# Short generators without quantum computers: the case of multiquadratics

Jens Bauch[1], Daniel J. Bernstein[2,3], Henry de Valence[2], Tanja Lange[2], and Christine van Vredendaal[2]

[1] Simon Fraser University
Department of Mathematics,
8888 University Dr, Burnaby, BC V5A 1S6, Canada
jbauch@sfu.ca

[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, NL
hdevalence@hdevalence.ca, tanja@hyperelliptic.org, c.v.vredendaal@tue.nl

[3] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to

**Abstract.** Finding a short element $g$ of a number field, given the ideal generated by $g$, is a classic problem in computational algebraic number theory. Solving this problem recovers the private key in cryptosystems introduced by Gentry, Smart–Vercauteren, Gentry–Halevi, Garg–Gentry–Halevi, et al. Work over the last few years has shown that for some number fields this problem has a surprisingly low *post-quantum* security level. This paper shows, and experimentally verifies, that for some number fields this problem has a surprisingly low *pre-quantum* security level.

**Keywords:** Public-key encryption, lattice-based cryptography, ideal lattices, Soliloquy, Gentry, Smart–Vercauteren, units, multiquadratic fields.

## 1 Introduction

Gentry's breakthrough ideal-lattice-based homomorphic encryption system at STOC 2009 [37] was shown several years later to be breakable by a fast quan-

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported by the Netherlands Organisation for Scientific Research (NWO) under grants 613.001.011 and 639.073.005; by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO), project number 643161 (ECRYPT-NET), and project number 645421 (ECRYPT-CSA); and by the U.S. National Science Foundation under grant 1314919. Calculations were carried out on the Saber cluster of the Cryptographic Implementations group at Technische Universiteit Eindhoven, and the Saber2 cluster at the University of Illinois at Chicago. Permanent ID of this document: 62a8ffc6a6765bdc6d92754e8ae99f1d. Date: 2017.05.01.

tum algorithm *if* the underlying number field[4] is chosen as a cyclotomic field (with "small $h^+$", a condition very frequently satisfied). Cyclotomic fields were considered in Gentry's paper ("As an example, $f(x) = x^n \pm 1$"), in a faster cryptosystem from Smart–Vercauteren [55], and in an even faster cryptosystem from Gentry–Halevi [39]. Cyclotomic fields were used in all of the experiments reported in [55] and [39]. Cyclotomic fields are also used much more broadly in the literature on lattice-based cryptography, although many cryptosystems are stated for more general number fields.

The secret key in the systems of Gentry, Smart–Vercauteren, and Gentry–Halevi is a short element $g$ of the ring of integers $\mathcal{O}$ of the number field. The public key is the ideal $g\mathcal{O}$ generated by $g$. The attack has two stages:

- Find some generator of $g\mathcal{O}$, using an algorithm of Biasse and Song [15], building upon a unit-group algorithm of Eisenträger, Hallgren, Kitaev, and Song [31]. This is the stage that uses quantum computation. The best known pre-quantum attacks (see, e.g., [11]) reuse ideas from NFS, the number-field sieve for integer factorization, and take time exponential in $N^{c+o(1)}$ for a real number $c$ with $0 < c < 1$ where $N$ is the field degree. If $N$ is chosen as an appropriate power of the target security level then the pre-quantum attacks take time exponential in the target security level, but the Biasse–Song attack takes time polynomial in the target security level.
- Reduce this generator to a short generator, using an algorithm introduced by Campbell, Groves, and Shepherd [22, page 4]: "A simple generating set for the cyclotomic units is of course known. The image of $\mathcal{O}^\times$ under the logarithm map forms a lattice. The determinant of this lattice turns out to be much bigger than the typical log-length of a private key $\alpha$ [i.e., $g$], so it is easy to recover the causally short private key given *any* generator of $\alpha\mathcal{O}$ *e.g.* via the LLL lattice reduction algorithm."[5] This is the stage that relies on the field being cyclotomic.

A quantum algorithm for the first stage was stated in [22] before [15], but the effectiveness of this algorithm was disputed by Biasse and Song (see [14]) and was not defended by the authors of [22]. The algorithm in [22, page 4] quoted above for the second stage does not rely on quantum computers, and its effectiveness is easily checked by experiment.

It is natural to ask whether quantum computers play an essential role in this polynomial-time attack. It is also natural to ask whether the problem of finding $g$ given $g\mathcal{O}$ is weak for *all* number fields, or whether there is something that makes cyclotomic fields particularly weak.

[4] We assume some familiarity with algebraic number theory, although we also review some background as appropriate.

[5] Beware that the analysis in [22, page 4] is incomplete: the analysis correctly states that the secret key is short, but fails to state that the textbook basis for the cyclotomic units is a very good basis; LLL would not be able to find the secret key starting from a bad basis. A detailed analysis of the basis appeared in a followup paper [28] by Cramer, Ducas, Peikert, and Regev.

### 1.1 Why focus on the problem of finding $g$ given $g\mathcal{O}$?

There are many other lattice-based cryptosystems that are not broken by the Biasse–Song–Campbell–Groves–Shepherd attack. For example, the attack does not break a more complicated homomorphic encryption system introduced in Gentry's thesis [36,38]; it does not break the classic NTRU system [40]; and it does not break the BCNS [17] and New Hope [4] systems. But the simple problem of finding $g$ given $g\mathcal{O}$ remains of interest for several reasons.

First, given the tremendous interest in Gentry's breakthrough paper, the scientific record should make clear whether Gentry's original cryptosystem is completely broken, or is merely broken for some special number fields.

Second, despite burgeoning interest in post-quantum cryptography, most cryptographic systems today are chosen for their pre-quantum security levels. Fast quantum attacks have certainly not eliminated the interest in RSA and ECC, and also do not end the security analysis of Gentry's system.

Third, the problem of finding a generator of a principal ideal has a long history of being considered hard. There is a list of five "main computational tasks of algebraic number theory" in [25, page 214], and the problem of finding a generator is the fifth on the list. Smart and Vercauteren describe their key-recovery problem as an "instance of a classical and well studied problem in algorithmic number theory", point to the Buchmann–Maurer–Möller cryptosystem [18] a decade earlier relying on the hardness of this problem, and summarize various slow solutions. The solutions scale poorly even if the output is allowed to be a long generator of an ideal that actually has a short generator.

Fourth, this problem has been reused in various attempts to build secure multilinear maps, starting with the Garg–Gentry–Halevi construction [35]. We do not mean to overstate the security or applicability of multilinear maps (see, e.g., [24,27]), but there is a clear pattern of this problem appearing in the design of advanced cryptosystems. Future designers need to understand whether this problem should simply be discarded, or whether it can be a plausible foundation for security.

Fifth, even when cryptosystems rely on more complicated problems, it is natural for cryptanalysts to begin by studying the security of simpler problems. Successful attacks on complicated problems are usually outgrowths of successful attacks on simpler problems. As explained in Appendix B, the Biasse–Song–Campbell–Groves–Shepherd attack has already been reused to attack a more complicated problem.

### 1.2 Contributions of this paper

We introduce a *pre-quantum* algorithm that, for a large class of number fields, computes a short $g$ given $g\mathcal{O}$. Plausible heuristic assumptions imply that, for a wide range of number fields in this class, this algorithm (1) has success probability converging rapidly to 100% as the field degree increases and (2) takes time *quasipolynomial* in the field degree.

One advantage of building pre-quantum algorithms is that the algorithms can be tested experimentally. We have implemented our algorithm within the Sage computer-algebra system; the resulting measurements are consistent with our analysis of the performance of the algorithm.

The number fields that we target are *multiquadratics*, such as the degree-256 number field $\mathbb{Q}(\sqrt{2},\sqrt{3},\sqrt{5},\sqrt{7},\sqrt{11},\sqrt{13},\sqrt{17},\sqrt{19})$, or more generally any $\mathbb{Q}(\sqrt{d_1},\sqrt{d_2},\ldots,\sqrt{d_n})$. Sometimes we impose extra constraints for the sake of simplicity: for example, in a few steps we require $d_1,\ldots,d_n$ to be coprime and squarefree, and in several steps we require them to be positive.

A preliminary step in the attack (see Section 5.1) is to compute a full-rank subgroup of "the unit group of" the number field (which by convention in algebraic number theory means the unit group of the ring of integers of the field): namely, the subgroup generated by the units of all real quadratic subfields. We dub this subgroup the set of "multiquadratic units" by analogy to the standard terminology "cyclotomic units", with the caveat that "multiquadratic units" (like "cyclotomic units") are not guaranteed to be *all* units.

The degree-256 example above has exactly 255 real quadratic subfields

$$\mathbb{Q}(\sqrt{2}),\mathbb{Q}(\sqrt{3}),\mathbb{Q}(\sqrt{6}),\ldots,\mathbb{Q}(\sqrt{2\cdot3\cdot5\cdot7\cdot11\cdot13\cdot17\cdot19}).$$

Each of these has a unit group quickly computable by standard techniques. For example, the units of $\mathbb{Q}(\sqrt{2})$ are $\pm(1+\sqrt{2})^{\mathbb{Z}}$, and the units of the last field are $\pm(69158780182494876719+22205900901368228\sqrt{2\cdot3\cdot5\cdot7\cdot11\cdot13\cdot17\cdot19})^{\mathbb{Z}}$.

This preliminary step generally becomes slower as $d_1,\ldots,d_n$ grow, but it takes time quasipolynomial in the field degree $N$, assuming that $d_1,\ldots,d_n$ are quasipolynomial in $N$.

In the next step (the rest of Section 5) we go far beyond the multiquadratic units: we quickly compute the *entire* unit group of the multiquadratic field. This is important because the gap between the multiquadratic units and all units would interfere, potentially quite heavily, with the success probability of our algorithm, the same way that a "large $h^+$" (a large gap between cyclotomic units and all units) would interfere with the success probability of the cyclotomic attacks. Note that computing the unit group is another of the five "main computational tasks of algebraic number theory" listed in [25]; furthermore, starting from these unit groups one can efficiently compute "class groups" as explained in, e.g., [43], solving yet another of these computational tasks. There is an earlier algorithm by Wada [60] to compute the unit group of a multiquadratic field, but that algorithm takes exponential time.

We then go even further (Section 6), quickly computing a generator of the input ideal. The generator algorithm uses techniques similar to, but not the same as, the unit-group algorithm. The unit-group computation starts from unit groups computed recursively in three subfields, while the generator computation starts from generators computed recursively in those subfields *and* from the unit group of the top field.

There is a very easy way to extract *short* generators when $d_1,\ldots,d_n$ are large enough, between roughly $N$ and any quasipolynomial bound. This condition is satisfied by a wide range of fields of each degree.

We do more work to extend the applicability of our attack to allow smaller $d_1, \ldots, d_n$, using LLL to shorten units and indirectly generators. Analysis of this extension is difficult, but experiments suggest that the success probability converges to 1 even when $d_1, \ldots, d_n$ are chosen to be as small as the first $n$ primes starting from $n^2$.

There are many obvious opportunities for precomputation in our algorithm, and in particular the unit group can be reused for attacking many targets $g\mathcal{O}$ in the same field. We separately measure the cost of computing the unit group and the cost of subsequently finding a generator.

### 1.3 Why focus on multiquadratics?

Automorphisms and subfields play critical roles in several strategies to attack discrete logarithms. These strategies complicate security analysis, and in many cases they have turned into successful attacks. For example, small-characteristic multiplicative-group discrete logarithms are broken in quasipolynomial time; there are ongoing disputes regarding a strategy to attack small-characteristic ECC; and very recently pairing-based cryptography has suffered a significant drop in security level, because of new optimizations in attacks exploiting subfields of the target field. See, e.g., [6], [33], and [41].

Do automorphisms and subfields also damage the security of lattice-based cryptography? We chose multiquadratics as an interesting test case because they have a huge number of subfields, presumably amplifying and clarifying any impact that subfields might have upon security.

A degree-$2^n$ multiquadratic field is Galois: i.e., it has $2^n$ automorphisms, the maximum possible for a degree-$2^n$ number field. The Galois group, the group of automorphisms, is isomorphic to $(\mathbb{Z}/2)^n$. The number of subfields of the field is the number of subgroups of $(\mathbb{Z}/2)^n$, i.e., the number of subspaces of an $n$-dimensional vector space over $\mathbb{F}_2$. The number of $k$-dimensional subspaces is the 2-binomial coefficient

$$\binom{n}{k}_2 = \frac{(2^n-1)(2^{n-1}-1)\cdots(2^1-1)}{(2^k-1)(2^{k-1}-1)\cdots(2^1-1)(2^{n-k}-1)(2^{n-k-1}-1)\cdots(2^1-1)},$$

which is approximately $2^{n^2/4}$ for $k \approx n/2$. This turns out to be overkill from the perspective of our attack: as illustrated in Figures 5.1 and 5.2, the number of subfields we use ends up essentially linear in $2^n$.

### 1.4 Priority dates

We made preliminary announcements of two corners of this work in February 2014 [9] and April 2015 [10, last slide], in both cases focusing on the problem of reducing a generator to a short generator, and in both cases using the idea of first computing relative norms of the short generator in all proper subfields of the original field. The first announcement was that this idea would benefit from having a "large number of subfields of small relative degree", and in particular

would take "slightly subexponential" time for cyclotomic fields whose conductor "has enough small prime factors". The second announcement highlighted multiquadratics as a more extreme case in which each reduction step "becomes trivial", although this used a superpolynomial number of subfields (we now use an essentially linear number of subfields), presumed that the unit group was already known (we now compute the unit group), and did nothing to suggest that the problem of finding a generator in the first place was easy (we now compute a generator).

Peikert posted slides in July 2015 [51] that also highlighted multiquadratics and that asked about the gap between multiquadratic units and all units. By computing all units we make this question easy to answer experimentally, and we also avoid having to answer it.

## 2 Multiquadratic fields

A **multiquadratic field** is, by definition, a field that can be written in the form $\mathbb{Q}(\sqrt{r_1},\ldots,\sqrt{r_m})$ where $(r_1,\ldots,r_m)$ is a finite sequence of rational numbers. The notation $\mathbb{Q}(\sqrt{r_1},\ldots,\sqrt{r_m})$ means the smallest subfield of $\mathbb{C}$, the field of complex numbers, that contains $\sqrt{r_1},\ldots,\sqrt{r_m}$.

When we write $\sqrt{r}$ for a nonnegative real number $r$, we mean specifically the nonnegative square root of $r$. When we write $\sqrt{r}$ for a negative real number $r$, we mean specifically $i\sqrt{-r}$, where $i$ is the standard square root of $-1$ in $\mathbb{C}$; for example, $\sqrt{-2}$ means $i\sqrt{2}$. These choices do not affect the definition of $\mathbb{Q}(\sqrt{r_1},\ldots,\sqrt{r_m})$, but many other calculations rely on each $\sqrt{r}$ having a definite value.

**Theorem 2.1.** *Let $n$ be a nonnegative integer. Let $d_1,\ldots,d_n$ be integers such that, for each nonempty subset $J \subseteq \{1,\ldots,n\}$, the product $\prod_{j\in J} d_j$ is not a square. Then the $2^n$ complex numbers $\prod_{j\in J}\sqrt{d_j}$ for all subsets $J \subseteq \{1,\ldots,n\}$ form a basis for the multiquadratic field $\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n})$ as a $\mathbb{Q}$-vector space. Furthermore, for each $j \in \{1,\ldots,n\}$ there is a unique field automorphism of $\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n})$ that preserves $\sqrt{d_1},\ldots,\sqrt{d_n}$ except for mapping $\sqrt{d_j}$ to $-\sqrt{d_j}$.*

Consequently $\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n})$ is a degree-$2^n$ number field.

*Proof.* $\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n})$ is a multiquadratic field by definition.

For $n = 0$ the basis statement is easy: the only number is 1 (from $J = \{\}$), and $\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n}) = \mathbb{Q}$. The automorphism statement is vacuous. Assume from now on that $n \geq 1$, and induct on $n$.

The inductive hypothesis states that the $2^{n-1}$ complex numbers $\prod_{j\in J}\sqrt{d_j}$ for all subsets $J \subseteq \{1,\ldots,n-1\}$ form a basis for $K = \mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_{n-1}})$ as a $\mathbb{Q}$-vector space; and, for each $j \in \{1,\ldots,n-1\}$, there is a unique field automorphism $\sigma_j$ of $K$ that preserves $\sqrt{d_1},\ldots,\sqrt{d_{n-1}}$ except for mapping $\sqrt{d_j}$ to $-\sqrt{d_j}$.

Note that an element $x = \sum_J x_J \prod_{j\in J} \sqrt{d_j} \in K$ satisfies $\sigma_k(x) = x$ if and only if $x_J = 0$ whenever $k \in J$. Similarly, $x$ satisfies $\sigma_k(x) = -x$ if and only if $x_J = 0$ whenever $k \notin J$. Hence a nonzero $x$ satisfies the $n-1$ conditions $\sigma_1(x), \ldots, \sigma_{n-1}(x) \in \{x, -x\}$ if and only if there is exactly one subset $J$ for which $x_J \neq 0$: namely, the set $J$ of indices $k$ for which $\sigma_k(x) = -x$.

Define $L$ as the $K$-vector space $\left\{a + b\sqrt{d_n} : a, b \in K\right\}$. Note that the field $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ contains $K$ and $\sqrt{d_n}$, so it contains $L$.

Suppose that $L = K$. Then $\sqrt{d_n} \in K$; i.e., $x^2 = d_n$ for some $x \in K$. Hence $\sigma_k(x)^2 = \sigma_k(d_n) = d_n$ for each $k$. The only square roots of $d_n$ are $x$ and $-x$, so each $\sigma_k(x)$ must be $x$ or $-x$, so $x$ has the form $x_J \prod_{j\in J} \sqrt{d_j}$ for a single set $J$. Hence $x_J^2 \prod_{j\in J} d_j = d_n$, so $\prod_{j\in J\cup\{n\}} d_j$ is a square, contradicting the non-squareness hypothesis.

Hence the $K$-dimension of $L$ is at least 2. But $(1, \sqrt{d_n})$ generates $L$ over $K$; so it must be a $K$-basis of $L$. Furthermore, $L$ is a field: for multiplication observe that $(a + b\sqrt{d_n})(a' + b'\sqrt{d_n}) = (aa' + bb'd_n) + (ab' + ba')\sqrt{d_n}$, and for division observe that any nonzero $a + b\sqrt{d_n}$ has nonzero $D = a^2 - b^2 d_n$ and reciprocal $a/D - (b/D)\sqrt{d_n}$.

By construction $L$ contains $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-1}})$ and $\sqrt{d_n}$, so it also contains $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$. Thus $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n}) = L$. Multiplying $1, \sqrt{d_n}$ by the above $\mathbb{Q}$-basis of $K$ produces the claimed $\mathbb{Q}$-basis of $L$.

For each $j \in \{1, \ldots, n-1\}$, the map $a + b\sqrt{d_n} \mapsto \sigma_j(a) + \sigma_j(b)\sqrt{d_n}$ is a field automorphism of $L$ that preserves $\sqrt{d_1}, \ldots, \sqrt{d_n}$ except for mapping $\sqrt{d_j}$ to $-\sqrt{d_j}$; and $a + b\sqrt{d_n} \mapsto a - b\sqrt{d_n}$ is a field automorphism of $L$ that preserves $\sqrt{d_1}, \ldots, \sqrt{d_{n-1}}$ while mapping $\sqrt{d_n}$ to $-\sqrt{d_n}$. Finally, uniqueness follows from the fact that specifying how a field automorphism acts on $\sqrt{d_1}, \ldots, \sqrt{d_n}$ also specifies how it acts on each basis element $\prod_{j\in J} \sqrt{d_j}$. □

**Theorem 2.2.** *Every multiquadratic field can be expressed in the form of Theorem 2.1 with each $d_j$ squarefree.*

*Proof.* We show by induction on $m$ that any field $\mathbb{Q}(\sqrt{r_1}, \ldots, \sqrt{r_m})$ with rational $r_1, \ldots, r_m$ can be expressed as $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ for some admissible $d_1, \ldots, d_n$, i.e., some squarefree $d_1, \ldots, d_n$ meeting the hypotheses of Theorem 2.1. For $m = 0$ simply take $n = 0$.

For $m \geq 1$, write $K = \mathbb{Q}(\sqrt{r_1}, \ldots, \sqrt{r_{m-1}})$. The inductive hypothesis states that $K$ can be expressed as $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ for some admissible $d_1, \ldots, d_n$. All that remains is to prove the same for $K(\sqrt{r_m})$.

If $r_m = 0$ then $K(\sqrt{r_m}) = K$ and we are done. Otherwise write $r_m$ as $N/D$ for nonzero integers $N, D$, and then write $ND$ as $s^2 d_{n+1}$ for a squarefree integer $d_{n+1}$. Now $\sqrt{r_m} = \sqrt{ND}/D = s\sqrt{d_{n+1}}/D$, so $K(\sqrt{r_m}) = K(\sqrt{d_{n+1}})$.

If $\prod_{j\in J} d_j$ is non-square for all $J \subseteq \{1, \ldots, n+1\}$ then we are done. Otherwise $\prod_{j\in J} d_j$ is square for some $J \subseteq \{1, \ldots, n+1\}$. The admissibility of $d_1, \ldots, d_n$ forces $n+1 \in J$, in turn forcing $d_{n+1}$ to be a square divided by some of $d_1, \ldots, d_n$, so $\sqrt{d_{n+1}}$ is an integer divided by an element of $K$, so $K(\sqrt{d_{n+1}}) = K$ and again we are done. □

# 3 Fast arithmetic in multiquadratic fields

There are well-known polynomial-time algorithms that perform basic arithmetic operations (addition, subtraction, multiplication, division, and square root) on elements of number fields represented in a standard way. See, e.g., [25]. These algorithms are readily available in the easy-to-use Sage [30] computer-algebra system, which adds many mathematical libraries to Python.

However, we care about performance in more detail than "polynomial time", for three reasons:

- Even though a polynomial-time or quasipolynomial-time attack is *usually* a security problem, it is not *necessarily* a security problem. Assessing concrete security levels requires more precise analysis and optimization of attacks.
- More precise analyses help algorithm designers compare predicted performance to the performance observed in experiments, reducing the chance of error.
- Optimizations allow larger-scale experiments, providing more data for comparison, further reducing the chance of error.

This section analyzes the cost of basic arithmetic in multiquadratic fields. We assume that the reader is already familiar with fast arithmetic in $\mathbb{Z}$: basic arithmetic operations on $B$-bit numbers in $\mathbb{Z}$ take time essentially $B$, where "essentially" suppresses lower-order factors such as $\log B$.

For general multiquadratics there are two important cost parameters: $N = 2^n$, the degree of the multiquadratic field; and $B$, the number of bits in the largest integer used. Each algorithm discussed here has time bounded by essentially $N^e B$ for some algorithm-specific exponent $e$. Our primary goal is to minimize $e$.

## 3.1 Representation of fields and field elements

We represent a multiquadratic field as a sequence of integers $d_1, \ldots, d_n$ such that $\prod_{j \in J} d_j$ is non-square for each nonempty subset $J \subseteq \{1, \ldots, n\}$. Then $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ is a field of degree $N = 2^n$, with automorphisms that negate any desired $\sqrt{d_j}$; see Theorem 2.1.

We represent an element of the field $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ as a fraction, specifically an element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ divided by a positive integer. The ring $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ consists of $\mathbb{Z}$-linear combinations of the $2^n$ products $\prod_{j \in J} \sqrt{d_j}$ for subsets $J \subseteq \{1, \ldots, n\}$. We represent an element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ as a vector of coefficients for $J = \{\}$, $J = \{1\}$, $J = \{2\}$, $J = \{1, 2\}$, etc. In other words, we represent an element of the ring $\mathbb{Z}[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$ as a vector of coefficients of 1, $x_1$, $x_2$, $x_1 x_2$, etc.; this ring is isomorphic to $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ under the isomorphism that maps each $x_j$ to $\sqrt{d_j}$.

Addition and subtraction in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ are performed coefficientwise and take time $O(NB)$ if each integer has $O(B)$ bits. More complicated operations such as multiplication are the topics of subsequent subsections.

Operations in the field $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ decompose easily into operations in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ and fast operations on integer denominators. Reducing a fraction to lowest terms means dividing all coefficients and the denominator by their gcd; this also takes time bounded by essentially $NB$ by standard algorithms.

An alternative, *assuming* $d_1, \ldots, d_n \in 1 + 4\mathbb{Z}$, is to replace $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ with the larger ring $\mathbb{Z}[(1+\sqrt{d_1})/2, \ldots, (1+\sqrt{d_n})/2]$, which has a similar advantage of allowing each square root to be handled separately. As a special case, for squarefree coprime $d_1, \ldots, d_n \in 1 + 4\mathbb{Z}$, this larger ring is the ring of integers of $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$, avoiding denominators in some computations. However, denominators do not noticeably slow down our computations, and focusing on $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ lets us handle much more general $d_1, \ldots, d_n$.

### 3.2 Finding good primes

The $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ multiplication algorithm in Section 3.3 relies on precomputing a sufficiently large pool of primes $q$ such that each of $d_1, \ldots, d_n$ is a nonzero square modulo $q$. Variants of the same problem appear elsewhere in the paper: for example, arranging for $d_1$ to be a non-square modulo $q$ while $d_2, \ldots, d_n$ are nonzero squares modulo $q$.

The goal of Algorithm 3.1, GoodPrime, is to find one such prime. The inputs $\mu_1, \ldots, \mu_n \in \{-1, 1\}$ select non-squareness or nonzero squareness for $d_1, \ldots, d_n$ respectively. Nonempty subsequences of $d_1, \ldots, d_n$ are not allowed to have square products, so there cannot be any dependencies between these squareness conditions.

GoodPrime simply tries a uniform random $\lambda$-bit integer $q$, where $\lambda$ is an algorithm parameter, and then tries again if $q$ is not a prime or the desired squareness does not hold. An integer $q$ in this range has probability approximately $1/(\lambda \ln 2)$ of being prime. The probability of satisfying $n$ squareness conditions converges to $1/N$ as $\lambda$ increases, so it is reasonable to estimate that GoodPrime will have to try $N\lambda \ln 2$ choices of $q$ on average. Each choice requires a primality test taking time $\lambda^{O(1)}$, and at most $n$ Legendre-symbol computations, each of which takes time $\lambda^{O(1)}$ assuming that each $d_j$ has $\lambda^{O(1)}$ bits.

Beware that if $\lambda$ is chosen too small then the algorithm can run forever: for example, $\lambda = n$ is likely to fail. If GoodPrime is being asked to generate $P$ different primes then, to ensure a reasonable chance of success, one must take $2^{\lambda-1}/(\lambda \ln 2)$ somewhat larger than $NP$.

Inside a quasipolynomial-time algorithm, $P$ is quasipolynomial in $N$; i.e., $\log P \in n^{O(1)}$. This is compatible with taking $\lambda \in n^{O(1)}$, so the time for each call to GoodPrime is essentially $N$; recall that "essentially" disregards logarithmic factors such as $n$.

An alternative approach, for the application in Section 3.3 with $\mu = (1, \ldots, 1)$, is to search $q$ in the arithmetic progression of integers congruent to 1 modulo all of $4d_1, \ldots, 4d_n$. This has two advantages: first, it reduces the estimated number of $q$'s from $N\lambda \ln 2$ to just $\lambda \ln 2$; second, it saves time in handling each $q$. The point here is that $d_1, \ldots, d_n$ are guaranteed to be square modulo $q$ by quadratic

**Algorithm 3.1:** GoodPrime$(d, \mu)$

**Input**: An integer sequence $d = (d_1, \ldots, d_n)$ such that $\prod_{j \in J} d_j$ is non-square for each nonempty subset $J \subseteq \{1, \ldots, n\}$; and a sequence $\mu = (\mu_1, \ldots, \mu_n)$ where each $\mu_j \in \{-1, 1\}$. As a side input, an integer parameter $\lambda \geq 3$.

**Output**: A $\lambda$-bit prime number $q$ such that, for each $j$, the integer $d_j$ is a nonzero square modulo $q$ if $\mu_j = 1$, and is a non-square modulo $q$ if $\mu_j = -1$.

1 Choose a uniform random element $q \in \{2^{\lambda-1}, 2^{\lambda-1}+1, \ldots, 2^{\lambda}-1\}$.
2 If $q$ is not prime, go back to Step 1.
3 **for** $j \in \{1, \ldots, n\}$ **do**
4 If the Legendre symbol of $d_j$ modulo $q$ is not $\mu_j$, go back to Step 1.
5 **return** $q$

reciprocity, so the Legendre-symbol tests can be skipped for each $q$, and the only question is whether $q$ is prime.

A disadvantage of this alternative approach is that it needs larger $\lambda$: specifically, to generate $P$ different primes, one must take $2^{\lambda-1}/(\lambda \ln 2)$ somewhat larger than $4d_1 \cdots d_n P$ (assuming $d_1, \ldots, d_n$ are coprime). This is still compatible with taking $\lambda \in n^{O(1)}$, since we assume that each $d_j$ is quasipolynomial.

Yet another approach is to instead allow any appropriate remainder for $q$ modulo $4d_1$, any appropriate remainder for $q$ modulo $4d_2$, etc. There are standard techniques to merge the lists of appropriate remainders (see, e.g., [8]), skipping quickly past many useless choices of $q$.

### 3.3 Multiplication

One reason for interest in cyclotomics is that polynomial multiplication can be carried out efficiently by fast Fourier transforms (FFTs). A classic FFT requires the base ring to contain appropriate roots of unity: one nice case is multiplying in $\mathbb{F}_q[x]/(x^N+1)$ where $q$ is an odd prime, $N = 2^n$, and $\mathbb{F}_q$ has an $N$th root of $-1$. One can multiply in $\mathbb{Z}[x]/(x^N+1)$ in time essentially $NB$ by multiplying in $\mathbb{F}_q[x]/(x^N+1)$ for enough such primes $q$, if each output coefficient is guaranteed to be below $2^B$.

Multiquadratic rings $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ support analogous fast-multiplication strategies. FFTs on cyclic groups are replaced by simpler FFTs on $(\mathbb{Z}/2)^n$, i.e., Hadamard–Walsh transforms, twisted by $\sqrt{d_1}, \ldots, \sqrt{d_n}$. The base ring is required to contain $1/2$ and to contain invertible square roots of $d_1, \ldots, d_n$. To find appropriate base fields $\mathbb{F}_q$ we use the GoodPrime subroutine from Section 3.2. This section presents the details.

*Twisted Hadamard–Walsh transforms over appropriate finite fields.* Given $f, g \in \mathbb{F}_q[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$, we compute $h = fg$ as follows. We assume here that $q$ is an odd prime with $n^{O(1)}$ bits, and that all of $d_1, \ldots, d_n$ are nonzero squares in $\mathbb{F}_q$.

Precompute square roots $s_1, \ldots, s_n$ of $d_1, \ldots, d_n$ respectively in $\mathbb{F}_q$, along with their reciprocals $1/s_1, \ldots, 1/s_n$. This takes negligible time, namely $n^{O(1)}$, by standard algorithms.

Define two homomorphisms $\varphi^+, \varphi^-$ from $\mathbb{F}_q[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$ to $R = \mathbb{F}_q[x_1, \ldots, x_{n-1}]/(x_1^2 - d_1, \ldots, x_{n-1}^2 - d_{n-1})$ as follows: $\varphi^+(f_0 + x_n f_1) = f_0 + s_n f_1$ and $\varphi^-(f_0 + x_n f_1) = f_0 - s_n f_1$ for any $f_0, f_1 \in R$. Note that adding these two outputs and dividing by 2 produces $f_0$, while subtracting these two outputs and dividing by $2s_n$ produces $f_1$; i.e., one can efficiently recover $f$ given $\varphi^+(f)$ and $\varphi^-(f)$.

Compute $\varphi^+(f), \varphi^-(f), \varphi^+(g), \varphi^-(g)$. Compute $\varphi^+(h) = \varphi^+(f)\varphi^+(g)$ and $\varphi^-(h) = \varphi^-(f)\varphi^-(g)$, using the same method recursively to multiply in $R$. Recover $h$ from $\varphi^+(h)$ and $\varphi^-(h)$.

This handles a size-$N$ multiplication using two size-$(N/2)$ multiplications and $O(N)$ simple coefficient operations in $\mathbb{F}_q$: additions, subtractions, multiplications by $s_n$ and $1/s_n$, and multiplications by $1/2$ (which are easily merged across levels of recursion into final multiplications by $1/N$). In total a size-$N$ multiplication uses $O(Nn)$ operations in $\mathbb{F}_q$, including $N$ base-case multiplications in $\mathbb{F}_q$.

*Reducing $\mathbb{Z}$ to $\mathbb{F}_q$.* We multiply $f, g \in \mathbb{Z}[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$ as follows.

Use Theorem 3.1 to find $B$ so that each coefficient of $h = fg$ has absolute value at most $2^{B-1} - 1$. Repeatedly call GoodPrime to find $\lceil B/(\lambda - 1) \rceil$ distinct odd primes $q$ where all of $d_1, \ldots, d_n$ are nonzero squares. Note that $\prod_q q = 2^{\sum_q \log_2 q} \geq 2^{\sum_q (\lambda - 1)} \geq 2^B$, so each coefficient of $h$ is determined by its remainder modulo $\prod_q q$.

The average number of GoodPrime calls will be approximately $\lceil B/(\lambda - 1) \rceil$ if the parameter $\lambda$ is chosen properly in Section 3.2: the outputs $q$ from GoodPrime will almost never repeat. Each call takes time essentially $N$, for a total time essentially $NB$. We cache each $q$ for reuse in subsequent multiplications.

Reduce each coefficient of $f$ and $g$ modulo all of the $q$'s. For each $q$ multiply $f$ mod $q$ by $g$ mod $q$ as indicated above, obtaining $h$ mod $q$. Use the Chinese remainder theorem to compute the coefficients of $h$ by interpolation. All of this takes time essentially $NB$ by standard fast-arithmetic algorithms such as remainder trees.

An alternative strategy, which we used in our first implementation, is as follows. Merge all the primes $q$ into a single modulus, their product $\prod_q q$. Similarly merge the square roots of each $d_j$ modulo all the primes $q$ into a square root of $d_j$ modulo $\prod_q q$. Then apply the twisted Hadamard–Walsh transform, replacing $\mathbb{F}_q$ with $\mathbb{Z}/\prod_q q$. This again takes time essentially $NB$. This has the advantage of working with a smaller number of larger integers; Sage has considerable overhead for each integer operation, and this overhead becomes much less noticeable as the integers grow. However, our primary concern is scalability, and a closer look at asymptotic performance shows that working separately with each $q$ gains a logarithmic factor. Our current software merges batches of 8 primes, gaining the same logarithmic factor with lower overhead.

**Theorem 3.1.** *Let $n$ be a nonnegative integer. Let $d_1, \ldots, d_n$ be integers. Let $f, g$ be elements of $\mathbb{Z}[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$, and define $h = fg$. Write $f$ as $\sum_{J \subseteq \{1,\ldots,n\}} f_J \prod_{j \in J} x_j$ and $g$ as $\sum_{J \subseteq \{1,\ldots,n\}} g_J \prod_{j \in J} x_j$. Assume that $|f_J| \leq F$ and $|g_J| \leq G$ for all $J$. Define $H = FG(1 + |d_1|) \cdots (1 + |d_n|)$. Then $h = \sum_{J \subseteq \{1,\ldots,n\}} h_J \prod_{j \in J} x_j$ with $|h_J| \leq H$.*

With slightly more effort one can compute a "balanced" bound involving $f_J \prod_{j \in J} \sqrt{|d_j|}$ etc., but we have not noticed this producing significant speedups.

*Proof.* The coefficient $h_J$ is exactly $\sum_S f_S g_{J \oplus S} \prod_{j \in S - J} d_j$, where $J \oplus S$ means $(J \cup S) - (J \cap S)$. This sum is bounded in absolute value by $\sum_S FG \prod_{j \in S-J} |d_j| \leq FG \sum_S \prod_{j \in S} |d_j| = H$. □

### 3.4 Norm computation

Fix a multiquadratic field $L = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ as in Theorem 2.1, and fix $\mu = (\mu_1, \ldots, \mu_n) \in \{-1, 1\}^n$. Write $\sigma$ for the unique automorphism of $L$ that maps each $\sqrt{d_j}$ to $\mu_j \sqrt{d_j}$. This is the product, over all $j$ with $\mu_j = -1$, of the $\sqrt{d_j} \mapsto -\sqrt{d_j}$ automorphisms from Theorem 2.1.

Assume that $(\mu_1, \ldots, \mu_n) \neq (1, \ldots, 1)$, i.e., that $\sigma \neq 1$; this forces $n \geq 1$. Define $K$ as the subfield of $L$ fixed by $\sigma$, i.e., the set of all $x \in L$ with $\sigma(x) = x$. Then $K$ is a multiquadratic field of degree $2^{n-1}$. Explicitly:

$$K = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-1}}) \quad \text{if } \mu = (1, \ldots, 1, -1), \tag{1a}$$

$$K = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_{n-1}d_n}) \quad \text{if } \mu = (1, \ldots, 1, -1, -1), \tag{1b}$$

$$K = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-3}}, \sqrt{d_{n-2}d_{n-1}}, \sqrt{d_{n-1}d_n}) \quad \text{if } \mu = (1, \ldots, 1, -1, -1, -1),$$

etc. Permuting indices in the $(1, \ldots, 1, -1)$ example covers all choices of $\mu$ with exactly one $-1$; permuting indices in the $(1, \ldots, 1, -1, -1)$ example covers all choices of $\mu$ with exactly two $-1$s; etc.

Define $N_{L:K}(f)$ as $f\sigma(f)$ for each $f \in L$. Then $N_{L:K}(f) \in K$. This is the "relative norm of $f$ from $L$ to $K$". The $N_{L:K}(f)$ notation is usable as $L$ and $\mu$ vary, since $L$ and $K$ together determine $\sigma$.

This section analyzes the cost of computing $N_{L:K}(f)$, focusing in particular on two cases used later. The primary case is $K = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-1}})$. The secondary case is $K = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_{n-1}d_n})$.

As before we represent an element of $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ as an element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ divided by a positive integer. The norm is the norm of the numerator divided by the norm of the denominator, and the norm of the denominator is simply the square of the denominator. The remaining problem is to compute $f\sigma(f)$ for $f \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$.

Primary case: Write $f$ as $f_0 + f_1\sqrt{d_n}$ where $f_0, f_1 \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{n-1}}]$. Then $\sigma(f) = f_0 - f_1\sqrt{d_n}$, and $f\sigma(f) = f_0^2 - f_1^2 d_n$. We compute $f_0^2$, compute $f_1^2$, multiply by $d_n$, and subtract. This is faster (by a constant factor) than general multiplication in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$. We actually save more time by merging the

inverse transforms involved in the computations of $f_0^2$ and $f_1^2$: transform $f_0$, square, transform $f_1$, square, multiply by $d_n$, subtract, inverse transform. The norm computation takes time essentially $NB$.

Secondary case: Write $f$ as $f_0 + f_1\sqrt{d_{n-1}} + f_2\sqrt{d_n} + f_3\sqrt{d_{n-1}}\sqrt{d_n}$. Then $\sigma(f) = f_0 - f_1\sqrt{d_{n-1}} - f_2\sqrt{d_n} + f_3\sqrt{d_{n-1}}\sqrt{d_n}$, and

$$f\sigma(f) = (f_0 + f_3\sqrt{d_{n-1}}\sqrt{d_n})^2 - f_1^2 d_{n-1} - f_2^2 d_n - 2f_1 f_2\sqrt{d_{n-1}}\sqrt{d_n}.$$

This is visibly in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_{n-1}d_n}]$, since $\sqrt{d_{n-1}}\sqrt{d_n} = \pm\sqrt{d_{n-1}d_n}$; the $\pm$ here is $-1$ if both $d_{n-1}$ and $d_n$ are negative. There are again various constant-factor speedups, such as computing $2f_1f_2$ as $(f_1+f_2)^2 - f_1^2 - f_2^2$. The norm computation again takes time essentially $NB$.

As an application of the primary case: Write $K_j = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_j})$. Then $N_{K_1:K_0}(N_{K_2:K_1}(\cdots(N_{K_n:K_{n-1}}(f))\cdots))$ is the "absolute norm of $f$" in $\mathbb{Q}$, written $N_{L:\mathbb{Q}}(f)$, equal to the product of $\tau(f)$ across all automorphisms $\tau$ of $L$. There are $n$ relative-norm steps here, and each step takes time essentially $NB$, since at each step the number of bits per coefficient approximately doubles while the number of coefficients is halved. Overall computing an absolute norm takes time essentially $NB$; again recall that "essentially" suppresses logarithmic factors such as $n$.

The literature contains various methods to compute absolute norms without using chains of subfields. All of the non-subfield methods that we investigated turned out to be slower than computing a sequence of relative norms. Gentry–Halevi [39] implicitly use a chain of subfields in an analogous way in the power-of-2 cyclotomic case.

### 3.5 Exact division

This section analyzes the cost of reconstructing $f \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ given a nonzero $g \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ and given $h = fg$.

Recall that Section 3.3 computed $h$ from $fg$ (modulo $q$, or modulo $\prod_q q$) by computing $\varphi^{\pm}(f)$ and $\varphi^{\pm}(g)$, multiplying recursively in a half-size ring to obtain $\varphi^{\pm}(h)$, and then reconstructing $h$.

To divide $h$ by $g$, simply run this process in reverse: compute $\varphi^{\pm}(h)$ and $\varphi^{\pm}(g)$, divide recursively in a half-size ring to obtain $\varphi^{\pm}(f)$, and then reconstruct $f$. This takes time essentially $NB$, just like the algorithm of Section 3.3.

This approach raises two issues not present in Section 3.3. First, the base-case divisions will fail if they involve divisions by 0. By hypothesis $g \neq 0$, but $g$ is then transformed into $N$ elements of $\mathbb{F}_q$, and any of these elements could be 0. This is equivalent to saying that $g$ might be contained in some prime ideal of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ lying over $q$. This cannot happen unless $q$ divides the absolute norm of $g$. The absolute norm is nonzero, and computing a bound on the size of the absolute norm easily produces a bound on the number of $\lambda$-bit primes dividing the norm. This bound is essentially $NB$, so if $\lambda$ is chosen to be reasonably large then there is negligible chance of failure, and in case of failure one can simply try again with a different prime $q$.

The second issue is that the size of the coefficients of $f$ is not obvious in advance: multiplying $f$ by $g$ can produce considerable cancellation, meaning that $f$ is actually much larger than one would guess from comparing the sizes of coefficients of $g$ and $h$. Given the context of how division is used in our higher-level algorithms (e.g., obtaining units with bounded logarithms) we could compute a-priori bounds on $f$, but these bounds would usually be quite loose.

One way to find the actual size of $f$ is as follows. Start with a small guess for the size. If the resulting $f$ does not satisfy $h = fg$, then double the number of primes and try again. This takes time essentially $NB$.

We use the following slightly more complicated approach. Compute floating-point approximations to the complex embeddings of $h$ and $g$, sufficiently precise to distinguish all embeddings of $g$ from 0. Divide to obtain approximations to the complex embeddings of $f$. Interpolate approximations to the coefficients of $f$. Use interval arithmetic to convert the approximations into proven bounds on the coefficients of $f$.

This approach has several advantages. First, the user is allowed to call the function even without knowing that $g$ divides $h$. The final check that $h = fg$ raises a prompt exception rather than restarting and eventually running out of memory.

Second, if the user does know that $g$ divides $h$, then the final check that $h = fg$ is skipped. Dividing $fg$ by $g$ can be even more efficient than multiplying $f$ by $g$, since there is less output.

Third, this approach generalizes immediately from computing $h/g$ to computing a product $h_1^{e_1} h_2^{e_2} h_3^{e_3} \cdots$ for any $e_1, e_2, e_3, \cdots \in \mathbb{Z}$, or computing a sequence of such products. These products of powers appear in the higher-level algorithms later in this paper, and experiments show that intermediate quantities such as $h_1^{e_1} h_2^{e_2}$ often have relatively large coefficients, while the inputs and the final product are relatively small.

### 3.6 Twisted square roots

This section analyzes the cost of a subroutine used in Section 3.7: namely, figuring out which of the $N$ rational numbers

$$h, \frac{h}{d_1}, \frac{h}{d_2}, \frac{h}{d_1 d_2}, \ldots, \frac{h}{d_1 d_2 \cdots d_n}$$

are integer squares. The inputs here are $h, d_1, \ldots, d_n \in \mathbb{Z}$. As usual we assume that $\prod_{j \in J} d_j$ is non-square for each nonempty subset $J \subseteq \{1, \ldots, n\}$.

There is no difficulty for $h = 0$, so we assume $h \neq 0$. Then at most one of the above $N$ rational numbers is a square: for example, if $h/(d_1 d_2)$ and $h/(d_1 d_3 d_4)$ were both squares, then $d_2 d_3 d_4$ would also be square, contradiction.

Algorithm 3.2, TwistedSquareRoot, quickly identifies the square and returns its square root, or raises an exception if there are no squares. TwistedSquareRoot uses GoodPrime to find an odd prime $q$ such that $d_1$ is a non-square modulo $q$ while $d_2, \ldots, d_n$ are nonzero squares modulo $q$. If $q$ turns out to divide $h$

**Algorithm 3.2:** TwistedSquareRoot$(d, h)$

**Input**: An integer sequence $d = (d_1, \ldots, d_n)$ such that $\prod_{j \in J} d_j$ is non-square for each nonempty subset $J \subseteq \{1, \ldots, n\}$; and an integer $h$.

**Output**: An integer $g$ and a set $S \subseteq \{1, \ldots, n\}$ such that $h = g^2 \prod_{j \in S} d_j$; or an exception if no such pair $(g, S)$ exists.

1 **if** $h = 0$ **then**
2 **return** $(0, \{\})$.
3 **for** $j \in \{1, \ldots, n\}$ **do**
4 $\mu \leftarrow (1, \ldots, 1, -1, 1, \ldots, 1)$ where the $-1$ is in the $j$th position.
5 $q_j \leftarrow \text{GoodPrime}(d, \mu)$.
6 If $h \bmod q_j = 0$ for any $j$, go back to step 3.
7 $S \leftarrow \{j \in \{1, \ldots, n\} :$ the Legendre symbol of $h$ modulo $q_j$ is $-1\}$.
8 Raise an exception if $h / \prod_{j \in S} d_j$ is not an integer square.
9 **return** $(\sqrt{h / \prod_{j \in S} d_j}, S)$.

then TwistedSquareRoot starts over with another $q$. If $h = g^2 \prod_{j \in S} d_j$ then $h$ must be a non-square modulo $q$ if $1 \in S$, and a square modulo $q$ if $1 \notin S$. TwistedSquareRoot computes the Legendre symbol of $h$ modulo $q$ to see whether $1 \in S$, and similarly uses other $q$'s to detect the other elements of $S$.

The GoodPrime computations take time essentially $N$, assuming as usual that $\lambda$ is chosen large enough. A remainder tree computes $h \bmod q_1, \ldots, h \bmod q_n$ in time essentially $B$. The Legendre-symbol computations take time essentially 1. Standard methods compute the final square root in time essentially $B$. Overall TwistedSquareRoot takes time essentially $N + B$: i.e., essentially $N$ or essentially $B$, whichever is larger.

### 3.7 Square-root computation

This section analyzes the cost of computing square roots in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$: i.e., recovering, up to sign, an element $f \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ given $h = f^2$. "Up to sign" means that the output is $\pm f$.

As motivation for the algorithm in this section, we review a standard method of computing square roots in the ring $\mathbb{Z}[\sqrt{-1}]$ of Gaussian integers: recovering, up to sign, $f = f_0 + f_1\sqrt{-1}$ given $f^2 = h_0 + h_1\sqrt{-1}$. First compute the norm $h_0^2 + h_1^2$ and its nonnegative real square root $s = f_0^2 + f_1^2$. Then note that $h_0 = f_0^2 - f_1^2$; compute $\pm f_0$ as the nonnegative real square root of $(h_0 + s)/2 = f_0^2$. Then note that $h_1 = 2f_0f_1$; compute $\pm f_1$ as $h_1/(\pm 2f_0)$. (This fails if $f_0 = 0$; we handle this case separately.) Finally $\pm f = (\pm f_0) + (\pm f_1)i$.

The same method might seem to generalize immediately to computing $f = f_0 + f_1\sqrt{d}$ for any $f_0, f_1, d \in \mathbb{Z}$ with non-square $d$, given $f^2 = h_0 + h_1\sqrt{d}$; and beyond this to square roots in arbitrary multiquadratics. First compute the norm $h_0^2 - h_1^2 d$ and its square root $s = f_0^2 - f_1^2 d$. Then compute $\pm f_0$ as the square root of $(h_0 + s)/2$, and compute $\pm f_1$ as $h_1/(\pm 2f_0)$.

**Algorithm 3.3:** SquareRoot$(d, m, h)$

**Input**: An integer sequence $d = (d_1, \ldots, d_n)$ such that $\prod_{j \in J} d_j$ is non-square for each nonempty subset $J \subseteq \{1, \ldots, n\}$; an integer $m \in \{0, 1, \ldots, n\}$; and an element $h \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$ such that $h = f^2 \prod_{j \in R} d_j$ for some $f \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$ and some $R \subseteq \{m+1, \ldots, n\}$.

**Output**: Some $(g, S)$ such that $g \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$; $S \subseteq \{m+1, \ldots, n\}$; $h = g^2 \prod_{j \in S} d_j$; and $S = \{\}$ if $g = 0$.

**if** $h = 0$ **then**
  **return** $(0, \{\})$.
**if** $m = 0$ **then**
  **return** TwistedSquareRoot$(d, h)$.
Write $h$ as $h_0 + h_1\sqrt{d_m}$ with $h_0, h_1 \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{m-1}}]$.
$(s, S_0) \leftarrow$ SquareRoot$((d_1, \ldots, d_{m-1}), m-1, h_0^2 - h_1^2 d_m)$.
**if** $s = h_0$ **then**
  $(t, S) \leftarrow$ SquareRoot$(d, m-1, h_0)$.
**else**
  $(t, S) \leftarrow$ SquareRoot$(d, m-1, (h_0 - s)/2)$.
$\pi \leftarrow \prod_{j \in S - \{m\}} d_j$.
$u \leftarrow (h_1/(2\pi))/t$.
**if** $m \in S$ **then**
  **return** $(u + t\sqrt{d_m}, S - \{m\})$.
**return** $(t + u\sqrt{d_m}, S)$.

What goes wrong here is that computing "the" square root $s$ of $h_0^2 - h_1^2 d$ might actually produce $-(f_0^2 - f_1^2 d)$ instead of $f_0^2 - f_1^2 d$. This problem did not occur in the Gaussian case: taking $s$ as the *nonnegative* square root forced $s$ to equal $f_0^2 + f_1^2$. The same trick does not work for $d > 0$.

If in fact $s = -(f_0^2 - f_1^2 d)$ then $(h_0 + s)/2 = f_1^2 d$. At this point we feed $(h_0 + s)/2$ to a generalized square-root function that produces $\pm f_0$ given $f_0^2$, and produces $\pm f_1$ given $f_1^2 d$, also reporting which of these cases occurred. Using the same idea recursively, starting from $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$, produces more and more potential combinations of $d$'s, eventually leading to the identifying-squares problem solved efficiently in Section 3.6.

Algorithm 3.3, SquareRoot, handles all levels of this recursion. This algorithm splits a size-$N$ problem, with coefficients below $2^B$, into two size-$(N/2)$ problems, each with coefficients below about $2^{2B}$. Each split and recombination takes time bounded by essentially $NB$, so the total time is bounded by essentially $N^2 B$. Actually, the performance of SquareRoot is somewhat better than this, bounded by essentially $N^{\log_2 3} B$, since the second recursive call uses only $B$-bit coefficients.

**Theorem 3.2.** *Let $m, n$ be integers with $0 \le m \le n$. Let $d_1, \ldots, d_n$ be integers such that $\prod_{j \in J} d_j$ is non-square for each nonempty subset $J \subseteq \{1, \ldots, n\}$. Let $f$ be an element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$. Let $R$ be a subset of $\{m+1, \ldots, n\}$. De-*

*fine* $d = (d_1, \ldots, d_n)$ *and* $h = f^2 \prod_{j \in R} d_j$. *Then* SquareRoot$(d, m, h) = (g, S)$ *for some* $g \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$ *and some* $S \subseteq \{m+1, \ldots, n\}$ *such that* $h = g^2 \prod_{j \in S} d_j$. *Furthermore,* $S = \{\}$ *if* $g = 0$.

*Proof.* If $h = 0$ then $(g, S) = (0, \{\})$; see Step 2 of SquareRoot. Hence $g \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$, $S \subseteq \{m+1, \ldots, n\}$, and $h = 0 = g^2 \prod_{j \in S} d_j$.

Assume from now on that $h \neq 0$. Induct on $m$.

If $m = 0$ then by hypothesis $h = f^2 \prod_{j \in R} d_j$ for some $f \in \mathbb{Z}$ and some $R \subseteq \{1, \ldots, n\}$. Consequently, in Step 4 of SquareRoot, the TwistedSquareRoot$(d, h)$ call returns some $g \in \mathbb{Z}$ and some $S \subseteq \{1, \ldots, n\}$ such that $h = g^2 \prod_{j \in S} d_j$. By assumption $h \neq 0$, so $g \neq 0$, vacuously satisfying the conclusion that $S = \{\}$ if $g = 0$.

Assume from now on that $m \geq 1$. Write $f$ as $f_0 + f_1\sqrt{d_m}$, and write $h$ as $h_0 + h_1\sqrt{d_m}$, where $f_0, f_1, h_0, h_1 \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{m-1}}]$.

Define $\phi = \prod_{j \in R} d_j$. By hypothesis $h = f^2\phi = (f_0 + f_1\sqrt{d_m})^2\phi$; i.e., $h_0 = (f_0^2 + f_1^2 d_m)\phi$ and $h_1 = 2f_0 f_1 \phi$.

Note that $h_0^2 - h_1^2 d_m = (f_0^2 - f_1^2 d_m)^2 \phi^2$. By the inductive hypothesis, $s = \pm(f_0^2 - f_1^2 d_m)\phi$ and $S_0 = \{\}$ in Step 6. At this point the analysis splits into four cases, depending on whether $\pm = +$ and on whether $s = h_0$.

Case 1: $\pm = +$ and $s = h_0$. Then $f_1 = 0$ so $h_0 = f_0^2\phi$. Note that $f_0 \neq 0$ since $h \neq 0$. Now Step 8 produces $t = \pm f_0$ and $S = R$, and $u = 0$ since $h_1 = 0$, so the algorithm outputs $(t, S) = (\pm f, R)$ as desired.

Case 2: $\pm = -$ and $s = h_0$. Then $f_0 = 0$ so $h_0 = f_1^2 d_m \phi$. Note that $f_1 \neq 0$. Now Step 8 produces $t = \pm f_1$ and $S = R \cup \{m\}$, and $u = 0$ since $h_1 = 0$, so the algorithm outputs $(t\sqrt{d_m}, S - \{m\}) = (\pm f, R)$ as desired.

Case 3: $\pm = +$ and $s \neq h_0$. Then $0 \neq (h_0 - s)/2 = f_1^2 d_m \phi$. Now Step 10 produces $t = \pm f_1$ and $S = R \cup \{m\}$, Step 11 produces $\pi = \phi$, Step 12 produces $u = \pm f_0$, and the algorithm outputs $(u + t\sqrt{d_m}, S - \{m\}) = (\pm f, R)$ as desired.

Case 4: $\pm = -$ and $s \neq h_0$. Then $0 \neq (h_0 - s)/2 = f_0^2\phi$; Step 10 produces $t = \pm f_0$ and $S = R$; Step 11 produces $\pi = \phi$; Step 12 produces $u = \pm f_1$; and the algorithm outputs $(t + u\sqrt{d_m}, S) = (\pm f, R)$ as desired. □

**Theorem 3.3.** *Let* $n$ *be a nonnegative integer. Let* $d_1, \ldots, d_n$ *be integers such that* $\prod_{j \in J} d_j$ *is non-square for each nonempty subset* $J \subseteq \{1, \ldots, n\}$. *Let* $f$ *be an element of* $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$. *Then* SquareRoot$((d_1, \ldots, d_n), n, f^2) = (\pm f, \{\})$.

*Proof.* Define $m = n$, $d = (d_1, \ldots, d_n)$, $R = \{\}$, and $h = f^2 = f^2 \prod_{j \in R} d_j$. All the hypotheses of Theorem 3.2 are satisfied, so SquareRoot$(d, m, h) = (g, S)$ for some $g \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_m}]$ and some $S \subseteq \{m+1, \ldots, n\}$ such that $h = g^2 \prod_{j \in S} d_j$. Evidently $S = \{\}$, so $h = g^2$, so $g = \pm f$. □

*History and alternatives.* The idea of computing square roots recursively in multiquadratics is not new. Wada [60] introduced an exponential-time algorithm to compute unit groups of multiquadratic fields, as mentioned earlier; one of Wada's subroutines was a recursive square-root algorithm. However, Wada's algorithm was much slower and more complicated than SquareRoot, using many more recursive calls and many more cases. See [60, pages 202–204].

We also explored various square-root methods described in [21] and in more recent NFS literature. Computing square roots in all complex embeddings (to high enough precision) takes exponential time $2^{\Theta(N)}$, since each square root has its own choice of $\pm$. Computing square roots modulo many different prime ideals produces the same problem. Thomé's multi-prime method in [56] uses a subset-sum computation, again taking exponential time; see [56, Table 1]. Computing square roots modulo a large power of an inert prime would avoid this problem, but there are no inert primes for multiquadratics beyond degree 2 (as explained in more detail in Appendix A). We did try computing square roots modulo large powers of a first-degree prime ideal generated by $q, \sqrt{d_1} - s_1, \ldots, \sqrt{d_n} - s_n$, and this takes essentially linear time to find the image of $\pm f$ in $\mathbb{Z}/q^e$, but the fastest method that we found to reconstruct the coefficients of $\pm f$ uses LLL and takes more than quadratic time.

*A caveat regarding denominators of square roots.* Consider the problem of computing a square root in the field $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$. It might seem sufficient to multiply the denominator into the numerator and then compute a square root in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$. However, an element $f \in \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ can have $f^2 \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ without $f \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$: consider, for example, $f = (\sqrt{2} + \sqrt{6})/2$.

A workaround is to compute $\delta f \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ as the square root of $\delta^2 f^2 \in \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$. Here $\delta \neq 0$ is chosen in advance to guarantee that $\delta\mathcal{O} \subseteq \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$, where $\mathcal{O}$ is the ring of integers of $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$. For coprime squarefree $d_1, \ldots, d_n$ one can take $\delta = N$. For comparison, [21] takes $\delta = F'(x)$ to handle the same problem for arbitrary monogenic rings $\mathbb{Z}[x]/F(x)$.

We write the output of this computation, using SquareRoot as a subroutine, as $\sqrt{f^2}$, without further comment on the sign.

## 4 Recognizing squares

This section explains how to recognize squares in a multiquadratic field $L = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$. The method does not merely check whether a single element $u \in L$ is a square: given nonzero $u_1, \ldots, u_r \in L$, the method rapidly identifies the set of exponent vectors $(e_1, \ldots, e_r) \in \mathbb{Z}^r$ such that $u_1^{e_1} \cdots u_r^{e_r}$ is a square.

The method here was introduced by Adleman [2] as a speedup to NFS. The idea is to apply a group homomorphism $\chi$ from $L^\times$ to $\{-1, 1\}$, or more generally from $T$ to $\{-1, 1\}$, where $T$ is a subgroup of $L^\times$ containing $u_1, \ldots, u_r$. Then $\chi$ reveals a linear constraint, hopefully nontrivial, on $(e_1, \ldots, e_r)$ modulo 2. Combining enough constraints reveals the space of $(e_1, \ldots, e_r)$ mod 2.

One choice of $\chi$ is the sign of a real embedding of $L$, but this is a limited collection of $\chi$ (and empty if $L$ is complex). Adleman suggested instead taking $\chi$ as a quadratic character defined by a prime ideal. There is an inexhaustible supply of prime ideals, and thus of these quadratic characters.

Section 3.6 used this idea for $L = \mathbb{Q}$, but only for small $r$ (namely $r = n$), where one can afford to try $2^r$ primes. This section handles arbitrary multiquadratics and allows much larger $r$.

### 4.1 Computing quadratic characters

Let $q$ be an odd prime number modulo which all the $d_i$ are nonzero squares. For each $i$, let $s_i$ be a square root of $d_i$ modulo $q$. The map $\mathbb{Z}[x_1, \ldots, x_n] \to \mathbb{F}_q$ defined by $x_i \mapsto s_i$ and reducing coefficients modulo $q$ induces a homomorphism $\mathbb{Z}[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n) \to \mathbb{F}_q$, or equivalently a homomorphism $\varphi : \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}] \to \mathbb{F}_q$.

Let $\mathfrak{P}$ be the kernel of $\varphi$. Then $\mathfrak{P}$ is a degree-1 prime ideal of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ above $q$, i.e., a prime ideal of prime norm $q$. Write $\mathcal{O}_L$ for the ring of integers of $L$; then $\mathfrak{P}$ extends to a unique degree-1 prime ideal of $\mathcal{O}_L$. The map $\varphi$ extends to the set $R_\varphi$ of all $u \in L$ having nonnegative valuation at this prime ideal. For each $u \in R_\varphi$ define $\chi(u) \in \{-1, 0, 1\}$ as the Legendre symbol of $\varphi(u) \in \mathbb{F}_q$. Then $\chi(uu') = \chi(u)\chi(u')$, since $\varphi(uu') = \varphi(u)\varphi(u')$ and the Legendre symbol is multiplicative. In particular, $\chi(u^2) \in \{0, 1\}$.

More explicitly: Given a polynomial $u \in \mathbb{Z}[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$ represented as coefficients of $1$, $x_1$, $x_2$, $x_1x_2$, etc., first take all coefficients modulo $q$ to obtain $u \bmod q \in \mathbb{F}_q[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$. Then substitute $x_n \mapsto s_n$: i.e., write $u \bmod q$ as $u_0 + u_1 x_n$, where $u_0, u_1 \in \mathbb{F}_q[x_1, \ldots, x_{n-1}]/(x_1^2 - d_1, \ldots, x_{n-1}^2 - d_{n-1})$, and compute $u_0 + u_1 s_n$. Inside this result substitute $x_{n-1} \mapsto s_{n-1}$ similarly, and so on through $x_1 \mapsto s_1$, obtaining $\varphi(u) \in \mathbb{F}_q$. Finally compute the Legendre symbol modulo $q$ to obtain $\chi(u)$.

As in Section 3, assume that each coefficient of $u$ has at most $B$ bits, and choose $q$ (using the GoodPrime function from Section 3.2) to have $n^{O(1)}$ bits. Then the entire computation of $\chi(u)$ takes time essentially $NB$, mostly to reduce coefficients modulo $q$. The substitutions $x_j \mapsto s_j$ involve a total of $O(N)$ operations in $\mathbb{F}_q$, and the final Legendre-symbol computation takes negligible time.

More generally, any element of $L$ is represented as $u/h$ for a positive integer denominator $h$. Assume that $q$ is coprime to $h$; this is true with overwhelming probability when $q$ is chosen randomly. (It is also guaranteed to be true for any $u/h \in \mathcal{O}_L$ represented in lowest terms, since $q$ is coprime to $2d_1 \cdots d_n$.) Then $\varphi(u/h)$ is simply $\varphi(u)/h$, and computing the Legendre symbol produces $\chi(u/h)$.

### 4.2 Recognizing squares using many quadratic characters

Let $\chi_1, \ldots, \chi_m$ be quadratic characters. Define $T$ as the subset of $L$ on which all $\chi_i$ are defined and nonzero. Then $T$ is a subgroup of $L^\times$, the intersection of the unit groups of the rings $R_\varphi$ defined above. Define a group homomorphism $X : T \to (\mathbb{Z}/2)^m$ as $u \mapsto (\log_{-1} \chi_1, \ldots, \log_{-1} \chi_m)$.

Given nonzero $u_1, \ldots, u_r \in L$, choose $m$ somewhat larger than $r$, and then choose $\chi_1, \ldots, \chi_m$ randomly using GoodPrime. Almost certainly $u_1, \ldots, u_r \in T$; if any $\chi_i(u_j)$ turns out to be undefined or zero, simply switch to another prime.

Define $U$ as the subgroup of $T$ generated by $u_1, \ldots, u_r$. If a product $\pi = u_1^{e_1} \cdots u_r^{e_r}$ is a square in $L$ then its square root is in $T$ so $X(\pi) = 0$, i.e., $e_1 X(u_1) + \cdots + e_r X(u_r) = 0$. Conversely, if $X(\pi) = 0$ and $m$ is somewhat larger than $r$ then almost certainly $\pi$ is a square in $L$, as we now explain.

The group $U/(U \cap L^2)$ is an $\mathbb{F}_2$-vector space of dimension at most $r$, so its dual group $\mathrm{Hom}(U/(U \cap L^2), \mathbb{Z}/2)$ is also an $\mathbb{F}_2$-vector space of dimension at most $r$. As in [21, Section 8], we heuristically model $\log_{-1} \chi_1, \ldots, \log_{-1} \chi_m$ as independent uniform random elements of this dual; then they span the dual with probability at least $1 - 1/2^{m-r}$ by [21, Lemma 8.2]. If they do span the dual, then any $\pi \in U$ with $X(\pi) = 0$ must have $\pi \in U \cap L^2$.

The main argument for this heuristic is the fact that, asymptotically, prime ideals are uniformly distributed across the dual. Restricting to degree-1 prime ideals does not affect this heuristic: prime ideals are counted by norm, so asymptotically 100% of all prime ideals have degree 1. Beware that taking more than one prime ideal over a single prime number $q$ would not justify the same heuristic.

Computing $X(u_1), \ldots, X(u_r)$ involves $mr \approx r^2$ quadratic-character computations, each taking time essentially $NB$. We do better by using remainder trees to merge the reductions of $B$-bit coefficients mod $q$ across all $r$ choices of $q$; this reduces the total time from essentially $r^2NB$ to essentially $rN(r + B)$.

We write EnoughCharacters$(L, (v_1, \ldots, v_s))$ for a list of $m$ randomly chosen characters that are defined and nonzero on $v_1, \ldots, v_s$. In higher-level algorithms in this paper, the group $\langle v_1, \ldots, v_s \rangle$ can always be expressed as $\langle u_1, \ldots, u_r \rangle$ with $r \leq N + 1$, and we choose $m$ as $N + 64$, although asymptotically one should replace 64 by, e.g., $\sqrt{N}$. The total time to compute $X(u_1), \ldots, X(u_r)$ is essentially $N^2(N + B)$. The same heuristic states that these characters have probability at most $1/2^{63}$ (or asymptotically at most $1/2^{\sqrt{N}-1}$) of viewing some non-square $u_1^{e_1} \cdots u_r^{e_r}$ as a square. Our experiments have not encountered any failing square-root computations.

## 5 Computing units

This section presents a fast algorithm to compute the unit group $\mathcal{O}_L^\times$ of a multiquadratic field $L$. For simplicity we assume that $L$ is real, i.e., that $L \subseteq \mathbb{R}$. Note that a multiquadratic field is real if and only if it is totally real, i.e., every complex embedding $L \to \mathbb{C}$ has image in $\mathbb{R}$. For $L = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ this is equivalent to saying that each $d_j$ is nonnegative.

Like Wada [60], we recursively compute unit groups for three subfields $K_\sigma$, $K_\tau$, $K_{\sigma\tau}$, and then use the equation $u^2 = N_{L:K_\sigma}(u) N_{L:K_\tau}(u) / \sigma(N_{L:K_{\sigma\tau}}(u))$ to glue these groups together into a group $U$ between $\mathcal{O}_L^\times$ and $(\mathcal{O}_L^\times)^2$. At this point Wada resorts to brute-force search to identify the squares in $U$, generalizing an approach taken by Kubota in [42] for degree-4 multiquadratics ("biquadratics"). We reduce exponential time to polynomial time by using quadratic characters as explained in Section 4.

### 5.1 Fundamental units of quadratic fields

A **quadratic field** is, by definition, a degree-2 multiquadratic field; i.e., a field of the form $\mathbb{Q}(\sqrt{d})$, where $d$ is a non-square integer.

Fix a positive non-square integer $d$. Then $L = \mathbb{Q}(\sqrt{d})$ is a real quadratic field, and the unit group $\mathcal{O}_L^\times$ is

$$\{\ldots, -\varepsilon^2, -\varepsilon, -1, -\varepsilon^{-1}, -\varepsilon^{-2}, \ldots, \varepsilon^{-2}, \varepsilon^{-1}, 1, \varepsilon, \varepsilon^2, \ldots\}$$

for a unique $\varepsilon \in \mathcal{O}_L^\times$ with $\varepsilon > 1$. This $\varepsilon$, the smallest element of $\mathcal{O}_L^\times$ larger than 1, is the **normalized fundamental unit** of $\mathcal{O}_L$. For example, the normalized fundamental unit is $1+\sqrt{2}$ for $d = 2$; $2+\sqrt{3}$ for $d = 3$; and $(1+\sqrt{5})/2$ for $d = 5$.

Sometimes the literature says "fundamental unit" instead of "normalized fundamental unit", but sometimes it defines all of $\varepsilon$, $-\varepsilon$, $1/\varepsilon$, $-1/\varepsilon$ as "fundamental units". The phrase "normalized fundamental unit" is unambiguous.

The size of the normalized fundamental unit $\varepsilon$ is conventionally measured by the **regulator** $R = \ln(\varepsilon)$. A theorem by Hua states that $R < \sqrt{d}(\ln(4d) + 2)$, and experiments suggest that $R$ is typically $d^{1/2+o(1)}$, although it is often much smaller. Write $\varepsilon$ as $a + b\sqrt{d}$ with $a, b \in \mathbb{Q}$; then both $2a$ and $2b\sqrt{d}$ are very close to $\exp(R)$, and there are standard algorithms that compute $a, b$ in time essentially $R$, i.e., at most essentially $d^{1/2}$. See generally [45] and [61].

For our time analysis we assume that $d$ is quasipolynomial in $N$, i.e., $\log d \in (\log N)^{O(1)}$. Then the time to compute $\varepsilon$ is also quasipolynomial in $N$.

Take, for example, $d = d_1 \cdots d_n$, where $d_1, \ldots, d_n$ are the first $n$ primes, and write $N = 2^n$. The product of primes $\leq y$ is approximately $\exp(y)$, so $\ln d \approx n \ln n = (\log_2 N) \ln \log_2 N$. As a larger example, if $d_1, \ldots, d_n$ are primes between $N^3$ and $N^4$, and again $d = d_1 \cdots d_n$, then $\log_2 d$ is between $3n^2$ and $4n^2$, i.e., between $3(\log_2 N)^2$ and $4(\log_2 N)^2$. In both of these examples, $d$ is quasipolynomial in $N$.

*Subexponential algorithms.* There are much faster algorithms that compute $\varepsilon$ as a product of powers of smaller elements of $L$. There is a deterministic algorithm that provably takes time essentially $R^{1/2}$, i.e., at most essentially $d^{1/4}$; see [16]. Heuristic algorithms take subexponential time $\exp((\ln(d))^{1/2+o(1)})$, and thus time polynomial in $N$ if $\ln(d) \in O((\log N)^{2-\epsilon})$; see [20,1,25,58]. Quantum algorithms are even faster, as mentioned in the introduction, but in this paper we focus on pre-quantum algorithms.

This representation of units is compatible with computing products, quotients, quadratic characters (see Section 4), and automorphisms, but we also need to be able to compute square roots. One possibility here is to generalize from "product of powers" to any algebraic algorithm, i.e., any chain of additions, subtractions, multiplications, and divisions. This seems adequate for our square-root algorithm in Section 3.7: for example, $h_0$ inside Algorithm 3.3 can be expressed as the chain $(h + \sigma(h))/2$ for an appropriate automorphism $\sigma$, and the base case involves square roots of small integers that can be computed explicitly. However, it is not clear whether our recursive algorithms produce chains of polynomial size. This paper does not explore this possibility further.

### 5.2 Units in multiquadratic fields

Let $d_1, \ldots, d_n$ be integers satisfying the conditions of Theorem 2.1. Assume further that $d_1, \ldots, d_n$ are positive. Then $L = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ is a real multiquadratic field.

This field has $N - 1 = 2^n - 1$ quadratic subfields, all of which are real. Each quadratic subfield is constructed as follows: take one of the $N-1$ nonempty subsets $J \subseteq \{1, \ldots, n\}$; define $d_J = \prod_{j \in J} d_j$; the subfield is $\mathbb{Q}(\sqrt{d_J})$. We write the normalized fundamental units of these $N-1$ quadratic subfields as $\varepsilon_1, \ldots, \varepsilon_{N-1}$.

The set of **multiquadratic units** of $L$ is the subgroup $\langle -1, \varepsilon_1, \ldots, \varepsilon_{N-1} \rangle$ of $\mathcal{O}_L^\times$; equivalently, the subgroup of $\mathcal{O}_L^\times$ generated by $-1$ and all units of rings of integers of quadratic subfields of $L$. (The "$-1$ and" can be suppressed except for $L = \mathbb{Q}$.) A unit in $\mathcal{O}_L$ is not necessarily a multiquadratic unit, but Theorem 5.2 states that its $N$th power must be a multiquadratic unit.

The group $\mathcal{O}_L^\times$ is isomorphic to $(\mathbb{Z}/2) \times \mathbb{Z}^{N-1}$ by Dirichlet's unit theorem. For $N \geq 2$ this isomorphism takes the $N$th powers to $\{0\} \times (N\mathbb{Z})^{N-1}$, a subgroup having index $2^{1+n(N-1)}$. The index of the multiquadratic units in $\mathcal{O}_L^\times$ is therefore a divisor of $2^{1+n(N-1)}$. One corollary is that $\varepsilon_1, \ldots, \varepsilon_{N-1}$ are multiplicatively independent: if $\prod \varepsilon_j^{a_j} = 1$, where each $a_j \in \mathbb{Z}$, then each $a_j = 0$.

**Lemma 5.1** *Let $L$ be a real multiquadratic field and let $\sigma, \tau$ be distinct non-identity automorphisms of $L$. Define $\sigma\tau = \sigma \circ \tau$. For $\ell \in \{\sigma, \tau, \sigma\tau\}$ let $K_\ell$ be the subfield of $L$ fixed by $\ell$. Define $U = \mathcal{O}_{K_\sigma}^\times \cdot \mathcal{O}_{K_\tau}^\times \cdot \sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)$. Then*

$$(\mathcal{O}_L^\times)^2 \leq U \leq \mathcal{O}_L^\times.$$

*Proof.* $\mathcal{O}_{K_\sigma}^\times$, $\mathcal{O}_{K_\tau}^\times$, and $\mathcal{O}_{K_{\sigma\tau}}^\times$ are subgroups of $\mathcal{O}_L^\times$. The automorphism $\sigma$ preserves $\mathcal{O}_L^\times$, so $\sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)$ is a subgroup of $\mathcal{O}_L^\times$. Hence $U$ is a subgroup of $\mathcal{O}_L^\times$.

For the first inclusion, let $u \in \mathcal{O}_L^\times$. Then $N_{L:K_\ell}(u) \in \mathcal{O}_{K_\ell}^\times$ for $\ell \in \{\sigma, \tau, \sigma\tau\}$. Each non-identity automorphism of $L$ has order 2, so in particular each $\ell \in \{\sigma, \tau, \sigma\tau\}$ has order 2 (if $\sigma\tau$ is the identity then $\sigma = \sigma\sigma\tau = \tau$, contradiction), so $N_{L:K_\ell}(u) = u \cdot \ell(u)$. We thus have

$$\frac{N_{L:K_\sigma}(u) N_{L:K_\tau}(u)}{\sigma(N_{L:K_{\sigma\tau}}(u))} = \frac{u \cdot \sigma(u) \cdot u \cdot \tau(u)}{\sigma(u \cdot \sigma\tau(u))} = u^2.$$

Hence $u^2 = N_{L:K_\sigma}(u) N_{L:K_\tau}(u) \sigma(N_{L:K_{\sigma\tau}}(u^{-1})) \in U$. This is true for each $u \in \mathcal{O}_L^\times$, so $(\mathcal{O}_L^\times)^2$ is a subgroup of $U$. □

**Theorem 5.2** *Let $L$ be a real multiquadratic field of degree $N$. Let $Q$ be the group of multiquadratic units of $L$. Then $\mathcal{O}_L^\times = Q$ if $N = 1$, and $(\mathcal{O}_L^\times)^{N/2} \leq Q$ if $N \geq 2$. In both cases $(\mathcal{O}_L^\times)^N \leq Q$.*

*Proof.* Induct on $N$. If $N = 1$ then $L = \mathbb{Q}$ so $\mathcal{O}_L^\times = \langle -1 \rangle = Q$. If $N = 2$ then $L$ is a real quadratic field so $\mathcal{O}_L^\times = \langle -1, \varepsilon_1 \rangle = Q$ where $\varepsilon_1$ is the normalized fundamental unit of $L$.

Assume from now on that $N \geq 4$. By Theorem 2.2, $L$ can be expressed as $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ where $d_1, \ldots, d_n$ are positive integers meeting the conditions of Theorem 2.1 and $N = 2^n$.

Define $\sigma$ as the automorphism of $L$ that preserves $\sqrt{d_1}, \ldots, \sqrt{d_n}$ except for negating $\sqrt{d_n}$. The field $K_\sigma$ fixed by $\sigma$ is $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-1}})$, a real multiquadratic field of degree $N/2$. Write $Q_\sigma$ for the group of multiquadratic units of $K_\sigma$. By the inductive hypothesis, $(\mathcal{O}_{K_\sigma}^\times)^{N/4} \leq Q_\sigma \leq Q$.

Define $\tau$ as the automorphism of $L$ that preserves $\sqrt{d_1}, \ldots, \sqrt{d_n}$ except for negating $\sqrt{d_{n-1}}$. Then the field $K_\tau$ fixed by $\tau$ is $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_n})$, and the field $K_{\sigma\tau}$ fixed by $\sigma\tau$ is $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_{n-1}d_n})$. Both of these are real multiquadratic fields of degree $N/2$, so $(\mathcal{O}_{K_\tau}^\times)^{N/4} \leq Q$ and $(\mathcal{O}_{K_{\sigma\tau}}^\times)^{N/4} \leq Q$. The automorphism $\sigma$ preserves $Q$, so $\sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)^{N/4} \leq Q$.

By Lemma 5.1, $(\mathcal{O}_L^\times)^2 \leq \mathcal{O}_{K_\sigma}^\times \cdot \mathcal{O}_{K_\tau}^\times \cdot \sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)$. Simply take $(N/4)$th powers: $(\mathcal{O}_L^\times)^{N/2} \leq (\mathcal{O}_{K_\sigma}^\times)^{N/4} \cdot (\mathcal{O}_{K_\tau}^\times)^{N/4} \cdot \sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)^{N/4} \leq Q$. □

### 5.3 Representing units: logarithms and approximate logarithms

Sections 5.4 and 5.5 will use Lemma 5.1, quadratic characters, and square-root computations to obtain a list of generators for $\mathcal{O}_L^\times$. However, this is usually far from a minimal-size list of generators. Given this list of generators we would like to produce a **basis** for $\mathcal{O}_L^\times$. This means a list of $N-1$ elements $u_1, \ldots, u_{N-1} \in \mathcal{O}_L^\times$ such that each element of $\mathcal{O}_L^\times$ can be written uniquely as $\zeta u_1^{e_1} \cdots u_{N-1}^{e_{N-1}}$ where $\zeta$ is a root of unity; i.e., as $\pm u_1^{e_1} \cdots u_{N-1}^{e_{N-1}}$. In other words, it is a list of independent generators of $\mathcal{O}_L^\times / \{\pm 1\}$.

A basis $u_1, \ldots, u_{N-1}$ for $\mathcal{O}_L^\times$ is traditionally viewed as a lattice basis in the usual sense: specifically, as the basis $\operatorname{Log} u_1, \ldots, \operatorname{Log} u_{N-1}$ for the lattice $\operatorname{Log} \mathcal{O}_L^\times$, where Log is Dirichlet's logarithm map. However, this view complicates the computation of a basis. We instead view a basis $u_1, \ldots, u_{N-1}$ for $\mathcal{O}_L^\times$ as a basis $\operatorname{ApproxLog} u_1, \ldots, \operatorname{ApproxLog} u_{N-1}$ for the lattice $\operatorname{ApproxLog} \mathcal{O}_L^\times$, where ApproxLog is an "approximate logarithm map". We define our approximate logarithm map here, explain why it is useful, and explain how we use the approximate logarithm map in our representation of units. In Section 5.5 we use ApproxLog to reduce a list of generators to a basis.

*Dirichlet's logarithm map.* Let $\sigma_1, \sigma_2, \ldots, \sigma_N$ be (in some order) the embeddings of $L$ into $\mathbb{C}$, i.e., the ring homomorphisms $L \to \mathbb{C}$. Since $L$ is Galois, these are exactly the automorphisms of $L$. **Dirichlet's logarithm map** $\operatorname{Log} : L^\times \to \mathbb{R}^N$ is defined as follows:

$$\operatorname{Log}(u) = (\ln |\sigma_1(u)|, \ln |\sigma_2(u)|, \ldots, \ln |\sigma_N(u)|).$$

This map has several important properties. It is a group homomorphism from the multiplicative group $L^\times$ to the additive group $\mathbb{R}^N$. The kernel of Log restricted to $\mathcal{O}_L^\times$ is the cyclic group of roots of unity in $L$, namely $\{1, -1\}$. The image $\operatorname{Log}(\mathcal{O}_L^\times)$ forms a lattice of rank $N-1$, called the *log-unit lattice*.

Given units $u_1, \ldots, u_b$ generating $\mathcal{O}_L^\times$, one can compute $\mathrm{Log}(u_1), \ldots, \mathrm{Log}(u_b)$ in $\mathbb{R}^N$, and then reduce these images to linearly independent vectors in $\mathbb{R}^N$ by a chain of additions and subtractions, obtaining a basis for the log-unit lattice. Applying the corresponding chain of multiplications and divisions to the original units produces a basis for $\mathcal{O}_L^\times$.

However, elements of $\mathbb{R}$ are conventionally represented as nearby rational numbers. "Computing" $\mathrm{Log}(u_1), \ldots, \mathrm{Log}(u_b)$ thus means computing nearby vectors of rational numbers. The group generated by these vectors usually has rank larger than $N-1$: instead of producing $N-1$ linearly independent vectors and $b-(N-1)$ zero vectors, reduction can produce as many as $b$ linearly independent vectors.

One can compute approximate linear dependencies by paying careful attention to floating-point errors. An alternative is to use $p$-adic techniques as in [12]. Another alternative is to represent logarithms in a way that allows all of the necessary real operations to be carried out without error: for example, one can verify that $\ln|\sigma_1(u)| > \ln|\sigma_1(v)|$ by using interval arithmetic in sufficiently high precision, and one can verify that $\mathrm{Log}\, u = \mathrm{Log}\, v$ by checking that $u/v$ is a root of unity.

*Approximate logarithms.* We instead sidestep these issues by introducing an approximate logarithm function ApproxLog as a replacement for the logarithm function Log. This new function is a group homomorphism from $\mathcal{O}_L^\times$ to $\mathbb{R}^N$. Its image is a lattice of rank $N-1$, which we call the *approximate unit lattice*. Its kernel is the group of roots of unity in $L$. The advantage of ApproxLog over Log is that all the entries of $\mathrm{ApproxLog}(u)$ are rationals, allowing exact linear algebra.

To define ApproxLog, we first choose $N$ linearly independent vectors

$$\mathrm{ApproxLog}(\varepsilon_1), \ldots, \mathrm{ApproxLog}(\varepsilon_{N-1}), (1,1,\ldots,1) \in \mathbb{Q}^N,$$

where $\varepsilon_1, \ldots, \varepsilon_{N-1}$ are the normalized fundamental units of the quadratic subfields of $L$ as before; $(1,1,\ldots,1)$ is included here to simplify other computations. We then extend the definition by linearity to the group $\langle -1, \varepsilon_1, \ldots, \varepsilon_{N-1}\rangle$ of multiquadratic units: if

$$u = \pm \prod_{j=1}^{N-1} \varepsilon_j^{e_j}$$

then we define $\mathrm{ApproxLog}(u)$ as $\sum_j e_j \mathrm{ApproxLog}(\varepsilon_j)$. Finally, we further extend the definition by linearity to all of $\mathcal{O}_L^\times$: if $u \in \mathcal{O}_L^\times$ then $u^N$ is a multiquadratic unit by Theorem 5.2, and we define $\mathrm{ApproxLog}(u)$ as $\mathrm{ApproxLog}(u^N)/N$. It is easy to check that ApproxLog is a well-defined group homomorphism.

For example, one can take $\mathrm{ApproxLog}(\varepsilon_1) = (1,0,\ldots,0,0)$, $\mathrm{ApproxLog}(\varepsilon_2) = (0,1,\ldots,0,0)$, and so on through $\mathrm{ApproxLog}(\varepsilon_{N-1}) = (0,0,\ldots,1,0)$. Then $\mathrm{ApproxLog}(u) = (e_1/N, e_2/N, \ldots, e_{N-1}/N, 0)$ if $u^N = \pm\varepsilon_1^{e_1}\varepsilon_2^{e_2}\cdots\varepsilon_{N-1}^{e_{N-1}}$. In other words, write each unit modulo $\pm 1$ as a product of powers of $\varepsilon_1, \ldots, \varepsilon_{N-1}$; ApproxLog is then the exponent vector.

We actually define ApproxLog to be numerically much closer to Log. We choose a precision parameter $\beta$, and we choose each entry of $\mathrm{ApproxLog}(\varepsilon_j)$ to be a multiple of $2^{-\beta}$ within $2^{-\beta}$ of the corresponding entry of $\mathrm{Log}(\varepsilon_j)$. Specifically, we build $\mathrm{ApproxLog}(\varepsilon_j)$ as follows:

- Compute the regulator $R = \ln(\varepsilon_j)$ to slightly more than $\beta + \log_2 R$ bits of precision.
- Round the resulting approximation to a (nonzero) multiple $R'$ of $2^{-\beta}$.
- Build a vector with $R'$ at the $N/2$ positions $i$ for which $\sigma_i(\varepsilon_j) = \varepsilon_j$, and with $-R'$ at the remaining $N/2$ positions $i$.

The resulting vectors $\mathrm{ApproxLog}(\varepsilon_1), \ldots, \mathrm{ApproxLog}(\varepsilon_{N-1})$ are orthogonal to each other and to $(1, 1, \ldots, 1)$.

*How units are represented.* Each unit in Algorithms 5.1 and 5.2 is implicitly represented as a pair consisting of (1) the usual representation of an element of $L$ and (2) the vector $\mathrm{ApproxLog}(u)$. After the initial computation of $\ln(\varepsilon_j)$ for each $j$, all subsequent units are created as products (or quotients) of previous units, with sums (or differences) of the ApproxLog vectors; or square roots of previous units, with the ApproxLog vectors multiplied by $1/2$. This approach ensures that we do not have to compute $\ln|\sigma(u)|$ for the subsequent units $u$.

As mentioned in Section 5.1, we assume that each quadratic field $\mathbb{Q}(\sqrt{d})$ has $\log d \in (\log N)^{O(1)} = n^{O(1)}$, so $\log R \in n^{O(1)}$. We also take $\beta \in n^{O(1)}$, so each entry of $\mathrm{ApproxLog}(\varepsilon_j)$ has $n^{O(1)}$ bits. One can deduce an $n^{O(1)}$ bound on the number of bits in any entry of any ApproxLog vector used in our algorithms, so adding two such vectors takes time $n^{O(1)}N$, i.e., essentially $N$.

For comparison, recall that multiplication takes time essentially $NB$, where $B$ is the maximum number of bits in any *coefficient* of the field elements being multiplied. For normalized fundamental units, this number of bits is essentially $R$, i.e., quasipolynomial in $N$, rather than $\log R$, i.e., polynomial in $n$.

### 5.4 Pinpointing squares of units inside subgroups of the unit group

Algorithm 5.1, UnitsGivenSubgroup, is given generators $u_1, \ldots, u_b$ of any group $U$ with $(\mathcal{O}_L^\times)^2 \le U \le \mathcal{O}_L^\times$. It outputs generators of $\mathcal{O}_L^\times/\{\pm 1\}$.

The algorithm begins by building enough characters $\chi_1, \ldots, \chi_m$ that are defined and nonzero on $U$. Recall from Section 4.2 that $m$ is chosen to be slightly larger than $N$.

For each $u \in U$ define $X(u)$ as the vector $(\log_{-1}(\chi_1(u)), \ldots, \log_{-1}(\chi_m(u))) \in (\mathbb{Z}/2)^m$. If $u \in (\mathcal{O}_L^\times)^2$ then $X(u) = 0$. Conversely, if $u \in U$ and $X(u) = 0$ then (heuristically, with overwhelming probability) $u = v^2$ for some $v \in L$; this $v$ must be a unit, so $u \in (\mathcal{O}_L^\times)^2$.

The algorithm assembles the rows $X(u_1), \ldots, X(u_b)$ into a matrix $M$; computes a basis $S$ for the left kernel of $M$; lifts each element $(S_{i1}, \ldots, S_{ib})$ of this basis to a vector of integers, each entry 0 or 1; and computes $s_i = u_1^{S_{i1}} \cdots u_b^{S_{ib}}$. By definition $X(s_i) = S_{i1}X(u_1) + \cdots + S_{ib}X(u_b) = 0$, so $s_i \in (\mathcal{O}_L^\times)^2$. The algorithm computes a square root $v_i$ of each $s_i$, and it outputs $u_1, \ldots, u_b, v_1, v_2, \ldots$.

**Algorithm 5.1:** UnitsGivenSubgroup($L$, $(u_1, \ldots, u_b)$)

**Input**: A real multiquadratic field $L$; elements $u_1, \ldots, u_b$ of $\mathcal{O}_L^\times$ such that $(\mathcal{O}_L^\times)^2 \subseteq \langle u_1, \ldots, u_b \rangle$.
**Result**: Generators for $\mathcal{O}_L^\times / \{\pm 1\}$.

$\chi_1, \ldots, \chi_m \leftarrow$ EnoughCharacters$(L, (u_1, \ldots, u_b))$
$M \leftarrow [\log_{-1}(\chi_k(u_j))]_{1 \le j \le b, 1 \le k \le m}$
$S \leftarrow$ Basis(LeftKernel$(M)$)
**for** $i = 1, \ldots, \#S$ **do**
  $s_i \leftarrow \prod_j u_j^{S_{ij}}$, interpreting exponents in $\mathbb{Z}/2$ as $\{0, 1\}$ in $\mathbb{Z}$
  $v_i \leftarrow \sqrt{s_i}$
**return** $u_1, \ldots, u_b, v_1, \ldots, v_{\#S}$

To see that $-1, u_1, \ldots, u_b, v_1, v_2, \ldots$ generate $\mathcal{O}_L^\times$, consider any $u \in \mathcal{O}_L^\times$. By definition $u^2 \in (\mathcal{O}_L^\times)^2$, so $u^2 \in U$, so $u^2 = u_1^{e_1} \cdots u_b^{e_b}$ for some $e_1, \ldots, e_b \in \mathbb{Z}$. Furthermore $X(u^2) = 0$ so $e_1 X(u_1) + \cdots + e_b X(u_b) = 0$; i.e., the vector $(e_1 \bmod 2, \ldots, e_b \bmod 2)$ in $(\mathbb{Z}/2)^b$ is in the left kernel of $M$. By definition $S$ is a basis for this left kernel, so $(e_1 \bmod 2, \ldots, e_b \bmod 2)$ is a linear combination of the rows of $S$ modulo 2; i.e., $(e_1, \ldots, e_b)$ is some $(2f_1, \ldots, 2f_b)$ plus a linear combination of the rows of $S$; i.e., $u^2$ is $u_1^{2f_1} \cdots u_b^{2f_b}$ times a product of powers of $s_i$; i.e., $u$ is $\pm u_1^{f_1} \cdots u_b^{f_b}$ times a product of powers of $v_i$.

*Complexity analysis and improvements.* Assume that the inputs $u_1, \ldots, u_b$ have at most $B$ bits in each coefficient. Each of the products $s_1, s_2, \ldots$ is a product of at most $b$ inputs, and thus has, at worst, essentially $bB$ bits in each coefficient.

Computing the character matrix $M$ takes time essentially $bN(b + B)$; see Section 4.2. Computing $S$ takes $O(N^3)$ operations by Gaussian elimination over $\mathbb{F}_2$; one can obtain a better asymptotic exponent here using fast matrix multiplication, but this is not a bottleneck in any case. Computing one product $s_i$ takes time essentially $bNB$ with a product tree, and computing its square root $v_i$ takes time essentially $bN^{\log_2 3}B$. There are at most $b$ values of $i$.

Our application of this algorithm has $b \in \Theta(N)$. The costs are essentially $N^3 + N^2B$ for characters, $N^3$ for kernel computation, $N^3B$ for products, and $N^{2+\log_2 3}B$ for square roots.

These bounds are too pessimistic, for three reasons. First, experiments show that products often have far fewer factors, and are thus smaller and faster to compute. Second, one can enforce a limit upon the output size by integrating the algorithm with lattice-basis reduction (see Section 5.5), computing products and square roots only after reduction. Third, we actually use the technique of Section 3.5 to compute products of powers.

### 5.5 A complete algorithm to compute the unit group

Algorithm 5.2 computes a basis for $\mathcal{O}_L^\times$, given a real multiquadratic field $L$.

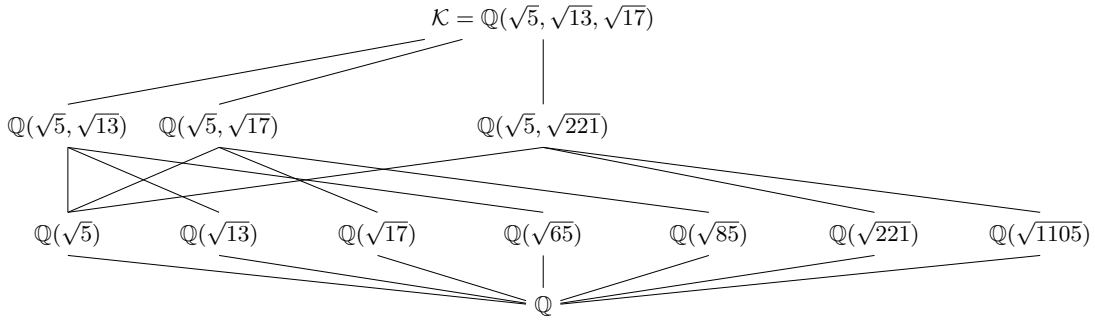



Fig. 5.1: How to pick subfields for the recursive algorithm for multiquadratic fields of degree 8.

As usual write $N$ for the degree of $L$. There is no difficulty if $N = 1$. For $N = 2$, the algorithm calls standard subroutines cited in Section 5.1. For $N \geq 4$, the algorithm calls itself recursively on three subfields of degree $N/2$; merges the results into generators for a subgroup $U \leq \mathcal{O}_L^\times$ such that $(\mathcal{O}_L^\times)^2 \leq U$; calls UnitsGivenSubgroup to find generators for $\mathcal{O}_L^\times$; and then uses lattice-basis reduction to find a basis for $\mathcal{O}_L^\times$. A side effect of lattice-basis reduction is that the basis is short, although it is not guaranteed to be minimal.

*The subgroup and the generators.* Lemma 5.1 defines $U = \mathcal{O}_{K_\sigma}^\times \cdot \mathcal{O}_{K_\tau}^\times \cdot \sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)$ where $\sigma, \tau$ are distinct non-identity automorphisms of $L$.

The three subfields used in the algorithm are $K_\sigma$, $K_\tau$, and $K_{\sigma\tau}$. The recursive calls produce lists of generators for $\mathcal{O}_{K_\sigma}^\times/\{\pm 1\}$, $\mathcal{O}_{K_\tau}^\times/\{\pm 1\}$, and $\mathcal{O}_{K_{\sigma\tau}}^\times/\{\pm 1\}$ respectively. The algorithm builds a list $G$ that contains each element of the first list; each element of the second list; $\sigma$ applied to each element of the third list; and $-1$. Then $G$ generates $U$. As a speedup, we sort $G$ to remove duplicates.

We cache the output of Units($L$) for subsequent reuse (without saying so explicitly in Algorithm 5.2). For example, if $L = \mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5})$, then the three subfields might be $\mathbb{Q}(\sqrt{2}, \sqrt{3})$, $\mathbb{Q}(\sqrt{2}, \sqrt{5})$, and $\mathbb{Q}(\sqrt{2}, \sqrt{15})$, and the next level of recursion involves $\mathbb{Q}(\sqrt{2})$ three times. We perform the Units($\mathbb{Q}(\sqrt{2})$) computation once and then simply reuse the results the next two times.

The overall impact of caching depends on how $\sigma$ and $\tau$ are chosen (which is also not specified in Algorithm 5.2). We use the following specific strategy. As usual write $L$ as $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$, where $d_1, \ldots, d_n$ are integers meeting the conditions of Theorem 2.1. Assume that $0 < d_1 < \cdots < d_n$. Choose $\sigma$ and $\tau$ such that $K_\sigma = \mathbb{Q}(\sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_{n-1}})$ and $K_\tau = \mathbb{Q}(\sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_n})$. We depict the resulting set of subfields in Figures 5.1 and 5.2. Notice that, in Figures 5.1 and 5.2, the leftmost field in each horizontal layer is a subfield used by all fields in the horizontal layer above it.

With this strategy, the recursion reaches exactly $2^{n-\ell+1} - 1$ subfields of degree $2^\ell$, namely the subfields of the form $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{\ell-1}}, \sqrt{D})$ where $D$ is a product of a nonempty subset of $\{d_\ell, \ldots, d_n\}$. With a less disciplined strategy, randomly picking 3 subfields of degree $N/2$ at each step, we would instead

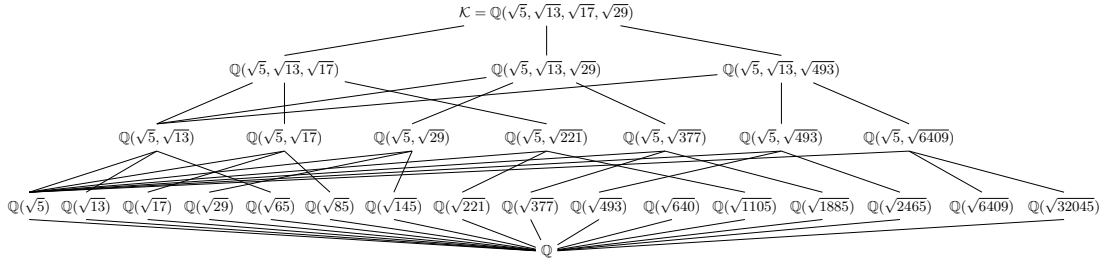



Fig. 5.2: How to pick subfields for the recursive algorithm for multiquadratic fields of degree 16.

end up with nearly $3^{n-\ell}$ subfields of degree $2^\ell$. "Nearly" accounts for accidental collisions and for the limited number of subfields of low degree.

*Finding short bases given generators.* Applying Pohst's modified LLL algorithm [53] to the vectors $\text{ApproxLog}(u_1), \ldots, \text{ApproxLog}(u_b)$ would find $b-(N-1)$ zero vectors and $N-1$ independent short combinations of the input vectors. The algorithm is easily extended to produce an invertible $b \times b$ transformation matrix $T$ that maps the input vectors to the output vectors. (The algorithm in [53] already finds the part of $T$ corresponding to the zero outputs.) We could simply use the entries of any such $T$ as exponents of $u_j$ in our algorithm. It is important to realize, however, that there are many possible choices of $T$ (except in the extreme case $b = N-1$), and the resulting computations are often much slower than necessary. For example, if $u_3 = u_1u_2$, then an output $u_1/u_2$ might instead be computed as $u_1^{1001}u_2^{999}/u_3^{1000}$.

We instead apply LLL to the matrix $A$ shown in Algorithm 5.2. This has three effects. First, if $H$ is chosen sufficiently large, then the right side of $A$ is reduced to $b-(N-1)$ zero vectors and $N-1$ independent short combinations of the vectors $H \cdot \text{ApproxLog}(u_1), \ldots, H \cdot \text{ApproxLog}(u_b)$. (We check that there are exactly $b-(N-1)$ zero vectors.) Second, the left side of $A$ keeps track of the transformation matrix that is used. Third, this transformation matrix is automatically reduced: short coefficients are found for the $b-(N-1)$ zero vectors, and these coefficients are used to reduce the coefficients for the $N-1$ independent vectors.

An upper bound on LLL cost can be computed as follows. LLL in dimension $N$, applied to integer vectors where each vector has $O(B)$ bits, uses $O(N^4B)$ arithmetic operations on integers with $O(NB)$ bits; see [44, Proposition 1.26] (and for lower exponents see, e.g., [50]). The total time is bounded by essentially $N^5B^2$. To bound $B$ one can bound each $H \cdot \text{ApproxLog}(\cdots)$. To bound $H$ one can observe that the transformation matrix has, at worst, essentially $N$ bits in each coefficient (see, e.g., [57]), while the required precision of ApproxLog is essentially 1, so it suffices to take essentially $N$ bits in $H$. The total time is, at worst, essentially $N^7$.

Our experiments show much better LLL performance for these inputs. We observe LLL actually using very few iterations; evidently the input vectors are

**Algorithm 5.2:** Units($L$)

**Input**: A real multiquadratic field $L$. As a side input, a parameter $H > 0$.
**Result**: Independent generators of $\mathcal{O}_L^\times/\{\pm 1\}$.

**if** $[L:\mathbb{Q}] = 1$ **then**
  **return** ()
**if** $[L:\mathbb{Q}] = 2$ **then**
  **return** the normalized fundamental unit of $L$
$\sigma, \tau \leftarrow$ distinct non-identity automorphisms of $L$
**for** $\ell \in \{\sigma, \tau, \sigma\tau\}$ **do**
  $G_\ell \leftarrow$ Units(fixed field of $\ell$)
$G \leftarrow -1, G_\sigma, G_\tau, \sigma(G_{\sigma\tau})$
$(u_1, \ldots, u_b) \leftarrow$ UnitsGivenSubgroup$(L, G)$
$A \leftarrow \begin{pmatrix} 1 & 0 & \ldots & 0 & H \cdot \text{ApproxLog}(u_1) \\ 0 & 1 & \ldots & 0 & H \cdot \text{ApproxLog}(u_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & H \cdot \text{ApproxLog}(u_b) \end{pmatrix}$
$A' \leftarrow \text{LLL}(A)$, putting shortest vectors first
**for** $i = 1, \ldots, N-1$ *where* $N = [L:\mathbb{Q}]$ **do**
  $w_i \leftarrow \prod_{1 \le j \le b} u_j^{A'_{b-(N-1)+i,j}}$
**return** $w_1, \ldots, w_{N-1}$

already very close to being reduced. It seems plausible to conjecture that the entries of the resulting transformation matrix have at most $n^{O(1)}$ bits, and that it suffices to take $H$ with $n^{O(1)}$ bits, producing $B$ bounded by $n^{O(1)}$. The total time might be as small as essentially $N^3$, depending on how many iterations there are.

## 6 Finding generators of ideals

This section presents the main contribution of this paper: a fast pre-quantum algorithm to compute a nonzero $g$ in a multiquadratic ring, given the ideal generated by $g$. For simplicity we focus on the real case, as in Section 5. The algorithm takes quasipolynomial time under reasonable heuristic assumptions if $d_1, \ldots, d_n$ are quasipolynomial.

The algorithm reuses the equation $g^2 = N_{L:K_\sigma}(g) N_{L:K_\tau}(g) / \sigma(N_{L:K_{\sigma\tau}}(g))$ that was used for unit-group computation in Section 5. To compute $N_{L:K}(g)$, the algorithm computes the corresponding norm of the input ideal, and then calls the same algorithm recursively.

The main algebraic difficulty here is that there are many generators of the same ideal: one can multiply $g$ by any unit, such as $-1$ or $1 + \sqrt{2}$, to obtain another generator. What the algorithm actually produces is some $ug$ where $u$ is a unit. This means that the recursion produces unit multiples of $N_{L:K_\sigma}(g)$

etc., and thus produces some $vg^2$ rather than $g^2$. The extra unit $v$ might not be a square, so we cannot simply compute the square root of $vg^2$. Instead we again use the techniques of Section 4, together with the unit group computed in Section 5, to find a unit $u$ such that $u(vg^2)$ is a square, and we then compute the square root.

### 6.1 Representing ideals and computing norms of ideals

Let $L$ be a real multiquadratic field of degree $N = 2^n$. Let $\mathcal{R}$ be an order inside $L$, such as $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ inside $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$. Our algorithm does not require $\mathcal{R}$ to be the ring of integers $\mathcal{O}_L$, although its output allows arbitrary units from the ring of integers; i.e., if the input is a principal ideal $\mathcal{I}$ of $\mathcal{R}$ then the output is some $g \in \mathcal{O}_L$ such that $g\mathcal{O}_L = \mathcal{I}\mathcal{O}_L$. Equivalently, one can (with or without having computed $\mathcal{O}_L$) view $\mathcal{I}$ as representing the ideal $\mathcal{I}\mathcal{O}_L$ of $\mathcal{O}_L$.

We consider three representations of an ideal $\mathcal{I}$ of $\mathcal{R}$:

- One standard representation is as a $\mathbb{Z}$-basis $\omega_1, \omega_2, \ldots, \omega_N \in \mathcal{R}$, i.e., a basis of $\mathcal{I}$ as a lattice.
- A more compact standard representation is the "two-element representation" $(\alpha_1, \alpha_2)$ representing $\mathcal{I} = \alpha_1\mathcal{R} + \alpha_2\mathcal{R}$, typically with $\alpha_1 \in \mathbb{Z}$. If $\mathcal{R} \neq \mathcal{O}_L$ then $\mathcal{I}$ might not have a two-element representation, but failure to convert $\mathcal{I}$ to a two-element representation reveals a larger order.
- Our target cryptosystem in Appendix A uses another representation that works for many, but certainly not all, ideals of $\mathcal{R} = \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$: namely, $(q, s_1, \ldots, s_n) \in \mathbb{Z}^{n+1}$, where each $s_j$ is a nonzero square root of $d_j$ modulo $q$ and where $q$ is odd, representing $\mathcal{I} = q\mathcal{R} + (\sqrt{d_1} - s_1)\mathcal{R} + \cdots + (\sqrt{d_n} - s_n)\mathcal{R}$.

Our algorithm works with any representation that allows basic ideal operations, such as ideal norms, which we discuss next. Performance depends on the choice of representation.

Let $\sigma$ be a nontrivial automorphism of $L$, and let $K$ be its fixed field; then $K$ is a subfield of $L$ with $[L : K] = 2$. Assume that $\sigma(\mathcal{R}) = \mathcal{R}$, and let $\mathcal{S}$ be the order $K \cap \mathcal{R}$ inside $K$. For example, if $\mathcal{R} = \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ and $\sigma$ preserves $\sqrt{d_1}, \ldots, \sqrt{d_{n-1}}$ while negating $\sqrt{d_n}$, then $\mathcal{S} = \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{n-1}}]$; if $\mathcal{R} = \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ and $\sigma$ preserves $\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}$ while negating $\sqrt{d_{n-1}}, \sqrt{d_n}$, then $\mathcal{S} = \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_{n-1}d_n}]$.

The relative norm $N_{L:K}(\mathcal{I})$ is, by definition, $\mathcal{I}\sigma(\mathcal{I}) \cap K$, which is the same as $\mathcal{I}\sigma(\mathcal{I}) \cap \mathcal{S}$. This is an ideal of $\mathcal{S}$. It has two important properties: it is not difficult to compute; and if $\mathcal{I} = g\mathcal{R}$ then $N_{L:K}(\mathcal{I}) = N_{L:K}(g)\mathcal{S}$. See, e.g., [26].

Given a $\mathbb{Z}$-basis of $\mathcal{I}$, one can compute a $\mathbb{Z}$-basis of $N_{L:K}\mathcal{I}$ by computing $\{\omega_i \cdot \sigma(\omega_j) : 1 \leq i \leq j \leq N\}$, transforming this into a Hermite-Normal-Form (HNF) basis for $\mathcal{I}\sigma(\mathcal{I})$, and intersecting with $\mathcal{S}$. A faster approach appears in [7]: compute a two-element representation of $\mathcal{I}$; multiply the two elements by a $\mathbb{Z}$-basis for $\sigma(\mathcal{I})$; convert to HNF form; and intersect with $\mathcal{S}$, obtaining a $\mathbb{Z}$-basis for $N_{L:K}\mathcal{I}$. This takes total time essentially $N^5B$.

**Algorithm 6.1:** IdealSqrt($L$, $h$)

**Input**: A real multiquadratic field $L$; an element $h$ of $\mathcal{O}_L^\times \cdot (L^\times)^2$.
**Result**: Some $g \in L^\times$ such that $h/g^2 \in \mathcal{O}_L^\times$.

$u_1, \ldots, u_{N-1} \leftarrow \text{Units}(L)$
$u_0 \leftarrow -1$
$\chi_1, \ldots, \chi_m \leftarrow \text{EnoughCharacters}(L, (u_0, \ldots, u_{N-1}, h))$
$M \leftarrow [\log_{-1} \chi_j(u_i)]_{0 \le i \le N-1, 1 \le j \le m}$
$V \leftarrow [\log_{-1} \chi_j(h)]_{1 \le j \le m}$
$[e_0, \ldots, e_{N-1}] \leftarrow \textsc{SolveLeft}(M, V)$
$u \leftarrow \prod_j u_j^{e_j}$, interpreting exponents in $\mathbb{Z}/2$ as $\{0, 1\}$ in $\mathbb{Z}$
$g \leftarrow \sqrt{uh}$
**return** $g$

The $(q, s_1, \ldots, s_n)$ representation allows much faster norms, and is used in our software. The norm to $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{n-1}}]$ is simply $(q, s_1, \ldots, s_{n-1})$, and the norm to $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_{n-2}}, \sqrt{d_{n-1}d_n}]$ is simply $(q, s_1, \ldots, s_{n-2}, s_{n-1}s_n)$.

### 6.2 Computing a generator of $\mathcal{I}$ from a generator of $\mathcal{I}^2$

Assume now that we have a nonzero principal ideal $\mathcal{I} \subseteq \mathcal{O}_L$, and a generator $h$ for $\mathcal{I}^2$. To find a generator $g$ for $\mathcal{I}$, it is sufficient to find a square generator for $\mathcal{I}^2$ and take its square root. To this end we seek a unit $u \in \mathcal{O}_L^\times$ such that $uh = g^2$ for some $g$. Applying the map $X$ from Section 4.2 to this equation, we obtain

$$X(uh) = X(g^2) = 2X(g) = 0.$$

Therefore $X(u) = X(h)$.

We start by computing $X(h)$ from $h$. We then compute a basis $u_1, \ldots, u_{N-1}$ for $\mathcal{O}_L^\times$, and we define $u_0 = -1$, so $u_0, u_1, \ldots, u_{N-1}$ generate $\mathcal{O}_L^\times$. We then solve the matrix equation

$$[e_0, e_1, \ldots, e_{N-1}] \begin{bmatrix} X(u_0) \\ X(u_1) \\ \vdots \\ X(u_{N-1}) \end{bmatrix} = X(h)$$

for $[e_0, e_1, \ldots, e_{N-1}] \in (\mathbb{Z}/2)^N$ and set $u = \prod_j u_j^{e_j}$. Then $uh$ is (almost certainly) a square, so its square root $g$ is a generator of $\mathcal{I}$. This algorithm is displayed as Algorithm 6.1.

The subroutine $\textsc{SolveLeft}(M, V)$ solves the matrix equation $eM = V$ for the vector $e$. One can save time by precomputing the inverse of an invertible full-rank submatrix of $M$, and using only the corresponding characters.

Note that for this computation to work we need a basis of the full unit group. If we instead use units $v_1, \ldots, v_{N-1}$ generating, e.g., the group $U = (\mathcal{O}_L^\times)^2$, and

**Algorithm 6.2:** ShortenGen$(L, h)$

**Input**: A real multiquadratic field $L$, and a nonzero element $h \in L$. As a side input, a positive integer parameter $\beta$.

**Result**: A short $g \in L$ with $g/h \in \mathcal{O}_L^\times$.

$u_1, \ldots, u_{N-1} \leftarrow \text{Units}(L)$

$$M \leftarrow \begin{pmatrix} \text{ApproxLog}(u_1) \\ \vdots \\ \text{ApproxLog}(u_{N-1}) \\ 1 \quad 1 \quad \ldots \quad 1 \quad 1 \end{pmatrix}$$

$v \leftarrow$ approximation to $\text{Log}(h)$ within $2^{-\beta}$ in each coordinate

$e \leftarrow \lfloor -vM^{-1} \rceil$

$g \leftarrow h u_1^{e_1} \cdots u_{N-1}^{e_{N-1}}$

**return** $g$

if $h = vg^2$ for some $v \in \mathcal{O}_L^\times - U$, then $uh$ cannot be a square for any $u \in U$: if it were then $h$ would be a square (since every $u \in U$ is a square), so $v$ would be a square, so $v$ would be in $U$, contradiction.

There are several steps in this algorithm beyond the unit-group precomputation. Characters for $u_0, \ldots, u_{N-1}$ take time essentially $N^3 + N^2B$ and can also be precomputed. Characters for $h$ take time essentially $N^2 + NB$. Linear algebra mod 2 takes time essentially $N^3$, or better with fast matrix multiplication; most of this can be precomputed, leaving time essentially $N^2$ to multiply a precomputed inverse by $X(h)$. The product of powers takes time essentially $N^2B$, and the square root takes time essentially $N^{1+\log_2 3}B$, although these bounds are too pessimistic for the reasons mentioned in Section 5.4.

### 6.3 Shortening

Algorithm 6.2, ShortenGen, finds a bounded-size generator $g$ of a nonzero principal ideal $\mathcal{I} \subseteq \mathcal{O}_L$, given any generator $h$ of $\mathcal{I}$. See Section 8 for analysis of the success probability of this algorithm at finding the short generators used in a cryptosystem.

Recall the log-unit lattice $\text{Log}(\mathcal{O}_L^\times)$ defined in Section 5.3. The algorithm finds a lattice point $\text{Log}\, u$ close to $\text{Log}\, h$, and then computes $g = h/u$.

In more detail, the algorithm works as follows. Start with a basis $u_1, \ldots, u_{N-1}$ for $\mathcal{O}_L^\times$. Compute $\text{Log}\, h$, and write $\text{Log}\, h$ as a linear combination of the vectors $\text{Log}(u_1), \ldots, \text{Log}(u_{N-1}), (1, 1, \ldots, 1)$; recall that $(1, 1, \ldots, 1)$ is orthogonal to each $\text{Log}(u_j)$. Round the coefficients in this combination to integers $(e_1, \ldots, e_N)$. Compute $u = u_1^{e_1} \cdots u_{N-1}^{e_{N-1}}$ and $g = h/u$.

The point here is that $\text{Log}\, h$ is close to $e_1 \text{Log}(u_1) + \cdots + e_{N-1} \text{Log}(u_{N-1}) + e_N(1, 1, \ldots, 1)$, and thus to $\text{Log}\, u + e_N(1, 1, \ldots, 1)$. The gap $\text{Log}\, g = \text{Log}\, h - \text{Log}\, u$ is between $-0.5$ and $0.5$ in each of the $\text{Log}(u_j)$ directions, plus some irrelevant amount in the $(1, 1, \ldots, 1)$ direction.

**Algorithm 6.3:** QPIP$(Q, \mathcal{I})$

**Input**: Real quadratic field $Q$ and a principal ideal $\mathcal{I}$ of an order inside $Q$
**Result**: A short generator $g$ for $\mathcal{I}\mathcal{O}_Q$

```
h ← FindQGen(Q, I)
g ← ShortenGen(Q, h)
return g
```

Normally the goal is to find a generator that is known in advance to be short. If the logarithm of this target generator is between $-0.5$ and $0.5$ in each of the $\operatorname{Log}(u_j)$ directions then this algorithm will find this generator (modulo $\pm 1$). See Section 8 for further analysis of this event.

*Approximations.* The algorithm actually computes $\operatorname{Log} h$ only approximately, and uses $\operatorname{ApproxLog} u_j$ instead of $\operatorname{Log} u_j$, at the expense of marginally adjusting the 0.5 bounds mentioned above.

Assume that $h$ has integer coefficients with at most $B$ bits. (We discard the denominator in any case: it affects only the irrelevant coefficient of $(1, 1, \ldots, 1)$.) Then $|\sigma_j(h)| \leq 2^B \prod_i (1 + \sqrt{|d_i|})$, so $\ln |\sigma_j(h)| \leq B \ln 2 + \sum_i \ln(1 + \sqrt{|d_i|})$. By assumption each $d_i$ is quasipolynomial in $N$, so $\ln |\sigma_j(h)| \leq B \ln 2 + n^{O(1)}$.

To put a *lower* bound on $\ln |\sigma_j(h)|$, consider the product of the other conjugates of $h$. Each coefficient of this product is between $-2^C$ and $2^C$ where $C$ is bounded by essentially $NB$. Dividing this product by the absolute norm of $h$, a nonzero integer, again produces coefficients between $-2^C$ and $2^C$, but also produces exactly $1/\sigma_j(h)$. Hence $\ln |1/\sigma_j(h)| \leq C \ln 2 + n^{O(1)}$.

In short, $\ln |\sigma_j(h)|$ is between essentially $-NB$ and $B$, so an approximation to $\ln |\sigma_j(h)|$ within $2^{-\beta}$ uses roughly $\beta + \log(NB)$ bits. We use interval arithmetic with increasing precision to ensure that we are computing $\operatorname{Log} h$ accurately; the worst-case precision is essentially $NB$. Presumably it would save time here to augment our representation of ideal generators to include approximate logarithms, the same way that we augment our representation of units, but we have not implemented this yet.

*Other reduction approaches.* Finding a lattice point close to a vector, with a promised bound on the distance, is called the *Bounded-Distance Decoding Problem* (BDD). There are many BDD algorithms in the literature more sophisticated than simple rounding: for example, Babai's nearest-plane algorithm [5]. See generally [32].

Our experiments show that, unsurprisingly, failures in rounding are triggered most frequently by the shortest vectors in our lattice bases. One cheap way to eliminate these failures is to enumerate small combinations of the shortest vectors.

**Algorithm 6.4:** MQPIP$(L, \mathcal{I})$

**Input**: Real multiquadratic field $L$ and a principal ideal $\mathcal{I}$ of an order inside $L$
**Result**: A short generator $g$ for $\mathcal{I}\mathcal{O}_L$

**if** $[L:\mathbb{Q}] = 1$ **then**
  **return** the smallest positive integer in $\mathcal{I}$
**if** $[L:\mathbb{Q}] = 2$ **then**
  **return** QPIP$(L, \mathcal{I})$
$\sigma, \tau \leftarrow$ distinct non-identity automorphisms of $L$
**for** $\ell \in \{\sigma, \tau, \sigma\tau\}$ **do**
  $K_\ell \leftarrow$ fixed field of $\ell$
  $\mathcal{I}_\ell \leftarrow N_{L:K_\ell}(\mathcal{I})$
  $g_\ell \leftarrow$ MQPIP$(K_\ell, \mathcal{I}_\ell)$
$h \leftarrow g_\sigma g_\tau / \sigma(g_{\sigma\tau})$
$g' \leftarrow$ IdealSqrt$(L, h)$
$g \leftarrow$ ShortenGen$(L, g')$
**return** $g$

## 6.4 Finding generators of ideals for quadratics

We now have all the ingredients for the attack algorithm. It will work in a recursive manner and in this subsection we will treat the base case.

Recall from Section 5.1 that there are standard algorithms to compute the normalized fundamental unit $\varepsilon$ of a real quadratic field $\mathbb{Q}(\sqrt{d})$ in time essentially $R = \ln(\varepsilon)$, which is quasipolynomial under our assumptions. There is, similarly, a standard algorithm to compute a generator of a principal ideal of $\mathcal{O}_{\mathbb{Q}(\sqrt{d})}$ in time essentially $R + B$, where $B$ is the number of bits in the coefficients used in the ideal. We call this algorithm FindQGen.

There are also algorithms that replace $R$ by something subexponential in $d$; see [59], [19], and [13]. As in Section 5.1, these algorithms avoid large coefficients by working with products of powers of smaller field elements, raising other performance questions in our context.

Algorithm 6.3, QPIP, first calls FindQGen to find a generator $h$, and then calls ShortenGen from Section 6.3 to find a short generator $g$. For quadratics this is guaranteed to find a generator with a minimum-size logarithm, up to the limits of the approximations used in computing logarithms.

## 6.5 Finding generators of ideals for multiquadratics

Algorithm 6.4 recursively finds generators of principal ideals of orders in real multiquadratic fields. The algorithm works as follows.

Assume, as usual, that $d_1, \ldots, d_n$ are positive integers meeting the conditions of Theorem 2.1. Let $L$ be the real multiquadratic field $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ of degree $N = 2^n$. Let $\mathcal{I}$ be a principal ideal of an order inside $L$, for which we want to find a generator.

| Precomp? | Subroutine | Cost |
|---|---|---|
| yes | units for all quadratic fields | $NB$ |
| yes | characters of units (in UGS, IS) | $N^3 + N^2B$ |
| yes | linear algebra (in UGS, IS) | $N^3$ without fast matrix multiplication |
| yes | basis reduction (in Units) | $N^7$; experimentally closer to $N^3$ |
| yes | products (in UGS, Units) | $N^3B$ |
| yes | square roots (in UGS) | $N^{2+\log_2 3}B$ |
| no | generators for all quadratic fields | $NB$ |
| no | characters for $h$ (in IS) | $N^2 + NB$ |
| no | linear algebra for $h$ (in IS) | $N^2$ |
| no | products (in IS, SG, MQPIP) | $N^2B$ |
| no | square roots (in IS) | $N^{1+\log_2 3}B$ |

Table 6.1: Complexities of subroutines at the top and bottom levels of recursion of MQPIP. Logarithmic factors are suppressed. $B$ is assumed to be at least as large as regulators. "UGS" means UnitsGivenSubgroup; "IS" means IdealSqrt; "SG" means ShortenGen. "Precomp" means that the results of the computation can be reused for many inputs $\mathcal{I}$.

If $N = 1$ then there is no difficulty. If $N = 2$, we find the generator with the QPIP routine of the previous section. Assume from now on that $N \geq 4$.

As in Section 5.5, choose distinct non-identity automorphisms $\sigma, \tau$ of $L$, and let $K_\sigma, K_\tau, K_{\sigma\tau}$ be the fields fixed by $\sigma, \tau, \sigma\tau$ respectively. These are fields of degree $N/2$.

For each $\ell \in \{\sigma, \tau, \sigma\tau\}$, compute $\mathcal{I}_\ell = N_{L:K_\ell}(\mathcal{I})$ as explained in Section 6.1, and call MQPIP$(K_\ell, \mathcal{I}_\ell)$ recursively to compute a generator $g_\ell$ for each $\mathcal{I}_\ell \mathcal{O}_{K_\ell}$. Notice that if $g$ is a generator of $\mathcal{I}\mathcal{O}_L$, then $g\ell(g)$ generates $\mathcal{I}_\ell \mathcal{O}_{K_\ell}$, so $g_\ell = u_\ell g\ell(g)$ for some $u_\ell \in \mathcal{O}_{K_\ell}^\times$. Therefore

$$\frac{g_\sigma g_\tau}{\sigma(g_{\sigma\tau})} = \frac{u_\sigma g\sigma(g) u_\tau g\tau(g)}{\sigma(u_{\sigma\tau} g\sigma\tau(g))} = g^2 u_\sigma u_\tau \sigma(u_{\sigma\tau}^{-1}),$$

so that $h = g_\sigma g_\tau / \sigma(g_{\sigma\tau})$ is a generator of $\mathcal{I}^2\mathcal{O}_L$. Now use IdealSqrt to find a generator of $\mathcal{I}\mathcal{O}_L$, and ShortenGen to find a bounded-size generator.

Table 6.1 summarizes the scalability of the subroutines inside MQPIP. Many of the costs are in precomputations that we share across many ideals $\mathcal{I}$, and these costs involve larger powers of $N$ than the per-ideal costs. On the other hand, the per-ideal costs can dominate when the ideals have enough bits $B$ per coefficient.

## 7 Timings

This section reports experiments on the timings of our software for our algorithms: specifically, the number of seconds used for various operations in the Sage [30] computer-algebra system on a single core of a 4GHz AMD FX-8350 CPU.

| $n$ | $2^n$ | mult | square | relnorm | absnorm | div | sqrt |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 0.0005 | 0.0004 | 0.0004 | 0.0023 | 0.0007 | 0.0130 |
| 4 | 16 | 0.0007 | 0.0006 | 0.0006 | 0.0049 | 0.0012 | 0.0528 |
| 5 | 32 | 0.0013 | 0.0010 | 0.0009 | 0.0109 | 0.0022 | 0.1352 |
| 6 | 64 | 0.0030 | 0.0022 | 0.0018 | 0.0250 | 0.0043 | 0.5408 |
| 7 | 128 | 0.0057 | 0.0042 | 0.0035 | 0.0574 | 0.0101 | 1.8964 |
| 8 | 256 | 0.0116 | 0.0085 | 0.0067 | 0.1332 | 0.0193 | 6.4766 |

Table 7.1: Observed time for basic operations in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$, with $d_1 = 2$, $d_2 = 3$, $d_3 = 5$, etc., using $\lambda = 64$. The "mult" column is the time to compute $h = fg$ where $f, g$ have each coefficient chosen randomly between $-2^{1000}$ and $2^{1000}-1$. The "square" column is the time to compute $f^2$. The "relnorm" column is the time to compute $f\sigma(f)$ where $\sigma$ is any of the automorphisms in Theorem 2.1. The "absnorm" column is the time to compute $N_{\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n}):\mathbb{Q}}f$. The "div" column is the time to divide $h = fg$ by $g$, recovering $f$. The "sqrt" column is the time to recover $\pm f$ from $f^2$. Each timing is the median of 101 measurements.

### 7.1 Basic subroutine timings

Table 7.1 shows the time taken for multiplication, squaring, etc., rounded to the nearest 0.0001 seconds: e.g., 0.0116 seconds to multiply two elements of a degree-256 multiquadratic ring, each element having random 1000-bit coefficients. The table is consistent with the analysis earlier in the paper: e.g., doubling the degree approximately doubles the cost of multiplication, and approximately triples the cost of square roots.

We have, for comparison, also explored the performance of multiquadratics using Sage's tower-field functions, Sage's absolute-number-field functions (using the polynomial $F$ defined in Appendix A), and Sage's ring constructors. The underlying polynomial-arithmetic code inside Sage is written in C, avoiding Python overhead, but suffers from poor algorithm scalability. Sage's construction of degree-2 relative extensions (in towers of number fields or in towers of rings) uses Karatsuba arithmetic, losing a factor of 3 for each extension, with no obvious way to enable FFTs. Working with one variable modulo $F$ produces good scalability for multiplication but makes norms difficult. Division is very slow in any case: for example, it takes 0.14 seconds, 1.5 seconds, and 31.5 seconds in degrees 32, 64, and 128 respectively using the tower-field representation, and it takes 0.12 seconds, 0.98 seconds, and 9.7 seconds in degrees 32, 64, and 128 respectively using the single-variable representation, while we use just 0.0101 seconds in degree 128 and 0.0193 seconds in degree 256.

### 7.2 Timings to compute the unit group and generators

The difference in scalability is much more striking for unit-group computation, as shown in Table 7.2. Our algorithm uses 2.33 seconds for degree 16, 6.61 seconds for degree 32, 23.30 seconds for degree 64, 93.02 seconds for degree 128, etc.,

| $n$ | $2^n$ | tower | absolute | new | new2 | new3 | attack | attack2 | attack3 |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 8 | 0.05 | 0.03 | 0.90 | 0.92 | 0.91 | 0.07 | 0.07 | 0.07 |
| 4 | 16 | 0.48 | 0.24 | 2.33 | 2.28 | 2.39 | 0.20 | 0.19 | 0.19 |
| 5 | 32 | 6.75 | 4.73 | 6.61 | 6.49 | 7.36 | 0.56 | 0.54 | 0.51 |
| 6 | 64 | >700000 | >700000 | 23.30 | 22.97 | 37.51 | 1.51 | 1.45 | 1.51 |
| 7 | 128 | | | 93.02 | 100.01 | 1560.49 | 4.95 | 5.18 | 7.29 |
| 8 | 256 | | | 463.91 | 650.92 | 31469.23 | 27.95 | 30.91 | 100.65 |

Table 7.2: Observed time to compute (once) the unit group of $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$; and to find a generator for the public key in the cryptosystem presented in Appendix A. The "tower" column is the time used by Sage's tower-field unit-group functions (with `proof=False`) with $d_1 = 2$, $d_2 = 3$, $d_3 = 5$, etc.; for $n = 6$ these functions ran out of memory after approximately 710000 seconds. The "absolute" column is the time used by Sage's absolute-field unit-group functions (also with `proof=False`), starting from the polynomial $F$ defined in Appendix A. The "new" column is the time used by this paper's unit-group algorithm. The "new2" column replaces $d_1, \ldots$ with the first $n$ primes $\geq n$. The "new3" column replaces $d_1, \ldots$ with the first $n$ primes $\geq n^2$. The "attack", "attack2", and "attack3" columns are 0.001 times the total time to find generators for 1000 public keys, including precomputation of the unit group.

slowing down by only a moderate factor for each doubling in the degree. Sage's internal C library uses under 5 seconds for degree 32, but we did not see it successfully compute a unit group for degree 64.

Table 7.2 also shows that our short-generator algorithm has similar scaling to our unit-group algorithm, as one would expect from the structure of the algorithms. As inputs we used public keys from a Gentry-style multiquadratic cryptosystem; see Appendix A. The number of bits per coefficient in this cryptosystem grows almost linearly with $2^n$, illustrating another dimension of scalability of our algorithm. See Section 8 for analysis of the success probability of the algorithm as an attack against the cryptosystem.

The dimensions we used in these experiments are below the $N = 8192$ recommended by Smart and Vercauteren for $2^{100}$ security against standard lattice-basis-reduction attacks, specifically BKZ. However, the Smart–Vercauteren analysis shows that BKZ scales quite poorly as $N$ increases; see Appendix A. Our attack should still be feasible for $N = 8192$, and a back-of-the-envelope calculation suggests that $N \approx 2^{20}$ is required for $2^{100}$ security against our attack.

## 8 Key-recovery probabilities

In this section we analyze the success probability of our algorithm recovering the secret key $g$ in a Gentry-style multiquadratic cryptosystem.

The specific system that we target is the system defined in Appendix A, the same system used for timings in Section 7.2. The secret key $g$ in this cryptosystem

is $g_0 + g_1\sqrt{d_1} + g_2\sqrt{d_2} + g_3\sqrt{d_1}\sqrt{d_2} + \cdots + g_{N-1}\sqrt{d_1}\cdots\sqrt{d_n}$, where $g_0, g_1, g_2, \ldots$ are independent random integers chosen from intervals

$$[-G, G], [-G/\sqrt{d_1}, G/\sqrt{d_1}], [-G/\sqrt{d_2}, G/\sqrt{d_2}], \ldots.$$

The distribution within each interval is uniform, except for various arithmetic requirements (e.g., $g$ must have odd norm) that do not appear to have any impact on the performance of our attack.

Section 8.1 presents heuristics for the expected size of $\operatorname{Log} g$ on the basis $\operatorname{Log}\varepsilon_1, \ldots, \operatorname{Log}\varepsilon_{N-1}$ for the logarithms of multiquadratic units, a sublattice of the log-unit lattice. Section 8.2 presents experimental data confirming these heuristics. Section 8.3 presents experimental data regarding the size of $\operatorname{Log} g$ on the basis that we compute for the full log-unit lattice. Section 8.4 presents an easier-to-analyze way to find $g$ when $\operatorname{Log}\varepsilon_1, \ldots, \operatorname{Log}\varepsilon_{N-1}$ are large enough.

### 8.1 MQ unit lattice: heuristics for $\operatorname{Log} g$

Write $U_L$ for the group of multiquadratic units in $L$. Recall that $U_L$ is defined as the group $\langle -1, \varepsilon_1, \ldots, \varepsilon_{N-1}\rangle$, where $\varepsilon_1, \ldots, \varepsilon_{N-1}$ are the normalized fundamental units of the $N-1$ quadratic subfields $\mathbb{Q}(\sqrt{D_1}), \ldots, \mathbb{Q}(\sqrt{D_{N-1}})$.

The logarithms $\operatorname{Log}\varepsilon_1, \ldots, \operatorname{Log}\varepsilon_{N-1}$ form a basis for the **MQ unit lattice** $\operatorname{Log} U_L$. This is an orthogonal basis: for example, for $\mathbb{Q}(\sqrt{2}, \sqrt{3})$, the basis vectors are $(x, -x, x, -x)$, $(y, y, -y, -y)$, and $(z, -z, -z, z)$ with $x = \ln(1+\sqrt{2})$, $y = \ln(2+\sqrt{3})$, and $z = \ln(5+2\sqrt{6})$. The general pattern (as in Section 5.3) is that $\operatorname{Log}\varepsilon_j$ is a vector with $R_j = \ln\varepsilon_j$ at $N/2$ positions and $-R_j$ at the other $N/2$ positions, specifically with $R_j$ at position $i$ if and only if $\sigma_i(\varepsilon_j) = \varepsilon_j$.

One consequence of orthogonality is that rounding on this basis is a perfect solution to the closest-vector problem for the MQ unit lattice. If 0 is the closest lattice point to $\operatorname{Log} g$, and $u$ is any multiquadratic unit, then rounding $\operatorname{Log} gu$ produces $\operatorname{Log} u$. One can decode beyond the closest-vector problem by enumerating some combinations of basis vectors, preferably the shortest basis vectors, but for simplicity we skip this option.

Write $c_j$ for the coefficient of $\operatorname{Log} g$ on the $j$th basis vector $\operatorname{Log}\varepsilon_j$; note that if each $c_j$ is strictly between $-0.5$ and $0.5$ then 0 is the closest lattice point to $\operatorname{Log} g$. Another consequence of orthogonality is that $c_j$ is simply the dot product of $\operatorname{Log} g$ with $\operatorname{Log}\varepsilon_j$ divided by the squared length of $\operatorname{Log}\varepsilon_j$; i.e., the dot product of $\operatorname{Log} g$ with a pattern of $N/2$ copies of $R_j$ and $N/2$ copies of $-R_j$, divided by $NR_j^2$; i.e., $Y/(NR_j)$, where $Y$ is the dot product of $\operatorname{Log} g$ with a pattern of $N/2$ copies of 1 and $N/2$ copies of $-1$.

We heuristically model $g_0$ as a uniform random real number from the interval $[-G, G]$; $g_1$ as a uniform random real number from $[-G/\sqrt{d_1}, G/\sqrt{d_1}]$; etc. In this model, each conjugate $\sigma_i(g)$ is a sum of $N$ independent uniform random real numbers from $[-G, G]$. For large $N$, the distribution of this sum is close to a Gaussian distribution with mean 0 and variance $G^2N/3$; i.e., the distribution of $(G\sqrt{N/3})\mathcal{N}$, where $\mathcal{N}$ is a normally distributed random variable with mean

0 and variance 1. The distribution of $\ln|\sigma_i(g)|$ is thus close to the distribution of $\ln(G\sqrt{N/3}) + \ln|\mathcal{N}|$.

Recall that $\mathrm{Log}(g)$ is the vector of $\ln|\sigma_i(g)|$ over all $i$, so $Y$ is $\ln|\sigma_1(g)| - \ln|\sigma_2(g)| + \cdots$ modulo an irrelevant permutation of indices. The mean of $\ln|\sigma_1(g)|$ is close to the mean of $\ln(G\sqrt{N/3}) + \ln|\mathcal{N}|$, while the mean of $-\ln|\sigma_2(g)|$ is close to the mean of $-\ln(G\sqrt{N/3}) - \ln|\mathcal{N}|$, etc., so the mean of $Y$ is close to 0. (For comparison, the mean of the sum of entries of $\mathrm{Log}(g)$ is close to $N\ln(G\sqrt{N/3}) + Nc$. Here $c \approx -0.6351814227$ is a universal constant, the average of $\ln|\mathcal{N}|$; the difference $-2c - \ln(2)$ is Euler's constant.)

To analyze the variance of $Y$, we heuristically model $\sigma_1(g), \ldots, \sigma_N(g)$ as independent. Then the variance of $Y$ is the variance of $\ln|\sigma_1(g)|$ plus the variance of $-\ln|\sigma_2(g)|$ etc. Each term is close to the variance of $\ln|\mathcal{N}|$, a universal constant $V \approx 1.2337005501$, so the variance of $Y$ is close to $VN$. The deviation of $Y$ is thus close to $\sqrt{VN}$, and the deviation of $c_j = Y/(NR_j)$ is close to $\sqrt{V}/(\sqrt{N}R_j) \approx 1.11072/(\sqrt{N}R_j)$.

To summarize, this model predicts that the coefficient of $\mathrm{Log}\, g$ on the $j$th basis vector $\mathrm{Log}\,\varepsilon_j$ has average approximately 0 and deviation approximately $1.11072/(\sqrt{N}R_j)$, where $R_j = \ln\varepsilon_j$. Recall that $R_j$ typically grows as $D_j^{1/2+o(1)}$.

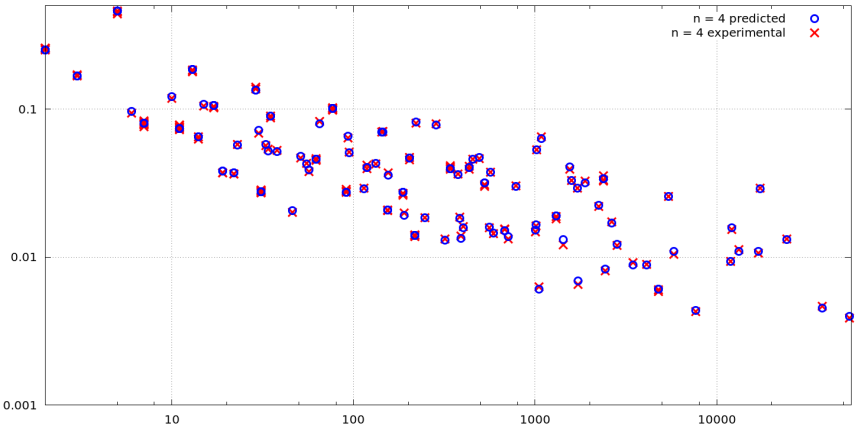
## 8.2 MQ unit lattice: experiments for $\mathrm{Log}\, g$

The experiments in Figure 8.1 confirm the prediction of Section 8.1. For each $n$, we took 10 possibilities for $n$ distinct primes $d_1, \ldots, d_n$ below $2n^2$. For each corresponding multiquadratic field, there are $N-1$ red crosses (sometimes overlapping). For each $D$ in $\{d_1, d_2, d_1d_2, \ldots, d_1d_2\cdots d_n\}$, one of these $N-1$ crosses is at horizontal position $D$. The vertical position is the observed average absolute coefficient of $\mathrm{Log}\, g$ in the direction of the basis vector corresponding to $D$, where $g$ ranges over 1000 secret keys for the $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$ cryptosystem. There is also a blue circle at the same horizontal position and at vertical position $1.11\sqrt{2/\pi}/(\sqrt{N}\cdot\ln\varepsilon_D)$; here $\sqrt{2/\pi}$ accounts for the average of $|\mathcal{N}|$.

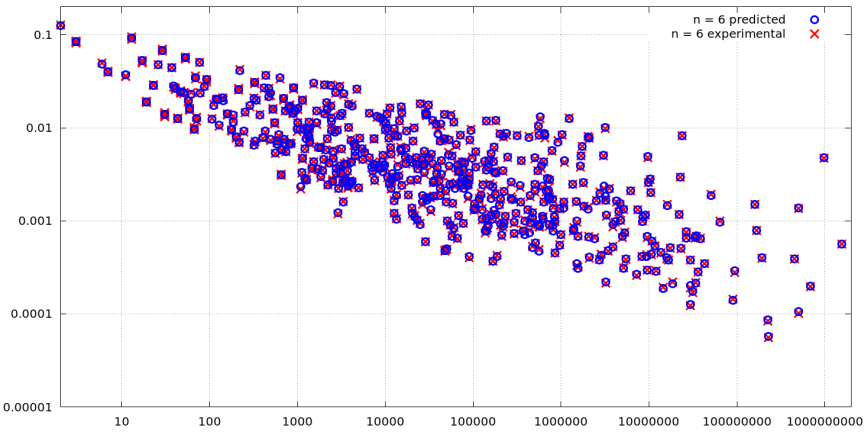
For all experiments we see a similar distribution in the blue circles (predictions) and the red crosses (experiment). We can even more strongly see this by rescaling the $x$-axis from $D$ to $1.11\sqrt{2/\pi}/(\sqrt{N}\cdot\ln\varepsilon_D)$, where $\varepsilon_D$ is again the normalized fundamental unit of $\mathbb{Q}(\sqrt{D})$. This rescaling of the crosses is shown in Figure 8.2.

After exploring these geometric aspects of the MQ unit lattice, we ran experiments on the success probability of rounding in the lattice. Figure 8.3 shows how often $\mathrm{Log}(g)$ is rounded to 0 (by simple rounding without enumeration) in our basis for the MQ unit lattice.

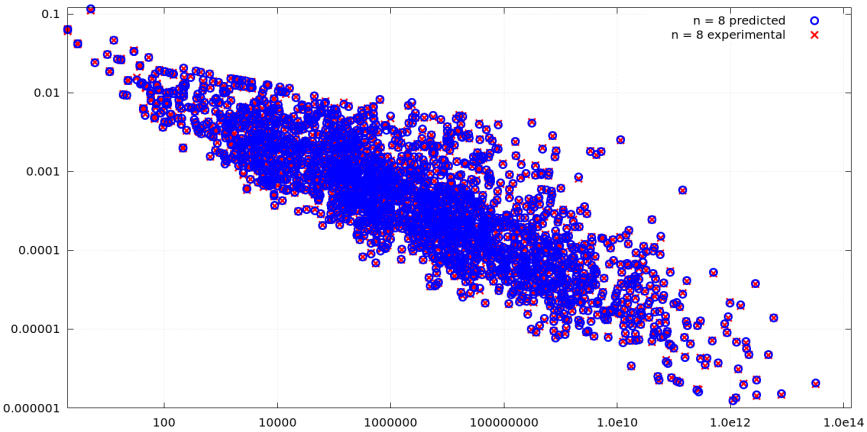
This graph shows a significant probability of failure if $d_1$ and $n$ are both small. Fields that contain the particularly short unit $(1+\sqrt{5})/2$ seem to be the worst case, as one would expect from our heuristics. However, even in this case, failures disappear as $n$ increases. The success probability seems to be uniformly bounded away from 0, seems to be above 90% for all fields with $d_1 \geq 7$ and $n \geq 4$, and seems to be above 90% for all fields with $n \geq 7$.




(a) $n = 4$




(b) $n = 6$




(c) $n = 8$

Fig. 8.1: Red crosses: For $n = 4, 6, 8$, the observed average absolute coefficient of $\mathrm{Log}(g)$ in the direction of the basis vector corresponding to $\mathbb{Q}(\sqrt{D})$. Blue circles: Predicted values.

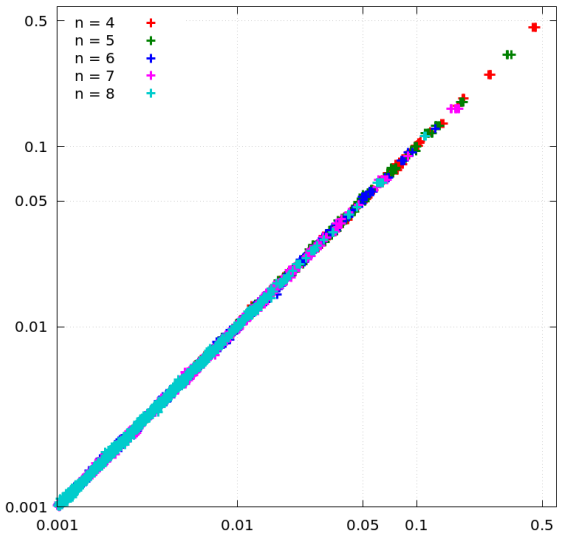



Fig. 8.2: Rescaling of the experiments of Figure 8.1, including $n \in \{4, 5, 6, 7, 8\}$.

## 8.3 Full unit lattice: experiments for Log $g$

Analyzing the full unit lattice is difficult, so we proceed directly to experiments. We first numerically compare the MQ unit lattice basis to the full unit lattice basis. The results of this are shown in Table 8.1. The index of $\mathrm{Log}(U_L)$ in $\mathrm{Log}(\mathcal{O}_L^\times)$ seems to grow as roughly $N^{0.3N}$.

In Table 8.2 we see the total success probability of the attack, with public keys provided as inputs, and with each successful output verified to match the corresponding secret key times $\pm 1$. The success probability is measured for 1000 trials after one precomputation. We noticed for $(n, j) = (7, 7)$ that running 1000 trials after another precomputation produced a significantly different success probability, presumably because random choices in the precomputation produced a significantly different basis. An attacker can try several precomputations and keep the one that achieves the maximum observed success probability.

We see that as the size and the number of the primes grow, the success probability increases, as was the case for the MQ unit basis. Specifically for the first $n$ primes after $n^2$ the success probability seems to rapidly converge towards 1, as was mentioned in Section 1.

## 8.4 Full unit lattice: an alternative strategy

The following alternative method of computing $g$ is easier to analyze asymptotically, because it does not require understanding the effectiveness of reduction in the full unit lattice. It does require $d_1, \ldots, d_n$ to be large enough compared to $N$, say larger than $N^{1.03}$, and it will obviously fail for many smaller $d_i$ where our experiments succeed, but it still covers a wide range of real multiquadratic number fields.

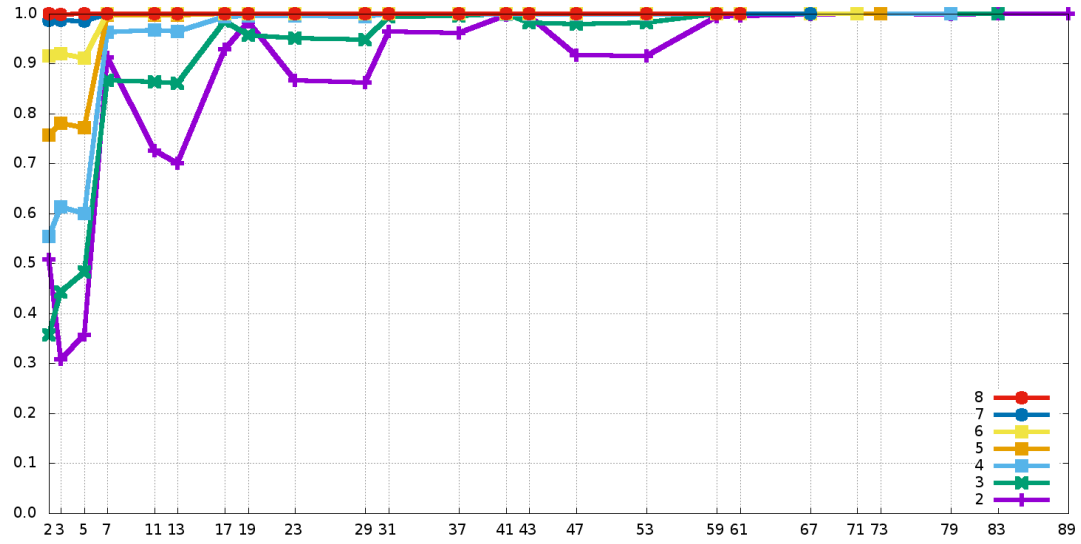



Fig. 8.3: Curves: $n = 2, 3, 4, 5, 6, 7, 8$. Horizontal axis: $d_1$, specifying $n$ consecutive primes $d_1, \ldots, d_n$. Vertical axis: Observed probability, for 10000 randomly drawn secret keys $g$ in the cryptosystem, that $\operatorname{Log} g$ is successfully rounded to 0 in the MQ unit lattice.

The point of requiring $d_1, \ldots, d_n$ to be larger than $N^{1.03}$ is that, for sufficiently large $N$ and most such choices of $d_1, \ldots, d_n$, the $n$ corresponding regulators $\ln(\varepsilon)$ are heuristically expected to be larger than $N^{0.51}$, and the remaining regulators for $d_1 d_2$ etc. are heuristically expected to be even larger. The coefficients of $\operatorname{Log} g$ on the MQ unit basis are then predicted to have deviation at most $1.11072/N^{1.01}$; see Section 8.1. We will return to this in a moment.

Compute, by our algorithm, some generator $gu$ of the public key $\mathcal{I}$. From Theorem 5.2 we know that $u^N$ is an MQ unit. Compute $N \operatorname{Log} gu$ and round in the MQ unit lattice. The coefficients of $N \operatorname{Log} g$ on the MQ unit basis are predicted to have deviation at most $1.11072/N^{0.01}$, so for sufficiently large $N$ these coefficients have negligible probability of reaching 0.5 in absolute value. Rounding thus produces $\operatorname{Log}(u^N)$ with high probability, revealing $\operatorname{Log}(g^N)$ and thus $\pm g^N$. Use a quadratic character to deduce $g^N$, compute the square root $\pm g^{N/2}$, use a quadratic character to deduce $g^{N/2}$, and so on through $\pm g$.

One can further extend the range of applicability of this strategy by finding a smaller exponent $e$ such that $u^e$ is always an MQ unit. Theorem 5.2 says $N/2$ for $N \geq 2$. By computing the MQ units for a particular field one immediately sees the minimum value of $e$ for that field; our computations suggest that $N/2$ is usually far from optimal.

## References

1. Christine Stefanie Abel. *Ein Algorithmus zur Berechnung der Klassenzahl und des Regulators reell-quadratischer Ordnungen.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1994.

| $n$ | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|
| average $\log_2 \|\|u^*\\|$ for $U_L$ | 1.113 | 1.623 | 2.338 | 2.953 | 3.504 | 4.089 |
| average $\log_2 \|\|u^*\\|$ for $\mathcal{O}_L^\times$ | 0.961 | 1.543 | 2.240 | 2.825 | 3.340 | 3.989 |
| average $\log_2(\#(\mathcal{O}_L^\times/U_L))$ | 5.730 | 17.230 | 43.560 | 108.140 | 255.200 | 580.960 |

Table 8.1: Experimental comparison of the MQ unit lattice $\mathrm{Log}(U_L)$, with basis formed by logarithms of the fundamental units of the quadratic subfields, and the full unit lattice $\mathrm{Log}(\mathcal{O}_L^\times)$, with basis produced by Algorithm 5.2. For each dimension $2^n$, $U_L$ and $\mathcal{O}_L^\times$ were computed for 100 random multiquadratic fields $L = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$, with $d_i$ primes bounded by $2n^2$. First row shows the average over these fields of $\log_2 ||u^*||$, where $||u^*||$ is the length of the smallest Gram–Schmidt vector of the basis for $U_L$. Second row shows the same for $\mathcal{O}_L^\times$. Third row shows the average of $\log_2$ of the index of $\mathrm{Log}(U_L)$ in $\mathrm{Log}(\mathcal{O}_L^\times)$.

| $n$ | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|
| $p_{\mathrm{suc}}(L_1)$ | 0.122 | 0.137 | 0.132 | 0.036 | 0.001 | 0.000 |
| $p_{\mathrm{suc}}(L_n)$ | 0.203 | 0.490 | 0.648 | 0.936 | 0.631 | 0.423 |
| $p_{\mathrm{suc}}(L_{n^2})$ | 0.784 | 0.981 | 1.000 | 1.000 | 1.000 | 1.000 |

Table 8.2: Observed attack success probabilities for various multiquadratic fields. By definition $L_j = \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$, with $d_i$ the first $n$ consecutive primes larger than or equal to $j$; and $p_{\mathrm{suc}}(L_j)$ is the fraction of keys out of 1000 trials that were successfully recovered without any enumeration by our attack on field $L_j$. Table covers $j \in \{1, n, n^2\}$.


2. Leonard M. Adleman. Factoring numbers using singular integers. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 64–71. ACM, 1991.
3. Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Des. Codes Cryptography*, 74(2):325–354, 2015.
4. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343. USENIX Association, 2016.
5. László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.
6. Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2014.

7. Karim Belabas. Topics in computational algebraic number theory. *Journal de Thorie des Nombres de Bordeaux*, 16(1):19–63, 2004.
8. Daniel J. Bernstein. Doubly focused enumeration of locally square polynomial values. In *High primes and misdemeanours: lectures in honour of the 60th birthday of Hugh Cowie Williams*, volume 41 of *Fields Inst. Commun.*, pages 69–76. Amer. Math. Soc., Providence, RI, 2004.
9. Daniel J. Bernstein. A subfield-logarithm attack against ideal lattices, 2014. https://blog.cr.yp.to/20140213-ideal.html.
10. Daniel J. Bernstein. Computational algebraic number theory tackles lattice-based cryptography, 2015. https://cr.yp.to/talks/2015.04.22/slides-djb-20150422-a4.pdf.
11. Jean-François Biasse, Thomas Espitau, Pierre-Alain Fouque, Alexandre Gélin, and Paul Kirchner. Computing generator in cyclotomic integer rings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 60–88, 2017. https://eprint.iacr.org/2017/142.
12. Jean-François Biasse and Claus Fieker. Improved techniques for computing the ideal class group and a system of fundamental units in number fields. In *Algorithmic Number Theory, 10th International Symposium, ANTS-IX, San Diego CA, USA, July 9-13, 2012. Proceedings*, volume 1 of *Open Book Series*, pages 113–133. Mathematical Sciences Publishers, 2012.
13. Jean-François Biasse, Michael J. Jacobson Jr., and Alan K. Silvester. Security estimates for quadratic field based cryptosystems. In *ACISP*, volume 6168 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2010.
14. Jean-François Biasse and Fang Song. On the quantum attacks against schemes relying on the hardness of finding a short generator of an ideal in $\mathbb{Q}(\zeta_{p^n})$, 2015. http://cacr.uwaterloo.ca/techreports/2015/cacr2015-12.pdf.
15. Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 893–902. SIAM, 2016.
16. Ingrid Biehl and Johannes Buchmann. Algorithms for quadratic orders. In *Mathematics of Computation 1943–1993: a half-century of computational mathematics*, volume 48 of *Proceedings of Symposia in Applied Mathematics*, pages 425–451. American Mathematical Society, 1994.
17. Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 553–570. IEEE Computer Society, 2015.
18. Johannes Buchmann, Markus Maurer, and Bodo Möller. Cryptography based on number fields with large regulator. *Journal de théorie des nombres de Bordeaux*, 12(2):293–307, 2000.
19. Johannes Buchmann and Ulrich Vollmer. *Binary Quadratic Forms: An Algorithmic Approach*. Algorithms and Computation in Mathematics. Springer Berlin Heidelberg, 2007.
20. Johannes A. Buchmann. A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In Catherine Goldstein, editor,

*Séminaire de Théorie des Nombres, Paris 1988–1989*, pages 27–41, Boston, 1990. Birkhauser.

21. Joe P. Buhler, Hendrik W. Lenstra, Jr., and Carl Pomerance. Factoring integers with the number field sieve. In *The development of the number field sieve*, volume 1554 of *Lecture Notes in Math.*, pages 50–94. Springer, Berlin, 1993.
22. Peter Campbell, Michael Groves, and Dan Shepherd. Soliloquy: a cautionary tale, 2014. ETSI 2nd Quantum-Safe Crypto Workshop, http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf.
23. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
24. Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 3–12, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
25. Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.
26. Henri Cohen. *Advanced Topics in Computational Number Theory*. Graduate Texts in Mathematics. Springer New York, 1999.
27. Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 607–628. Springer, 2016.
28. Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 559–585. Springer, 2016.
29. Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short Stickelberger class relations and application to Ideal-SVP. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 324–348, 2017. https://eprint.iacr.org/2016/885.
30. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.5.1)*, 2017. http://www.sagemath.org.
31. Kirsten Eisenträger, Sean Hallgren, Alexei Y. Kitaev, and Fang Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 293–302. ACM, 2014.

32. Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.
33. Steven D. Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Des. Codes Cryptography*, 78(1):51–72, 2016.
34. Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, EUROCRYPT'08, pages 31–51, Berlin, Heidelberg, 2008. Springer-Verlag.
35. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.
36. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. https://crypto.stanford.edu/craig.
37. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.
38. Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, 2010.
39. Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.
40. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
41. Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 543–571. Springer, 2016.
42. Tomio Kubota. Über den bizyklischen biquadratischen Zahlkörper. *Nagoya Math. J.*, 10:65–85, 1956.
43. Franz Lemmermeyer. Kuroda's class number formula. *Acta Arith.*, 66(3):245–260, 1994.
44. Arjen K. Lenstra, Hendrik W. Lenstra, and Laszlo Lovasz. Factoring polynomials with rational coefficients. *MATH. ANN*, 261:515–534, 1982.
45. Hendrik W. Lenstra, Jr. Solving the Pell equation. *Notices Amer. Math. Soc.*, 49(2):182–192, 2002.

46. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011*, CT-RSA'11, pages 319–339, Berlin, Heidelberg, 2011. Springer-Verlag.
47. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, November 2013.
48. Daniele Micciancio. Foundations of lattice cryptography, 2013. http://www.math.uci.edu/~asilverb/Lattices/Slides/Daniele1uci13-handout.pdf.
49. Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*. The Springer International Series in Engineering and Computer Science. Springer US, 2002.
50. 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.
51. Chris Peikert. Recovering short generators of principal ideals: extensions and open problems, 2015. http://www.math.uci.edu/~asilverb/Conference2015/Slides/Peikert-slides-uci.pdf.
52. Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.
53. Michael Pohst. A modification of the LLL reduction algorithm. *J. Symb. Comput.*, 4(1):123–127, 1987.
54. Oded Regev. On the complexity of lattice problems with polynomial approximation factors. In Phong Q. Nguyen and Brigitte Vallée, editors, *The LLL Algorithm - Survey and Applications*, Information Security and Cryptography, pages 475–496. Springer, 2010.
55. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
56. Emmanuel Thomé. Square root algorithms for the number field sieve. In Ferruh Özbudak and Francisco Rodríguez-Henríquez, editors, *Arithmetic of Finite Fields - 4th International Workshop, WAIFI 2012, Bochum, Germany, July 16-19, 2012. Proceedings*, volume 7369 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2012. https://members.loria.fr/EThome/files/nfs-sqrt.pdf.
57. Wilberd van der Kallen. Complexity of an extended lattice reduction algorithm, 1998. http://www.staff.science.uu.nl/~kalle101/complexity.pdf.
58. Ulrich Vollmer. Asymptotically Fast Discrete Logarithms in Quadratic Number Fields. In Wieb Bosma, editor, *Algorithmic Number Theory: 4th International Symposium, ANTS-IV Leiden, The Netherlands, July 2-7, 2000. Proceedings*, pages 581–594, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
59. Ulrich Vollmer. *Rigorously Analyzed Algorithms for the Discrete Logarithm Problem in Quadratic Number Fields*. PhD thesis, Technische Universität Darmstadt, October 2004.
60. Hideo Wada. On the class number and the unit group of certain algebraic number fields. *J. Fac. Sci., Univ. Tokyo, Sect. I*, 13:201–209, 1966.
61. Hugh C. Williams. Solving the Pell equation. In *Number theory for the millennium, III (Urbana, IL, 2000)*, pages 397–435. A K Peters, Natick, MA, 2002.

# A A multiquadratic cryptosystem

To generate realistic targets for our attack, we have implemented a Gentry-style cryptosystem using multiquadratic fields, including various optimizations. Most of the optimizations are due to Smart–Vercauteren [55] and Gentry–Halevi [39], although in some cases the original optimizations were specific to cyclotomic fields and required some adaptations for multiquadratics.

## A.1 Smart–Vercauteren for multiquadratics

The cryptosystem defined by Smart and Vercauteren [55, Section 3.1, "The Scheme"] uses $\mathbb{Z}[x]/F$ for any "monic irreducible polynomial" $F \in \mathbb{Z}[x]$. The cryptosystem does not actually require the ring $\mathbb{Z}[x]/F$ to be the ring of integers of the field.

Take $d_1, \ldots, d_n$ as in Theorem 2.1. The algebraic integer $\sqrt{d_1} + \cdots + \sqrt{d_n}$ is a root of a unique monic irreducible polynomial $F \in \mathbb{Z}[x]$. Explicitly, $\sqrt{d_1} + \cdots + \sqrt{d_n}$ has exactly $2^n$ distinct conjugates in the field $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$, namely $\mu_1\sqrt{d_1} + \cdots + \mu_n\sqrt{d_n}$ where $\mu_1, \ldots, \mu_n \in \{-1, 1\}^n$, so $F$ is the product of the linear polynomials $x - (\mu_1\sqrt{d_1} + \cdots + \mu_n\sqrt{d_n}) \in \mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})[x]$, and $\deg F = 2^n$. The field $\mathbb{Q}[x]/F$ is isomorphic to $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_n})$.

The Smart–Vercauteren cryptosystem repeatedly generates a short secret element $g$ of $\mathbb{Z}[x]/F$ until finding that $q$, the absolute value of $N_{\mathbb{Q}[x]/F:\mathbb{Q}}(g)$, is a prime number. There is then a unique ring homomorphism $\varphi : \mathbb{Z}[x]/F \to \mathbb{Z}/q$ that takes $g$ to 0. The public key in the cryptosystem is the pair $(q, r)$ where $r = \varphi(x)$, i.e., $r$ is a root of $g(x)$ modulo $q$. The ideal generated by $q$ and $x - r$ in $\mathbb{Z}[x]/F$ is the same as the ideal generated by $g$.

Gentry's original cryptosystem did not require $q$ to be prime, but it worked with a much larger, much slower representation for the public ideal generated by $g$. Smart–Vercauteren suggested instead using the classic "two-element representation" of an ideal, and in particular computing $r$ via standard algorithms to find roots of polynomials in the field $\mathbb{Z}/q$. Gentry–Halevi, in the special case of power-of-2 cyclotomics, suggested constructing $\varphi$ in a different way that works efficiently for any squarefree $q$ (and for some other $q$'s; the main failure cases are norms divisible by squares of *small* primes), avoiding the many iterations needed to find a prime.

Here is a multiquadratic adaptation of the Gentry–Halevi idea. View $g$ as an element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$. As before, let $q = \left|N_{\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n}):\mathbb{Q}}(g)\right|$. Compute $N_{\mathbb{Q}(\sqrt{d_1},\ldots,\sqrt{d_n}):\mathbb{Q}(\sqrt{d_1})}(g)$, obtaining some $a + b\sqrt{d_1}$ with $a, b \in \mathbb{Z}$. If $\gcd\{b, q\} > 1$, start over with a new $g$. (Note that $\pm q = a^2 - b^2 d_1$, so any prime dividing both $b$ and $q$ must also divide $a$, but then the square of the prime divides $q$; in other words, this restart will not happen if $q$ is squarefree.) If $\gcd\{b, q\} = 1$, find $k, \ell \in \mathbb{Z}$ with $kb + \ell q = 1$. Now $g$ divides both $a + b\sqrt{d_1}$ and $q\sqrt{d_1}$ in $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$, so it divides $k(a + b\sqrt{d_1}) + lq\sqrt{d_1} = ka + \sqrt{d_1}$; i.e., the image of $\sqrt{d_1}$ in $\mathbb{Z}/q$ is exactly $-ka$. Similarly compute the images of $\sqrt{d_2}, \ldots, \sqrt{d_n}$, and define $r$ as the sum of the images of $\sqrt{d_1}, \ldots, \sqrt{d_n}$.

The messages encrypted by Smart–Vercauteren are short elements $m$ of $\mathbb{Z}[x]/F$. The ciphertext $c$ corresponding to $m$ is simply $\varphi(m) \in \mathbb{Z}/q$. Addition and multiplication of messages, up to some limit, are simply addition and multiplication of ciphertexts in $\mathbb{Z}/q$.

Given $c$, the receiver lifts $c$ to $\{0, 1, \ldots, q-1\}$, computes $c/g$ in the field $\mathbb{Q}[x]/F$, rounds coefficients to integers to obtain an element of $\mathbb{Z}[x]/F$, multiplies by $g$, and subtracts from $c$. It is easy to see that this produces $m$ if each coefficient of $m/g \in \mathbb{Q}[x]/F$ is strictly between $-1/2$ and $1/2$.

Smart and Vercauteren choose the "short" in $g$ longer than the "short" in $m$ with the goal of limiting each coefficient of $m/g$ in this way. Beware that Silverberg later pointed out a flaw in the Smart–Vercauteren size analysis, creating decryption failures if some coefficients of $1/g$ happen to be larger than expected; the easiest fix is for the key generator to check the coefficients of $1/g$ and restart if necessary.

### A.2 Does the Smart–Vercauteren cryptosystem take polynomial time?

We observe that the Smart–Vercauteren key-generation algorithm is asymptotically much slower than polynomial time for some number fields. In particular, for degree-$2^n$ multiquadratic fields, the Smart–Vercauteren key-generation time is exponential in essentially $2^n$. Smart and Vercauteren suggest taking the field degree to be essentially quadratic in the target security level; the key-generation time for multiquadratics is then exponential in essentially the *square* of the target security level.

The issue is the expected number of choices of $g$ before the norm $q$ is prime. The Gentry–Halevi speedup does not help much: finding a squarefree norm (more precisely, a norm that survives the $\gcd\{b, q\} = 1$ requirement mentioned above) also takes many tries.

Sizes are chosen so that the total number of bits in the norm is bounded by a polynomial in $N = \deg F$, so one might guess that a prime norm will appear after a polynomial number of tries, and one might guess that a squarefree norm will appear after a constant number of tries. But these guesses turn out to be quite wrong for multiquadratic fields.

To understand the issue, fix a prime number $p$, and fix a field $k$ of size $p^2$. All integers are squares in $k$, so in particular $d_1, \ldots, d_n$ are squares in $k$. Choose square roots $s_1, \ldots, s_n$. Then the image of $F$ in $k[x]$ is the product of the linear polynomials $x - (\mu_1 s_1 + \cdots + \mu_n s_n)$.

Assume that the image of $F$ in $\mathbb{F}_p[x]$ factors as $h_1^{e_1} h_2^{e_2} \cdots$ where $h_1, h_2, \ldots$ are distinct irreducible polynomials in $\mathbb{F}_p[x]$. Then each $h_j$ divides the image of $F$ in $k[x]$, so $h_j$ factors into linear polynomials in $k[x]$, so the field $\mathbb{F}_p[x]/h_j$ maps to $k$ and is thus isomorphic to a subfield of $k$.

Each $\deg h_j$ must therefore be either 1 or 2. This limit on $\deg h_j$ is an extremely special property of multiquadratics. The limit almost guarantees that there are many irreducibles $h_1, h_2, \ldots$; the only other possibility would be surprisingly large powers in the factorization of $F$. Experiments confirm that there

are in fact many irreducibles, far more than what one would expect for a "random" polynomial $F$.

A uniform random element of $\mathbb{F}_p[x]/F$ has probability $1/p^2$ of being divisible by a quadratic $h_j$. The randomly chosen element $g$ of $\mathbb{Z}[x]/F$ has a distribution reasonably close to uniform modulo $p$ (if $p$ is not very large), so it has probability approximately $1/p^2$ of being divisible by $h_j$ modulo $p$. In this case the norm of $g$ must be divisible by $p^2$. For $g = 0$ this is trivial, and for $g \neq 0$ it follows from the fact that the absolute value of the norm of $g$ is the size of the ring $(\mathbb{Z}[x]/F)/g$, which has a ring homomorphism onto the field $\mathbb{F}_p[x]/h_j$ of size $p^2$.

Slightly less obvious is that divisibility by a *linear* $h_j$, something that occurs with much higher probability $1/p$, forces $p^2$ (not just $p$) to divide the norm of $g$, except in the rare case that all of $d_1, \ldots, d_n$ are squares in $\mathbb{F}_p$. The point here is that if any of $d_1, \ldots, d_n$ is a non-square in $\mathbb{F}_p$ then the ring of integers $\mathcal{O}$ cannot have any degree-1 primes lying over $p$. The exponent of $p$ in the norm of $g$ is the sum (with appropriate multiplicities) of the exponents of $p$ in the norms of ideals of $\mathcal{O}$ containing $g$, so it is either 0 or $\geq 2$.

These probabilities are approximately independent across $j$, and across $p$. In particular, there are approximately $2^n/\ln(2^n)$ primes $p \leq 2^n$, and experiments show that a typical prime contributes $\Theta(p)$ linear factors $h_j$, each of which is avoided by $g$ with probability approximately $1 - 1/p$. The product for each $p$ is approximately constant, and the product over all $p \leq 2^n$ is inverse exponential in $2^n/\ln(2^n)$. Quadratics and larger primes contribute further, less severe, slowdowns.

For $2^n = 16$, a computation applying the above analysis to the factors of $F$ for all $p < 10000$ predicts that norms will be squarefree with probability only about $0.09084 \approx 2^{-3.46}$. Experiments with 100000 random norms found 9099 norms avoiding all of these $p^2$. For $2^n = 64$, the prediction drops to $0.00934 \approx 2^{-6.74}$, and experiments with 100000 random norms found 963 norms avoiding all of these $p^2$. For $2^n = 256$, the prediction drops to $2^{-11.2}$. For $2^n = 1024$, the prediction drops to $2^{-21.9}$.

### A.3 Speeding up key generation

To avoid a seemingly neverending series of norm computations, we tweak the Smart–Vercauteren/Gentry–Halevi approach to allow much faster key generation for multiquadratics.

We begin by generalizing the approach to support $n$ polynomial variables, directly supporting the multiquadratic ring $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ rather than artificially working with the minimal polynomial $F$ of $\sqrt{d_1} + \cdots + \sqrt{d_n}$. Specifying $\varphi : \mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}] \to \mathbb{Z}/q$ then means specifying, for each $j$, a square root $r_j$ of $d_j$ in $\mathbb{Z}/q$. We compute these square roots using norms to $\mathbb{Q}(\sqrt{d_j})$ as explained above.

We force $g$ to be invertible modulo all primes $p \leq y$. Specifically, for each $p$, we construct a uniform random invertible element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]/p$ as explained in Appendix A.4. We glue these elements together into a uniform random element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]/\prod_{p \leq y} p$, which we view as an element $u$ of

$\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$ with each coefficient bounded in absolute value by $(1/2)\prod_{p \le y} p$. We then take $g$ in the form $u + s\prod_{p \le y} p$, where $s$ is a short secret element of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$.

We choose the parameter $y$ around $0.25N/\ln(N)$ where $N = 2^n$, so $\prod_{p \le y} p$ has roughly $0.35N/\ln(N)$ bits. We choose each coefficient of $s$ to have around $0.5N/\ln(N)$ bits. We give slightly more bits to the earlier coefficients, balancing the coefficients against the natural weights of 1, $\sqrt{d_1}$, $\sqrt{d_2}$, etc.

Here is a heuristic analysis indicating that taking $y$ on the scale of $N/\ln(N)$ produces a significant probability of squarefree norms. A typical prime $p > y$ splits into $N/2$ degree-2 prime ideals, each of which contains $g$ with probability $1/p^2$, so the norm of $g$ is coprime to $p$ with probability $(1-1/p^2)^{N/2}$. The product of these probabilities over all $p > y$ is $1/\exp(\Theta(N/(y \log y))$. Similar comments apply to the occasional primes $p$ that split differently. Even if $y$ is merely on the scale of $N/(\ln(N))^2$, the probability is inverse polynomial in $N$.

It is of course conceivable that our tweaks have introduced some serious weakness, allowing our target cryptosystem to be broken in a way that would not have been applicable to, e.g., the original Smart–Vercauteren system using $F$, or to another multiquadratic system that achieves faster key generation in a different way. We emphasize, however, that the objective of our experiments is to help confirm the performance of *our* attack. Recovering secret keys $g$ from this cryptosystem's public keys demonstrates that our attack scales smoothly to rather large coefficients in $g$. We are putting our attack software online to allow other people to carry out experiments with it, further reducing the risk of an unnoticed obstruction.

### A.4 Uniform random invertible ring elements modulo primes

A subroutine used above is to generate a uniform random invertible element of $R = k[x_1, \ldots, x_n]/(x_1^2 - d_1, \ldots, x_n^2 - d_n)$, where $d_1, \ldots, d_n$ are integers and $k$ is a finite field. We handle this as follows.

If $n = 0$, generate a uniform random nonzero element of $k$. From now on assume that $n \ge 1$, and write $R' = k[x_2, \ldots, x_n]/(x_2^2 - d_2, \ldots, x_n^2 - d_n)$. Then $R = R'[x_1]/(x_1^2 - d_1)$.

Case 1: $2d_1 = 0$ in $k$. Recursively generate a uniform random invertible element $f_0 \in R'$, and, independently of $f_0$, a uniform random element $f_1 \in R'$. Output $f_0 + f_1(x_1 - d_1)$.

To see that this is invertible, first note that $d_1^2 + d_1 = 0$ in $k$: if $d_1 = 0$ in $k$ then $d_1^2 + d_1 = 0$ in $k$, and if $2 = 0$ in $k$ then $d_1^2 + d_1 = 0$ in $k$ since $d_1^2 + d_1 \in 2\mathbb{Z}$. Hence $x_1^2 - d_1 = (x_1 - d_1)^2$ in $R'[x_1]$. The product of $f_0 + f_1(x_1 - d_1)$ and $f_0 - f_1(x_1 - d_1)$ is $f_0^2$ in $R$.

Conversely, each element of $R$ can be written uniquely as $f_0 + f_1(x_1 - d_1)$ with $f_0, f_1 \in R'$, and if $f_0 + f_1(x_1 - d_1)$ has inverse $g_0 + g_1(x_1 - d_1)$ in $R$ then $1 = f_0 g_0 + (f_0 g_1 + f_1 g_0)(x_1 - d_1)$ so $f_0 g_0 = 1$ in $R'$.

Case 2: $d_1$ is a nonzero square in $k$, and $2 \ne 0$ in $k$. Say $d_1 = s^2$ with $s \in k$. Recursively generate independent uniform random invertible elements $a, b \in R'$. Output $(a + b)/2 + (a - b)x_1/(2s)$.

The point here is that $(a, b) \mapsto (a+b)/2 + (a-b)x_1/(2s)$ is an isomorphism from $R' \times R'$ to $R$. Each element of $R$ can be written as $(a+b)/2+(a-b)x_1/(2s)$, and is invertible if and only if $a, b$ are both invertible in $R'$.

Case 3: $d_1$ is non-square in $k$. Write $k' = k[x_1]/(x_1^2 - d_1)$. Then $k'$ is a finite field. Recursively generate and output a uniform random invertible element of $k'[x_2, \ldots, x_n]/(x_2^2 - d_2, \ldots, x_n^2 - d_n)$.

### A.5 BKZ attack on Smart–Vercauteren

We follow Smart–Vercauteren's suggestion of $N$ as a power of the target security level $\lambda$, specifically $\lambda^{2+o(1)}$. Concretely, Smart and Vercauteren suggested $N = 8192$ for $2^{100}$ security using cyclotomics. These suggestions were based on conventional lattice-basis-reduction attacks. We now adapt the analysis of such attacks to multiquadratics, showing that $N \in \lambda^{2+o(1)}$ provides ample security asymptotically. Accurate analysis of BKZ performance for cryptographic sizes is difficult, but existing estimates suggest that our cryptosystem is extremely difficult to break by these attacks for the same $N = 8192$.

The first attack target is the secret key $g$. The public key $\mathcal{I} = gR$ is specified as its norm $q$ and the image $c_i \bmod q$ of each $\sqrt{d_i}$ modulo $I$. A ring element $f_0 + f_1\sqrt{d_1} + \cdots$ is in $I$ if and only if $f_0 + f_1 c_1 + \cdots$ is a multiple of $q$, i.e., if and only if $(f_0, f_1, \ldots)$ is some combination of the rows of the matrix

$$\Lambda = \begin{pmatrix} q & \mathbf{0} \\ -\mathbf{c} & I_{N-1} \end{pmatrix},$$

where $\mathbf{c} \in (\mathbb{Z}/q)^{N-1}$ consists of the products of all nonempty subsets of $c_i$. It is natural to multiply the columns by weights $(1, \sqrt{d_1}, \ldots)$, obtaining a matrix of determinant $q(d_1 \cdots d_n)^{N/2}$ whose row space contains the short vector $(g_0, g_1\sqrt{d_1}, \ldots)$.

Quantitatively, the cryptosystem takes each component of this vector as a random number between $-G$ and $G$ for some large $G$, so the vector is expected to have length approximately $G\sqrt{N/3}$. For comparison, as mentioned in Section 8, $\ln|q|$ is expected to be approximately $N\ln(G\sqrt{N/3}) + Nc$ where $c \approx -0.63518$ is a universal constant, the average of $\ln|\mathcal{N}|$. Hence $(q(d_1 \cdots d_n)^{N/2})^{1/N} \approx \gamma G\sqrt{N/3}$ where $\gamma = \sqrt{d_1 \cdots d_n}\exp c \approx 0.52984\sqrt{d_1 \cdots d_n}$.

In short, this short vector is smaller than the $N$th root of the determinant by a factor $\gamma$. Note that, since we have limited each $d_i$ to quasipolynomial in $N$, this approximation factor is also quasipolynomial in $N$.

Smart and Vercauteren also considered, as a better attack target, the plaintext message $m$. As mentioned above, $m$ needs to be somewhat shorter than $g$ for reliable decryption (several bits shorter in our experiments), correspondingly increasing the factor $\gamma$, but the factor is still quasipolynomial.

The standard estimate is that, in time $2^\lambda$, the smallest approximation factor reachable by BKZ (for most lattices) is asymptotically exponential in $(N \log \lambda)/\lambda$. For Smart and Vercauteren this is exponential in $\lambda^{1+o(1)}$, since $N \in \lambda^{2+o(1)}$. For comparison, the target approximation factor $\gamma$ is far smaller: it is exponential in

$(\log N)^{O(1)}$, i.e., exponential in $(\log \lambda)^{O(1)}$, which is far smaller than exponential in $\lambda^{1+o(1)}$. Reaching this approximation factor with BKZ takes time $2^{\lambda^{2+o(1)}}$, obviously very poor scalability for an attack when the target security level is $2^\lambda$.

The reason that Smart and Vercauteren take $N$ so large is to allow much shorter messages $m$: specifically, they allow $m$ to be shorter than $g$ by a factor as large as $2^{\lambda^{1+o(1)}}$. This reduces the security against BKZ to $2^{\lambda^{1+o(1)}}$. From this perspective, the difference between cyclotomics and multiquadratics is only a minor difference in the allowable gap between $m$ and $g$.

We can also see the poor scalability of BKZ by looking at concrete values of $N$. If we assume a multiquadratic field of the first $n$ primes bigger than respectively $1, n$ and $n^2$, then after computing the approximation factor $\gamma$, we need lattice-basis reduction on $\Lambda$ with a Hermite factor as shown in the following table:

| $n$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\delta_1 = \gamma_1^{1/N}$ | 1.1359 | 1.1064 | 1.0732 | 1.0475 | 1.0293 | 1.0176 | 1.0105 | 1.0061 | 1.0034 | 1.0019 |
| $\delta_n = \gamma_n^{1/N}$ | 1.2542 | 1.1706 | 1.1204 | 1.0725 | 1.0460 | 1.0264 | 1.0150 | 1.0084 | 1.0048 | 1.0027 |
| $\delta_{n^2} = \gamma_{n^2}^{1/N}$ | 1.4107 | 1.2964 | 1.1849 | 1.1157 | 1.0686 | 1.0402 | 1.0231 | 1.0133 | 1.0074 | 1.0041 |

Experiments from [34] show that the LLL algorithm gives a Hermite factor of 1.0219 for most lattices. LLL has comparable performance to our algorithm (see also Figure A.1), but is conjectured to not find the secret key beyond $n = 10$. In a sample size of 1 on the field $\mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11}, \sqrt{13}, \sqrt{17}, \sqrt{19}, \sqrt{23})$, the LLL attack returned a key of norm approximately $2^{9729.6} N_{L:\mathbb{Q}}(g)$, where $g$ is the secret key, in 11.1 days. For comparison, our attack on the field $\mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11}, \sqrt{13}, \sqrt{17}, \sqrt{19}, \sqrt{23}, \sqrt{29})$ took 4.3 days.

For BKZ the estimates of realistic running time vary in the literature (see e.g. [34,46,23,3]). For this purpose we will use the approach of [3], which estimates the running time of BKZ as

$$\log_2 t_{\mathrm{BKZ2.0}} = \frac{0.009}{\log_2^2 \delta} - 27.$$

Using this estimate, in Figure A.1 we graph the running times of an LLL attack on $\Lambda$, along with our attack results of Section 7.2. Beware that this BKZ estimate is not meaningful for $\delta$ above 1.0219 (i.e., for $n \leq 10$), and also does not account for the large sizes of our input coefficients, which is presumably why our observed LLL time is far above this BKZ estimate.

We see that our attack is conjectured to outperform BKZ greatly from around $n = 11$. For smaller multiquadratic fields this cross-over point is even earlier: e.g., for the multiquadratic field consisting of the first $n$ primes, the intersection occurs at $n = 10$.

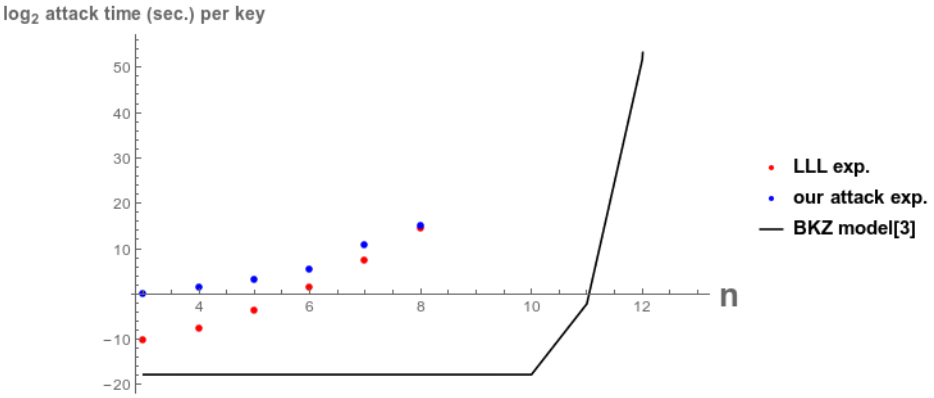



Fig. A.1: Experimental running times for LLL (red) and our attack (blue) compared to estimated BKZ running times (black). Experimental data was averaged over 1000 random keys for the field consisting of the first $n$ consecutive primes after $n^2$.

## B Recent progress in attacking Ideal-SVP

$\mathrm{SVP}_\gamma$ is the problem of finding, in a given lattice, a nonzero vector whose length is at most $\gamma$ times the length of the shortest nonzero vector in the lattice.

Micciancio and Goldwasser wrote in 2002 [49] that "To date, the best known polynomial time (possibly randomized) approximation algorithms for SVP and CVP achieve worst-case (over the choice of the input) approximation factors $\gamma(n)$ that are *essentially exponential* in the rank $n$" (emphasis added). The same approximation factors are featured on the right side of a picture displayed, for example, by Regev in 2010 [54]:

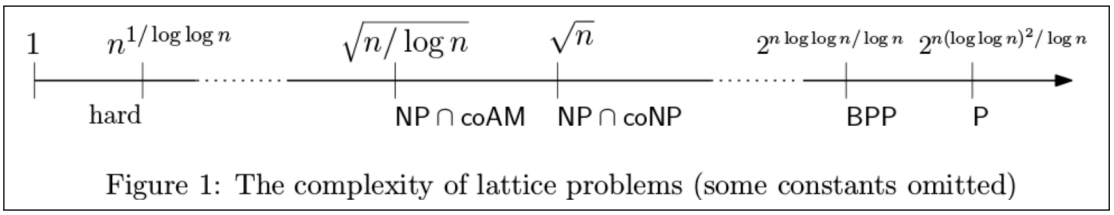



Figure 1: The complexity of lattice problems (some constants omitted)

Micciancio wrote in 2013 [48] that there is a "smooth trade-off between running time and approximation: $\gamma \approx 2^{O(n \log\log T/\log T)}$". Taking a polynomial run time $T$ produces the same essentially exponential approximation factor $\gamma$. Taking an exponential run time $T$ produces a polynomial approximation factor $\gamma$. In the middle is a run time $T$ of the form $\exp(\Theta(\sqrt{n \ln n}))$, i.e., $\exp(n^{1/2+o(1)})$, producing an approximation factor of the same form.

A new attack taking polynomial time—or quantum polynomial time—to reach an $\exp(n^{1/2+o(1)})$ approximation factor would be tremendous progress, shredding the standard tradeoff picture and jumping halfway to the $\exp(n^{o(1)})$ (e.g., polynomial) approximation factors commonly used in cryptography. For comparison, a similar jump to $\exp(n^{1/2+o(1)})$ in the time for integer factorization was followed by a jump to $\exp(n^{1/3+o(1)})$, and the underlying algo-

rithms broke widely used RSA key sizes. A similar jump in the time for small-characteristic multiplicative-group discrete logarithms was followed by a jump to $\exp(n^{1/3+o(1)})$, and then $\exp(n^{1/4+o(1)})$, and then $\exp(n^{o(1)})$, more specifically quasipolynomial.

This impressive jump to an $\exp(n^{1/2+o(1)})$ approximation factor is exactly what has happened for SVP for an important class of lattices, namely ideals for various cyclotomic fields. $\mathrm{SVP}_\gamma$ restricted to this class of inputs is typically called "Ideal-$\mathrm{SVP}_\gamma$" or "approximate Ideal-SVP" or simply "Ideal-SVP".

This jump began with the Biasse–Song–Campbell–Groves–Shepherd attack. This attack was initially described as handling an even more restricted type of input, namely ideals with short generators, and for this case it reaches an $\exp(n^{o(1)})$ approximation factor. Principal ideals very rarely have short generators. However, the same attack works for *arbitrary* principal ideals, provided that one allows an $\exp(n^{1/2+o(1)})$ approximation factor. See [28, Theorem 6.5] for a variant of this attack.

Most ideals in cyclotomic rings are not principal. However, Cramer, Ducas, and Wesolowski recently [29] used the Biasse–Song–Campbell–Groves–Shepherd attack as a subroutine to break *worst-case* Ideal-SVP inputs, again with an $\exp(n^{1/2+o(1)})$ approximation factor, again for various cyclotomic fields, under plausible assumptions.

This history illustrates the general point that successful attacks are usually outgrowths of successful attacks on simpler problems. From a cryptanalyst's perspective, ideals of the form $g\mathcal{O}$ are a natural and important example of ideal lattices, and they are an obvious starting point even if the objective is to handle more general ideals.

It is interesting to compare this progress to the comments just a few years ago from Lyubashevsky, Peikert, and Regev in [47, 2013 version online] regarding the difficulty of "SVP and other problems on ideal lattices": namely, "despite considerable effort, no significant progress in attacking these problems has been made. The best known algorithms for ideal lattices perform essentially no better than their generic counterparts, both in theory and in practice."

The Ideal-SVP instances highlighted in [47] (as the security foundation for "Ring-LWE"; see [52] for a survey of various related problems) are worst-case, so they are not broken by the Biasse–Song–Campbell–Groves–Shepherd attack. They also use $\exp(n^{o(1)})$ approximation factors, so they are not broken by the Cramer–Ducas–Wesolowski extension of the attack. Perhaps $\exp(n^{1/2+o(1)})$ is the end of the worst-case story, and the $\exp(n^{o(1)})$ worst-case problem will never be broken. On the other hand, perhaps $\exp(n^{1/2+o(1)})$ is an important step towards a complete break. The only way to find out is to continue to explore cryptanalytic algorithms.