying two ideals I and J where $\mathcal{O}_{\mathrm{R}}(I) = \mathcal{O}_{\mathrm{L}}(J)$ using lattice multiplication remains possible, and the result will be a left ideal of any order of which I is a left ideal.

Algorithms on ideals and orders

The colon lattice algorithm given in Section 7.4 does not generalize to ideals of non-maximal orders. However, if it is replaced by the colon lattice algorithm for lattices applied to the lattice representation of the involved ideals, the isomorphism and right and left order computations still work.

The connecting ideal computation in connecting does not generalize to non-maximal orders.

A.2 Benchmark tables

Modification	$+_{Lat}$	$\times_{Lat}$	$\cap_{Lat}$	$\mathsf{colon}_{\mathsf{R}}$	$\mathsf{order}_{\mathsf{R}}$	connecting
NIST submission	0,61	7,9	10	30	6,1	4,5
Clean multiplication	0,70	7,9	11	30	6,0	3,5
New colon lattice	0,63	8,5	10	13	3,6	3,5
New connecting ideal	0,69	8,4	10	13	3,5	3,5
Triangular matrix	0,64	8,6	11	14	3,5	1,7
Large HNF	0,69	7,4	10	14	2,9	1,5
Ideal colon lattice	0,61	7,3	11	14	1,5	1,3
HNF with modulo	0,93	9,9	19	18	2,7	2,6

Figure A.1: Benchmarks of incremental modifications. Input integer size 300 bit, prime p^I_{1223} from [4]. Median of 1000 iterations, with time in millions of cycles, rounded to the second non-zero decimal cipher or the same decimal cipher after the comma as the smallest value in the column

Modification	$+_{Lat}$	$\times_{Lat}$	$\cap_{Lat}$	$\mathsf{colon}_\mathsf{R}$	$\mathsf{order}_\mathsf{R}$	connecting
NIST submission	2,1	40	56	170	18,3	10,1
Clean multiplication	2,3	40	56	170	17,1	9,1
New colon lattice	2,1	39	66	84	9,0	12,5
New connecting ideal	2,1	40	65	67	9,4	8,9
Triangular matrix	2,2	40	57	65	9,1	3,8
Large HNF	2,1	34	57	72	6,8	3,4
Ideal colon lattice	2,2	35	56	71	3,5	3,5
HNF with modulo	2,5	41	98	75	5,9	4,7

Figure A.2: Benchmarks of incremental modifications. Input integer size 1000 bit, prime p^I_{1223} from [4]. Median of 1000 iterations, with time in millions of cycles, rounded to the second non-zero decimal cipher or the same decimal cipher after the comma as the smallest value in the column

Modification	$+_{Lat}$	$\times_{Lat}$	$\cap_{Lat}$	$\mathsf{colon}_\mathsf{R}$	$\mathsf{order}_\mathsf{R}$	connecting
NIST submission	9,2	190	290	900	90	37
Clean multiplication	9,1	190	280	880	79	38
New colon lattice	8,9	190	290	310	36	37
New connecting ideal	9,4	190	280	320	35	37
Triangular matrix	9,2	190	280	330	39	13
Large HNF	9,1	160	280	360	27	18
Ideal colon lattice	9,2	160	270	350	14	13
HNF with modulo	10,1	190	460	360	18	14

Figure A.3: Benchmarks of incremental modifications. Input integer size 3000 bit, prime p^I_{1223} from [4]. Median of 1000 iterations, with time in millions of cycles, rounded to the second non-zero decimal cipher or the same decimal cipher after the comma as the smallest value in the column

Modification	$+_{Lat}$	$\times_{Lat}$	$\cap_{Lat}$	$\mathsf{colon}_\mathsf{R}$	$\mathsf{order}_\mathsf{R}$	connecting
NIST submission	19	400	600	2100	190	82
Clean multiplication	19	400	590	1900	150	74
New colon lattice	19	390	580	730	79	79
New connecting ideal	19	390	580	690	72	73
Triangular matrix	20	400	590	690	78	24
Large HNF	20	350	590	740	55	26
Ideal colon lattice	20	350	610	750	27	23
HNF with modulo	21	380	930	740	36	37

Figure A.4: Benchmarks of incremental modifications. Input integer size 5000 bit, prime p^{III}_{5563} from [4]. Median of 1000 iterations, with time in millions of cycles, rounded to the second non-zero decimal cipher or the same decimal cipher after the comma as the smallest value in the column

Modification	$+_{Lat}$	$\times_{Lat}$	$\cap_{Lat}$	$\mathsf{colon}_\mathsf{R}$	$\mathsf{order}_\mathsf{R}$	connecting
NIST submission	32	610	940	3300	230	130
Clean multiplication	32	660	990	3500	290	120
New colon lattice	32	650	990	1100	120	130
New connecting ideal	31	630	940	1100	120	120
Triangular matrix	32	630	950	1100	130	43
Large HNF	31	540	940	1200	89	49
Ideal colon lattice	32	520	950	1200	45	45
HNF with modulo	35	600	1500	1200	55	57

Figure A.5: Benchmarks of incremental modifications. Input integer size 7000 bit, prime p^V_{40609} from [4]. Median of 1000 iterations, with time in millions of cycles, rounded to the second non-zero decimal cipher or the same decimal cipher after the comma as the smallest value in the column

Bibliography

[1] T. Banachiewicz. Zur Berechnung der Determinanten, wie auch der Inversen, und zur darauf basierten Auflösung der Systeme linearer Gleichungen. *Acta Astronomica, Série C*, pages 41–67, 1937.

[2] Daniel J Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *Open Book Series*, 4(1):39–55, 2020.

[3] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology – INDOCRYPT 2014*, pages 428–442, Cham, 2014. Springer International Publishing.

[4] 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. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures, last consulted on September 12, 2023.

[5] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, New York, NY, USA, 1993.

[6] Maria Corte-Real Santos and Giacomo Pope. Learning to SQI. https://learningtosqi.github.io/, 2022. Last accessed June 11, 2023.

[7] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Paper 2006/291, 2006. https://eprint.iacr.org/2006/291, visited on September 7, 2023.

[8] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQISignHD: New Dimensions in Cryptography. Cryptology ePrint Archive, Paper 2023/436, 2023. https://eprint.iacr.org/2023/436, visited on September 6, 2023.

[9] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New Algorithms for the Deuring Correspondence - Towards Practical and Secure SQISign Signatures. 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 659–690, 2023.

[10] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Des. Codes Cryptogr.*, 78(2):425–440, 2016.

[11] David Eberly. The Laplace expansion theorem: Computing the determinants and inverses of matrices, 2008. Available at https://www.geometrictools.com/Documentation/LaplaceExpansionTheorem.pdf, last accessed May 3, 2023.

[12] Kirsten Eisenträger, Sean Hallgren, Kristin Lauter, Travis Morrison, and Christophe Petit. Supersingular isogeny graphs and endomorphism rings: Reductions and solutions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 329–368, Cham, 2018. Springer International Publishing.

[13] 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, Paper 2023/106, 2023. https://eprint.iacr.org/2023/106, visited on July 27, 2023.

[14] 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*, pages 64–93, 2020.

[15] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[16] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985.

[17] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU, 2020. Available at https://falcon-sign.info/falcon.pdf", last accessed at September 13, 2023.

[18] Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. 33(1):130–175, January 2020.

[19] David Jacquemin, Anisha Mukherjee, Sujoy SINHA ROY, and Péter Kutas. Towards a constant-time implementation of isogeny-based signature, SQISign. Cryptology ePrint Archive, Paper 2023/807, 2023. https://eprint.iacr.org/2023/807, visited on September 6, 2023.

[20] H.W. jr. Lenstra, A.K. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[21] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion ℓ-isogeny path problem. Cryptology ePrint Archive, Report 2014/505, 2014. https://eprint.iacr.org/2014/505, visited on July 27, 2023.

[22] Antonin Leroux. Quaternion algebras and isogeny-based cryptography. Thesis available at https://www.lix.polytechnique.fr/Labo/Antonin.LEROUX/manuscrit_these.pdf, 2022. Last accessed March 23, 2023.

[23] Kaizhan Lin, Weize Wang, Zheng Xu, and Chang-An Zhao. A faster software implementation of SQISign. Cryptology ePrint Archive, Paper 2023/753, 2023. https://eprint.iacr.org/2023/753, visited on September 7, 2023.

[24] Artur Mariano, Thijs Laarhoven, Fábio Correia, Manuel Rodrigues, and Gabriel Falcão. A practical view of the state-of-the-art of lattice-based cryptanalysis. *IEEE Access*, 5:24184–24202, 2017.

[25] Daniele Micciancio. Lattices Algorithms and Applications, 2014. Course CSE206A from Spring 2014 at UC San Diego, available at https://cseweb.ucsd.edu/classes/sp14/cse206A-a/index.html (accessed on July 19, 2023).

[26] Daniele Micciancio and Michael Walter. *Fast Lattice Point Enumeration with Minimal Overhead*, pages 276–294.

[27] François Morain and Jean-Louis Nicolas. On Cornacchia's algorithm for solving the diophantine equation $u^2 + dv^2 = m$, 1990. Last accessed on July 27, 2023.

[28] Phong Q. Nguyen and Damien Stehlé. Low-Dimensional Lattice Basis Reduction Revisited. In Duncan Buell, editor, *Algorithmic Number Theory*, pages 338–357, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[29] Christophe Petit and Spike Smith. An improvement to the quaternion analogue of the l-isogeny problem. *Presentation at MathCrypt*, 2018.

[30] Benjamin Schraen. Introduction à la géométrie algébrique et courbes elliptiques, 2021. 2021 edition of lecture notes for MAT562, Département de Mathématiques, École polytechnique, France.

[31] Joseph H Silverman. *The Arithmetic of Elliptic Curves*. Graduate texts in mathematics. Springer, Dordrecht, 2009.

[32] Arne Storjohann and Thom Mulders. Fast Algorithms for Linear Algebra Modulo N. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms — ESA' 98*, pages 139–150, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[33] Andrew Sutherland. Elliptic curves, 2017. 2017 edition of course 17.783 at Massachusetts Institute of Technology, available at https://math.mit.edu/classes/18.783/2017/lectures.html (last accessed on August 4, 2023).

[34] Jacques Vélu. Isogénies entre courbes elliptiques. *Comptes Rendus de l'Académie des Sciences de Paris, Séries A*, 273:238–241, 1971.

[35] John Voight. *Quaternion Algebras*. Springer Graduate Texts in Mathematics series, 2021.

[36] Benjamin Wesolowski. The supersingular isogeny path and endomorphism ring problems are equivalent. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1100–1111, 2021.