Constant time lattice reduction in dimension 4 with application to SQIsign

Otto Hanyecz[1], Alexander Karenin[2], Elena Kirshanova[3], Péter Kutas[1,4] and Sina Schaeffler[5,6]

[1] Eötvős Loránd University, Budapest, Hungary, ohanyecz@inf.elte.hu
[2] Technology Innovation Institute, Abu Dhabi, UAE, alexander.karenin@tii.ae,
[3] Technology Innovation Institute, Abu Dhabi, UAE, elenakirshanova@gmail.com
[4] University of Birmingham, Birmingham, UK, kutasp@gmail.com
[5] ETH Zürich, Zürich, Switzerland,
[6] IBM Research Europe, Zürich, Switzerland, sschaeffle@ethz.ch

The authors are listed in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf.

Abstract. In this paper we propose a constant time lattice reduction algorithm for integral dimension-4 lattices. Motivated by its application in the SQIsign post-quantum signature scheme, we provide for the first time a constant time LLL-like algorithm with guarantees on the length of the shortest output vector. We implemented our algorithm and ensured through various tools that it indeed operates in constant time. Our experiments suggest that in practice our implementation outputs a Minkowski reduced basis and thus can replace a non constant time lattice reduction subroutine in SQIsign.

Keywords: LLL · BKZ · constant time · isogenies · SQIsign

1 Introduction

Due to the recent advancements in quantum technology it is highly important to design primitives that remain secure even if an adversary possesses a large-scale quantum computer. The National Institute of Standards and Technology (NIST) launched its first post-quantum standardization effort in 2016 which partially concluded in 2022 by the selection of one key exchange and three digital signature schemes (https://www.nist.gov/pqcrypto). Even though three digital signatures were selected for standardization, several important issues remained: Dilithium signatures and public keys are large, Falcon is using floating-point arithmetic (thus hard to implement securely) and SPHINCS+ is impractical for many use cases, and suggested mostly as a conservative choice for special purpose applications. Most importantly, the two main choices both rely on structured lattices and it is prudent to have alternative options in case the hard problems underlying structured lattice-based schemes admit efficient (classical or quantum) algorithms.

Thus NIST launched a new call specifically for digital signature schemes explicitly to address the aforementioned issues. One of the candidates is SQIsign [CSSDF+23], an isogeny-based signature scheme. Even though previous NIST candidate SIKE suffered a devastating attack [CD23], SQIsign is unaffected as the attack heavily relies on extra information provided in SIKE. The main advantage of SQIsign is its small public keys and signatures. The drawback of SQIsign is slow signing and verification. However, it has to be noted that verification is considerably faster than signing and recent advances [BFD+24],[NO24],[DF24],[AAA+25] have provided significant improvements.

The signing procedure in SQIsign is relatively complex, especially compared to other NIST submissions. Thus at the time of the first round submission, none of the major subroutines were implemented in constant-time (and there has been no official update since). Signing entails computations involving integers (dubbed the quaternion side) and computations in finite fields (dubbed the elliptic curve side). Algorithms used on the quaternion side are very different in nature compared to any other NIST submissions thus making them constant time requires novel techniques to ensure protection against timing attacks. This paper focuses on lattice algorithm aspects relevant to SQIsign. The lattices occurring on SQIsign's quaternion side are different from lattices used in lattice-based cryptography: they have very low dimension (dimension 4 and 2) but large determinant (usually around 512 bits). Furthermore, lattice reduction techniques are primarily used in cryptanalysis, hence have not been considered in a constant time setting.

Our contributions. We present a constant time LLL-like algorithm for dimension-4 integral lattices. Here "constant time" means that the running time is independent of the input lattice once the size of integers is fixed, as long as the input is valid. Existing LLL-type algorithms [LLL82, NS09b] do not have this feature: conditional swaps inside LLL depend on the shape of the lattice. Our algorithm is reminiscent of the BKZ reduction with block size 2 [Sch87]. As the main subroutine it uses our constant time implementation of the shortest vector algorithm for dimension-2 lattices.

There are three main challenges proving that our algorithm is constant time. First, we show how to bound the number of iterations in the dimension-2 shortest vector algorithm known as Lagrange reduction. For that we adapt the analysis of the Lagrange/Gauss algorithm due to Vallée [Val91]. Second, we need to bound the number of 'bkz-tours' (a value proportional to the amount of required Lagrange calls). Adapting the analysis of BKZ reduction from [HPS11] to our setting, we show how to bound the number of bkz-tours while having a guarantee on the length of the shortest output vector. Third, in order to guarantee constant time arithmetic we need to bound the sizes of integers that appear during all computations. As our algorithm internally operates on the same objects as the original LLL algorithm [LLL82], we adapt the integer size analysis from there.

We implemented our algorithm in C. We rely on our non-optimized constant time implementation of integers and rationals which partially replaces the variable-time functions on multi-precision integers from the library GMP [Gt] which the NIST submission used.

We provide extensive experimental evidences that in our implementation we output bases that satisfy the requirements of SQIsign. Concretely, in practice our algorithm returns a Minkowski reduced basis (which is a very strong notion of reducedness) after a small number of bkz-tours. We ensured our implementation is constant time using ctgrind [Lan10] and the statistical analyzer RTLF [DME+24, ME24]. To the best of our knowledge, RTLF implements the most advanced statistical evaluation methodology and is able to detect statistical differences better than other existing tools. Finally we compared our implementation to the LLL implementation used in [CSSDF+23] with respect to performance. We note that we do not analyze our implementation against active side-channels attacks, e.g., fault injections as this is beyond the scope of this work. We should also remark that even though our LLL routine makes a significant step towards constant time SQIsign, it is not the only routine that has to be taken care of to claim constant timeness of the whole signature.

The code together with various test results are available at https://github.com/CTlll-SQIsign/CTlll-SQIsign.

Impact on SQIsign variants. SQIsign is a isogeny- and quaternion-based scheme, as explained in Section 2. In the original SQIsign variants [DFKL+20],[DFLLW23] as well as the Après-SQI variant [CEMR24] (which uses essentially the same signing procedure),

lattice reduction is required by the KLPT algorithm. This algorithm is needed for transforming an ideal in a quaternion algebra into an equivalent ideal of very smooth and not too large norm suitable for conversion into a representable isogeny. Since the attacks on SIDH [CD23] have allowed us to efficiently represent some non-smooth degree isogenies, several new variants of SQIsign emerged: SQIsignHD [DLRW24], SQIsign2D-West [BFD+24], SQIsign2D-East [NO24],[CCI+24] and SQIPrime [DF24], of which the latter three will be published at Asiacrypt 2024. These variants do not use the full KLPT [KLPT14] algorithm anymore (even though SQIsign2DEast uses its SpecialEichlerNorm variant), since they only require an equivalent ideal of short norm (which is not necessarily smooth). This shortness requirement means that there is still need for lattice reduction.

Related work. The study of the quaternion side of SQIsign with respect to constant-timeness was begun in [JMKR23]. This paper focused exclusively on Cornacchia's algorithm (and how recovering inputs to Cornacchia's algorithm lead to a key recovery attack) and its use in the original SQIsign scheme from [DFKL+20, DFLLW23], proposing both a constant-time and a masking approach for this algorithm. [JMKR23, Algorithm 10] is a proposal for dimension 2 lattice reduction used for representing integers as sum of two squares. This algorithm takes the input basis obtained from the integer to be represented and randomizes it in a suitable fashion, then runs the usual Lagrange reduction. Its runtime is not actually constant, but at best independent from the input integers. No proof of this independence is given, but only experimental validation using a t-test. Our paper goes beyond that approach in many ways. First, we provide a proven constant-time version of Lagrange reduction. Second, since SQIsign actually requires lattice reduction for dimension 4 lattices, while Lagrange reduction only applies directly to dimension 2, we also provide constant-time algorithms for the dimension 4 case together with rigorous proofs. Finally, we use RTLF instead of the standard t-test for experimental validation of our implementation.

Another paper on side-channel attacks on SQIsign focusing on fault attacks appeared in 2024 [LHK+24], which did not focus on lattice reduction. To the best of our knowledge, there are no other works on securing the quaternion side of SQIsign against side-channel attacks.

Organization of the paper. We first fix notations and terminology in Section 2. That section also explains where in SQIsign lattice reduction is required, and introduces classical concepts and algorithms for lattice reduction. Then we present our constant-time lattice reduction algorithm in Section 3. First we describe constant-time adaptations of its building blocks including the Lagrange algorithm in Section 3.1, then we state and analyse our BKZ-2 variant in Section 3.2. Finally, we explain our implementation in Section 4. First we study which inputs it will be given in SQIsign in Section 4.1. We verify its correctness in Section 4.2, then its constant-timeness in Section 4.3 and finally its performance when compared to to the LLL implementation from [CSSDF+23] in Section 4.4.

Acknowledgements. This work was supported in part by SNSF Consolidator Grant CryptonIs 213766. Kutas is partly supported by EPSRC through grant number EP/V011324/1. Kutas is supported by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Quantum Information National Laboratory Program and by grant "EXCELLENCE-151343". Kutas is also supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

We also want to thank Robert Merget for help with RTLF, and Olivér Facklam for helping us to find a hyperthreading issue.

2 Preliminaries

2.1 Notations and conventions

We denote matrices as bold upper case letters (e.g. $\mathbf{B}$). Their (column) vectors are denoted as bold lower case letters (e.g. $\mathbf{b}_i$). We denote by $\|\mathbf{b}\|$ the Euclidean norm of $\mathbf{b}$. The transpose of $\mathbf{B}$ is denoted as $\mathbf{B}^T$. A submatrix $(\mathbf{b}_i, \ldots, \mathbf{b}_{i'})$ of $\mathbf{B}$ is denoted as $\mathbf{B}[i, \ldots, i'], i' \geqslant i$.

For a statement s we define $\mathbb{I}(s)$ to be equal to 1 if s holds and 0 otherwise. The *bitsize* of an integer n is defined as $\lceil \log_2 |n| \rceil$.

In this work we assume that the following operations are constant time as long as the bitsizes of their operands are explicitly bounded from above.

- Addition, subtraction, multiplication of rationals and integral division of integers.
- Comparisons $=, >, <, \geqslant, \leqslant$ of two rationals and rounding towards left $\lfloor \bullet \rfloor$, right $\lceil \bullet \rceil$ and nearest integer $\lfloor \bullet \rceil$.
- Assigning (denoted $:=$), swapping and copying numbers. Also conditionally, denoted $\text{assign}_c(\cdot, \cdot)$ (assigns the first value if c is true, otherwise the second) and $\text{swap}_c(\cdot, \cdot)$ (swaps if c is true) as long as the condition c can be evaluated in constant time.
- Accessing an i-th element of an array for some $i \geqslant 0$, in time independent of the content of the array (but access time can depend on the index i).

These assumptions apply to large integers and rationals. They are not satisfied by most libraries for big integers. Even GMP [Gt] which proposes some constant-time operations, does not cover all of these operations. Care must therefore be used when implementing our results.

For $n \in \mathbb{N}$ we say that $q : \mathbb{R}^n \to \mathbb{R}^+$ is a *quadratic form* if the following conditions hold:

- For all $\lambda \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^n$ we have: $q(\lambda \cdot \mathbf{v}) = \lambda^2 \cdot q(\mathbf{v})$
- We have that $\langle \mathbf{u}, \mathbf{v} \rangle_q$ defined as $\frac{1}{2}(q(\mathbf{u}+\mathbf{v}) - q(\mathbf{u}) - q(\mathbf{v}))$ is a positive definite symmetric bilinear form. This implies $q(\mathbf{v}) = \langle \mathbf{v}, \mathbf{v} \rangle_q$. For brevity we will refer to positive definite symmetric bilinear forms as "inner products".

We will be focusing on the Euclidean norm here since every statement in this work holds for any norm induced by a quadratic form (see [Coh93, Section 2.5]).

2.2 Isogenies and quaternions

Let E_1 and E_2 be elliptic curves defined over a finite field of charactersitic p. An isogeny between E_1 and E_2 is a non-constant rational map which is also a group homomorphism. The degree of a (separable) isogeny is the size of its kernel. An isogeny from E to itself is called an endomorphism and endomorphisms (together with the 0 map) form a ring under addition and composition. An elliptic curve is called supersingular if its endomorphism ring is non-commutative.

One can actually give a much more precise statement on the structure of these endomorphism rings. A rational quaternion algebra is a $\mathbb{Q}$-algebra generated by a basis $1, i, j, k$ with the relations $ij = k, ij = -ji$ and $i^2 = a, j^2 = b$ where $a, b \in \mathbb{Q}$. An order in a quaternion algebra is a subring containing 1 which is also a four-dimensional $\mathbb{Z}$-lattice. An order is maximal if it is maximal with respect to inclusion. The Deuring correspondence tells us that the endomorphism rings of supersingular elliptic curves are maximal orders in a certain rational quaternion algebra. If $p \equiv 3 \pmod 4$, then this quaternion algebra is given by the relations $i^2 = -1, j^2 = -p$ (in full generality the quaternion algebra in question is the one

ramified at p and ∞). This quaternion algebra is denoted by $B_{p,\infty}$. The conjugate of an element $x = x_1 + x_i i + x_j j + x_{ij} ij \in B_{p,\infty}$ is $\bar{x} = x_1 - x_i i - x_j j - x_{ij} ij$, and the reduced norm of x is $\mathrm{nrd}(x) = x\bar{x} \in \mathbb{Q}$. This "norm" therefore is a quadratic form, and defines a symmetric bilinear form $< a, b >= \frac{1}{2}(\mathrm{nrd}(a+b) - \mathrm{nrd}(a) - \mathrm{nrd}(b)) = a_1 b_1 + a_i b_i + p(a_j b_j + a_{ij} b_{ij})$.

The Deuring correspondence actually provides a deeper connection between supersingular elliptic curves and maximal orders. Namely isogenies from E_1 to E_2 correspond to left ideals of $\mathrm{End}(\mathrm{E}_1)$ and right ideals of $\mathrm{End}(\mathrm{E}_2)$. The norm of an ideal can be defined as the greatest common divisors of the reduced norm of all the elements. The norm of an ideal corresponds to the degree of an isogeny. There is a relatively simple geometric interpretation of the correspondence between left ideals and isogenies. Namely one can look at the set of all isogenies between E_1 and E_2 denoted by $\mathrm{Hom}(\mathrm{E}_1, \mathrm{E}_2)$. This is clearly a $\mathbb{Z}$-module but the degree function also equips it with a scalar product, so one can actually view $\mathrm{Hom}(\mathrm{E}_1, \mathrm{E}_2)$ as a Euclidean lattice. This is indeed a four-dimensional lattice with determinant p^2. Voight in [Voi21, Lemma 42.2.8] gives a precise connection between left ideals and $\mathrm{Hom}(\mathrm{E}_1, \mathrm{E}_2)$ (i.e., the Gram matrices of these lattices only differ by multiplication of the norm of an ideal).

2.3 SQIsign variants

The main hard problem in isogeny-based cryptography is to find isogenies (often of a certain degree) between supersingular elliptic curves. On the quaternion side this problem is actually easy, one particular solution is given in [KV10]. The reason that the isogeny problem is still hard is that computing endomorphism rings is believed to be hard, i.e., it is hard to move from the elliptic curve world to the quaternion world. This motivates the concept of the SQIsign identification scheme from [DFKL+20]. Let E_0 be a special supersingular elliptic curve of known endomorphism ring. Let $\tau : E_0 \to E_A$ be a secret isogeny and let E_A be public. Then the prover computes a commitment isogeny $E_0 \to E_1$ and the verifier responds with a challenge isogeny $\varphi : E_A \to E_2$. The prover then responds with a certain isogeny $\sigma : E_1 \to E_2$. The verifier accepts if it is indeed an isogeny between the two curves and some other technical condition (composed with the dual of φ it is still cyclic) is satisfied. This technical condition is needed to avoid the simple attack where a prover generates the commitment isogeny from E_A. This is summarized in Figure 1.

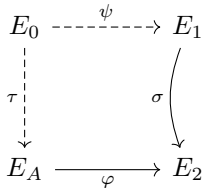

Figure 1: SQIsign

Remark 1. Actually in [DFKL+20] the scheme is not exactly like Figure 1. Namely, the challenge is generated from E_1 and the response is an isogeny between E_A and E_2. However, this layout changed to Figure 1 in later SQIsign variants and is equivalent to the original layout from a security standpoint.

The main point of this concept is that a prover who knows the witness knows the endomorphism ring of every single curve in Figure 1 hence can respond with a suitable isogeny. Whereas a dishonest prover does not know the endomorphism ring of E_A hence cannot fully transform the problem to the quaternion side.

Even though this concept is relatively simple, actually making it practical is quite daunting. The first problem one encounters is that [KV10] returns some left ideal of

random norm. Therefore, when one naturally translates it to an isogeny it is likely to have a non-smooth degree hence there is no obvious way to represent it to ensure fast verification. This problem is resolved via [KLPT14] and [DFKL+20] where one computes a connecting ideal whose norm is a power of 2 (thus can be represented as a chain of degree 2 isogenies). The main drawback of this construction is that the isogeny returned is quite long (which actually poses some technical difficulties) and the zero knowledge property relies on an ad hoc assumption. This is the high level concept of the original SQIsign protocol.

Newer variants of SQIsign [DLRW24],[BFD+24],[NO24],[DF24] circumvent this problem in a different manner. Namely one can actually represent a non-smooth degree isogeny using higher dimensional isogenies. This eventually leads to faster and more secure versions of SQIsign. Nevertheless, finding more efficient versions of SQIsign is still an active research area thus both types of methods need some consideration.

One common point in all methods however, is finding a relatively small norm connecting ideal between two maximal orders (i.e., a short vector in $\mathrm{Hom}(\mathrm{E}_1, \mathrm{E}_2)$), which requires to use some form of a lattice reduction. This is important in all variants as it governs the degree of the response isogeny, on which depends the speed of verification, as well as the size of the signature. Hence it is highly important to have subroutines for finding elements of small reduced norm in a left ideal, or equivalently in $\mathrm{Hom}(\mathrm{E}_1, \mathrm{E}_2)$. Since this procedure is a crucial step in computing the response isogeny (i.e., in a Fiat-Shamir setting in signing), one has to have a constant-time implementation of this step to avoid potential timing attacks. blue Finally, we believe that since lattice reduction plays an important role providing compact representations of ideals, any quaternion-based scheme will need it as a subroutine. However, so far no lattice reduction algorithm with input-independent runtime and sufficient guarantees on its output quality is known.

2.4 Lattices

A lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$ endowed with an inner product $\langle \bullet, \bullet \rangle$. Here $n \in \mathbb{N}$ is called the dimension of the ambient space. We consider the case of full-rank lattices where $\mathcal{L} = \bigoplus_{i=0}^{n-1} \mathbb{Z}^n \cdot \mathbf{b}_i$ for some linearly independent $\mathbf{b}_i \in \mathbb{R}^n, 0 \leqslant i < n$. In that case the lattice $\mathcal{L}$ is said to have its rank equal to n. The corresponding matrix with columns $(\mathbf{b}_0, \ldots, \mathbf{b}_{n-1})$ is called a basis of $\mathcal{L}$. For $n \geqslant 2$ any dimension-n lattice admits an infinite number of bases. If $\mathbf{B}$ and $\mathbf{B}'$ are bases of $\mathcal{L}$ then it holds that $\mathbf{B} = \mathbf{B}' \cdot \mathbf{U}$ for some $\mathbf{U} \in \mathbb{Z}^n$ with $\det \mathbf{U} = \pm 1$.

A lattice with basis $\mathbf{B}$ is denoted as $\mathcal{L}(\mathbf{B})$. The matrix

$$\mathbf{G} = \left(\langle \mathbf{b}_i, \mathbf{b}_j \rangle\right)_{0 \leqslant i,j < n} \in \mathbb{R}^{n \times n}$$

is called the Gram matrix of $\mathcal{L}(\mathbf{B})$. The determinant of $\mathcal{L}(\mathbf{B})$ is defined as $(\det \mathbf{G})^{1/2}$. The norm of a shortest nonzero vector of $\mathcal{L}$ is denoted as $\lambda_1(\mathcal{L})$. The problem of finding a vector that attains λ_1 is called the Shortest Vector Problem (SVP).

For the vector space $\mathbb{R}^n$ we recall the notion of the Hermite constant defined as $\gamma_n = \max \frac{\min_{\mathbf{v} \in \mathbb{Z}^n \setminus \{0\}} q(\mathbf{v})}{\det(q)^{1/n}}$ where the maximum is taken over all q being positive definite quadratic forms over $\mathbb{R}^n$. The maximum exists [Ngu09] and by definition is independent of a considered quadratic form q.

The Gram-Schmidt vectors $\{\mathbf{b}_0^*, \ldots, \mathbf{b}_{n-1}^*\}$ of a basis $\mathbf{B}$ are defined as follows. The first Gram-Schmidt vector $\mathbf{b}_0^*$ is equal to $\mathbf{b}_0$. The $(n-1)$ remaining Gram-Schmidt vectors are defined recursively as $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=0}^{i-1} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \cdot \mathbf{b}_j^*$. We call the quantities

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}, 0 \leqslant i < n, 0 \leqslant j < i \quad \text{and} \quad r_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle, 0 \leqslant i < n, 0 \leqslant j \leqslant i$$

the Gram-Schmidt coefficients. It holds that $r_{i,i} = \|\mathbf{b}_i^*\|^2, \mu_{i,j} = \frac{r_{i,j}}{r_{j,j}}$. Given the Gram matrix of a basis one can compute the corresponding Gram-Schmidt coefficients efficiently using the Cholesky factorization algorithm [NS09a, Fig. 4]. We will make use of the projections orthogonally to the first $i < n$ Gram-Schmidt vectors which we denote as

$$\pi_i(\mathbf{v}) = \mathbf{v} - \sum_{j<i} \frac{\langle \mathbf{v}, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \cdot \mathbf{b}_j^*.$$

By the i'th projective sublattice of a rank $\beta < n-i$ we refer to the lattice $\mathcal{L}(\pi_i([\mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta-1}]))$ where the projection is applied to each vector of $\mathbf{B}[i, \ldots, i+\beta-1]$. It also holds that $\det \mathcal{L}(\mathbf{B}) = \prod_{i=0}^{n-1} \|\mathbf{b}_i^*\|$.

2.5 Lattice reduction

Here we define the notions of lattice reductions, namely size reduction, Lagrange reduction, and LLL reduction.

Size reduction There are several notions of reducedness. A basis $\mathbf{B}$ is called *size reduced* if $|\mu_{i,j}| \leqslant 1/2, 0 \leqslant i < n, j < i$. Algorithm 2.1 shows how to obtain a size reduced basis.

Algorithm 2.1 Size reduction

Input: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ – a basis of a lattice.
Output: $\mathbf{B} \in \mathbb{Z}^{n \times n}$ – a size reduced basis of $\mathcal{L}(\mathbf{B})$

1: **for** $i = 0$ to $n-1$ **do**
2: **for** $j = i-1$ to 0 **do**
3: Compute $\{\mu_{\iota,j}\}_{0 \leqslant \iota \leqslant i, j < \iota}$ corresponding to $\mathbf{B}$
4: $\mathbf{b}_i := \mathbf{b}_i - \lfloor \mu_{i,j} \rceil \cdot \mathbf{b}_j$
return B

At step 4 of Algorithm 2.1 the current value of $\mu_{i,j}$ is replaced by $\mu_{i,j} - \lfloor \mu_{i,j} \rceil$ while all $\mu_{i,j'}$ remain unchanged for all $j' > j$. Hence, after the execution all $|\mu_{i,j}| \leqslant 1/2$ and the resulting basis is size reduced.

Lagrange reduction A basis $\mathbf{B}$ of rank-2 lattice $\mathcal{L} \subset \mathbb{Q}^m, m \in \mathbb{N}, m > 1$ is said to be *Lagrange reduced* if the two following conditions are satisfied: $\|\mathbf{b}_0\| \leqslant \|\mathbf{b}_1\|$ and $\langle \mathbf{b}_0, \mathbf{b}_1 \rangle \leqslant |\mathbf{b}_0\|^2/2$ where the norm corresponds to a quadratic form q. An algorithm that performs the Lagrange reduction of a basis is presented in Algorithm 2.2. It receives as input a basis matrix $\mathbf{B}$ and the corresponding Gram matrix $\mathbf{G}$ and shortens the projections of $\mathbf{b}_i$ onto $\mathbf{b}_j^*, 0 \leqslant i < i$.

We call a size reduced basis $\mathbf{B}$ of $\mathcal{L}$ *HKZ reduced* if $\|\mathbf{b}_0\| = \lambda_1(\mathcal{L})$ and $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(\mathbf{B}))$. In dimension 2 Lagrange reduced bases are also HKZ reduced [Val91, Proposition 1]. In order to proceed we need the following lemma

Lemma 1. *For any HKZ reduced basis* $\mathbf{B}$ *we have* $\forall i < n-1$,

$$\|\mathbf{b}_i^*\| \leqslant \sqrt{\gamma_{n-i}} \cdot \left(\prod_{j=i}^{n-1} \|\mathbf{b}_j^*\| \right)^{\frac{1}{n-i}}.$$

Algorithm 2.2 Lagrange reduction

Input: $\mathbf{B} \in \mathbb{Q}^{m\times 2}$ – a basis of a lattice
$\mathbf{G} = (\mathbf{B}^T \cdot \mathbf{B})$ – a Gram matrix corresponding to $\mathbf{B}$.
Output: $\mathbf{B} \in \mathbb{Z}^{m\times 2}$ – a Lagrange reduced basis of $\mathcal{L}(\mathbf{B})$
$\mathbf{G} = (\mathbf{B}^T \cdot \mathbf{B})$ – a Gram matrix corresponding to reduced $\mathbf{B}$.

repeat
  $\mu := \frac{\langle \mathbf{b}_1, \mathbf{b}_0 \rangle}{\|\mathbf{b}_0\|^2}$
  $\mathbf{b}_1 := \mathbf{b}_1 - \lfloor \mu \rceil \mathbf{b}_0$ ▷ "Size reduction"
  $\mathbf{G}_{1,1} := \mathbf{G}_{1,1} - (2\lfloor \mu \rceil \cdot \mathbf{G}_{1,0} - \lfloor \mu \rceil^2 \cdot \mathbf{G}_{0,0})$
  $\mathbf{G}_{1,0} := \mathbf{G}_{1,0} - \lfloor \mu \rceil \cdot \mathbf{G}_{0,0}$
  $\mathbf{b}_0, \mathbf{b}_1 := \mathbf{b}_1, \mathbf{b}_0$ ▷ "Swap"
  $\mathbf{G}_{0,0}, \mathbf{G}_{1,1} := \mathbf{G}_{1,1}, \mathbf{G}_{0,0}$
until $\|\mathbf{b}_0\| \geqslant \|\mathbf{b}_1\|$
$\mathbf{b}_0, \mathbf{b}_1 := \mathbf{b}_1, \mathbf{b}_0$
$\mathbf{G}_{0,0}, \mathbf{G}_{1,1} := \mathbf{G}_{1,1}, \mathbf{G}_{0,0}$
return $\mathbf{B}$, $\mathbf{G}$

LLL reduction Since our constant time lattice reduction algorithm will be partially based on the celebrated LLL algorithm [LLL82] we define for $1/2 < \delta < 1$ the notion of δ-LLL reducedness as follows. A basis $\mathbf{B} \in \mathbb{Z}^{n\times n}$ is said to be LLL reduced if $|\mu_{i,j}| \leqslant 1/2 \ \forall\ 0 \leqslant i < n, j < i$ and $\delta^2 r_{i,i} \leqslant \mu_{i,i+1}^2 \cdot r_{i,i} + r_{i+1,i+1}$ where the GSO coefficients are computed w.r.t. the inner product corresponding to a quadratic form q.

To claim constant time, we need to bound the bitsizes of integers involved in the computations. For this we use a bound proved in [LLL82]. Even though our algorithm is not classical LLL, internally we are computing the same objects, namely the GSO coefficients, and they determine the required integer sizes as proved in [LLL82, Proposition 1.26]. We formulate here a precise version of this result, its proof can be directly reconstructed from the proof of [LLL82, Proposition 1.26].

Lemma 2 ([LLL82, Proposition 1.26])**.** *Let $\mathbf{B} \in \mathbb{Z}^{n\times n}$ be a basis of an integer dimension-n lattice and $B = \max_i \|\mathbf{b}_i\|^2$. Then all corresponding $r_{i,j}, \mu_{i,j}$ can be represented with fractions of integers of size of at most $n + \frac{3(n-1)}{2} \cdot \log B + \frac{\log n}{2} + 1$ bits.*

3 Constant time LLL

In this section we first introduce the building blocks our variant of lattice reduction algorithm will utilize. Next, we present our constant time BKZ-2 algorithm, argue on its constant timeness, as well as on guarantee of the output basis.

3.1 Constant time building blocks

Our lattice reduction algorithm requires several subroutines that allow us to compute and update the Gram-Schmidt coefficients. In this section we show constant-time algorithms for these tasks.

GSO via Cholesky In order to keep all coefficients of a Gram matrix updated we use the Cholesky algorithm given in [NS09a]. It takes as input the Gram matrix $\mathbf{G}$ of a basis $\mathbf{B}$ and outputs the corresponding Gram-Schmidt coefficients $\{\mu_{i,j}\}_{0\leqslant i<4, j<i}$ and $\{r_{i,j}\}_{0\leqslant i<4, j<i}$ valid for all $i \leqslant \rho$ for a given parameter $\rho \in \{0,1,2,3\}$ and not necessarily valid for $i > \rho$. The reason why we introduce the indices ℓ and ρ is the fact that we are usually interested in updating the Gram-Schmidt coefficients in a lazy manner after updating some basis

vectors, saving us some computations. This routine is presented in Algorithm 3.1. From the presented pseudocode it follows that the algorithm is constant time provided that we can bound the sizes of the integers the algorithm manipulates with.

Algorithm 3.1 Cholesky

Input: $\mathbf{G} = (\mathbf{B}^T \cdot \mathbf{B})$ – a Gram matrix corresponding to $\mathbf{B} \in \mathbb{Z}^{4\times 4}$.
$\{\mu_{i,j}\}_{0\leqslant i<4, j<i}$, $\{r_{i,j}\}_{0\leqslant i<4, j<i}$ – corresponding Gram-Schmidt coefficients valid for all $i < \ell$,
$\ell, \rho \in \{0, 1, 2, 3\}, \ell \leqslant \rho$.

Output: $\{\mu_{i,j}\}_{0\leqslant i<4, j<i}$, $\{r_{i,j}\}_{0\leqslant i<4, j<i}$ – corresponding Gram-Schmidt coefficients valid for $0 \leqslant i \leqslant \rho$.

1: **for** $i = \ell$ to ρ **do**
2: **for** $j = 0$ to $i - 1$ **do**
3: $r_{i,j} := \mathbf{G}_{i,j}$
4: **for** $k = 0$ to $j - 1$ **do**
5: $r_{i,j} := r_{i,j} - \mu_{j,k} r_{i,k}$
6: $\mu_{i,j} = r_{i,j}/r_{j,j}$
7: $r_{i,i} = \mathbf{G}_{i,i}$
8: **for** $j = 0$ to $i - 1$ **do**
9: $r_{i,i} := r_{i,i} - \mu_{i,j} r_{i,j}$
return $\{\mu_{i,j}\}_{0\leqslant i<4, j<i}$, $\{r_{i,j}\}_{0\leqslant i<4, j<i}$

Constant time size reduction Below we introduce Algorithm 3.2 that, when called on a basis $\mathbf{B}$ for $0 \leqslant i < n$, results in a size reduced basis of $\mathcal{L}(\mathbf{B})$. While Algorithm 2.1 is already constant time for a fixed dimension and precision, we use its modified version that uses solely the Gram matrix of a basis. The design rationale behind this decision is that the Gram matrix of an integral basis is exact and by performing operations on a basis we do not lose the accuracy. In addition, that simplifies the required arithmetic operations and allows us SQIsign-specific optimizations explained in Section 4.4.

Lemma 3. *On its input Algorithm 3.2 size reduces the i'th vector of* $\mathbf{B}$. *If* $B = \max_i \|\mathbf{b}_i\|^2 = \max_i \mathbf{G}_{i,i}$ *is bounded from above, the algorithm's complexity depends only on* $i \in \{1, 2, 3\}$. *The algorithm operates with integers of bitsize bounded by* $\lceil 6 + \frac{11}{2} \cdot \log B \rceil$.

Proof. Let $\mathbf{B} \in \mathbb{Z}^{4\times 4}$ be a matrix, $\mathbf{G} = \mathbf{B}^T \cdot \mathbf{B}$ and $i \in \{1, 2, 3\}$. First, notice that $\{\mu_{\iota,\kappa}\}_{\iota\leqslant i, \kappa<\iota}$ and $\{r_{\iota,\kappa}\}_{\iota\leqslant i, \kappa\leqslant\iota}$ are valid at the beginning of each operation of the main loop. Within an iteration of this loop, Steps 2–8 perform the size-reduction using the Gram-matrix, Step 10 updates all Gram-Schmidt coefficients for $\iota < i + 1$.

For a fixed i each line of the algorithm is executed a constant number of times. We have no branching and all arithmetic operations are constant time except maybe in i. The algorithm operates on $\mathbf{G}$ and the GSO coefficients $\mu_{ij}, r_{i,j}$. Entries of $\mathbf{G}$ require $\lceil \log B \rceil$ bits, while the bitsize of entries of GSO are bounded from above in Algorithm 3.2. Thus, the bound on the bitsize of integers involved comes from substituting $n = 4$ into Lemma 2 for a non-reduced basis $\mathbf{B}$. □

For our purposes we also need an algorithm that updates the Gram matrix of $\mathbf{B}$ after a transformation of the form $[\mathbf{b}_i, \mathbf{b}_{i+1}] \leftarrow [\mathbf{b}_i, \mathbf{b}_{i+1}] \cdot \mathbf{U}$ for some $\mathbf{U} \in \mathrm{SL}_2(\mathbb{Z})$.

The intuition behind Algorithm 3.3 is straightforward. Let $\mathbf{G}$ be a non-updated Gram matrix and let $\mathbf{G}'$ denote the updated one. It will be easier to look at the three zones of the elements of $\mathbf{G}$ as in Figure 2. The following are the results of straightforward computations. In region I, we have the relations from the Step 3. Analogously, the values

Algorithm 3.2 size_red

Input: $\mathbf{B} \in \mathbb{Z}^{4\times 4}$ – a basis.
$i \in \{1,2,3\}$ – an index.
$\mathbf{G} \in \mathbb{Z}^{4\times 4}$ – the Gram matrix of the input basis $\mathbf{B} \in \mathbb{Z}^{4\times 4}$.
$\{\mu_{\iota,j}\}, \{r_{\iota,j}\}$ – Gram-Schmidt coefficients valid for $0 \leqslant \iota < i+1$.
Output: $\mathbf{B} \in \mathbb{Z}^{4\times 4}$ – a basis after the size reduction of $\mathbf{b}_i$.
$\mathbf{G}$ – a Gram matrix after the size reduction of $\mathbf{b}_i$.
$\{\mu_{\iota,j}\}_{0\leqslant\iota<n,j<\iota}, \{r_{\iota,j}\}_{0\leqslant\iota<n,j\leqslant\iota}$ – corresponding Gram-Schmidt coefficients valid up to the $(i+1)$'th column.

1: **for** $j = i-1$ to 0 **do**
2: $\quad \mu := \lfloor \mu_{i,j} \rceil$
3: $\quad g^{(i,j)}, g^{(j,j)} := \mathbf{G}_{i,j}, \mathbf{G}_{j,j}$
4: $\quad$ **for** $\kappa = 0$ to i **do**
5: $\quad\quad X := \text{assign}_{\kappa\leqslant j}(\mathbf{G}_{j,\kappa}, \mathbf{G}_{\kappa,j})$ ▷ Conditional assignment
6: $\quad\quad \mathbf{G}_{i,\kappa} = \mathbf{G}_{i,\kappa} - \mu \cdot X$
7: $\quad$ **for** $\kappa = i+1$ to 3 **do**
8: $\quad\quad \mathbf{G}_{\kappa,i} = \mathbf{G}_{\kappa,i} - \mu \cdot \mathbf{G}_{j,\kappa}$
9: $\quad \mathbf{G}_{i,i} := \mathbf{G}_{i,i} - 2\mu g^{(i,j)} - \mu^2 g^{(j,j)}$
10: $\quad \{\mu_{i,\kappa}\}_{\kappa<i}, \{r_{i,\kappa}\}_{\kappa\leqslant i} := \mathsf{cholesky}(\mathbf{G}, \{\mu_{i,\kappa}\}_{\kappa<i}, \{r_{i,\kappa}\}_{\kappa\leqslant i}, i, i)$
11: $\quad \mathbf{B}_i := \mathbf{B}_i - \mu \cdot \mathbf{B}_j$
12: **return** $\mathbf{B}$, $\mathbf{G}$,$\{\mu_{\iota,j}\}_{0\leqslant\iota<n,j<\iota}$, $\{r_{\iota,j}\}_{0\leqslant\iota<n,j\leqslant\iota}$

Algorithm 3.3 update_after_svp

Input: $\mathbf{G} \in \mathbb{Z}^{4\times 4}$ – a Gram matrix of a basis $\mathbf{B} \in \mathbb{Z}^{4\times 4}$.
$\mathbf{U} \in \mathbb{Z}^{2\times 2}$ – a unimodular matrix.
$i \in \{0,1,2\}$ – an index.
Output: $\mathbf{G} \in \mathbb{Z}^{4\times 4}$ – a Gram matrix of basis after the transformation $[\mathbf{b}_i, \mathbf{b}_{i+1}] \leftarrow [\mathbf{b}_i, \mathbf{b}_{i+1}] \cdot \mathbf{U}$.

1: $g_0, g_1, g_2 := \mathbf{G}_{i,i}, \mathbf{G}_{i+1,i+1}, \mathbf{G}_{i+1,i}$
2: **for** $j = 0$ to $i-1$ **do**
3: $\quad \mathbf{G}_{i,j}, \mathbf{G}_{i+1,j} := \mathbf{U}_{0,0} \cdot \mathbf{G}_{i,j} + \mathbf{U}_{1,0} \cdot \mathbf{G}_{i+1,j}$, $\mathbf{U}_{0,1} \cdot \mathbf{G}_{i,j} + \mathbf{U}_{1,1} \cdot \mathbf{G}_{i+1,j}$
4: **for** $\iota = i+2$ to 3 **do**
5: $\quad \mathbf{G}_{\iota,i}, \mathbf{G}_{\iota,i+1} := \mathbf{U}_{0,0} \cdot \mathbf{G}_{\iota,i} + \mathbf{U}_{1,0} \cdot \mathbf{G}_{\iota,i+1}, \mathbf{U}_{0,1} \cdot \mathbf{G}_{\iota,i} + \mathbf{U}_{1,1} \cdot \mathbf{G}_{\iota,i+1}$
6: $\mathbf{G}_{i,i} := \mathbf{U}_{0,0}^2 \cdot g_0 + 2\mathbf{U}_{0,0} \cdot \mathbf{U}_{1,0} \cdot g_2 + \mathbf{U}_{1,0}^2 \cdot g_1$
7: $\mathbf{G}_{i+1,i+1} := \mathbf{U}_{0,1}^2 \cdot g_0 + 2\mathbf{U}_{0,0} \cdot \mathbf{U}_{0,1} \cdot g_2 + \mathbf{U}_{1,1}^2 \cdot g_1$
8: $\mathbf{G}_{i+1,i} := \mathbf{U}_{0,1} \cdot \mathbf{U}_{0,0} \cdot g_0 + (\mathbf{U}_{0,1} \cdot \mathbf{U}_{0,1} + \mathbf{U}_{0,0} \cdot \mathbf{U}_{1,1}) \cdot g_2 + \mathbf{U}_{1,0} \cdot \mathbf{U}_{1,1} \cdot g_1$
9: **return** $\mathbf{G}$

of the region III are updated in Step 5. The explicit formulas for the three elements from zone II are the ones used in Steps 6-8.

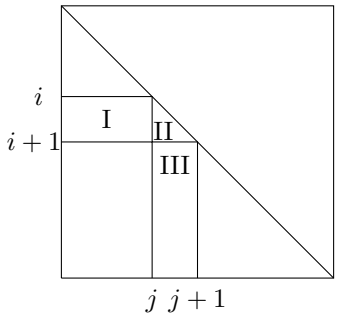

Figure 2: Regions of the Gram matrix to be updated

Lemma 4. *On its input Algorithm 3.3 returns a valid Gram matrix* $\mathbf{G}$ *of* $\mathbf{B}$ *after the transformation* $[\mathbf{b}_i, \mathbf{b}_{i+1}] \leftarrow [\mathbf{b}_i, \mathbf{b}_{i+1}] \cdot \mathbf{U}$. *If* i *is fixed and both* $\max_j \langle \mathbf{b}_j, \mathbf{b}_j \rangle, 0 \leqslant j \leqslant 3$ *and* $\max_{i,j \in \{0,1\}} |\mathbf{U}_{i,j}|$ *are bounded from above, its complexity is independent of an input. The algorithm operates with integers of bitsizes bounded by*

$$\max \left\{ \log_2 \left(4 \cdot \max_i |\mathbf{U}_{i,j}|^2 \cdot \max_i \|\mathbf{b}_i\|^2 \right), n + \frac{3(n-1)}{2} \cdot \log B + \frac{\log n}{2} + 1 \right\}.$$

Proof. The correctness follows by technical, but straightforward calculations. Hence it suffices to prove the bound on the bitsizes of the integers involved.

All variables computed in Algorithm 3.3 are integral. The bitsizes are dominated by the values used in Lines 6-8. For Line 6 and the updated value of $\mathbf{G}'_{i,i}$ we have:

$$|\mathbf{G}'_{i,i}| \leqslant \left|\mathbf{U}_{0,0}^2 \mathbf{G}_{i,i}\right| + 2\left|\mathbf{U}_{0,0}\mathbf{U}_{0,0}\mathbf{G}_{i,i}\right| + \left|\mathbf{U}_{1,1}^2 \mathbf{G}_{i+1,i+1}\right| \leqslant 4 \cdot \max_i |\mathbf{U}_{i,j}|^2 \cdot \max_i{}^2 \|\mathbf{b}_i\|.$$

The same bound also applies to Line 7 and to Line 8 since $|\mathbf{U}_{0,1} \cdot \mathbf{U}_{0,1} + \mathbf{U}_{0,0} \cdot \mathbf{U}_{1,1}| \leqslant 2|\max_i \mathbf{U}_{i,j}|$. Taking the maximum of this bound and the one from Lemma 2 applied to the entire lattice gives the result. □

Constant time Lagrange reduction Next we describe a constant time version of the Lagrange algorithm that works with Gram matrices. We are interested in an algorithm reducing bases $\mathbf{B} \in \mathbb{Q}^{4\times 2}$, as such bases will appear in our constant time version of LLL reduction. The routine is presented in Algorithm 3.4.

Lemma 5. *Let* $\mathbf{B} = (\mathbf{b}_0, \mathbf{b}_1) \in \mathbb{Q}^{4\times 2}$ *be a basis of a dimension-2 lattice* $\mathcal{L}$ *with* $\det(\mathbf{B}^T \cdot \mathbf{B})^{1/2} \geqslant 1$. *Let* $\mathbf{G}$ *be its Gram matrix and* $d \in \mathbb{N}$ *is the smallest natural number such that* $d \cdot \mathbf{B} \in \mathbb{Z}^{4\times 2}$. *Then given* $\mathbf{G}$ *and* $T = \max_i \log_{\sqrt{3}} \|d\mathbf{b}_i\| + 2$, *Algorithm 3.4 returns* $\mathbf{U}$ *such that* $\mathbf{B} \cdot \mathbf{U}$ *is Lagrange reduced. If the bitsizes of all* $\mathbf{G}_{i,j}$ *are bounded from above, the runtime of Algorithm 3.4 is independent of its input. The maximal bitsize of the integers used during the computations is bounded by* $\lceil \max_{i,j} \log \left(d^2 |\mathbf{G}_{i,j}|\right) \rceil + 1$.

Proof. Let $\ell = \|d\mathbf{b}_0\|^2 + \|d\mathbf{b}_1\|^2 \leqslant 2 \cdot \max_i \|d\mathbf{b}_i\|^2$. Then the number of iterations of the main loop to guarantee that the output is Lagrange reduced is $\frac{1}{2} \log_{\sqrt{3}} d^2\ell + 2 \leqslant \max_i \left(\log_{\sqrt{3}} \|d\mathbf{b}_i\|\right) + 2$ due to [Val91, Corollary 1].

Since we cannot guarantee that $\|\mathbf{b}_0\| \leqslant \|\mathbf{b}_1\|$ after the execution of the loop we swap $\mathbf{b}_1$ and $\mathbf{b}_0$ conditionally.

Algorithm 3.4 Lagrange

Input: $\mathbf{G} \in \mathbb{Q}^{2\times 2}$ – a Gram matrix of a basis $\mathbf{B} \in \mathbb{Q}^{4\times 2}$.
$T \in \mathbb{N}$ – number of iterations.
Output: $\mathbf{U} \in \mathbb{Z}^{2\times 2}$ – a unimodular matrix such that $\mathbf{B} \cdot \mathbf{U}$ is Lagrange reduced.

1: $\mathbf{U} := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.
2: **for** $c = 1$ to T **do**
3: $\quad \mu := \mathbf{G}_{1,0}/\mathbf{G}_{0,0}$
4: $\quad \mathbf{G}_{1,1} = \mathbf{G}_{1,1} - (2\lfloor\mu\rceil \cdot \mathbf{G}_{1,0} - \lfloor\mu\rceil^2 \cdot \mathbf{G}_{0,0})$
5: $\quad \mathbf{G}_{1,0} = \mathbf{G}_{1,0} - \lfloor\mu\rceil \cdot \mathbf{G}_{0,0}$
6: $\quad \mathbf{U}_1 = \mathbf{U}_1 - \lfloor\mu\rceil \cdot \mathbf{U}_0$
7: $\quad \mathbf{G}_{0,0}, \mathbf{G}_{1,1} := \mathbf{G}_{1,1}, \mathbf{G}_{0,0}$
8: $\quad \mathbf{U}_{0,0}, \mathbf{U}_{1,1} := \mathbf{U}_{1,1}, \mathbf{U}_{0,0}$
9: $\mathrm{swap}_{\mathbf{G}_{1,1}<\mathbf{G}_{0,0}}(\mathbf{U}_0, \mathbf{U}_1)$
10: **return** $\mathbf{U}$

To prove the constant timeness we need to prove that we have an upper bound on the size of the largest integer encountered by the algorithm and that all operations can be performed in constant time provided that their input is of bounded bitsize. In Step 3 we have $\mu = \mathbf{G}_{1,0}/\mathbf{G}_{0,0}$. For its numerator we have that $|\mathbf{G}_{1,0}| \leqslant \max_i(|\mathbf{G}_{0,0}|, |\mathbf{G}_{1,1}|)$ the same bound is applied to the denominator and, thus, the bitsize of μ is bounded. In Steps 4-5 the arithmetic is integral and the values of $\mathbf{G}_{1,0}, \mathbf{G}_{1,1}$ cannot increase and thus their bitsizes are bounded as well. Steps 7-8 are swaps and the sum of the bitsizes remains constant.

At Step 6 the values $|\mathbf{U}_{i,j}|_{i,j\in\{0,1\}}$ are bounded from above as

$$|\mathbf{U}_{i,j}|_{i,j\in\{0,1\}} \leqslant \max(\|\mathbf{b}_i\|) \cdot \max_i(\|\mathbf{b}'_i\|)/|\det \mathbf{B}| \leqslant \max(\|\mathbf{b}_i\|) \cdot \max_i(\|\mathbf{b}'_i\|)/|\det \mathbf{B}|$$

due to [NS16, Eq. (2)] for $\mathbf{B}' = \mathbf{B}\cdot\mathbf{U}$. Since the value of $\max_i(\|\mathbf{b}'_i\|)$ cannot increase during the algorithm we have that $\max \|\mathbf{b}'_i\| \leqslant \max \|\mathbf{b}_i\|$ and, hence, the norms of the coefficients of $\mathbf{U}$ are bounded from above at each loop as $|\mathbf{U}_{i,j}|_{i,j\in\{0,1\}} \leqslant \frac{\max_i^2(\|\mathbf{b}_i\|)}{|\det \mathbf{B}|}$. Since $\mathbf{U}$ is integral and $|\det \mathbf{B}| \geqslant 1$, the maximal bitsize of the entries of $\mathbf{U}$ is at most $\log_2 \max_i \|\mathbf{b}_i\|^2$. The bound on the bitsizes of the involved integers follows from the bound on the numerator of $\mu_{1,0}$ which is equal to $\mathbf{G}_{1,0} \leqslant \max_{i,j} \mathbf{G}_{i,j}$ and the fact that $d \cdot \mathbf{G} \in \mathbb{Z}^{2\times 2}$ and hence $d^2 \cdot \mathbf{B} \in \mathbb{Z}^{4\times 2}$. Add one bit to account for the sign to get the bound. □

3.2 Constant-time BKZ-2 in dimension 4

As all necessary routines have been presented, we are ready to discuss an algorithm that reduces 4-dimensional lattices. For that one could use the celebrated LLL algorithm [LLL82]. Yet, it is not straightforward how to make this algorithm constant time due to nontrivial branching within the algorithm. To obtain an LLL-like lattice reduction algorithm we propose Algorithm 3.5 mimicking the BKZ algorithm [Sch87] with block size 2.

Our algorithm: BKZ-2 The BKZ lattice reduction algorithm proceeds by considering consequent projective lattices $\pi_i([\mathbf{b}_i, \mathbf{b}_{i+1}])$ for $0 \leqslant i < 3$. It Lagrange reduces these lattices and returns the corresponding transformation matrix $\mathbf{U}$. This matrix is then applied to $[\mathbf{b}_i, \mathbf{b}_{i+1}]$ ensuring that the projection of these vectors against first $\max(0, i-1)$ basis vectors is small. To shorten the projection *on* these first basis vectors Algorithm 3.2

is invoked. After we reduce the 'last' projective lattice $\mathcal{L}(\pi_2([\mathbf{b}_2, \mathbf{b}_3]))$ we say that a tour has passed, so we then consider the 0-th projective lattice $\mathcal{L}(\pi_0([\mathbf{b}_0, \mathbf{b}_1]))$ again.

We run exactly T_{BKZ} tours before the algorithm terminates. Bounding this number of tours is a nontrivial task and is considered below. There we also provide the analysis of the output quality of Algorithm 3.5.

Our algorithm uses the constant-time subroutines introduced in Section 3.1 and iterates through them a fixed number of times, in a fixed order independent of the lattice. Parameters of the subroutines on which their runtime depends (such as the number of iterations in Lagrange's algorithm), are also given independently of the lattice. It is therefore constant-time.

Algorithm 3.5 BKZ-2

Input: $\mathbf{B} \in \mathbb{Z}^{4\times 4}$ – a basis matrix.
$\mathbf{G} \in \mathbb{Z}^{2\times 2}$ – a Gram matrix of the basis $\mathbf{B}$.
$T_{\mathrm{BKZ}} \in \mathbb{N}$ – the number of tours.
$T_{\mathrm{Lagr}} \in \mathbb{N}$ – the number of iterations in Lagrange.
Output: $\mathbf{B} \in \mathbb{Z}^{4\times 4}$ – a basis matrix.

Compute $\mu_{1,0}$ and $r_{i,j}$ for $0 \leqslant i \leqslant 1, j \leqslant i$.
for $c = 1$ to T_{BKZ} **do**
  for $i = 0$ to 2 **do**
    $\mathbf{B}, \mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\} := \mathsf{size\_red}(\mathbf{B}, i+1, \mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\})$
    $\mathbf{H} := \begin{pmatrix} r_{i,i} & \mu_{i+1,i} \cdot r_{i,i} \\ \mu_{i+1,i} \cdot r_{i,i} & \mu_{i+1,i}^2 r_{i,i} + r_{i+1,i+1} \end{pmatrix}$
    $\mathbf{U} := \mathsf{Lagrange}(\mathbf{H}, T_{\mathrm{Lagr}})$
    $\mathbf{B}[i, i+1] := \mathbf{B}[i, i+1] \cdot \mathbf{U}$
    $\mathbf{G} := \mathsf{update\_after\_svp}(\mathbf{G}, \mathbf{U}, i)$
    $\{\mu_{i',j'}\}, \{r_{i',j'}\} := \mathsf{Cholesky}(\mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\}, i, i+1)$
    $\mathbf{B}, \mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\} := \mathsf{size\_red}(\mathbf{B}, i, \mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\})$
    $\mathbf{B}, \mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\} := \mathsf{size\_red}(\mathbf{B}, i+1, \mathbf{G}, \{\mu_{i',j'}\}, \{r_{i',j'}\})$
return B

BKZ-2 analysis In the analysis of our Algorithm 3.5 we rely on the Sandpile Model Assumption (SMA). It has been introduced in [MV10] as a tool to model and to analyze the BKZ algorithm [HPS11]. Informally, the assumption replaces inequalities of Lemma 1 by equalities and models the profile $(\|\mathbf{b}_i^*\|)_{0\leqslant i<\beta}$ of a dimension-β HKZ reduced basis of a lattice $\mathcal{L}$ as $\|\mathbf{b}_i^*\| = \frac{1}{2}\log\gamma_{\beta-i} + \frac{1}{\beta-i}\sum_{j=i}^{\beta-1}\|\mathbf{b}_i^*\|$. We focus here on the particular case of SMA applied to dimension-4 bases with $\beta = 2$, we refer the reader to [HPS11] for the general case. The lemma below provides us with a bound on the number of BKZ tours T_{BKZ} and the bound the number of Lagrange tours T_{Lagr} sufficient to argue on the length of the shortest vector in the output basis of BKZ-2.

Lemma 6. *[adaptation of [HPS11, Lemma 12]] Let* $\mathbf{B}$ *be an upper triangular basis of a lattice* $\mathcal{L} \subset \mathbb{Z}^4$ *with* $D = \det(\mathcal{L})$. *Let* $\mathbf{G}$ *be its Gram matrix and* $r_{i,j}$ *be its Gram-Schmidt coefficients. Then on its input Algorithm 3.5 returns a basis* $\mathbf{B}'$ *of* $\mathcal{L}$ *that satisfies:*

$$\|\mathbf{b}_0'^*\| \leqslant 2 \cdot \left(\frac{4}{3}\right)^{3/2} \cdot \left(\prod_{i=0}^{3}\|\mathbf{b}_i'^*\|\right)^{1/4} \tag{1}$$

given that

$$T_{BKZ} \geqslant \frac{2\log\left(\log\max_i \frac{\|\mathbf{b}_i^*\|}{(D)^{1/4}} + \sqrt{5}\cdot(\log(4/3))^{1/2}\right)}{\log(8/7)}. \tag{2}$$

If $B = \max_i \|\mathbf{b}_i\|$ *is bounded, then is it sufficient to set* $T_{Lagr} = 2 + 2\lceil(\log_{\sqrt{3}} 2) \cdot (9 \log_2 B + 12)\rceil$ *to ensure Equation* (1) *in constant time.*

Remark 2. We note that the statement of the above lemma holds both for Euclidean norm and the square root of the reduced norm used in SQIsign.

Proof. The existence of T_{BKZ} such that the output of Algorithm 3.5 satisfies Equation (1) is provided by the proof of the original lemma [HPS11, Lemma 12]. To deduce an upper bound of the exact value of T_{BKZ} we will specialize the proof of [HPS11, Thm. 2] by taking the dimension n of the lattice equal to 4 and the blocksize β equal to 2. In order to specialize the proof, we borrow the following notations:

- Denote by $\mathbf{B}^{(t)}$ the basis obtained after t tours of BKZ-2 and define
$$\mathbf{x}_i^{(t)} = \log\left(\|(\mathbf{b}_i^{(t)})^*\|\right) - \log(\det \mathcal{L})/4.$$
- Denote by $\mathbf{B}^{(0)}$ the input basis to the BKZ-2 algorithm and define for $r_{i,j}$the Gram-Schmidt coefficients of $\mathbf{B}^{(0)}$.
$$\mathbf{x}_i^{(0)} = (\log(r_{i,i})/2 - \log(\det(\mathcal{L}))/4)_{0 \leqslant i < 4}\,.$$
- Denote by $\mathbf{B}^{\infty}$ a BKZ-2 reduced basis and define $\mathbf{x}_i^{\infty} = \mathbf{x}_3^{\infty} - (3-i) \cdot \log(4/3)/2$ for $0 \leq i \leq 3$. The SMA applied to dimension-4 lattices with blocksize $\beta = 2$ yields [HPS11, Section 4.2]
$$\mathbf{x}^{\infty} = ((3/2)\log\sqrt{4/3}, (1/2)\log\sqrt{4/3}, -(1/2)\log\sqrt{4/3}, -(3/2)\log\sqrt{4/3}). \quad (3)$$

In the proof of [HPS11, Thm. 2] for the t-th BKZ tour, Hanrot, Pujol and Stehlé deduce that
$$\|\mathbf{x}^{(t)} - \mathbf{x}^{\infty}\|_2 \leqslant \left(1 - \frac{4}{2 \cdot 4^2}\right)^{t/2} \cdot \|\mathbf{x}^0 - \mathbf{x}^{\infty}\|_2 = \left(\frac{7}{8}\right)^{t/2} \cdot \|\mathbf{x}^0 - \mathbf{x}^{\infty}\|_2. \quad (4)$$

As soon as $\|\mathbf{x}^{(t)} - \mathbf{x}^{\infty}\|_2 \leqslant 1$, we have $\forall\ 0 \leqslant j < 4 : |\mathbf{x}_j^{(t)} - \mathbf{x}_j^{\infty}|^2 \leqslant 1$ and, hence, $|\mathbf{x}_0^{(t)} - \mathbf{x}_0^{\infty}| \leqslant 1$. Substituting SMA estimation for $\mathbf{x}_0^{\infty}$ into the latter inequality, we get $\mathbf{x}_0^{(t)} \leqslant \frac{3}{2} \cdot \log\frac{4}{3} + 1$. By definition of $\mathbf{x}_i^{(t)}$ we then have
$$\|(\mathbf{b}_0^*)^{(t)}\| \leqslant 2 \cdot \left(\frac{4}{3}\right)^{3/2} \cdot (\prod_{i=1}^{4}(||\mathbf{b}_i^*||))^{1/4},$$
which gives the claimed norm bound for a sufficiently large t which value we want to estimate.

To estimate the value of t we now deduce when the upper bound of Equation (4) is at most 1. We have that $\|\mathbf{x}^0 - \mathbf{x}^{\infty}\|_2 \leqslant \|\mathbf{x}^0\| + \|\mathbf{x}^{\infty}\|_2$ by triangular inequality. By definition of $\mathbf{x}^0$ we have $\|\mathbf{x}^0\|_2 \leqslant \log \max_i \frac{\|\mathbf{b}_i^*\|}{(D)^{1/4}}$ and straightforward calculations using Equation (3) give us $\|\mathbf{x}^{\infty}\|_2 = \sqrt{5} \cdot (\log(4/3))^{1/2}$. Hence we have $\|\mathbf{x}^0 - \mathbf{x}^{\infty}\|_2 \leqslant R$ for $R = \log \max_i \frac{\|\mathbf{b}_i^*\|}{(D)^{1/4}} + \sqrt{5} \cdot (\log(4/3))^{1/2}$. Then finding t boils down to solving the following inequality:
$$\left(\frac{7}{8}\right)^{t/2} \cdot R \leqslant 1.$$
By taking logarithm we obtain:
$$t/2 \geqslant \frac{\log -R}{\log(7/8)},$$

$$t \geqslant \frac{2\log R}{\log(8/7)} = \frac{2\log\left(\log \max_i \frac{\|\mathbf{b}_i^*\|}{(D)^{1/4}} + \sqrt{5}\cdot(\log(4/3))^{1/2}\right)}{\log(8/7)},$$

hence Equation (2).

Algorithm 3.1 and Algorithm 3.3 are constant time. Algorithm 3.2 is constant time for a given $i \in \{1,2,3\}$. At each iteration of the main loop it is being called 3 times for $i \in \{1,2\}$, 2 times for $i = 4$ and the overall time spent within Algorithm 3.2 in any given loop is constant. Thus, we only need to prove a bound on T_{Lagr} for Algorithm 3.4. At each iteration of the inner loop $\mathbf{H}$ corresponds to the following basis

$$\pi_i(\mathbf{B}[i:i+1]) = \begin{pmatrix}\mathbf{b}_i^* & \mathbf{b}_{i+1}^* + \mu_{i+1,i}\cdot\mathbf{b}_i^*\end{pmatrix} \in \mathbb{Q}^{4\times 2}.$$

Both Euclidean norms of $\mathbf{b}_{i+1}^*$ and $\mathbf{b}_{i+1}^* + \mu_{i+1,i}\cdot\mathbf{b}_i^*$ are bounded from above by B respectively for each $i < 3$. By Lemma 2 both denominators d_1 and d_2 of $\mathbf{b}_{i+1}^*$ and $\mathbf{b}_{i+1}^* + \mu_{i+1,i}\cdot\mathbf{b}_i^*$ respectively are bounded by $2^{4+(9/2)\cdot\log_2 B+2}$. If we scale $\mathcal{L}(\pi_i(\mathbf{B}))$ by $S = \operatorname{lcm}(d_1,d_2) \leqslant 2^{8+9\log_2 B+4} = 2^{9\log_2 B+12}$ we obtain an integral lattice with $\lambda_1 \geqslant 1$.

Notice that all dimension-2 projective lattices $\pi_i(\mathbf{B})$ have their determinants greater than or equal to 1 since it is true at the initial step (since $\mathbf{B} \subseteq \mathbb{Z}^4$ and is upper triangular) and $\min_i \det(\pi_i(\mathbf{B}))$ cannot decrease during the execution of the algorithm. Using Lemma 5 we obtain

$$T_{\text{Lagr}} \leqslant \max_i \log_{\sqrt{3}} \|S\mathbf{b}_i\| + 2 \leqslant (9\log_2 B + 12)\log_{\sqrt{3}} 2 + \log_{\sqrt{3}} B + 2$$

which gives the statement for T_{Lagr}. □

4 Constant time implementation of BKZ-2

In order to test and verify our algorithm and its performance, we implemented it in C. More precisely, we implemented it so that it could replace the variable-time LLL implementation from [CSSDF+23] as easily as possible, which required several adaptations. First, we do not use the usual Euclidean norm, but the reduced norm of the quaternion algebra. Second, we can assume that our input is an integer matrix in Hermite Normal Form (HNF), with columns representing algebra elements. This is the case because lattices in [CSSDF+23] are represented by such matrices and a common denominator, and the denominator is not passed to LLL. Finally, we had to write constant-time integer and rational arithmetic which works as a drop-in replacement for the subset of the *intbig* module of the implementation of the NIST submission [CSSDF+23] used by LLL, because that implementation lacks constant-time arithmetic so far.

Therefore, we implemented the necessary integer and rational arithmetic in constant time using fixed-size integers, the secure low-level functions of the GMP library [Gt] and a self-made implementation of the constant-time greatest common divisor (gcd) algorithm from [BY19]. Our integers ensure compatibility of our implementation with the NIST submission, but it is not optimized, as integer arithmetic was not our main focus. This compatibility allows us to run the LLL implementation from [CSSDF+23] on our integers, in order to compare its performance to ours as we do in Section 4.4. We will refer to the LLL implementation from [CSSDF+23] using our constant-time integers instead of variable-time *intbig* module of the NIST submission, as *SQIsign LLL* in the following.

Before benchmarking in Section 4.4, we will ensure a realistic test setting by studying which lattices LLL gets called on in SQIsign in Section 4.1, and present the results of various experiments we did to ensure correctness (Section 4.2) and constant-timeness (Section 4.3) of our code. Our implementation and test data can be found at `https://github.com/CTlll-SQIsign/CTlll-SQIsign`.

4.1 Use of lattice reduction in SQIsign

Given our goal is to provide SQIsign with a constant-time lattice reduction algorithm, we need to get an understanding of SQIsign's use of lattice reduction. This will allow us to run the tests in the remainder of this section in a realistic setting.

LLL calls in SQIsign We therefore studied the arguments of LLL in the calls it received in executions of SQIsign key generation and signing. In the submitted code, there are a total of 6 different calls to LLL, three from within the Special Eichler Norm (SEN) algorithm, which is a variant of KLPT [KLPT14] used in the ideal-to-isogeny translation [DFLLW23], and three directly from the signature and key generation protocols.

All of these calls receive integer matrices in Hermite Normal Form (HNF) which represent integral ideals in a quaternion algebra (the common denominator is left out during the reduction). An integral ideal is a rank 4 lattice which is closed under multiplication in the algebra. The algebra is fixed for each security level. An ideal's size can be measured by its norm, which is defined as the gcd of the reduced norms ($N(x) = x\bar{x}$, cf [Voi21] Chapter 3.3) of all its elements.

Since all of the 6 call are in while loops, the number of executions of each call can be larger than one. In order to see which inputs to LLL are the most frequent, we therefore measured the average number of occurrences of each of the 6 calls to LLL in 1000 executions of SQIsign at level 1 and 100 executions at both higher levels, as well as the maximum size in bits of the norm of an ideal input in each call. The results are in Table 1.

Table 1: Integer sizes in bits and frequency of the 6 calls to LLL in [CSSDF+23]

LLL call	Keygen	Sign rand	Sign	SEN O_0	SEN alternate	SEN rare
LVL1 max norm	76	469	137	258	385	192
LVL1 call count	1,0	1,5	1,5	21	3,1	0,01
LVL3 max norm	98	684	198	383	482	–
LVL3 call count	1,0	1,6	1,6	24	2,0	0
LVL5 max norm	146	919	259	506	757	–
LVL5 call count	1,0	1,5	1,5	21	3,0	0

Conclusions The most frequent of these 6 calls to LLL is therefore the one corresponding to the Special Eichler Norm algorithm (SEN) using LLL on a lattice which is a left O_0-ideal. Here and in the following, O_0 is the lattice generated by $(1, i, (i+j)/2, (1+ij)/2)$, which is a subring of the algebra. This call occurs more than 20 times on average when running key generation followed by signature, and has therefore the highest impact on the performance of SQIsign. Runtime measures to confirm the relative impact of these LLL calls on the current, variable-time SQIsign implementation's runtime failed because the LLL runtimes are negligible, as all LLL calls together take less than 3% of the signing time.

As a consequence, most of our tests and benchmarks in the remainder of this section will focus on left O_0-ideals of norm not more than 2 bits larger or 8 bits smaller than the maxima observed for the SEN O_0 calls in our experiments.

4.2 Output quality of constant time BKZ-2

The different SQIsign variants use reduced lattice bases for different means, and therefore have slightly different requirements on the output of LLL. Most often, they require at least one or two of the vectors to be close to the minima of the lattice. To summarize these requirement, most of the SQIsign literature [DFKL+20, Ler22, DLRW24] refers to Minkowski-reduced bases as a sufficiently good output (even though this is not always necessary).

We will therefore assume in the following that the purpose of lattice reduction in SQIsign is to obtain a Minkowski reduced basis of an ideal I of norm $N(I)$. This means we ask for a basis $\mathbf{b}_0, \dots \mathbf{b}_3$ of I such that for all $0 \leq i \leq 3$, $\mathbf{b}_i$ is the shortest possible that allows $\mathbf{b}_0 \dots \mathbf{b}_i$ to be extended into a basis. Even though we do not prove that our BKZ-2 algorithm guarantees to output such basis, in practice it holds due to small dimension. Since in dimension 4 Minkowski reduced basis attains successive minima [Wae56], the product of the norms of output basis vectors should satisfy Minkowski's second theorem. In the SQIsign setting Minkowski's second theorem translates to the inequality [KLPT14, Section 3.1]:

$$16 \cdot \mathrm{nrd}(\mathbf{b}_0) \cdot \mathrm{nrd}(\mathbf{b}_1) \cdot \mathrm{nrd}(\mathbf{b}_2) \cdot \mathrm{nrd}(\mathbf{b}_3) \leq 4p^2(N(I))^4, \tag{5}$$

where nrd is defined in Section 2.2 and p is the prime that defines the quaternion algebra.

We ran experiments to verify that our implementation outputs bases that satisfy Equation (5) for LVL1, LVL3, and LVL5 parameters. We execute the reduction for different $T_{\mathrm{BKZ}}, T_{\mathrm{Lagr}}$ on 30 different lattices for each security level and check how often Equation (5) is violated by the output basis. For each $1 \leq T_{\mathrm{BKZ}} \leq 20$ we run our BKZ2 with different T_{Lagr} values varying from 1 to 12. The result for the three security levels are shown in Figure 3, where for readability we plot only the results for $T_{\mathrm{BKZ}} \leq 5$.

While for small T_{Lagr} the output bases sometimes fail to be Minkowski reduced, for $T_{\mathrm{Lagr}} \geq 10$ and $T_{\mathrm{BKZ}} \geq 3$ all 30 reductions were successful. The result shows that in practice $T_{\mathrm{BKZ}}, T_{\mathrm{Lagr}}$ can be chosen relatively small (much smaller than the theory predicts). We remark that on non-failed instances, not only Equation (5) was satisfied, but also the guarantee on the shortest vector in the returned basis from Lemma 6.

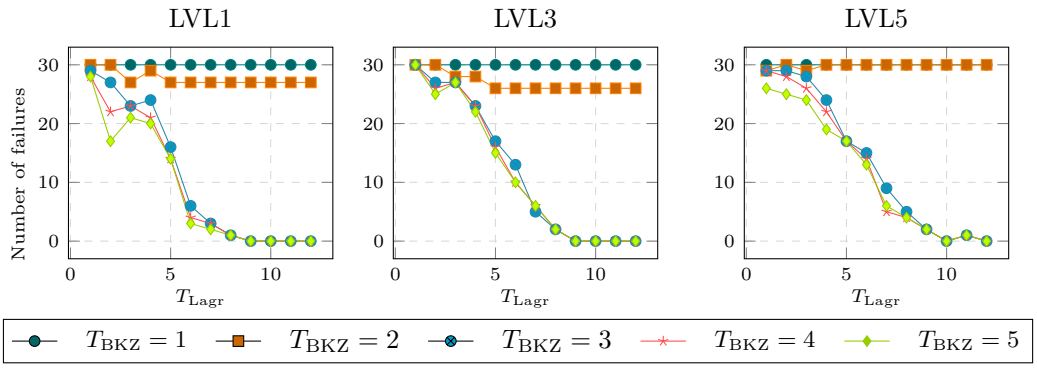

Figure 3: Number of failures of our BKZ-2 output on NIST LVL1–5 SQIsign lattices

Additionally, we present the 'reverted' plots that lead to the same conclusion. In Figure 4 we group the quality results on the same inputs as above as follows: for each fixed value of T_{Lagr}, we compute how often our BKZ2 failed for different values of T_{BKZ}. As expected, each plot is non-increasing illustrating that a tour of our BKZ2 algorithm can never increase the norm of the first basis vector. Inputs to these experiments are the same as in Figure 3.

4.3 Verification of the constant-time implementation

In order to ensure that our implementation is constant-time, we used the tools ctgrind [Lan10] and RTLF [DME+24, ME24].

ctgrind: For a first check whether our implementation is constant-time, we used ctgrind [Lan10]. The tool did not detect secret-dependent branching or other secret-dependent

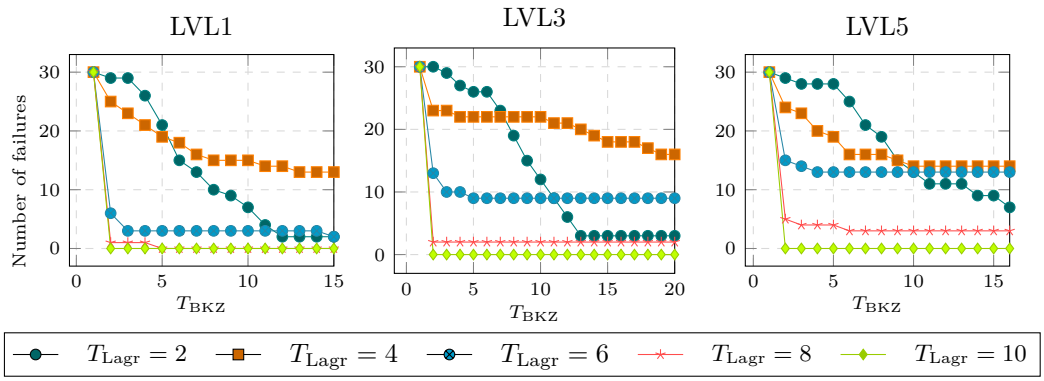

Figure 4: Number of failures of our BKZ-2 output on NIST LVL1–5 SQIsign lattices

program behaviour in our BKZ-2 implementation compiled with gcc and the -DNDEBUG flag and no additional compiler flags on a ThinkPad P1 Gen4 running RHEL 8.10.

RTLF: To further verify constant-timeness we used RTLF [DME+24, ME24]. It is a tool that runs statistical tests given on input two samples $x = (x_1, \ldots, x_N)$ and $y = (y_1, \ldots, y_N)$ representing runtime measurements of a routine we want to prove (or disprove) to be constant time. RTLF tries to detect statistical differences between x and y. The output either says that the tests found no differences (meaning that x and y *likely* follow the same distribution), or it detects a difference between x and y concluding that the distributions are likely not the same.

Our setting. We ran 144000 iterations of our BKZ-2 implementation (compiled by gcc with flag -DNDEBUG) on as many distinct inputs in four threads in parallel on a laptop with an Intel i7-9750H processor running Ubuntu 24.10 with TurboBoost disabled. The computation was launched after warming up the processor, and after the last samples all threads continued running for a while to ensure that all of them were done when the first one stopped. Besides separating the machine from the network, no other countermeasures against noise were taken.

The inputs were left O_0-ideals (see Section 4.1 for the definition) of norm between $n_i = 12890707586852862$ and $n_u = 187181532229268607942$ in the quaternion algebra $B_{p,\infty}$ where $p = 335065877789763716363842902011413 87$. BKZ-2 was run with $T_{\mathrm{BKZ}} = 10$, $T_{\mathrm{Lagr}} = 6$, and the integer size was $p_{\mathrm{BKZ}} = 704$ bits. These parameters where chosen because they are small enough to allow us to run a large number of samples in reasonable time, but large enough that p does not fit in a 64-bit word, while all entries of the lattice do so for the smaller half of the samples.

RTLF results on norm: Our input lattices are in the HNF form (as in Section 4.1). We separate the measurements with respect to the size of the coefficient N_1 in the upper left corner of the (upper triangular) HNF. In O_0-ideals in HNF, N_1 is usually equal to the ideal norm up to a small factor, and therefore it is the largest entry in the lattice basis. That is, our vector x stores 72 000 measurements for the lattices with smaller N_1's, and y stores 72 000 measurements for the lattices with larger N_1's. RTLF detected no difference in the runtimes of the 2 sets with the type-1 error parameter 0.09. The measurement are presented on the scatter plot in orange in Figure 5. It shows runtimes in cycles (on the vertical axis) for all of our sample ordered in increasing N_1 along the horizontal axis.

More details of the test, including our complete data, is available on our github repository https://github.com/CTlll-SQIsign/CTlll-SQIsign.

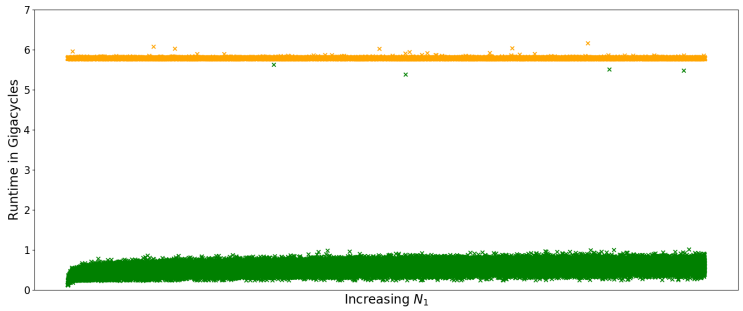

Figure 5: Plots of the runtimes (clockcycles) of our BKZ-2 (orange) and SQIsign LLL (green), on random lattices. The left half corresponds to set x (norm below the median), the right to y (norm above the median).

RTLF results on skewness: In addition, we ran the same test for the orthogonal defect of the same BKZ-2 reduced bases. The orthogonal defect of a basis $\mathbf{B} \in \mathbb{Z}^{4\times 4}$ can be considered as a measure of the "skewness" of $\mathbf{B}$ and is defined as:

$$\theta(\mathbf{B}) = \left(\frac{\prod_{i=0}^{3} \|\mathbf{b}_i^*\|}{\prod_{i=0}^{3} \|\mathbf{b}_i\|} \right)^{1/4}.$$

The orthogonal defect $\theta(\mathbf{B})$ is equal to one if and only if $\mathbf{B}$ is orthogonal. Similarly to the test above, no difference in the runtimes of BKZ2 was detected by RTLF as can be seen in Figure 6.

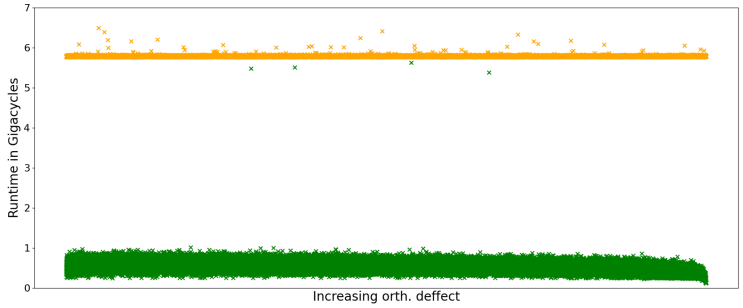

Figure 6: Plots of the runtimes (clockcycles) of our BKZ-2 (orange) and SQIsign LLL (green), on random lattices. The left half corresponds to runtimes on lattices with orthogonality defect below the median, the right half – to runtimes on lattices with orthogonality defect above the median.

Control RTLF on SQIsign LLL: To ensure that our test set separation in two subsets x and y following the first coefficient of the lattice (which corresponds to the ideal norm) is reasonable, we ran SIQsign LLL on the 144 000 exact same inputs and in the same conditions as the BKZ-2 runs for RTLF. With the same separation in two sets x and y as for BKZ-2 above, RTLF detected a strong dependency of the SQIsign LLL runtime on these sets, concretely it outputs "Test determined difference for the following quantile indices: 1, 2, 3, 4, 5, 6, 7, 8, 9.". The lower green plot of Figure 5 shows that the runtime

of LLL grows (slowly but visibly) with N_1. With the same settings but a separation on the skewness instead of the norm, RTLF also detected a difference for SQIsign LLL.

Further verification: Given that the independence of the runtime from the norm and skewness of the lattice only shows that BKZ2 is better than SQIsignLLL with respect to constant-time, but not that there is no property of the lattices on which the BKZ2 runtime depends, we ran an additional measure. On 10 threads in parallel and on a laptop with an Intel i7-9850H processor running Ubuntu 20.04 (with TurboBoost disabled), we run BKZ2 on 120000 randomly generated O_0-ideals, then shuffled these inputs and run the measure again. We separated the lattices into 2 sets measurement (set X being the faster half, and set Y the slower), and run RTLF (with type-1 error parameter 0.09) with the runtimes from the second measure and these sets. The outcome is that for BKZ2 the runtime in the second measure (on the same lattices) is independent from the one in the first measure. This indicates that the runtime of our BKZ2 implementation does not depend on any property of lattices which occurred in our random input set.

4.4 Comparison against non constant time LLL in SQIsign

Given our goal is to provide SQIsign with a constant-time lattice reduction algorithm, we will now evaluate the performance of our algorithm compared to the LLL implementation of SQIsign's NIST submission [CSSDF+23] in a meaningful setting based on our study of SQIsign's use of LLL in Section 4.1.

Benchmark setup: Since the call through SEN O_0 occurs more than 20 times on average when running key generation and signature successively, and has therefore the highest impact on the performance of SQIsign, we set up our benchmarks as follows: As input, we randomly sampled 100 left integral O_0-ideals with a norm at most 2 bits above and 8 bits below the measured maxima for SEN O_0 calls for each level. These ideals were put into HNF (disregarding the denominator). The resulting integer matrices were used as inputs. Using the fact that the norms of elements in these matrices do almost reach the square of the ideal norm, we use this value to set the integer sizes, following the formula from Lemma 2. Since the experiments in Section 4.2 suggest that fairly small numbers of iterations are sufficient, we set the parameters accordingly. The chosen tour numbers do not respect the proven bounds from Lemma 6, but are such that even if either T_{Lagr} xor T_{BKZ} are decreased by one, our quality tests and tests on our benchmark inputs still pass. These experiments show that reduction failures with these parameters are unlikely, even if we cannot bound their probability mathematically. Applications which can accept that in some rare cases our algorithm might output bases which are not fully reduced can therefore use these parameters for constant-time lattice reduction with reasonable speed and a very low failure rate. Other choices are possible depending on the application.

For comparison, SQIsign LLL was used. This is the LLL implementation from [CSSDF+23], however, instead of the original, variable-time integer implementation, it is run using the same implementation of integer and rational arithmetic as our constant-time implementation. The change of integers was needed since the difference in speed of the two integer implementations would otherwise make any comparison meaningless for evaluation the efficiency of our constant-time lattice reduction, as the first column of Table 2 shows.

For the benchmarks, both implementations were compiled using gcc with the flag -DNDEBUG, on a intel i7-11850H processor with TurboBoost disabled. The results are given in Table 2, and show that in this setting, our implementation has an about $\times 5$ slowdown compared to SQIsign LLL.

Table 2: Average runtime over 100 inputs, in units of 1.000.000.000 cycles

Algorithm	original LLL old integers	SQIsign LLL our integers	our BKZ-2	T_{BKZ}	T_{Lagr}	integer size (64-bit words)
LVL1	0,0016	7,39	35,9	4	10	37
LVL3	0,0024	17,2	81,8	4	11	55
LVL5	0,0031	26,4	133	4	13	72

Optimizations: There are some possible optimizations of our implementation. A first point to investigate in future work would be the use of compiler optimization features and whether these negatively affect constant-timeness. Furthermore, profiling (using gprof [GKM82]) showed that more than 90% of time is spent on gcd computations to reduce fractions, which is inherent to the use of rational numbers. Using other number types or representations is therefore a promising direction for future research. However we did not explore this, since it would limit compatibility with the code from the NIST submission.

The high dependency of the runtime of the gcd on the size of integers motivated us to investigate whether we could further reduce the size of integers used in our algorithm, by taking advantage of the structure of the inputs it receives in SQIsign. Using the fact that the norm of an ideal (or half of it, if it is even) divides every entry of the Gram matrix of the ideal, we obtain a lower bound on the gcd G of all entries of the Gram matrix. Dividing G out before starting the Gram matrix, therefore allows us to use significantly smaller integers than SQIsign LLL, if a lower bound of the occurring norms is known. The exact value can be obtained by replacing $\log(B)$ by $\log(B) - b$ where b is a lower bound on the ideal norm in the formula from Lemma 2. In the same setting as above, the comparison of this optimized version against SQIsign LLL (therefore with different integer sizes) yields the results in Table 3.

Table 3: Average runtime over 100 inputs, in units of 1.000.000.000 cycles

Algorithm	SQIsign LLL	integer sizes (64-bit words)	our BKZ-2	T_{BKZ}	T_{Lagr}	integer size (64-bit words)
LVL1	7,39	37	9,59	4	10	20
LVL3	17,2	55	19,5	4	11	28
LVL5	26,4	72	38,9	4	13	37

Impact on SQIsign: The benchmarks of this optimized version show a smaller, but still significant slowdown, whose impact on SQIsign's signing time is unclear. First, it is not clear whether the multiple uses of lattice reduction in SQIsign can tolerate rare cases of insufficient reduction and for example restart without leaking secrets if an insufficient reduction occurs. Given the complex internals of SQIsign, which also rejects some fully reduced lattices after reduction because of the length of their successive minima, analysing the consequences of rare insufficient reductions is out of scope of this work. If rare cases of insufficient reduction can be tolerated, SQIsign could use our BKZ2 with T_{Lagr} and T_{BKZ} below the proven values. Assuming this to be the case, we attempted to benchmark the SQIsign implementation from [CSSDF+23] with its original, non-constant-time integers and our BKZ2-algorithm instead of the original LLL. Even with fairly high iteration counts ($T_{\mathrm{Lagr}} = 30, T_{\mathrm{BKZ}} = 100$) there was no noticeable slowdown of the overall key generation and signing procedures. This is not surprising since the runtime of LLL in the signature is negligible. However, using our not optimized, constant-time integers, the runtime of each of the considered lattice reduction algorithms is larger than the total signing time of SQIsign as reported in [CSSDF+23]. As a consequence, the impact of making lattice reduction constant-time cannot be more deeply studied before other parts of SQIsign, such

as its integer functions, have an optimized constant-time implementation. Implementing complex integer functions not required by LLL is however not the purpose of this work, so we leave that analysis for the time when such an implementation exists. Similarly, the exact impact of LLL's runtime on the signing time of a constant-time SQIsign implementation cannot be studied yet since no such implementation exists so far, and many complex algorithms in the scheme have no constant-time adaptation yet. Currently we can only note that, in the implementation from the NIST submission, the bottleneck are finite field multiplications in the ideal-to-isogeny translation, and LLL (or equivalently, our BKZ2) is almost negligible (less than 3% of the runtime if built in Release mode). This could however change if constant-time integer arithmetic was used. For other variants such as SQIsignHD [DLRW24] the efficiency of LLL is more important, and the impact of a slowdown could be more severe. In this context, the 1.5× slowdown reported in Table 3 seems encouraging, especially given that further optimizations are possible.

5 Conclusion

In this paper we presented a first constant time LLL-like algorithm for dimension-4 lattices. Our implementation has been verified for constant timeness through different tools (ct-grind, RLTF). In practice it outputs Minkowski reduced bases and thus can replace the (currently) non constant time LLL subroutine in SQIsign.

Additional work Finding Minkowski reduced bases is not the only lattice related task in SQIsign. Enumerating close vectors in dimension-2 lattices is another task used in some variants of the signature (e.g. in the SQIsign 1st Round NIST submission [CSSDF+23] and the SQIsign2d-East variant [NO24], but not in SQIsignHD [DLRW24] nor in SQIsign2d-West [BFD+24]). Enumerating close vectors asks, for given a 2-dimensional lattice and a target vector, to output all lattice vectors within a given distance to the target. Presently there is no known constant time realization of this subroutine. In our Github repository we provide a proof-of-concept implementation in Python of constant time close vector enumeration on dimension-2 lattices that can serve as a base for a more efficient constant time implementation of the routine. As it is currently unclear which SQIsign variant is the most promising for practical purposes, we omit a complete description of the algorithm here and leave a C-implementation of it as a future work in case the SQIsign variants that use this enumeration will be interesting from a practical viewpoint.

References

[AAA+25] Marius A Aardal, Gora Adj, Arwa Alblooshi, Diego F Aranha, Isaac A Canales-Martínez, Jorge Chávez-Saab, Décio Luiz Gazzoni Filho, Krijn Reijnders, and Francisco Rodríguez-Henríquez. Optimized one-dimensional sqisign verification on intel and cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):497–522, 2025. 1

[BFD+24] Andrea Basso, Luca De Feo, Pierrick Dartois, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. SQIsign2D-west: The fast, the small, and the safer. Cryptology ePrint Archive, Paper 2024/760, 2024. https://eprint.iacr.org/2024/760. 1, 3, 6, 22

[BY19] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):340–398, May 2019. 15

[CCI+24] Wouter Castryck, Mingjie Chen, Riccardo Invernizzi, Gioella Lorenzon, and Frederik Vercauteren. Breaking and repairing SQIsign2D-East. *Cryptology ePrint Archive*, 2024. https://eprint.iacr.org/2024/771. 3

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 423–447, Cham, 2023. Springer Nature Switzerland. 1, 3

[CEMR24] Maria Corte-Real Santos, Jonathan Komada Eriksen, Michael Meyer, and Krijn Reijnders. AprèsSQI: Extra fast verification for SQIsign using extension-field signing. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024*, pages 63–93, Cham, 2024. Springer Nature Switzerland. 2

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer, Graduate texts in mathematics series, New York, NY, USA, 1993. 4

[CSSDF+23] Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQIsign: algorithm specifications and supporting documentation (2023). Submission to NIST, accessible at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures, 2023. accessed repeatedly since July 023. 1, 2, 3, 15, 16, 20, 21, 22

[DF24] Max Duparc and Tako Boris Fouotsa. SQIPrime: A dimension 2 variant of SQISignHD with non-smooth challenge isogenies. Cryptology ePrint Archive, Paper 2024/773, 2024. https://eprint.iacr.org/2024/773. 1, 3, 6

[DFKL+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 64–93, Cham, 2020. Springer International Publishing. 2, 3, 5, 6, 16

[DFLLW23] Luca De Feo, Antonin Leroux, Patrick Longa, and Benjamin Wesolowski. New algorithms for the Deuring correspondence. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 659–690, Cham, 2023. Springer Nature Switzerland. 2, 3, 16

[DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. SQIsignHD: New dimensions in cryptography. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 3–32, Cham, 2024. Springer Nature Switzerland. 3, 6, 16, 22

[DME+24] Martin Dunsche, Marcel Maehren, Nurullah Erinola, Robert Merget, Nicolai Bissantz, Juraj Somorovsky, and Jörg Schwenk. With great power come great side channels: Statistical timing side-channel analyses with bounded type-1 errors, 2024. https://www.usenix.org/system/files/sec24fall-prepub-264-dunsche.pdf. 2, 17, 18

[GKM82] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. Gprof: A call graph execution profiler. In *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*, SIGPLAN '82, page 120–126, New York, NY, USA, 1982. Association for Computing Machinery. 21

[Gt] Torbjörn Granlund and the GMP development team. GNU MP: The GNU Multiple Precision Arithmetic Library. http://gmplib.org/. 2, 4, 15

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 447–464, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 2, 13, 14

[JMKR23] David Jacquemin, Anisha Mukherjee, Péter Kutas, and Sujoy Sinha Roy. Ready to SQI? safety first! towards a constant-time implementation of isogeny-based signature, SQIsign. Cryptology ePrint Archive, Paper 2023/807, 2023. https://eprint.iacr.org/2023/807. 3

[KLPT14] David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion ℓ-isogeny path problem. *LMS Journal of Computation and Mathematics*, 17(A):418–432, 2014. 3, 6, 16, 17

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

[Lan10] Adam Langley. Checking that functions are constant time with Valgrind. https://github.com/agl/ctgrind/, 2010. Accessed: 12 July 2024. 2, 17

[Ler22] Antonin Leroux. Quaternion algebras and isogeny-based cryptography. http://www.lix.polytechnique.fr/Labo/Antonin.LEROUX/manuscrit_these.pdf, accessed on 15 July 2024, 2022. 16

[LHK+24] Jeonghwan Lee, Donghoe Heo, Hyeonhak Kim, Gyusang Kim, Suhri Kim, Heeseok Kim, and Seokhie Hong. Fault attack on SQIsign. In *Post-Quantum Cryptography: 15th International Workshop, PQCrypto 2024, Oxford, UK, June 12–14, 2024, Proceedings, Part II*, page 54–76, Berlin, Heidelberg, 2024. Springer-Verlag. 3

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982. 2, 8, 12

[ME24] Marcel Maehren and Nurullah Erinola. RTLF - R-Time-Leak-Finder, 2024. https://github.com/tls-attacker/RTLF, accessed 13 July 2024. 2, 17, 18

[MV10] Manfred G. Madritsch and Brigitte Vallée. Modelling the LLL algorithm by sandpiles. In Alejandro López-Ortiz, editor, *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010. Proceedings*, volume 6034 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010. 13

[Ngu09] Phong Q Nguyen. Hermite's constant and lattice algorithms. In *The LLL Algorithm: Survey and Applications*, pages 19–69. Springer, 2009. 6

[NO24] Kohei Nakagawa and Hiroshi Onuki. SQIsign2D-East: A new signature scheme using 2-dimensional isogenies. Cryptology ePrint Archive, Paper 2024/771, 2024. https://eprint.iacr.org/2024/771. 1, 3, 6, 22

[NS09a] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM Jornal on Computing*, 39(3):874–903, 2009. 7, 8

[NS09b] Phong Q. Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Trans. Algorithms*, 5(4), nov 2009. 2

[NS16] Arnold Neumaier and Damien Stehlé. Faster LLL-type reduction of lattice bases. In Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao, editors, *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*, pages 373–380. ACM, 2016. 12

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2):201–224, 1987. 2, 12

[Val91] Brigitte Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12(4):556–572, 1991. 2, 7, 11

[Voi21] John Voight. *Quaternion Algebras*. Springer Graduate Texts in Mathematics series, 2021. 5, 16

[Wae56] B. L. Waerden. Die Reduktionstheorie Der Positiven Quadratischen Formen. *Acta Mathematica*, 96(none):265 – 309, 1956. 17