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] Department of Mathematics, Simon Fraser University,
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, The Netherlands
`djb@cr.yp.to`, `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

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 [29] was shown several years later to be breakable by a fast quantum

algorithm *if* the underlying number field[1] 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 [38], and in an even faster cryptosystem from Gentry–Halevi [31]. Cyclotomic fields were used in all of the experiments reported in [31,38]. 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 [10], building upon a unit-group algorithm of Eisenträger, Hallgren, Kitaev, and Song [24]. This is the stage that uses quantum computation. The best known pre-quantum attacks 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 [17, p. 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 α [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."[2] This is the stage that relies on the field being cyclotomic.

A quantum algorithm for the first stage was stated in [17] before [10], but the effectiveness of this algorithm was disputed by Biasse and Song (see [9]) and was not defended by the authors of [17]. The algorithm in [17, p. 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.

---

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

[2] Beware that the analysis in [17, p. 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 [22] 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 [28,30]; it does not break the classic NTRU system [32]; and it does not break the BCNS [12] and New Hope [3] 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—even if the ideal actually has a short generator, and even if the output is allowed to be a long generator. There is a list of five "main computational tasks of algebraic number theory" in [19, p. 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 [13] a decade earlier relying on the hardness of this problem, and summarize various slow solutions.

Fourth, this problem has been reused in various attempts to build secure multilinear maps, starting with the Garg–Gentry–Halevi construction [27]. We do not mean to overstate the security or applicability of multilinear maps (see, e.g., [18,21]), 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 (in the full version of this paper), 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 Sect. 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

$$\mathbb{Q}(\sqrt{2}), \mathbb{Q}(\sqrt{3}), \mathbb{Q}(\sqrt{6}), \ldots, \mathbb{Q}(\sqrt{2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19}).$$

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 $\pm(69158780182494876719 + 22205900901368228\sqrt{2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19})^{\mathbb{Z}}$.

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 Sect. 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 [19]. There is an earlier algorithm by Wada [42] to compute the unit group of a multiquadratic field, but that algorithm takes exponential time.

We then go even further (Sect. 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., [5, 26, 33].

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 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 Figs. 5.1 and 5.2, the number of subfields we use ends up essentially linear in 2^n.

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.

See full version of paper on multiquad.cr.yp.to for proofs.

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.

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

3   Fast Arithmetic in Multiquadratic Fields

See full version of paper on multiquad.cr.yp.to.

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 χ 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 χ 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 χ is the sign of a real embedding of L, but this is a limited collection of χ (and empty if L is complex). Adleman suggested instead taking χ 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 (in the full version of this paper) 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 φ. 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 φ extends to the set R_φ 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 $1, x_1, x_2, x_1 x_2$, 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 Sect. 3 (in the full version of this paper), assume that each coefficient of u has at most B bits, and choose q (using the GoodPrime function from Sect. 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 χ_i are defined and nonzero. Then T is a subgroup of $L^\times$, the intersection of the unit groups of the rings R_φ 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 $\chi(u_j)$ 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 π 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 [16, Sect. 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 [16, 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 [42], we recursively compute unit groups for three subfields K_σ, K_τ, $K_{\sigma\tau}$, 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 [34] for degree-4 multiquadratics ("biquadratics"). We reduce exponential time to polynomial time by using quadratic characters as explained in Sect. 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 ε, 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 ε 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 ε 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 [36,43].

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 ε 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 ε 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 [11]. 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 [1,15,19,40]. 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 Sect. 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 Sect. 3.7 (in the full version of this paper): for example, h_0 inside Algorithm 3.3 can be expressed as the chain $(h + \sigma(h))/2$ for an appropriate automorphism σ, 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. We do 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 Nth 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 Nth 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 σ, τ be distinct non-identity automorphisms of L. Define $\sigma\tau = \sigma \circ \tau$. For $\ell \in \{\sigma, \tau, \sigma\tau\}$ let K_ℓ be the subfield of L fixed by ℓ. 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 σ 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 ε_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 σ as the automorphism of L that preserves $\sqrt{d_1}, \ldots, \sqrt{d_n}$ except for negating $\sqrt{d_n}$. The field K_σ fixed by σ is $\mathbb{Q}(\sqrt{d_1}, \ldots, \sqrt{d_{n-1}})$, a real multiquadratic field of degree $N/2$. Write Q_σ for the group of multiquadratic units of K_σ. By the inductive hypothesis, $(\mathcal{O}_{K_\sigma}^\times)^{N/4} \leq Q_\sigma \leq Q$.

Define τ 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_τ fixed by τ 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 σ 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: $(\mathcal{O}_L^\times)^{N/2} \leq (\mathcal{O}_{K_\sigma}^\times)^{N/4} \cdot (\mathcal{O}_{K_\tau}^\times)^{N/4} \cdot \sigma(\mathcal{O}_{K_{\sigma\tau}}^\times)^{N/4} \leq Q$. $\qquad\square$

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 ζ 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 $\text{Log } u_1, \ldots, \text{Log } u_{N-1}$ for the lattice $\text{Log } \mathcal{O}_L^\times$, 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 $\text{ApproxLog } u_1, \ldots, \text{ApproxLog } u_{N-1}$ for the lattice $\text{ApproxLog } \mathcal{O}_L^\times$, 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 Sect. 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** $\text{Log} : L^\times \to \mathbb{R}^N$ is defined as follows:

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

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 $\text{Log}(\mathcal{O}_L^\times)$ 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 $\text{Log}(u_1), \ldots, \text{Log}(u_b)$ 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" $\text{Log}(u_1), \ldots, \text{Log}(u_b)$ 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 [7]. 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 Log $u >$ Log v by using interval arithmetic in sufficiently high precision, and one can verify that Log $u =$ 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 ApproxLog(u) are rationals, allowing exact linear algebra.

To define ApproxLog, we first choose N linearly independent vectors

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

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 ApproxLog(u) as $\sum_j e_j \text{ApproxLog}(\varepsilon_j)$. 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 ApproxLog(u) as ApproxLog(u^N)/N. It is easy to check that ApproxLog is a well-defined group homomorphism.

For example, one can take ApproxLog(ε_1) $= (1, 0, \ldots, 0, 0)$, ApproxLog (ε_2) $= (0, 1, \ldots, 0, 0)$, and so on through ApproxLog(ε_{N-1}) $= (0, 0, \ldots, 1, 0)$. Then 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 ± 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 β, and we choose each entry of ApproxLog(ε_j) to be a multiple of $2^{-\beta}$ within $2^{-\beta}$ of the corresponding entry of Log(ε_j). Specifically, we build ApproxLog(ε_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 Sect. 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 $(\mathcal{O}_L^\times)^2 \leq U \leq \mathcal{O}_L^\times$. 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 Sect. 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 $u_1, \ldots, u_b, v_1, v_2, \ldots$.

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

---

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 $\mathcal{O}_L^\times / \{\pm 1\}$.

1 $\chi_1, \ldots, \chi_m \leftarrow$ EnoughCharacters($L, (u_1, \ldots, u_b)$)
2 $M \leftarrow [\log_{-1}(\chi_k(u_j))]_{1 \leq j \leq b, 1 \leq k \leq m}$
3 $S \leftarrow$ Basis(LeftKernel(M))
4 for $i = 1, \ldots, \#S$ **do**
5 $\quad$ $s_i \leftarrow \prod_j u_j^{S_{ij}}$, interpreting exponents in $\mathbb{Z}/2$ as $\{0,1\}$ in $\mathbb{Z}$
6 $\quad$ $v_i \leftarrow \sqrt{s_i}$
7 return $u_1, \ldots, u_b, v_1, \ldots, v_{\#S}$

---

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 Sect. 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 $N^3 + N^2 B$ for characters, N^3 for kernel computation, $N^3 B$ 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 output size by integrating the algorithm with lattice-basis reduction (see Sect. 5.5), computing products and square roots only after reduction. Third, we actually use the technique of Sect. 3.5 (in the full version of this paper) 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.

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 Sect. 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.

Algorithm 5.2. Units(L)

Input: A real multiquadratic field L. As a side input, a parameter $H > 0$.
Result: Independent generators of $\mathcal{O}_L^\times/\{\pm1\}$.

1 **if** $[L : \mathbb{Q}] = 1$ **then**
2 $\quad$ **return** ()

3 **if** $[L : \mathbb{Q}] = 2$ **then**
4 $\quad$ **return** the normalized fundamental unit of L

5 $\sigma, \tau \leftarrow$ distinct non-identity automorphisms of L
6 **for** $\ell \in \{\sigma, \tau, \sigma\tau\}$ **do**
7 $\quad$ $G_\ell \leftarrow$ Units(fixed field of ℓ)

8 $G \leftarrow -1, G_\sigma, G_\tau, \sigma(G_{\sigma\tau})$
9 $(u_1, \ldots, u_b) \leftarrow$ UnitsGivenSubgroup(L, G)

10 $A \leftarrow \begin{pmatrix} 1\ 0\ \ldots\ 0\ H \cdot \mathrm{ApproxLog}(u_1) \\ 0\ 1\ \ldots\ 0\ H \cdot \mathrm{ApproxLog}(u_2) \\ \vdots\ \vdots\ \ddots\ \vdots\ \qquad\quad\ \vdots \\ 0\ 0\ \ldots\ 1\ H \cdot \mathrm{ApproxLog}(u_b) \end{pmatrix}$

11 $A' \leftarrow$ LLL(A), putting shortest vectors first
12 **for** $i = 1, \ldots, N - 1$ where $N = [L : \mathbb{Q}]$ **do**
13 $\quad$ $w_i \leftarrow \prod_{1 \leq j \leq b} u_j^{A'_{b-(N-1)+i,j}}$
**14** **return** $w_1, \ldots, w_{N-1}$

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 σ, τ are distinct non-identity automorphisms of L.

The three subfields used in the algorithm are K_σ, K_τ, and $K_{\sigma\tau}$. The recursive calls produce lists of generators for $\mathcal{O}_{K_\sigma}^\times/\{\pm1\}$, $\mathcal{O}_{K_\tau}^\times/\{\pm1\}$, and $\mathcal{O}_{K_{\sigma\tau}}^\times/\{\pm1\}$ respectively. The algorithm builds a list G that contains each element of the first list; each element of the second list; σ 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 σ and τ 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 σ and τ 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 Figs. 5.1 and 5.2. Notice that, in Figs. 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.
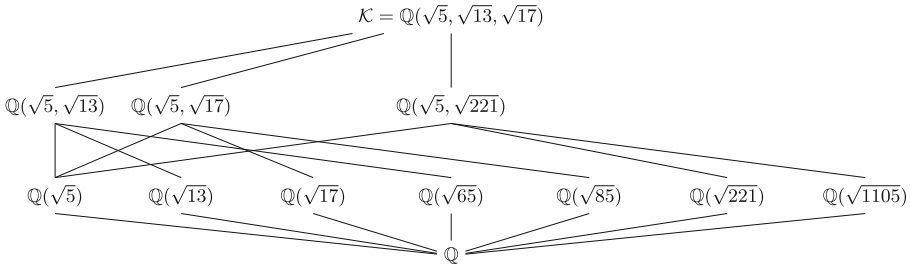
Fig. 5.1. How to pick subfields for the recursive algorithm for multiquadratic fields of degree 8.
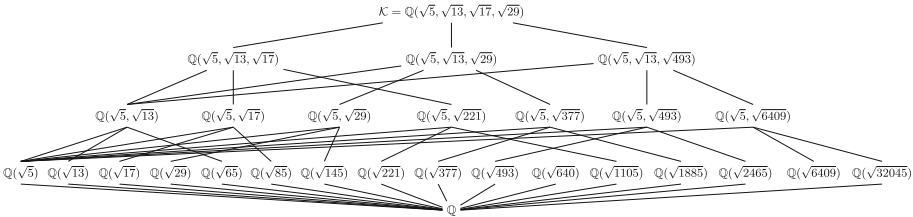
Fig. 5.2. How to pick subfields for the recursive algorithm for multiquadratic fields of degree 16.

With this strategy, the recursion reaches exactly $2^{n-\ell+1} - 1$ subfields of degree 2^{ℓ}, 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 end up with nearly $3^{n-\ell}$ subfields of degree 2^{ℓ}. "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 [37] to the vectors $\mathrm{ApproxLog}(u_1), \ldots, \mathrm{ApproxLog}(u_b)$ 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 [37] 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 $u_3 = u_1 u_2$, 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 $H \cdot \mathrm{ApproxLog}(u_1), \ldots, H \cdot \mathrm{ApproxLog}(u_b)$. (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 [35, Proposition 1.26]. The total time is bounded by essentially N^5B^2. To bound B one can bound each $H \cdot \mathrm{ApproxLog}(\cdots)$. To bound H one can observe that the transformation matrix has, at worst, essentially N bits in each coefficient (see, e.g., [39]), 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 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 Sect. 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 Sect. 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 Sect. 4, together with the unit group computed in Sect. 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" (α_1, α_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 σ 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 σ 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 σ 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., [20].

Given a $\mathbb{Z}$-basis of $\mathcal{I}$, one can compute a $\mathbb{Z}$-basis of $N_{L:K}\mathcal{I}$ by computing $\{\omega_i \cdot \sigma(\omega_j) \mid 1 \leq i \leq j \leq N\}$, 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 [6]: 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 $N^5 B$.

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 Sect. 4.2 to this equation, we obtain
$$X(uh) = X(g^2) = 2X(g) = 0.$$
Therefore $X(u) = X(h)$.

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 $h/g^2 \in \mathcal{O}_L^\times$.

1 $u_1, \ldots, u_{N-1} \leftarrow \text{Units}(L)$
2 $u_0 \leftarrow -1$
3 $\chi_1, \ldots, \chi_m \leftarrow \text{EnoughCharacters}(L, (u_0, \ldots, u_{N-1}, h))$
4 $M \leftarrow [\log_{-1} \chi_j(u_i)]_{0 \leq i \leq N-1, 1 \leq j \leq m}$
5 $V \leftarrow [\log_{-1} \chi_j(h)]_{1 \leq j \leq m}$
6 $[e_0, \ldots, e_{N-1}] \leftarrow \text{SOLVELEFT}(M, V)$
7 $u \leftarrow \prod_j u_j^{e_j}$, interpreting exponents in $\mathbb{Z}/2$ as $\{0, 1\}$ in $\mathbb{Z}$
8 $g \leftarrow \sqrt{uh}$
9 **return** g

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 summarized in Algorithm 6.1.

The subroutine $\text{SOLVELEFT}(M, V)$ 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 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 Sect. 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 Sect. 8 for analysis of the success probability of this algorithm at finding the short generators used in a cryptosystem.

Recall the log-unit lattice $\mathrm{Log}(\mathcal{O}_L^\times)$ defined in Sect. 5.3. The algorithm finds a lattice point $\mathrm{Log}\, u$ close to $\mathrm{Log}\, h$, 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 $\mathrm{Log}\, h$, and write $\mathrm{Log}\, h$ as a linear combination of the vectors $\mathrm{Log}(u_1), \ldots, \mathrm{Log}(u_{N-1}), (1, 1, \ldots, 1)$; recall that $(1, 1, \ldots, 1)$ is orthogonal to each $\mathrm{Log}(u_j)$. 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 $\mathrm{Log}\, h$ is close to $e_1 \mathrm{Log}(u_1) + \cdots + e_{N-1} \mathrm{Log}(u_{N-1}) + e_N(1, 1, \ldots, 1)$, and thus to $\mathrm{Log}\, u + e_N(1, 1, \ldots, 1)$. The gap $\mathrm{Log}\, g = \mathrm{Log}\, h - \mathrm{Log}\, u$ is between -0.5 and 0.5 in each of the $\mathrm{Log}(u_j)$ directions, plus some irrelevant amount in the $(1, 1, \ldots, 1)$ direction.

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 $\mathrm{Log}(u_j)$ directions then this algorithm will find this generator (modulo ± 1). See Sect. 8 for further analysis of this event.

Approximations. The algorithm actually computes $\mathrm{Log}\, h$ only approximately, and uses $\mathrm{ApproxLog}\, u_j$ instead of $\mathrm{Log}\, u_j$, 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 $|\sigma_j(h)| \le 2^B \prod_i (1 + \sqrt{|d_i|})$, so $\ln |\sigma_j(h)| \le B \ln 2 + \sum_i \ln(1 + \sqrt{|d_i|})$. By assumption each d_i is quasipolynomial in N, so $\ln |\sigma_j(h)| \le B \ln 2 + n^{O(1)}$.

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 $\ln |1/\sigma_j(h)| \le C \ln 2 + n^{O(1)}$.

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 $\mathrm{Log}\, h$ 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 [4]. See generally [25].

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 β.

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

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

2 $M \leftarrow \begin{pmatrix} \text{ApproxLog}(u_1) \\ \vdots \\ \text{ApproxLog}(u_{N-1}) \\ 1 \quad 1 \quad \ldots \quad 1 \quad 1 \end{pmatrix}$

3 $v \leftarrow$ approximation to $\text{Log}(h)$ within $2^{-\beta}$ in each coordinate

4 $e \leftarrow \left\lfloor -vM^{-1} \right\rceil$

5 $g \leftarrow h u_1^{e_1} \cdots u_{N-1}^{e_{N-1}}$

6 return g

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 $\mathcal{I}\mathcal{O}_Q$

1 $h \leftarrow \text{FindQGen}(Q, \mathcal{I})$

2 $g \leftarrow \text{ShortenGen}(Q, h)$

3 return g

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.

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 Sect. 5.1 that there are standard algorithms to compute the normalized fundamental unit ε 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 [8, 14, 41]. As in Sect. 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 Sect. 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.

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 $\mathcal{I}\mathcal{O}_L$

1 **if** $[L : \mathbb{Q}] = 1$ **then**

2 $\quad$ **return** the smallest positive integer in $\mathcal{I}$

3 **if** $[L : \mathbb{Q}] = 2$ **then**

4 $\quad$ **return** QPIP$(L, \mathcal{I})$

5 $\sigma, \tau \leftarrow$ distinct non-identity automorphisms of L

6 **for** $\ell \in \{\sigma, \tau, \sigma\tau\}$ **do**

7 $\quad$ $K_\ell \leftarrow$ fixed field of ℓ

8 $\quad$ $\mathcal{I}_\ell \leftarrow N_{L:K_\ell}(\mathcal{I})$

9 $\quad$ $g_\ell \leftarrow$ MQPIP$(K_\ell, \mathcal{I}_\ell)$

10 $h \leftarrow g_\sigma g_\tau / \sigma(g_{\sigma\tau})$

11 $g' \leftarrow$ IdealSqrt(L, h)

12 $g \leftarrow$ ShortenGen(L, g')

13 **return** g

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.

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 Sect. 5.5, choose distinct non-identity automorphisms σ, τ 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 Sect. 6.1, and call MQPIP$(K_\ell, \mathcal{I}_\ell)$ recursively to compute a generator g_ℓ 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.

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 $\mathcal{I}$.

Precomp?	Subroutine	Cost
Yes	Units for all quadratic fields	NB
Yes	Characters of units (in UGS, IS)	$N^3 + N^2 B$
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)	$N^3 B$
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)	$N^2 B$
No	Square roots (in IS)	$N^{1+\log_2 3} B$

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 [23] computer-algebra system on a single core of a 4 GHz AMD FX-8350 CPU.

7.1   Basic Subroutine Timings

Table 7.1 shows the time taken for multiplication, squaring, etc., rounded to the nearest 0.0001 s: e.g., 0.0627 s 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 0.2 s, 2.8 s, and 93 s in degrees 32, 64,

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., and $\lambda = 64$. 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 σ 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 21 measurements.

n	2^n	mult	square	relnorm	absnorm	div	sqrt
3	8	0.0084	0.0062	0.0080	0.0515	0.0140	0.3547
4	16	0.0100	0.0075	0.0088	0.1119	0.0153	1.0819
5	32	0.0132	0.0101	0.0106	0.2364	0.0176	3.3507
6	64	0.0209	0.0163	0.0145	0.5013	0.0231	10.2689
7	128	0.0347	0.0275	0.0221	1.0199	0.0341	31.2408
8	256	0.0627	0.0501	0.0367	2.1024	0.0573	93.9827

and 128 respectively using the tower-field representation, and it takes $0.15\,$s, $1.16\,$s, and $11.3\,$s in degrees 32, 64, and 128 respectively using the single-variable representation, while we use under $0.06\,$s 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 $2.34\,$s for degree 16, $7.80\,$s for degree

Table 7.2. Observed time to compute (once) the unit group of $\mathbb{Z}[\sqrt{d_1}, \ldots, \sqrt{d_n}]$, with $d_1 = 2$, $d_2 = 3$, $d_3 = 5$, etc.; and to break the cryptosystem presented in Appendix A. The "tower" column is the time used by Sage's tower-field unit-group functions (with `proof=False`); for $n = 6$ these functions ran out of memory after approximately $710000\,$s. 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 "attack" column is the time to find a generator of the public key, after the unit group is precomputed. In "new2" and "attack2" the same timings are given for the field with the first n consecutive primes after n. In "new3" and "attack3" the same timings are given for the field with the first n consecutive primes after n^2.

n	2^n	tower	absolute	new	new2	new3	attack	attack2	attack3
3	8	0.05	0.03	0.63	0.65	0.66	0.10	0.11	0.11
4	16	0.51	0.24	2.34	2.21	2.18	0.27	0.35	0.36
5	32	7.24	4.80	7.80	7.71	8.22	0.96	1.36	1.47
6	64	>700000	>700000	26.62	28.08	81.78	4.36	6.68	7.48
7	128			146.60	192.19	2332.79	26.14	37.23	42.30
8	256			942.36	2364.18	65932	181.26	239.05	239.90

32, 26.62 s for degree 64, 146.60 s for degree 128, etc., slowing down by a factor considerably below 2^5 for each doubling in the degree. Sage's internal C library uses 4.8 s 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;[3] 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 Sect. 8 for analysis of the success probability of the algorithm as an attack against the cryptosystem.

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 (in the full version of this paper), the same system used for timings in Sect. 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 $\mathrm{Log}\, g$ on the basis $\mathrm{Log}\, \varepsilon_1, \ldots, \mathrm{Log}\, \varepsilon_{N-1}$ 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 $\mathrm{Log}\, g$ 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 $\mathrm{Log}\, \varepsilon_1, \ldots, \mathrm{Log}\, \varepsilon_{N-1}$ are large enough.

8.1   MQ Unit Lattice: Heuristics for 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 $\mathrm{Log}\, \varepsilon_1, \ldots, \mathrm{Log}\, \varepsilon_{N-1}$ form a basis for the **MQ unit lattice** $\mathrm{Log}\, U_L$. 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 $x = \log(1 + \sqrt{2})$,

---

[3] 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.

$y = \log(2 + \sqrt{3})$, and $z = \log(5 + 2\sqrt{6})$. The general pattern (as in Sect. 5.3) is that $\mathrm{Log}\,\varepsilon_j$ 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 $\mathrm{Log}\,g$, and u is any multiquadratic unit, then rounding $\mathrm{Log}\,gu$ produces $\mathrm{Log}\,u$. 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 $\mathrm{Log}\,g$ on the jth basis vector $\mathrm{Log}\,\varepsilon_j$; note that if each c_j is strictly between -0.5 and 0.5 then 0 is the closest lattice point to $\mathrm{Log}\,g$. Another consequence of orthogonality is that c_j is simply the dot product of $\mathrm{Log}\,g$ with $\mathrm{Log}\,\varepsilon_j$ divided by the squared length of $\mathrm{Log}\,\varepsilon_j$; i.e., the dot product of $\mathrm{Log}\,g$ 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 $\mathrm{Log}\,g$ 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 is a universal constant, the average of $\ln|\mathcal{N}|$.)

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, 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 jth 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 Log g

The experiments in Fig. 8.1 confirm the prediction of Sect. 8.1. For each n, we took possibilities for n consecutive primes $d_1, \ldots, d_n$ below 100. For each corresponding multiquadratic field, there are $N - 1$ blue dots. For each D in $\{d_1, d_2, d_1d_2, \ldots, d_1d_2 \cdots d_n\}$, one of these $N-1$ dots is at horizontal position D.

(a) $n = 4$

(b) $n = 5$

(c) $n = 6$
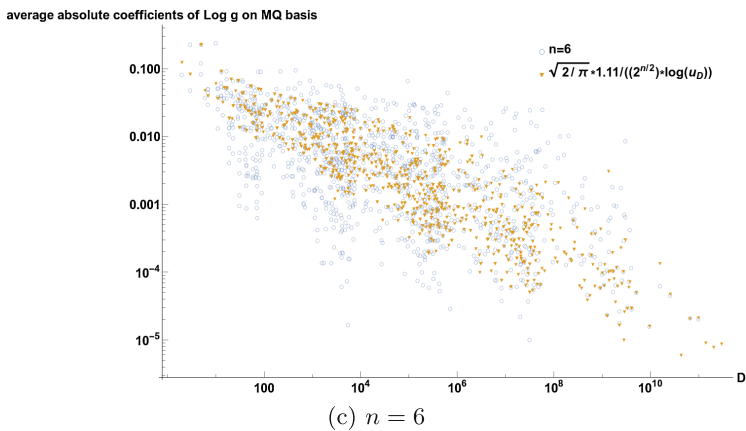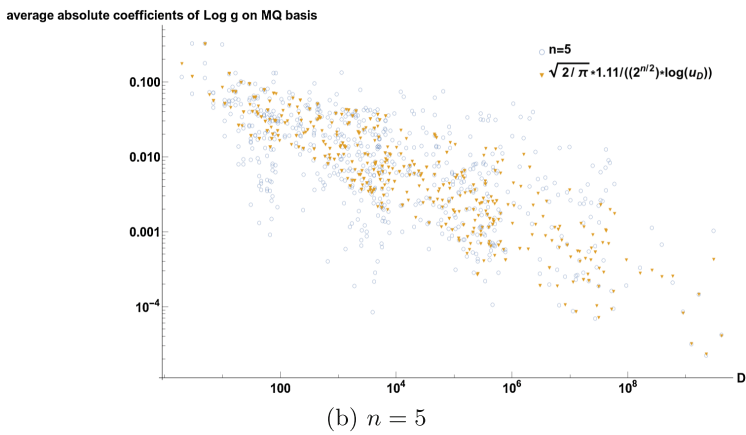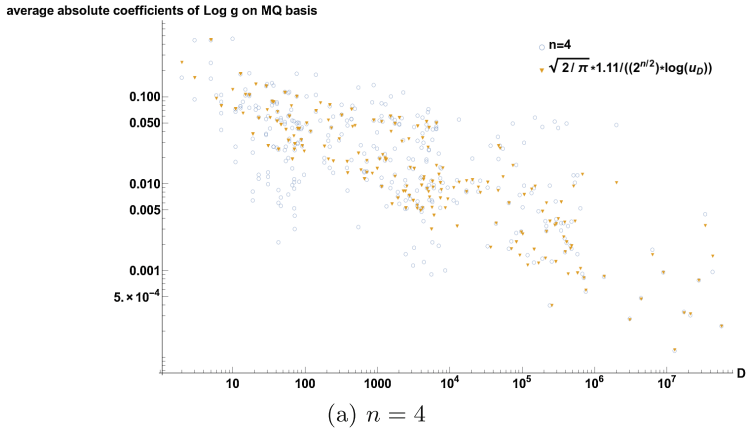
Fig. 8.1. Blue dots: For $n = 4, 5, 6$, the observed average absolute coefficient of $\mathrm{Log}(g)$ in the direction of the basis vector corresponding to $\mathbb{Q}(\sqrt{D})$. Yellow dots: Predicted values. (Color figure online)

The vertical position is the observed average absolute coefficient of 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 yellow dot 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 yellow dots (predictions) and the blue dots (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 ε_D is again the normalized fundamental unit of $\mathbb{Q}(\sqrt{D})$. This rescaling of the blue dots is shown in Fig. 8.2. In purple we compare these to the $x = y$ line.
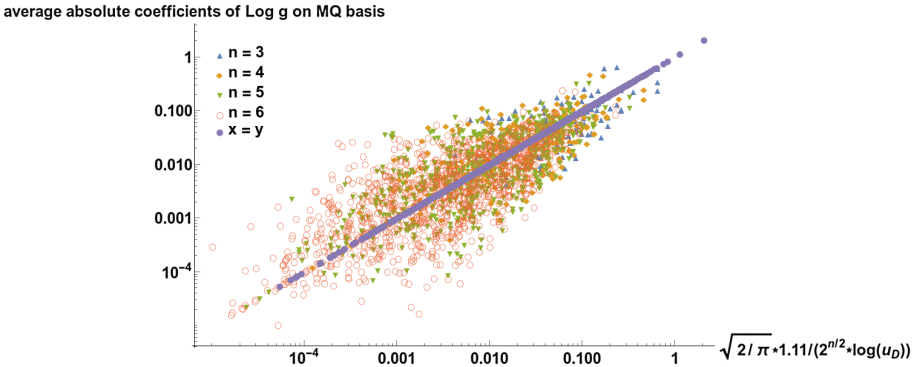

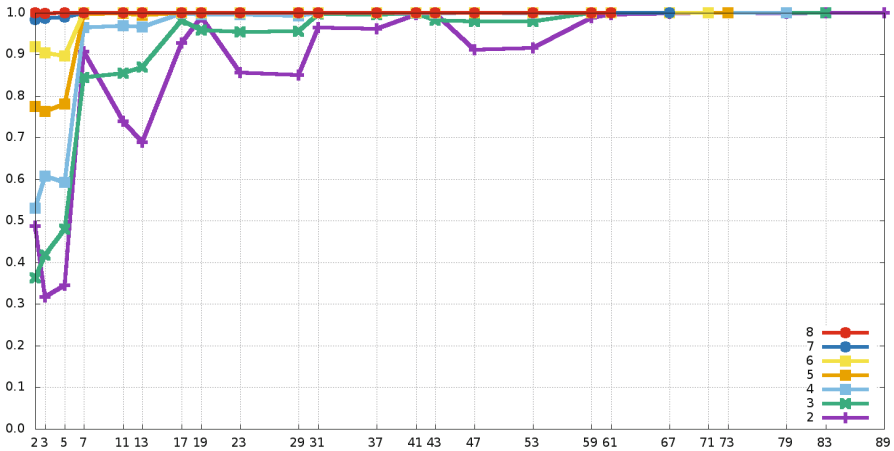**Fig. 8.2.** Rescaling of the experiments of Fig. 8.1, also including $n = 3$.

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 1000 randomly drawn secret keys g in the cryptosystem, that Log g is successfully rounded to 0 in the MQ unit lattice.

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 $n \geq 7$.

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 $N^{0.3N}$.

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 1130 (except for $n = 8$: first 832 that have finished) 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
Average $\log_2 \|u^*\|$ for U_L	1.095	1.762	2.345	2.899	3.487	4.040
Average $\log_2 \|u^*\|$ for $\mathcal{O}_L^{\times}$	0.964	1.642	2.223	2.797	3.386	3.926
Average $\log_2(\#(\mathcal{O}_L^{\times}/U_L))$	5.711	17.462	44.095	108.133	253.722	580.099

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 at least 1000 trials (except 100 for $n = 8, j = 1$) that were successfully recovered without any enumeration by our attack on field L_j. Table covers $j \in \{1, n, n^2\}$.

n	3	4	5	6	7	8
$p_{\mathrm{suc}}(L_1)$	0.112	0.130	0.145	0.084	0.003	0.00
$p_{\mathrm{suc}}(L_n)$	0.204	0.497	0.649	0.897	0.783	0.348
$p_{\mathrm{suc}}(L_{n^2})$	0.782	0.980	0.999	1.000	1.000	1.000

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 ± 1.

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 Sect. 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.

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 $\log \varepsilon$ 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 Log g on the MQ unit basis are then predicted to have deviation at most $1.11072/N^{1.01}$; see Sect. 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 $N\text{Log } gu$ and round in the MQ unit lattice. The coefficients of $N\text{Log } g$ 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 $\text{Log}(u^N)$ with high probability, revealing $\text{Log}(g^N)$ 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.

A   A Multiquadratic Cryptosystem

See full version of paper on multiquad.cr.yp.to.

B   Recent Progress in Attacking Ideal-SVP

See full version of paper on multiquad.cr.yp.to.

References

1. Abel, C.S.: Ein Algorithmus zur Berechnung der Klassenzahl und des Regulators reell-quadratischer Ordnungen. Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany (1994)

2. Adleman, L.M.: Factoring numbers using singular integers. In: STOC 1991, pp. 64–71 (1991)

3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. USENIX Security 2016, pp. 327–343 (2016)

4. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)

5. Barbulescu, R., Gaudry, P., Joux, A., Thomé, E.: A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 1–16. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55220-5_1

6. Belabas, K.: Topics in computational algebraic number theory. J. de Théorie des Nombres de Bordeaux **16**(1), 19–63 (2004)

7. Biasse, J.-F., Fieker, C.: Improved techniques for computing the ideal class group and a system of fundamental units in number fields. In: ANTS-IX. Open Book Series, vol. 1, pp. 113–133. Mathematical Sciences Publishers (2012)

8. Biasse, J.-F., Jacobson Jr., M.J., Silvester, A.K.: Security estimates for quadratic field based cryptosystems. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 233–247. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14081-5_15

9. Biasse, J.-F., Song, F.: 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

10. Biasse, J.-F., Song, F.: Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In: SODA 2016, pp. 893–902 (2016)

11. Biehl, I., Buchmann, J.: Algorithms for quadratic orders. In: Mathematics of Computation 1943–1993: A Half-century of Computational Mathematics, pp. 425–451. AMS (1994)

12. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: IEEE S&P 2015, pp. 553–570 (2015)

13. Buchmann, J., Maurer, M., Möller, B.: Cryptography based on number fields with large regulator. J. de Théorie des Nombres de Bordeaux **12**(2), 293–307 (2000)

14. Buchmann, J., Vollmer, U.: Binary Quadratic Forms: An Algorithmic Approach. Algorithms and Computation in Mathematics. Springer, Heidelberg (2007)

15. Buchmann, J.A.: A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In: Séminaire de Théorie des Nombres, Paris 1988–1989, pp. 27–41 (1990)

16. Buhler, J.P., Lenstra Jr., H.W., Pomerance, C.: Factoring integers with the number field sieve. In: Lenstra, A.K., Lenstra, H.W. (eds.) Algorithms and Computation in Mathematics. Springer, Heidelberg (2007)

17. Campbell, P., Groves, M., Shepherd, D.: Soliloquy: a cautionary tale (2014). http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf

18. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 3–12. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46800-5_1

19. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer, Heidelberg (1993)

20. Cohen, H.: Advanced Topics in Computational Number Theory. Springer, New York (1999)

21. Coron, J.-S., Lepoint, T., Tibouchi, M.: New multilinear maps over the integers. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 267–286. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47989-6_13

22. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 559–585. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49896-5_20

23. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 7.5.1) (2017). http://www.sagemath.org

24. Eisenträger, K., Hallgren, S., Lauter, K.: Weak instances of PLWE. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 183–194. Springer, Cham (2014). doi:10.1007/978-3-319-13051-4_11

25. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press, Cambridge (2012)

26. Galbraith, S.D., Gaudry, P.: Recent progress on the elliptic curve discrete logarithm problem. Des. Codes Cryptogr. **78**(1), 51–72 (2016)

27. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38348-9_1

28. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009). https://crypto.stanford.edu/craig

29. C. Gentry.: Fully homomorphic encryption using ideal lattices. In: STOC 2009, pp. 169–178 (2009)

30. Gentry, C.: Toward basing fully homomorphic encryption on worst-case hardness. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 116–137. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14623-7_7

31. Gentry, C., Halevi, S.: Implementing Gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20465-4_9

32. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). doi:10.1007/BFb0054868

33. Kim, T., Barbulescu, R.: Extended tower number field sieve: a new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 543–571. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53018-4_20

34. Kubota, T.: Über den bizyklischen biquadratischen Zahlkörper. Nagoya Math. J. **10**, 65–85 (1956)

35. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**, 515–534 (1982)

36. Lenstra, H.W.: Solving the Pell equation. Notices Amer. Math. Soc. **49**, 182–192 (2002)

37. Pohst, M.: A modification of the LLL reduction algorithm. J. Symb. Comput. **4**, 123–127 (1987)
38. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010). doi:10.1007/ 978-3-642-13013-7_25
39. van der Kallen, W.: Complexity of an extended lattice reduction algorithm (1998). http://www.staff.science.uu.nl/~kalle101/complexity.pdf
40. Vollmer, U.: Asymptotically fast discrete logarithms in quadratic number fields. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 581–594. Springer, Heidelberg (2000). doi:10.1007/10722028_39
41. Vollmer, U.: Rigorously analyzed algorithms for the discrete logarithm problem in quadratic number fields. Ph.D. thesis, Technische Universität, Darmstadt (2004)
42. Wada, H.: On the class number and the unit group of certain algebraic number fields. J. Fac. Sci. Univ. Tokyo Sect. I **13**(13), 201–209 (1966)
43. Williams, H.C.: Solving the Pell equation. In: Number theory for the millennium III, pp. 397–435. A K Peters (2002)