COMPUTING DISCRETE LOGARITHMS IN QUADRATIC ORDERS

MICHAEL J. JACOBSON, JR.

Abstract. We present efficient algorithms for computing discrete logarithms in the class group of a quadratic order and for principality testing in a real quadratic order, based on the work of Düllmann and Abel. We show how the idea of generating relations with sieving can be applied to improve the performance of these algorithms. Computational results are presented which demonstrate that our new techniques yield a significant increase in the sizes of discriminants for which these discrete logarithm problems can be solved.

1. Introduction

It is well-known that finite Abelian groups offer an excellent setting for cryptographic protocols [15], in particular, groups G in which the *discrete logarithm problem* (DLP) is intractable. That is, given $g, a \in G$, it should be beyond the reach of an adversary to recover an integer x such that $g^x = a$, or determine that no such x exists. Several types of finite Abelian groups have been proposed for this purpose, including the original idea of the multiplicative group of a finite field modulo a prime [15], the group of points on an elliptic curve over a finite field [20, 27], the group of points on a hyperelliptic curve [21] over a finite field, and others.

The ideal class group of an imaginary quadratic order has received attention in the context of key exchange protocols [9, 25], cryptosystems [18], and even an identification scheme [26]. Although not yet practical, such systems have the advantage that breaking them appears to be at least as difficult as factoring, and quite possibly harder. Real quadratic orders have also been proposed for use in these contexts [29, 2]. Breaking theses systems also appears to be at least as difficult as factoring, and they have the distinction of being the first discrete logarithm-based systems to make use of a structure which is *not* a group, namely the infrastructure of the principal class.

In quadratic orders, the general discrete logarithm problem is the following:

Problem 1.1. Given ideals $\mathfrak{a}, \mathfrak{b}$ of $\mathcal{O}_\Delta$, compute $\alpha \in \mathbb{Q}(\sqrt{\Delta})$ and $x \in \mathbb{Z}_{\geq 0}$ such that

$$\mathfrak{a} = \alpha \mathfrak{b}^x,$$

or determine that no such x and α exist.

Most cryptosystems based on quadratic orders rely on the difficulty of this problem, or some special case of it [9, 29]. In imaginary quadratic orders, the most difficult part of solving Problem 1.1 is computing the exponent x. Since every ideal class has a single unique reduced representative, the problem can be reduced to computing the discrete logarithm of two ideal classes in Cl_Δ. The class group is a finite Abelian group, so generic methods such as those in [6] can be directly applied. If desired, α can be found by computing the quadratic number α such that $(1/\alpha)\mathfrak{b}^x\mathfrak{a}^{-1}$ is reduced. However, the size of the class group is approximately $\sqrt{|\Delta|}$, so for large discriminants generic algorithms are too slow.

In real quadratic orders, the situation is reversed. In general, computing α is the most difficult part of solving Problem 1.1, since most real quadratic orders have small class numbers. If we are given the exponent x, then α is the generator of the principal ideal $\mathfrak{b}^{-x}\mathfrak{a}$. Thus, one needs to be able to do principal ideal testing

1991 *Mathematics Subject Classification.* Primary 11R11, 11R29, 11Y40; Secondary 11Y16.

Key words and phrases. Discrete logarithm, principal ideal testing, quadratic order, class group, computational number theory.

in order to solve Problem 1.1 in real quadratic orders. Baby-step giant-step methods are applicable here, but since the regulator is roughly as large as $\sqrt{\Delta}$ when the class number is small, they quickly become impractical for large discriminants.

Subexponential algorithms have been proposed for computing discrete logarithms in the class group [4] and for principality testing [1, 12]. Both algorithms require that the class group and regulator be computed, together with data accumulated during the course of the computation, after which any instance of the discrete logarithm problem can be solved relatively quickly. Buchmann and Düllmann have implemented their algorithm [4], and they were able to solve instances of the discrete logarithm problem for imaginary quadratic orders with discriminants of as many as 40 decimal digits. For real quadratic orders, class groups and regulators have been computed for orders with discriminants of as many as 41 decimal digits using an implementation of the algorithm described in [12], so principality testing is certainly possible for these orders as well.

The improved algorithm for computing class groups and regulators from [19] can be applied almost directly to these existing algorithms in order to speed up the precomputation step of computing the class group and regulator. The major development in this algorithm is the application of sieving to the process of generating random relations. An important component of both the discrete logarithm algorithm in the class group and the principality testing algorithm is computing a representation of a given ideal over the factor base, and the idea of relation generation with sieving can also be applied to speed up this step. Further improvements can be obtained in the linear algebra necessary to compute discrete logarithms in the class group and in computing an approximation of $\delta(\alpha)$ for principality testing. In this paper we will review the algorithms of [4] and [1, 12], and describe our improved versions of both, together with computational results demonstrating their efficiency.

2. Quadratic Orders

We give here a brief review of the concepts related to quadratic orders which we will need in this exposition. For further details and proofs, the interested reader is invited to refer to any standard text on the subject, for example [17, 10].

Given a non-square integer $\Delta \equiv 0, 1 \pmod 4$, the *quadratic order* of discriminant Δ is defined as the $\mathbb{Z}$-module

$$\mathcal{O}_\Delta = \left[\mathbb{Z} + \frac{\Delta + \sqrt{\Delta}}{2}\mathbb{Z}\right] \ .$$

$\mathcal{O}_\Delta$ is a Dedekind domain, and its field of quotients is the quadratic field $\mathbb{Q}(\sqrt{\Delta})$. The units in $\mathcal{O}_\Delta$ can be completely characterized as follows. If $\Delta < 0$, there are finitely many units given by

$$\begin{cases} \pm 1 & \text{if } \Delta < -4, \\ \pm 1, \pm i & \text{if } \Delta = -4, \\ \pm 1, \pm i, \frac{1 \pm \sqrt{-3}}{2} & \text{if } \Delta = -3 \ . \end{cases}$$

If $\Delta > 0$, the unit group is given by

$$\mathcal{O}_\Delta^* = \langle \varepsilon_\Delta \rangle \times \langle -1 \rangle \ .$$

In other words, the unit group of a real quadratic order is the direct product of the infinite cyclic subgroup generated by $\varepsilon_\Delta > 1$, the fundamental unit, and the torsion subgroup $\langle -1 \rangle$ or order 2. We define the *regulator* as $R_\Delta = \log \varepsilon_\Delta$.

A fractional ideal of $\mathcal{O}_\Delta$ can be represented by the $\mathbb{Z}$-module

$$\mathfrak{a} = q\left[a\mathbb{Z} + \frac{b + \sqrt{\Delta}}{2}\mathbb{Z}\right],$$

where $a, b \in \mathbb{Z}$, $b^2 - \Delta \equiv 0 \pmod{4a}$, and $q \in \mathbb{Q}$ [22, 8]. In this representation, a and q are unique, and b is unique modulo $2a$. The *norm* of $\mathfrak{a}$ is given by $\mathcal{N}(\mathfrak{a}) = q^2 a$. The product of two ideals is defined as $\mathfrak{a}\mathfrak{b} = \{\sum_{(a,b)\in U} ab : U \subset \mathfrak{a} \times \mathfrak{b} \text{ finite}\}$. For any ideal $\mathfrak{a}$ represented as above, if we have $\gcd(a, b, \frac{b^2-\Delta}{4a}) = 1$, then $\mathfrak{a}$ is invertible under this operation, and its inverse is given by

$$\mathfrak{a}^{-1} = \frac{1}{q}\left[\mathbb{Z} + \frac{-b+\sqrt{\Delta}}{2a}\mathbb{Z}\right] .$$

$\mathcal{O}_\Delta$ is a unit under ideal multiplication, and it is easy to see that $\mathcal{I}_\Delta$, the set of invertible ideals of $\mathcal{O}_\Delta$, forms a group under this operation. Efficient algorithms for ideal multiplication can be found in [5, 19].

It can be shown that every ideal of $\mathcal{O}_\Delta$ is generated by at most two elements of $\mathcal{O}_\Delta$, i.e., we can write the ideal $\mathfrak{a}$ as a $\mathcal{O}_\Delta$-module $[\alpha\mathcal{O}_\Delta + \beta\mathcal{O}_\Delta]$ for some $\alpha, \beta \in \mathcal{O}_\Delta$. If $\mathfrak{a} = \alpha\mathcal{O}_\Delta$, it is called a *principal* ideal. The set of invertible, principal ideals, $\mathcal{P}_\Delta$, forms a subgroup of $\mathcal{I}_\Delta$ under ideal multiplication. The *class group* of $\mathcal{O}_\Delta$, denoted by Cl_Δ is defined as the factor group $\mathcal{I}_\Delta/\mathcal{P}_\Delta$. The class group is a finite Abelian group, and we define the *class number* h_Δ as its order. Elements of Cl_Δ will be denoted by $[\mathfrak{a}]$. Clearly, if $\mathfrak{a}, \mathfrak{b} \in [\mathfrak{a}]$, then $\mathfrak{a} = \beta\mathfrak{b}$ for some $\beta \in \mathbb{Q}(\sqrt{\Delta})$. In this case we say that $\mathfrak{a}$ and $\mathfrak{b}$ are *equivalent*, written $\mathfrak{a} \sim \mathfrak{b}$.

We call the ideal $\mathfrak{a}$ *reduced* if

$$\begin{cases} -b < a < b, b > 0 \text{ when } a = \frac{b^2-\Delta}{4a} & \text{if } \Delta < 0 \\ \sqrt{\Delta} - 2a < b < \sqrt{\Delta} + 2a & \text{if } \Delta > 0 \ . \end{cases}$$

Any given ideal can be reduced quickly [10], and we define the operation in the class group as ideal multiplication followed by reduction, i.e., all arithmetic will be performed with reduced ideals. This has the advantage that the operands used are bounded by $\sqrt{|\Delta|}$ [10]. For $\Delta < 0$, there is exactly one reduced ideal per equivalence class. If $\Delta > 0$, this is unfortunately not the case, but there are finitely many reduced representatives. If $(\alpha)\mathcal{O}_\Delta = \mathfrak{a}$, we call α the *generator* of the principal ideal $\mathfrak{a}$, and we define the *distance* as $\delta(\mathfrak{a}) = \delta(\alpha) = \frac{1}{2}\log|\frac{\alpha}{\sigma(\alpha)}| \pmod{R_\Delta}$. For two reduced principal ideals $\mathfrak{a}$ and $\mathfrak{b}$, it can be shown that $\delta(\mathfrak{a}\mathfrak{b}) \approx \delta(\mathfrak{a}) + \delta(\mathfrak{b})$. Using this construction, Shanks showed that some structure can be imposed among the reduced principal ideals, called the *infrastructure* [31, 22, 19]. The infrastructure exhibits group-like properties, in fact, the only group axiom which fails is associativity.

The prime ideals of $\mathcal{O}_\Delta$ can be characterized as follows. If $\left(\frac{\Delta}{p}\right) \neq -1$ and b_p is the uniquely determined square root of $\Delta \pmod{4p}$ with $0 \leq b_p \leq p$, then

$$\mathfrak{p}(p) = \left[p\mathbb{Z} + \frac{b_p + \sqrt{\Delta}}{2}\mathbb{Z}\right]$$

are the prime ideals of norm p. Given any invertible integral ideal, we can represent it as a unique power-product of prime ideals.

Theorem 2.1. *If for some invertible ideal* $\mathfrak{a} = [a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}]$ *we have*

$$\mathcal{N}(\mathfrak{a}) = \prod_{p \ prime} p^{t(p)},$$

then $\mathfrak{a}$ *is equal to*

$$\prod_{p \ prime} \mathfrak{p}(p)^{e(p)t(p)}$$

where $e(p) \in \{-1, 1\}$ *such that* $b \equiv e(p)b_p \pmod{2p}$.

Thus, in order to factor an invertible integral ideal, one simply factors its norm (an integer) and applies Theorem 2.1.

Let $\mathfrak{a} = [a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}]$ be an integral ideal of $\mathcal{O}_\Delta$. The *norm form* of $\mathfrak{a}$ is the binary quadratic form $f = aX^2 + bXY + cY^2$, where $c = (b^2 - \Delta)/(4a)$. Every element $\alpha = ax + \frac{b+\sqrt{\Delta}}{2}y \in \mathfrak{a}$ for fixed $x, y \in \mathbb{Z}$ has norm $\alpha\sigma(\alpha) = af(x, y)$, where $\sigma(\alpha) = ax + \frac{b-\sqrt{\Delta}}{2}y$.

Proposition 2.2. *If f is the norm form of $\mathfrak{a} \in \mathcal{O}_\Delta$ and $f(x, y) = n$ for some $x, y \in \mathbb{Z}$, then there exists an ideal $\mathfrak{b} = [n\mathbb{Z} + \frac{b'+\sqrt{\Delta}}{2}\mathbb{Z}]$ such that $\mathfrak{a} \sim \mathfrak{b}$.*

Proof. Construct a form g by solving the linear Diophantine equation

$$ux + vy = 1$$

for u and v and then applying the transformation matrix

$$\begin{bmatrix} x & -v \\ y & u \end{bmatrix} \in GL(2, \mathbb{Z})$$

to f, yielding the form $g = f(x, y)X^2 + (2(asu + ctv) + b(sv + tu))XY + f(u, v)Y^2 = nX^2 + b'XY + c'Y^2$. Since g is obtained from f via a unimodular transformation of variables, we have $f \sim g$. Furthermore, g is the norm form of an ideal

$$\mathfrak{b} = [n\mathbb{Z} + \frac{b' + \sqrt{\Delta}}{2}\mathbb{Z}],$$

and it can be shown that $f \sim g$ implies $\mathfrak{a} \sim \mathfrak{b}$ [10]. □

3. Discrete Logarithms in Cl

Let $\mathcal{O}_\Delta$ be any quadratic order. In this section we will restrict our attention to solving the following problem:

Problem 3.1. Given reduced ideals $\mathfrak{a}, \mathfrak{b} \in Cl_\Delta$, compute $x \in \mathbb{Z}_{\geq 0}$ such that

$$[\mathfrak{a}] = [\mathfrak{b}]^x,$$

or determine that no such x exists.

In order to solve the more general Problem 1.1, it is first necessary to compute x by solving Problem 3.1. For cryptographic purposes, the intractability of this problem is especially important in the context of protocols based on imaginary quadratic orders [9, 25, 18, 26]. These protocols are usually not directly adapted to real quadratic orders since the class number is almost always small [13]. However, since instances of real quadratic orders with large class groups and small regulators can be constructed easily, for example $\Delta = 10^x + 1$ with x even, we will not omit this case in the following discussion.

Our algorithm stems from earlier work by Buchmann and Düllmann [4]. The main idea behind their algorithm is as follows. First, compute the structure of Cl_Δ as a direct product of cyclic subgroups,

$$Cl_\Delta \cong \bigotimes_{i=1}^{l} C(m_i),$$

together with generators $\mathfrak{g}_i$ of each cyclic subgroup (order of $\mathfrak{g}_i$ in Cl_Δ is m_i). Then compute the representations

$$\mathfrak{a} \sim \prod_{i=1}^{l} \mathfrak{g}_i^{a_i}$$

and

$$\mathfrak{b} \sim \prod_{i=1}^{l} \mathfrak{g}_i^{b_i}$$

of $\mathfrak{a}$ and $\mathfrak{b}$ over the generators. If Problem 3.1 is solvable, then there exists $x \in \mathbb{Z}_{\geq 0}$ such that

$$\prod_{i=1}^{l} \mathfrak{g}_i^{a_i} \sim \prod_{i=1}^{l} \mathfrak{g}_i^{x b_i},$$

and x can be found by solving the system of simultaneous congruences

$$a_i \equiv x b_i \pmod{m_i}, \quad 1 \leq i \leq l, \tag{3.1}$$

using the generalized Chinese remainder theorem. If (3.1) cannot be solved, then there is no solution to Problem 1.1.

3.1. **Computing the Class Group.** The first problem which must be solved in order to implement this method is to compute the structure of Cl_Δ. We used the method described in [19] (Algorithm 4.3). The underlying strategy of this algorithm is the same as that of Hafner and McCurley [16] and its variants [14], [3], [1], [11]. Suppose we have computed a factor base $FB = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$ consisting of invertible prime ideals such that the equivalence classes of some subset of FB generates Cl_Δ. For $\vec{v} \in \mathbb{Z}^k$ we define

$$FB^{\vec{v}} = \prod_{i=1}^{k} \mathfrak{p}_i^{v_i}$$

where $\mathfrak{p}_i \in FB$. We call $\vec{v}$ a *relation* if $FB^{\vec{v}} \sim \mathcal{O}_\Delta$, i.e., the ideal given by $FB^{\vec{v}}$ is principal. A generating system $L = \{\vec{v}_1, \ldots, \vec{v}_n\}$ of the *relation lattice*

$$\Lambda = \{\vec{v} \in \mathbb{Z}^k \mid FB^{\vec{v}} \sim \mathcal{O}_\Delta\} \tag{3.2}$$

is then produced, which is the kernel of the homomorphism

$$\mathbb{Z}^k \to Cl_\Delta, \quad \vec{v} \to FB^{\vec{v}} \ . \tag{3.3}$$

Since the equivalence classes of the ideals of FB generate the class group, it follows that the homomorphism (3.3) is surjective, and we have

$$Cl_\Delta \cong \mathbb{Z}^k/\Lambda \ .$$

This implies that Λ is a k-dimensional lattice and its determinant is equal to h_Δ. Also, the *relation matrix* $A = (\vec{v}_1^T, \ldots, \vec{v}_n^T)$, the matrix formed by taking the relations $\vec{v}_i$ as columns, has rank k. The diagonal elements which are greater than 1 in S, the Smith normal form of A, are precisely the elementary divisors of Cl_Δ.

This strategy can easily be extended to compute class groups and regulators of real quadratic orders [3, 1]. In this case, we compute relations of the form $(\vec{v}, \log|\gamma|)$ where $FB^{\vec{v}} = (\gamma)$, i.e., γ generates the principal ideal $FB^{\vec{v}}$. We produce a generating system

$$L' = \{(\vec{v}_1, \log|\gamma_1|), (\vec{v}_2, \log|\gamma_2|), \ldots, (\vec{v}_l, \log|\gamma_l|)\}$$

of the extended relation lattice

$$\Lambda' = \{(\vec{v}, \log|\gamma|) \in \mathbb{Z}^k \times \mathbb{R} \mid FB^{\vec{v}} = (\gamma)\} \ . \tag{3.4}$$

Then, if Λ is the part of Λ' in $\mathbb{Z}^k$, as before we have $Cl_\Delta \cong \mathbb{Z}^k/\Lambda$. Furthermore, it can be shown [3] that $\det(\Lambda') = h_\Delta R_\Delta$, so by computing this determinant and the structure of Cl_Δ we also get the regulator.

The major difference between our approach and that of [16], [14], etc., is in the way the generating system of the relation lattice is produced. The solution employed by earlier algorithms is to attempt to factor randomly produced ideals over the factor base. We replace this step by a sieve-based strategy similar to that used in the MPQS factoring algorithm [32]. The idea of employing sieving to compute relations in similar contexts was first suggested by Seysen [30], and later by Paulus [28].

In the MPQS, one sieves over quadratic polynomials $F(X) = aX^2 + 2bX + c$ in order to find values of x for which $F(x)$ completely factors over a finite factor base of prime integers. By sieving a polynomial $F(X)$ over an interval, we mean testing each value of x in a given interval as to whether all the prime factors of $F(x)$ are contained in a finite, given set. The observation that $F(x) \equiv F(x + ip) \pmod{p}$ for $i \in \mathbb{Z}$, p prime, allows one to use a sieve to perform this test rather than evaluating every value of $F(x)$ and attempting to factor it.

In our case, we first compute an ideal $\mathfrak{a}$ as a power-product of the prime ideals in our factor base FB, i.e., $\mathfrak{a} = FB^{\vec{e}}$ for some $\vec{e} \in \mathbb{Z}^k$. Then, we search for integers x and y such that $f(X,Y) = aX^2 + bXY + cY^2$, the norm form of $\mathfrak{a}$, factors over the norms of the ideals in FB. For each such pair (x,y), there exists a quadratic number γ such that $\mathfrak{a}/(\gamma) = \mathfrak{b}^{-1}$ splits over the factor base. We can explicitly compute $\mathfrak{b}$ and its decomposition over FB using Proposition 2.2. Since $\mathfrak{a}$ splits over FB by construction, we have that $\mathfrak{ab} = (\gamma)$ yields a relation.

The main work in generating relations with the strategy outlined above is finding smooth values of the quadratic polynomial $f(X,Y)$. It is certainly possible to sieve $f(X,Y)$ in two dimensions. However, most sieve-based factoring algorithms, including the MPQS, work exclusively with univariate quadratic polynomials. Hence, in order to parallel these factoring methods as closely as possible, we will also work with the univariate polynomials $F(X) = f(X,1) = aX^2 + bX + c$.

We have thus reduced the problem of finding relations for class group computation to the same problem for finding relations in the MPQS factoring algorithm. A large amount of effort has been invested in making the MPQS and its variants as efficient as possible, and we will attempt to make use of as many of these techniques as possible. most notable self-initialization. See [19] for more details and computational results.

3.2. **Computing a System of Generators.** Once the class group has been computed, the next step is to compute a system of generators of Cl_Δ. More generally, we need to be able to convert from representations of ideals as power-products of factor base elements,

$$\mathfrak{a} \sim FB^{\vec{v}},$$

to and from power-products of the generators,

$$\mathfrak{a} \sim \prod_{i=1}^{l} \mathfrak{g}_i^{a_i} \ .$$

Suppose we are given a Hermite normal form basis $H = (h_{i,j})_{k\times k}$ of the relation lattice Λ (3.2). The diagonal entries of S, the Smith normal form of H, which are greater than 1 correspond to the elementary divisors of Cl_Δ. During the course of the Smith normal form computation we also compute $U, V \in GL(k,\mathbb{Z})$ (recall k is the size of the factor base FB) such that

$$S = UHV \ .$$

If $U^{-1} = (u'_{ij})_{k\times k}$, then

$$\mathfrak{g}_i \sim \prod_{j=1}^{k} \mathfrak{p}_j^{u'_{ji}}, \quad 1 \le i \le k,$$

form a system of generators of Cl_Δ, if we ignore those $g_i \sim \mathcal{O}_\Delta$, i.e., $m_i = 1$. Conversely, if $U = (u_{ij})_{k\times k}$, then for each factor base element $\mathfrak{p}_j$ we have

$$\mathfrak{p}_j \sim \prod_{i=1}^{k} \mathfrak{g}_i^{u_{ij}} \ .$$

Thus, if $\mathfrak{a} \sim FB^{\vec{v}}$, it can be represented over the system of generators by

$$\mathfrak{a} \sim \prod_{i=1}^{k} \mathfrak{g}_i^{\left(\sum_{j=1}^{k} v_j u_{ij}\right)} \ . \tag{3.5}$$

For the purpose of computing discrete logarithms, all we need is the matrix U^{-1} — we never need to explicitly compute a system of generators. However, a partial verification of the class group is to check whether the orders of a system of generators are actually the elementary divisors. If the dimensions of H

are large, then it is very difficult to compute a system of generators, since it involves inverting a matrix of roughly the same dimension.

Algorithm 4.3 from [19] can easily be extended to compute a system of generators and all the information required to execute the transformations necessary for computing discrete logarithms, while avoiding the problem of computing the large transformation matrix U. Instead of computing the Smith normal form of the entire matrix H, we use only its essential part, i.e., the matrix formed by the rows and columns of H corresponding do diagonal entries greater than 1. Let H' denote the essential part of H. Clearly, the diagonal entries greater than 1 in SNF (H') are the same as those of SNF (H) , so we get the same elementary divisors of Cl_Δ. However, in practice the dimensions of H' are much smaller than those of H, and hence the Smith normal form transformation matrices will also be much smaller. Cohen and Lenstra [13] give heuristics which indicate that the class group is almost always cyclic or close to it, so we expect that the total number of elementary divisors will be small. The number of rows and columns in the essential part of H is equal to or greater than the rank of Cl_Δ. In practice, if they differ at all it is only a very slight difference (see for example [14, p.145]).

3.3. **Computing the Discrete Logarithm.** Once the structure of Cl_Δ is computed, we have to compute representations of $\mathfrak{a}$ and $\mathfrak{b}$ over a system of generators of Cl_Δ. If we know representations of $\mathfrak{a}$ and $\mathfrak{b}$ over the factor base FB used to compute Cl_Δ, then we can use (3.5). In practice, we will require that representations of $\mathfrak{a}$ and $\mathfrak{b}$ be computed over the prime ideals of FB corresponding to the essential part of the Hermite normal form relation matrix. However, in order to compute these representations, it is still necessary to first compute representations over the entire factor base.

Fortunately, computing these representations is no harder than finding a single relation corresponding to each of $\mathfrak{a}$ and $\mathfrak{b}$. As shown in Cohen [10], if we can find an ideal $\mathfrak{c}$ equivalent to $\mathfrak{a}FB^{\vec{v}}$ which factors over FB as $\mathfrak{c} = FB^{\vec{e}}$, then $\mathfrak{a}FB^{\vec{v}} \sim FB^{\vec{e}}$ and it follows that $\mathfrak{a} \sim FB^{\vec{e}-\vec{v}}$. In practice, we first compute

$$\mathfrak{d} = \mathfrak{a}FB^{\vec{e}} = \left[a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}\right]$$

for some random $\vec{e} \in \{-1, 0, 1\}^k$ such that

$$\mathcal{N}(\mathfrak{d}) \approx \frac{\sqrt{|\Delta|/2}}{M}, \tag{3.6}$$

where M is the sieve radius (all polynomials sieved over $-M \leq x \leq M$). If $f = aX^2 + bXY + cY^2$ is the norm form of $\mathfrak{d}$ and we find some $x \in [-M, M]$ such that $f(x, 1)$ factors over the norms of the prime ideals in FB, then we can compute γ and $\vec{w}$ such that $\mathfrak{d}/(\gamma) = FB^{\vec{w}}$ using Proposition 2.2. Thus, since $\mathfrak{d} = \mathfrak{a}FB^{\vec{e}}$ we can write

$$\mathfrak{a} = (\gamma)FB^{\vec{w}-\vec{e}} \ .$$

The reason we select the exponent vectors $\vec{e}$ in $\{-1, 0, 1\}$ satisfying (3.6) is so that our sieve polynomials $f(x, 1)$ resemble those used in the MPQS factoring algorithm as closely as possible. In [32], Silverman shows that if $\mathfrak{d}$ satisfies (3.6) for a given sieve radius M, then the values of $f(x, 1)$ for $x \in [-M, M]$ will be minimal compared to other quadratic polynomials of the same discriminant. Thus, we are more likely to find smooth values of $f(x, 1)$ in this case.

We present this method in the following algorithm.

Algorithm 3.1 (REPRESENT_OVER_FB).
Computes a representation of $\mathfrak{a}$ over the factor base.
Input: $\mathfrak{a}$, factor base $FB = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$
Output: $\vec{v} \in \mathbb{Z}^k$ and $\gamma \in \mathcal{O}_\Delta$ such that $\mathfrak{a} = (\gamma)FB^{\vec{v}}$

1. Select the sieve radius M.

2. Randomly select $\vec{e} \in \{-1, 0, 1\}$ and compute $\mathfrak{d}$ such that $\mathfrak{d} = \mathfrak{a}FB^{\vec{e}} = [a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}]$ and
$$\mathcal{N}(\mathfrak{d}) \approx \frac{\sqrt{|\Delta|/2}}{M} \quad .$$
3. Set $f = aX^2 + bXY + cY^2$ to be the norm form of $\mathfrak{d}$. Set $F(X) = f(X, 1)$ and sieve $F(X)$ over the interval $(-M, M)$.
4. If we find no $x \in (-M, M)$ such that $F(x)$ completely factors over the norms of the prime ideals in FB, go to Step 2.
5. For the smallest $x \in (-M, M)$ such that $F(x)$ completely factors over the norms of the prime ideals in FB :
   (a) Compute the exponents $\overline{w}_i$ such that
$$F(x) = \prod_{i=1}^{k} \mathcal{N}(\mathfrak{p}_i)^{\overline{w}_i} \quad .$$
   (b) Compute $g = nX^2 + b'XY + c'Y^2$ and $\mathfrak{c} = [n\mathbb{Z} + \frac{b'+\sqrt{\Delta}}{2}\mathbb{Z}] \sim \mathfrak{d}$ using Proposition 2.2 (g is the norm form of $\mathfrak{c}$).
   (c) Compute $\vec{w}$ such that $w_i = \pm\overline{w}_i$ and $\mathfrak{c} = FB^{\vec{w}}$ using Theorem 2.1.
   (d) Set $\vec{v} = \vec{w} - \vec{e}$.
   (e) $\gamma = ax + \frac{b+\sqrt{\Delta}}{2}$.

Given a representation of $\mathfrak{a}$ over the factor base FB, $\mathfrak{a} \sim FB^{\vec{v}}$, it is a simple matter to compute a representation of $\mathfrak{a}$ over a system of generators using (3.5). However, rather than computing SNF(H), we want to compute SNF(H') and the corresponding left transformation matrix $U = (u_{ij})_{k' \times k'}$, so we can also explicitly compute a system of generators. Hence, we need a method to compute a representation of $\mathfrak{a}$ over Q, where Q contains the prime ideals in FB corresponding to the rows of H taken in the essential part of H. If $\mathfrak{a} = Q^{\vec{w}}$ for $\vec{w} \in \mathbb{Z}^{k'}$, then a similar relation to (3.5) holds, namely
$$\mathfrak{a} \sim \prod_{i=1}^{l} \mathfrak{g}_i^{(\sum_{j=1}^{k'} w_j u_{ij}) \pmod{m_i}} \quad .$$

The following observation allows us to compute $\vec{w}$. The columns of H, the Hermite normal form of the relation matrix produced during the computation of the class group, form a basis of the relation lattice Λ. Furthermore, column i of H is a relation which involves only prime ideals $\mathfrak{p}_j$ for $j \leq i$, i.e.,
$$\mathcal{O}_\Delta \sim \mathfrak{p}_i^{h_{ii}} \prod_{j=1}^{i-1} \mathfrak{p}_j^{h_{ji}},$$
since $h_{ji} = 0$ for $j > i$. Thus, if $h_{ii} = 1$, we can represent $\mathfrak{p}_i$ in terms of $\mathfrak{p}_j$ for $j < i$:
$$\mathfrak{p}_i \sim \prod_{j=1}^{i-1} \mathfrak{p}_j^{-h_{ji}} \quad . \tag{3.7}$$

Assume we have computed $\vec{v} \in \mathbb{Z}^k$ such that $\mathfrak{a} \sim FB^{\vec{v}}$. Let i be the largest index such that $H_{ii} = 1$. If we substitute the representation of $\mathfrak{p}_i$ (3.7) into the representation $\mathfrak{a} \sim FB^{\vec{v}}$ we obtain
$$\mathfrak{a} \sim \prod_{\substack{j=1 \\ j \neq i}}^{k} \mathfrak{p}_j^{v_j} \prod_{j=1}^{i-1} \mathfrak{p}_j^{-v_i h_{ji}} \sim \prod_{\substack{j=1 \\ j \neq i}}^{k} \mathfrak{p}_j^{v_j - v_i h_{ji}} \sim FB^{\vec{v'}} \quad .$$

Note that we now have $\mathfrak{a} = FB^{\vec{v'}}$ with $v'_i = 0$, a representation of $\mathfrak{a}$ over FB which does not involve $\mathfrak{p}_i$. Also, the substitution of $\mathfrak{p}_i$ does not affect the entries in $\vec{v}$ with index greater than i, since (3.7) is a representation of $\mathfrak{p}_i$ consisting only of those $\mathfrak{p}_j$ with $j < i$. Hence, if we start with $i = k$ and repeat the substitution with successively decreasing indices i, we will eventually obtain a vector $\vec{w'} \in \mathbb{Z}^k$ such that $\mathfrak{a} = FB^{\vec{w'}}$ and $w'_j = 0$ for every j such that $H_{jj} = 1$. Taking $\vec{w}$ to be those entries of $\vec{w'}$ whose indices correspond to the diagonal entries of H greater than 1 (all other entries are zero), we have that $\mathfrak{a} = Q^{\vec{w}}$, as required.

The overall procedure for computing a representation of an ideal over a system of generators of Cl_Δ is summarized in the following algorithm.

Algorithm 3.2 (REPRESENT_OVER_GENS).
Computes a representation of $\mathfrak{a}$ over a system of generators of Cl_Δ.
Input: $\mathfrak{a}$, factor base FB, Hermite normal form basis of relation lattice H, left transformation matrix U for computing Smith normal form of essential part of H, elementary divisors of Cl_Δ m_i, $1 \le i \le l$
Output: $\vec{a} \in \mathbb{Z}^l$ such that $\mathfrak{a} \sim \prod_{i=1}^{l} \mathfrak{g}_i^{a_i}$

1. Compute $\vec{v} =$ REPRESENT_OVER_FB($\mathfrak{a}$, FB) (Algorithm 3.1).
2. $i = k = |FB|$.
3. If $H_{ii} = 1$, set $v_j = v_j - v_i h_{ji}$ for $1 \le j \le k$, $j \ne i$. Set $v_i = 0$. Decrement i and repeat while $i > 1$.
4. $i = 1$, $j = 1$.
5. If $H_{ii} \ne 1$, set $w_j = v_i$ and $j = j + 1$. Increment i and repeat while $i \le k$.
6. Set
$$a_i = \sum_{j=1}^{l} (w_j\, u_{ij}) \pmod{m_i}, \quad 1 \le i \le l\ .$$

Finally, we present our complete algorithm for computing x in Problem 3.1.

Algorithm 3.3 (DISCRETE_LOG).
Solves the discrete logarithm problem in the class group of $\mathcal{O}_\Delta$.
Input: Δ, $\mathfrak{a}$, $\mathfrak{b}$
Output: $x \in \mathbb{Z}_{\ge 0}$ such that $\mathfrak{a} \sim \mathfrak{b}^x$, or -1 if no such x exists

1. Compute the m_i such that $Cl_\Delta = C(m_1) \times \cdots \times C(m_l)$ using Algorithm 4.3 from [19, p.57]. Keep the factor base FB and the Hermite normal form basis of the relation lattice H.
2. $k = |FB|$, $H' = H$. For each $i \in \{1, \ldots, k\}$ such that $H_{ii} = 1$, remove row i and column i from H'.
3. Compute S and U such that $S = UH'V$.
4. Compute
$$\vec{a} = \text{REPRESENT\_OVER\_GENS}(\mathfrak{a},\ FB,\ H,\ U,\ \{m_i\})$$
and
$$\vec{b} = \text{REPRESENT\_OVER\_GENS}(\mathfrak{b},\ FB,\ H,\ U,\ \{m_i\})$$
using Algorithm 3.2.
5. Compute $x \in \mathbb{Z}_{\ge 0}$ such that
$$a_i \equiv x b_i \pmod{m_i}, \quad 1 \le i \le l\ .$$
If no such x exists, set $x = -1$.

4. Principality Testing

The second task necessary to solve Problem 1.1 once the exponent x has been computed is to determine the quadratic number α such that $\mathfrak{a} = \alpha\mathfrak{b}^x$. Computing α can be reduced to determining the generator of the principal ideal $\mathfrak{b}^{-x}\mathfrak{a}$, since we have $(\alpha) = \mathfrak{b}^{-x}\mathfrak{a}$. Thus, we now focus our attention on the principal ideal problem.

Problem 4.1. Given a reduced ideal $\mathfrak{a}$, determine whether $\mathfrak{a}$ is principal and if so, compute $\delta(\mathfrak{a})$.

Recall that $\delta(\mathfrak{a}) = \frac{1}{2}\log|\frac{\alpha}{\sigma(\alpha)}| \pmod{R_\Delta}$, where $(\alpha) = \mathfrak{a}$. Given this distance, it is not difficult to explicitly compute α if desired, or some quadratic numbers whose product is equal to α [8].

Cryptographic protocols based on real quadratic orders derive their security on the supposed intractability of Problem 4.1 [29, 2]. For these protocols, it is important that there be many reduced principal ideals, otherwise simple methods like exhaustive search or baby-step giant-step suffice to solve Problem 4.1. Hence, real quadratic orders with small regulators and imaginary quadratic orders are not suitable for these protocols.

Subexponential algorithms for principality testing have been proposed by Abel [1] and Cohen, Diaz y Diaz, and Olivier [12]. As in the algorithm for computing discrete logarithms in the class group, both algorithms require that the class group and regulator first be computed using an index calculus-type algorithm, after which any instance of Problem 4.1 in the same quadratic order is relatively easy.

The algorithm we will describe here follows that of [1]. The overall strategy is the same, but we introduce a number of practical improvements, primarily centered around using sieving methods. Given a reduced ideal $\mathfrak{a}$ and a quadratic order $\mathcal{O}_\Delta$, we first compute the class group and regulator of $\mathcal{O}_\Delta$ using Algorithm 4.3 from [19, p.57]. In addition to the usual output of the algorithm, we keep the factor base $FB = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$, matrix $H \in \mathbb{Z}^{k \times k}$ such that $\mathrm{HNF}\,(A) = [0 \mid H]$ where $A \in \mathbb{Z}^{k \times n}$ is the relation matrix (all relations produced appear as columns of A), the unimodular transformation matrices T_i, $1 \leq i \leq s$ such that $AT_1 \ldots T_s = \mathrm{HNF}\,(A)$, and the vector $\vec{\gamma} = \{\gamma_1, \ldots, \gamma_n\}$ containing the generators corresponding to the relations in A, i.e., if $\vec{a_i}$ is column i of A, then $FB^{\vec{a_i}} = (\gamma_i)$.

Notice that the columns of H form a basis of the relation lattice Λ. Thus, every principal ideal of $\mathcal{O}_\Delta$ can be represented by a vector $\vec{v} \in \mathbb{Z}^k$ where $\vec{v}$ is a linear combination of the columns of H. In order to determine whether an ideal $\mathfrak{a}$ is principal, we compute $\vec{v}$ such that $\mathfrak{a} = (\gamma)FB^{\vec{v}}$ and test whether there exists a solution $\vec{x} \in \mathbb{Z}^k$ of $H\vec{x} = \vec{v}$. If not, then $\mathfrak{a}$ is not principal. Otherwise, let T'_s be the matrix formed by the last k columns of T_s. Then, we have $AT_1 \ldots T_{s-1}T'_s = H$. Furthermore, we have that the elements in

$$\vec{r} = \vec{l}\,T_1 \ldots T_{s-1}T'_s,$$

where $l_i = \frac{1}{2}\log|\frac{\gamma_i}{\sigma(\gamma_i)}| \pmod{R_\Delta}$, are approximately the distances of the principal ideals corresponding to the columns of H. In other words, if column i of H is $\vec{h}_i$ and $\mathfrak{h}_i = FB^{\vec{h}_i} = (\alpha_i)$, then $r_i \approx \frac{1}{2}\log|\frac{\alpha_i}{\sigma(\alpha_i)}| \pmod{R_\Delta}$. It follows that the generator β of $FB^{\vec{v}}$ satisfies

$$\frac{1}{2}\log\left|\frac{\beta}{\sigma(\beta)}\right| \pmod{R_\Delta} \approx \vec{r}\,\vec{x} \pmod{R_\Delta},$$

since

$$\prod_{i=1}^{k} \mathfrak{h}_i^{x_i} \sim FB^{\vec{v}} \ .$$

Finally, since $\mathfrak{a} = (\gamma)FB^{\vec{v}}$ we have that

$$d = \frac{1}{2}\log\left|\frac{\gamma}{\sigma(\gamma)}\right| + \sum_{i=1}^{n} z_i l_i \pmod{R_\Delta}, \quad \vec{z} = T_1 \ldots T'_s\vec{x}, \tag{4.1}$$

is an approximation of $\delta(\mathfrak{a})$. We take as $\delta(\mathfrak{a})$ the value of $d + jR_\Delta$ for $j \in \mathbb{Z}$ such that $0 < d + jR_\Delta \leq R_\Delta$.

There are two main computational tasks required in this method, assuming that the class group and regulator have already been computed. The first is to compute a representation of $\mathfrak{a}$ over the factor base, which can be done using Algorithm 3.1. Recall that this algorithm makes use of our method for finding relations based on sieving techniques, and is hence more efficient in practice than the corresponding methods in [1] and [12] which do not use sieving. The linear algebra required, solving a linear system of equations over $\mathbb{Z}$, is very easy in our case, since the matrix H is in upper-triangular form (Hermite normal form). However, computing d can be very difficult, because the precision required in order to ensure an accurate approximation is very high. We used the tools of M. Maurer [24] to compute this approximation. Using the fact that $\mathfrak{a} = (\alpha)$ where

$$\alpha = \gamma \prod_{i=i}^{n} \gamma^{z_i},$$

and $\vec{z}$ is given in (4.1), we use Maurer's methods to determine the precision required to guarantee that the floating point approximation of $\delta(\mathfrak{a})$ we compute is sufficiently accurate. The floating point approximations are computed using routines from the `xbigfloat` class in LiDIA [23]. This class contains routines designed to compute with rational approximations of real numbers while keeping track of the errors incurred, and is described in [7].

Our method for principality testing is given by the following algorithm.

Algorithm 4.1 (PRINCIPAL).
Determines whether $\mathfrak{a}$ is principal, and if so computes its distance.
Input: Δ, $\mathfrak{a}$ reduced
Output: -1 if $\mathfrak{a}$ is not principal, otherwise $\delta(\mathfrak{a})$.

1. Compute R_Δ with the algorithm from [19]. Keep the factor base $FB = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$, Hermite normal form basis of the relation lattice H, transformation matrices $T_1, \ldots, T_s$ and vector $\vec{\gamma} = \{\gamma_1, \ldots, \gamma_n\}$ containing the generators of each relation generated.
2. Compute $(\vec{v}, \gamma) =$ REPRESENT_OVER_FB($\mathfrak{a}$, FB) (Algorithm 3.1).
3. Compute $\vec{x} \in \mathbb{Z}^k$ such that $H\vec{x} = \vec{v}$. If no such $\vec{x}$ exists, exit and return -1.
4. Compute $\vec{z} = T_1 \ldots T_s' \vec{x}$, where T_s' is the matrix formed by taking the last k columns of T_s. Evaluate the product from right to left, using only matrix-vector multiplication.
5. Compute $\vec{l}$ such that
$$l_i = \frac{1}{2} \log \left| \frac{\gamma_i}{\sigma(\gamma_i)} \right| \pmod{R_\Delta} .$$
6. Compute
$$d \approx \frac{1}{2} \log \left| \frac{\gamma}{\sigma(\gamma)} \right| + \sum_{i=1}^{k} z_i l_i .$$
Take $\delta(\mathfrak{a})$ to be the smallest value of $d + jR_\Delta$ such that $j \in Z$ and $0 < d + jR_\Delta \leq R_\Delta$.

5. Computational Results

5.1. **Discrete Logarithms in Cl_Δ.** We present some computational results of applying our implementation of Algorithm 3.3 in the LiDIA computer algebra system [23]. We have computed discrete logarithms for four test classes of discriminants, namely $\Delta = -4(10^x + 1)$ and $\Delta = -(10^x + 3)$ for imaginary quadratic orders and $\Delta = 4(10^x + 3)$ and $\Delta = 10^x + 1$ for real quadratic orders. In all cases, we have computed the average time for solving each of six random instances of Problem 3.1 for each different value of Δ. These run-times, given in CPU seconds (s), minutes (m), hours (h), or days (d) on a 296 MHz UltraSPARC-II processor, are presented in Tables 5.1 to 5.3. The average time for solving each of the six instances of Problem 3.1 is given by t_{dl}, and the time required to compute the structure of the class group is given by t_{Cl}.

Notice that in all cases, the time required to solve Problem 3.1 is very small once the class group has been computed. For the largest example, $-4 \times (10^{80} + 1)$, each instance requires about 4.5 hours, as compared to almost 5.5 days for computing the class group. The times for the real quadratic orders are somewhat smaller than those for the imaginary quadratic orders, since the bulk of the time is spent computing representations of the ideals $\mathfrak{a}$ and $\mathfrak{b}$ over a system of generators. The main part of this procedure is computing representations over the factor base (i.e., relation generation), and as observed in [19], relation generation is in general faster for real quadratic orders.

TABLE 5.1. Average discrete logarithm run-times for $\Delta < 0$.

	$\Delta = -4(10^x + 1)$		$\Delta = -(10^x + 3)$	
x	t_{Cl}	t_{dl}	t_{Cl}	t_{dl}
10	0.51 s	0.02 s	0.37 s	0.02 s
15	0.51 s	0.03 s	0.45 s	0.03 s
20	1.05 s	0.07 s	0.85 s	0.07 s
25	1.60 s	0.16 s	1.66 s	0.15 s
30	3.14 s	0.25 s	2.93 s	0.25 s
35	5.79 s	0.48 s	10.93 s	0.53 s
40	22.00 s	0.96 s	36.63 s	1.68 s
45	1.32 m	9.57 s	2.22 m	12.24 s
50	4.03 m	6.57 s	6.47 m	18.06 s
55	26.70 m	1.34 m	19.22 m	35.44 s
60	1.15 h	1.11 m	2.37 h	1.86 m
65	4.85 h	4.73 m	5.20 h	1.63 m
70	13.01 h	12.71 m	1.28 d	19.19 m
75	1.86 d	47.73 m	1.88 d	55.78 m
80	5.37 d	4.35 h	10.00 d	4.79 h

TABLE 5.2. Average discrete logarithm run-times for $\Delta = 4(10^x + 3)$.

x	t_{Cl}	t_{dl}
10	2.04 s	0.00 s
15	2.08 s	0.01 s
16	1.79 s	0.00 s
19	3.18 s	0.09 s
20	4.69 s	0.10 s
25	5.68 s	0.22 s
26	7.69 s	0.09 s
29	9.96 s	0.06 s
30	11.66 s	0.08 s
35	21.21 s	0.51 s
36	25.86 s	0.71 s
39	46.01 s	0.00 s
40	53.04 s	0.72 s
45	2.59 m	0.42 s
46	4.55 m	0.01 s
49	8.87 m	50.41 s
50	9.91 m	2.83 s
55	55.57 m	0.01 s
56	1.18 h	0.00 s
59	3.35 h	17.59 s
60	2.95 h	14.24 s
65	23.73 h	9.47 s
66	22.99 h	32.41 s

TABLE 5.3. Average discrete logarithm run-times for $\Delta = 10^x + 1$.

x odd			x even		
x	t_{Cl}	t_{dl}	x	t_{Cl}	t_{dl}
-	-	-	10	0.37 s	0.01 s
11	2.39 s	0.00 s	12	0.42 s	0.02 s
15	1.72 s	0.04 s	16	0.60 s	0.05 s
19	3.09 s	0.04 s	20	0.85 s	0.09 s
25	6.33 s	0.04 s	26	2.00 s	0.20 s
29	9.93 s	0.06 s	30	3.39 s	0.41 s
35	19.52 s	0.23 s	36	11.93 s	1.07 s
39	47.56 s	3.52 s	40	22.28 s	1.90 s
45	2.20 m	0.59 s	46	1.57 m	8.08 s
49	6.18 m	0.31 s	50	3.08 m	36.48 s
55	39.41 m	1.63 s	56	20.90 m	3.00 m
59	2.34 h	14.32 s	60	1.32 h	46.48 s
65	21.42 h	27.85 s	66	4.77 h	2.24 m
69	-	-	70	12.03 h	6.46 m
75	-	-	76	1.78 d	38.77 m
79	-	-	80	5.24 d	23.70 m

For the sake of comparison, we note that the largest imaginary quadratic order for which Problem 3.1 had been solved previously was $\mathcal{O}_{-4F_7}$, where F_7 is the seventh Fermat number and $\Delta = -4F_7$ has 40 decimal digits [4]. It took Buchmann and Düllmann about 6 days to compute the class group and about 114 seconds on a SPARCStation 1 to evaluate any individual discrete logarithm. We were able to compute the class group for this discriminant in only 15 seconds, and each DLP instance took less than one second. Using the rough figure that a UltraSPARC-II is 24 times faster than a SPARCStation1, we estimate that our algorithm using sieving computes each instance of the discrete logarithm problem about 4 times faster than without.

Tables 5.2 and 5.3 appear to contain the first examples of computing discrete logarithms in the class group of real quadratic orders. The most interesting examples are the case $\Delta = 10^x + 1$, x even, since the quadratic orders of these discriminants have very small regulators, and hence by the analytic class number formula the class number is large. The run-times in this case are actually somewhat faster than those for the imaginary quadratic orders. This is to be expected, since the Hermite normal form computation is the same as that for imaginary quadratic orders (the transformation matrix is not needed) and as observed in [19, Chapter 5], relation generation is in general faster for real quadratic orders.

5.2. **Principality Testing.** We present some computational results of applying our implementation of Algorithm 4.1 in the LiDIA computer algebra system [23]. Since principality testing is trivial in imaginary quadratic orders, we have only considered the two test classes of positive discriminants, $\Delta = 4(10^x + 3)$ and $\Delta = 10^x + 1$. Also, since principality testing is easy for $\Delta = 10^x + 1$, x even, we only consider odd x for this type of discriminant. In all cases, we have computed the average time for solving each of six random instances of Problem 4.1 for each different value of Δ. These run-times, given in CPU seconds (s), minutes (m), hours (h), or days (d) on a 296 MHz UltraSPARC-II processor, are presented in Tables 5.4 and 5.5. The average time for solving the each of the six instances of Problem 4.1 is given by t_{prin}, and the time required to compute the structure of the class group is given by t_{Cl}. We also give the average time required to compute the representation of each ideal over the factor base and the corresponding $\vec{z}$ (Steps 1 to 4 of Algorithm 4.1) by t_{rep}, and the average time required to compute an approximation of $\delta(\mathfrak{a})$ by t_{app}.

In most cases, the time required to compute a representation of $\mathfrak{a}$ over the factor base and to evaluate $\vec{z}$ was very small compared to that required to compute the class group and regulator. The main work in

TABLE 5.4. Average principality test run-times for $\Delta = 4(10^x + 3)$.

x	t_{Cl}	t_{rep}	t_{app}	t_{prin}
10	2.04 s	0.07 s	1.41 s	1.50 s
15	2.08 s	0.03 s	1.25 s	1.31 s
16	1.79 s	0.03 s	1.01 s	1.10 s
19	3.18 s	0.08 s	3.36 s	3.48 s
20	4.69 s	0.09 s	3.19 s	3.32 s
25	5.68 s	0.17 s	21.23 s	21.48 s
26	7.69 s	0.17 s	27.70 s	27.95 s
29	9.96 s	0.31 s	54.51 s	54.90 s
30	11.66 s	0.20 s	37.23 s	37.54 s
35	21.21 s	0.69 s	1.83 m	1.84 m
36	25.86 s	0.89 s	1.17 m	1.19 m
39	46.01 s	1.58 s	3.05 m	3.09 m
40	53.04 s	0.86 s	54.76 s	55.82 s
45	2.59 m	3.82 s	2.64 m	2.70 m
46	4.55 m	3.86 s	3.16 m	3.23 m
49	8.87 m	1.06 m	3.16 m	4.23 m
50	9.91 m	21.57 s	3.47 m	3.83 m
55	55.57 m	54.67 s	20.57 m	21.49 m
56	1.18 h	1.57 m	49.23 m	50.81 m
59	3.35 h	3.51 m	55.07 m	58.59 m
60	2.95 h	1.10 m	1.41 h	1.43 h
65	23.73 h	1.15 h	4.29 h	5.44 h
66	22.99 h	57.53 m	3.42 h	4.38 h

TABLE 5.5. Average principality test run-times for $\Delta = 10^x + 1$, x odd.

x	t_{Cl}	t_{rep}	t_{app}	t_{prin}
11	2.39 s	0.14 s	1.72 s	1.89 s
15	1.72 s	0.04 s	1.00 s	1.07 s
19	3.09 s	0.06 s	2.48 s	2.58 s
25	6.33 s	0.21 s	35.81 s	36.09 s
29	9.93 s	0.24 s	52.31 s	52.65 s
35	19.52 s	0.64 s	59.74 s	1.01 m
39	47.56 s	2.12 s	2.11 m	2.15 m
45	2.20 m	9.56 s	2.42 m	2.58 m
49	6.18 m	16.84 s	4.25 m	4.54 m
55	39.41 m	1.47 m	23.17 m	24.64 m
59	2.34 h	4.19 m	41.07 m	45.27 m
65	21.42 h	29.16 m	4.00 h	4.49 h

this part of the algorithm is computing one relation (for representing $\mathfrak{a}$ over the factor base), solving an upper-triangular linear system over $\mathbb{Z}$, and evaluating the matrix-vector products required to compute $\vec{z}$, none of which are very time-consuming. The reason for the large jump in t_{rep} between $x = 59$ and $x = 65$ is, as specified in [19], that the matrices T_i are stored in disk files when they have more than 2000 rows in order to conserve main memory.

At the moment, the bottleneck in solving Problem 4.1 is computing the floating point approximation of $\delta(\mathfrak{a})$. In fact, for $25 \leq x \leq 45$ the average time to compute this approximation alone was more than that required to compute the class group and regulator. If all we are interested in is the decision problem, i.e., simply determining whether $\mathfrak{a}$ is principal, then this is not a problem. In this case, there is no need to even compute the vector $\vec{z}$; it is sufficient to know that the system $H\vec{x} = \vec{v}$ can be solved. Also, if we only want the quadratic integer α, there may be no need to approximate $\delta(\mathfrak{a})$, since

$$\alpha = \gamma \prod_{i=1}^{n} \gamma_i^{z_i} \ .$$

From this representation, it should be possible to efficiently find either an explicit representation of α or a short representation like those described in [8]. However, at the moment the best way to handle these problems and to compute $\delta(\mathfrak{a})$ of which we are aware is the method outlined in Algorithm 4.1.

The difficulty in approximating $\delta(\mathfrak{a})$ is that the coefficients of $\vec{z}$ are very large, and extremely high precision is required in order to avoid round-off errors incurred during the course of the approximation. Furthermore, in order to get an accurate result modulo R_Δ, even greater precision must be used. Again, at the moment we know of know way around this problem. For discriminants with around 30 decimal digits and more we were unable to obtain accurate results in a reasonable amount of time without Maurer's method.

As with the regulator computation, the method described in [12] requires that the column operations performed during the Hermite normal form computation be performed directly on $\vec{l}$, the vector containing the distances corresponding to the relations in the relation matrix. In other words, the vector $\vec{r}$ is computed during the course of the Hermite normal form computation, rather than afterwards by making use of the transformation matrices. For smaller examples this approach works fine, but whenever modular techniques are used during the Hermite normal form computation, as is necessary for large examples, it is no longer possible.

Although algorithms are given in both [1] and [12] for principality testing, to the best of our knowledge the computations presented here are the first presented for this problem. Cohen et. al. have implemented their algorithm as part of the PARI computer algebra system, and in [11] they present computations where the regulators for some real quadratic orders are computed using their algorithm. The largest example they gave was the quadratic order with 41 digit discriminant $10^{40} + 1$. Since most of the approximations required for principality testing are computed during the computation of the regulator in their algorithm, it is reasonable to assume that they could solve Problem 4.1 for real quadratic orders with similar sized discriminants.

The largest example for which we were able to perform principality testing was the real quadratic order with 67-digit discriminant $4(10^{66} + 3)$. Even though we can now compute regulators for orders with discriminants in excess of 80 decimal digits [19], we are as yet unable to solve Problem 4.1 for these larger orders. The principal ideal decision problem is certainly possible in these cases, since a Hermite normal form basis of the relation lattice is always computed. However, as stated in [19] we are unable to compute transformation matrices with integer coefficients for these larger quadratic orders, and as a result we cannot directly apply our algorithm to compute $\delta(\mathfrak{a})$.

References

1. C.S. Abel, *Ein Algorithmus zur Berechnung der Klassenzahl und des Regulators reellquadratischer Ordnungen*, Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany, 1994.
2. I. Biehl, J. Buchmann, and C. Thiel, *Cryptographic protocols based on discrete logarithms in real-quadratic orders*, Advances in Cryptology - CRYPTO '94, Lecture Notes in Computer Science, vol. 839, 1995, pp. 56–60.
3. J. Buchmann, *A subexponential algorithm for the determination of class groups and regulators of algebraic number fields*, Séminaire de Théorie des Nombres (Paris), 1988-89, pp. 27–41.
4. J. Buchmann and S. Düllmann, *On the computation of discrete logarithms in class groups*, Advances in Cryptology - CRYPTO '90, Lecture Notes in Computer Science, vol. 537, 1991, pp. 134–139.
5. J. Buchmann, S. Düllmann, and H.C. Williams, *On the complexity and efficiency of a new key exchange system*, Advances in Cryptology - EUROCRYPT '89, Lecture Notes in Computer Science, vol. 434, 1990, pp. 597–616.

6. J. Buchmann, M.J. Jacobson, Jr., and E. Teske, *On some computational problems in finite abelian groups*, Math. Comp. **66** (1997), no. 220, 1663–1687.
7. J. Buchmann and M. Maurer, *Approximate evaluation of* $L(1, \chi_\Delta)$, Tech. Report TI-6/98, Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany, 1998.
8. J. Buchmann, C. Thiel, and H.C. Williams, *Short representation of quadratic integers*, Computational Algebra and Number Theory, Mathematics and its Applications **325**, Kluwer, Dordrecht, 1995, pp. 159–185.
9. J. Buchmann and H.C. Williams, *A key-exchange system based on imaginary quadratic fields*, Journal of Cryptology **1** (1988), 107–118.
10. H. Cohen, *A course in computational algebraic number theory*, Springer-Verlag, Berlin, 1993.
11. H. Cohen, F. Diaz y Diaz, and M. Olivier, *Calculs de nombres de classes et de régulateurs de corps quadratiques en temps sous-exponentiel*, Séminaire de Théorie des Nombres (Paris), 1993, pp. 35–46.
12. ———, *Subexponential algorithms for class and unit group computations*, J. Symb. Comp. **24** (1997), 433–441.
13. H. Cohen and H.W. Lenstra, Jr., *Heuristics on class groups*, Number Theory (Noordwijkerhout, 1983), Lecture Notes in Math., vol. 1052, Springer-Verlag, New York, 1984, pp. 26–36.
14. S. Düllmann, *Ein Algorithmus zur Bestimmung der Klassengruppe positiv definiter binärer quadratischer Formen*, Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany, 1991.
15. T. El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), 469–472.
16. J.L. Hafner and K.S. McCurley, *A rigorous subexpoential algorithm for computation of class groups*, J. Amer. Math. Soc. **2** (1989), 837–850.
17. L.K. Hua, *Introduction to number theory*, Springer-Verlag, New York, 1982.
18. D. Hühnlein, M.J. Jacobson, Jr., S. Paulus, and T. Takagi, *A cryptosystem based on non-maximal imaginary quadratic orders with fast decryption*, Advances in Cryptology - EUROCRYPT '98, Lecture Notes in Computer Science, vol. 1403, 1998, pp. 294–307.
19. M.J. Jacobson, Jr., *Subexponential class group computation in quadratic orders*, Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1999, In preparation.
20. N. Koblitz, *Elliptic curve cryptosystems*, Math. Comp. **48** (1987), 203–209.
21. ———, *Hyperelliptic cryptosystems*, Journal of Cryptology **1** (1989), 139–150.
22. H.W. Lenstra, Jr., *On the calculation of regulators and class numbers of quadratic fields*, London Math. Soc. Lecture Note Series **56** (1982), 123–150.
23. LiDIA, *http://www.informatik.tu-darmstadt.de/TI/LiDIA*, 1997.
24. M. Maurer, *To be announced*, Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1999, In preparation.
25. K.S. McCurley, *Cryptographic key distribution and computation in class groups*, Proc. NATO ASI on Number Theory and Applications (R.A. Mollin, ed.), Kluwer Academic Press, 1989, pp. 459–479.
26. A. Meyer, *Ein neues Identifikations- und Signaturverfahren über imaginär-quadratischen Zahlkörpern*, Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 1997.
27. V. Miller, *Use of elliptic curves in cryptography*, Advances in Cryptology - CRYPTO '85, Lecture Notes in Computer Science, vol. 218, 1986, pp. 417–426.
28. S. Paulus, *An algorithm of subexponential type computing the class group of quadratic orders over principal ideal domains*, Algorithmic Number Theory - ANTSII (Université Bordeaux I, Talence, France), Lecture Notes in Computer Science, vol. 1122, Springer-Verlag, Berlin, 1996, pp. 243–257.
29. R. Scheidler, J. Buchmann, and H.C. Williams, *A key-exchange protocol using real quadratic fields*, Journal of Cryptology **7** (1994), 171–199.
30. M. Seysen, *A probabilistic factorization algorithm with quadratic forms of negative discriminant*, Math. Comp. **48** (1987), 757–780.
31. D. Shanks, *The infrastructure of real quadratic fields and its applications*, Proc. 1972 Number Theory Conf., Boulder, Colorado, 1972, pp. 217–224.
32. R.D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987), 329–339.

Centre for Applied Cryptographic Research, University of Waterloo, Waterloo, Ontario, CANADA, N2L 3G1

E-mail address: `mjjacobs@cacr.math.uwaterloo.ca`