Algebraic techniques for number field computations (extended abstract)

Jean-François Biasse[1], Michael J. Jacobson, Jr.[2⋆], and Alan K. Silvester[3]

[1] École Polytechnique, 91128 Palaiseau, France
biasse@lix.polytechnique.fr
[2] Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
jacobs@cpsc.ucalgary.ca
[3] Department of Mathematics and Statistics, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
aksilves@math.ucalgary.ca

Abstract. We present improvements to the computations related to quadratic number fields and their application to cryptology.

1 Introduction

Quadratic number fields were proposed as a setting for public-key cryptosystems in the late 1980s by Buchmann and Williams [4, 5]. Their security relies on the hardness of the discrete logarithm problem in the imaginary case and the infrastructure discrete logarithm problem in the real case. The complexity of the algorithms for solving these problems is bounded by $L(1/2, O(1))$ where the subexponential function is defined as

$$L(\alpha, \beta) = e^{\beta \log |\Delta|^{\alpha} \log\log |\Delta|^{1-\alpha}},$$

where Δ is the discriminant of the order we are working with. This complexity is asymptotically slower than the one for factoring which reduces to the problem of computing the class number, and although the discrete logarithm problem in the Jacobian of elliptic curves remains exponential, there is no known reduction between this problem and the discrete logarithm problems in number fields either [14]. Therefore, studying the hardness of the discrete logarithm problem and of the principality testing problem on number fields is of cryptographic interest since they provide alternative cryptosystems whose security is unrelated to those currently being used.

Following the recommendations for securely choosing discriminants for use in quadratic field cryptography of [10] for the imaginary case and of [13] for the real case, we restricited our study to the case of prime discriminants. Indeed, in both imaginary and real cases, it usually suffices to use prime discriminants, as this forces the class number h_{Δ} to be odd. In the imaginary case, one then relies on

⋆ The second author is supported in part by NSERC of Canada.

the Cohen-Lenstra heuristics [9] to guarantee that the class number is not smooth with high probability. In the real case, one uses the Cohen-Lenstra heuristics to guarantee that the class number is very small (and that the infrastructure is therefore large) with high probability. This restriction also prevents ourselves against the attacks described by Castagnos and Laguillaumie in the imaginary case [7] and by Castagnos, Joux, Laguillaumie and Nguyen in the real case [6] which are designed for discriminants of the form $\Delta = \pm np^2$.

In this paper, we describe improvements to the algorithms for computing the group structure of the ideal class group $Cl(\mathcal{O}_\Delta)$ of the maximal order $\mathcal{O}_\Delta$, solving instances of the discrete logarithm problem in $Cl(\mathcal{O}_\Delta)$, computing the regulator of $\mathcal{O}_\Delta$ when $\Delta > 0$ and solving the infrastructure discrete logarithm problem. After a brief description of the necessary background concerning number fields, we describe the last improvements affecting the linear algebra phase. We begin with a dedicated Gaussian elimination strategy to reduce the dimensions of the relation matrix M. Then, we provide numerical data about an implementation of a new algorithm for computing the Hermite Normal Form of M and thus deduce the group structure of $Cl(\mathcal{O}_\Delta)$. We also describe a new algorithm for the regulator computation, and finally we show the impact of a new algorithm due to Vollmer [23] for solving instances of the discrete logarithm problem in $Cl(\Delta)$.

Note that most of this paper is not new material. Section 3 is taken from [1], and Section 4 from [2]. Vollmer's algorithm described in Section 5 for computing the HNF was already used in [1], but it had not been compared with the existing HNF algorithm. Section 6 is taken from [3]. The security estimates given in Section 7 were not described before.

2 Number fields

Let $K = \mathbb{Q}(\sqrt{\Delta})$ be the quadratic field of discriminant Δ, where Δ is a non-zero integer congruent to 0 or 1 modulo 4 with Δ or $\Delta/4$ square-free. The integral closure of $\mathbb{Z}$ in K, called the maximal order, is denoted by $\mathcal{O}_\Delta$. An ideal can be represented by the two dimensional $\mathbb{Z}$-module

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

where $a, b, s \in \mathbb{Z}$ and $4a \mid b^2 - \Delta$. The integers a and s are unique, and b is defined modulo $2a$. The norm of $\mathfrak{a}$ is given by $\mathcal{N}(\mathfrak{a}) = as^2$. Ideals can be multiplied using Gauss' composition formulas for integral binary quadratic forms. Ideal norm respects this operation. The prime ideals of $\mathcal{O}_\Delta$ have the form $p\mathbb{Z}+(b_p+\sqrt{\Delta})/2\mathbb{Z}$ where p is a prime that is split or inert in K, i.e., the Kronecker symbol $(\Delta/p) \neq -1$. As $\mathcal{O}_\Delta$ is a Dedekind domain, every ideal can be factored uniquely as a product of prime ideals. We define the ideal class group as $Cl(\Delta) := \mathcal{I}_\Delta/\mathcal{P}_\Delta$, where $\mathcal{I}_\Delta$ is the set of invertible ideals, and $\mathcal{P}_\Delta$ is the set of principal ideals of $\mathcal{O}_\Delta$. This way, given two ideals $\mathfrak{a}$ and $\mathfrak{b}$ we have

$$[\mathfrak{a}] = [\mathfrak{b}] \in Cl(\Delta) \iff \exists \alpha \in \mathbb{K} \ \mathfrak{b} = (\alpha)\mathfrak{a}.$$

$Cl(\Delta)$ is a finite group of cardinality h_Δ. To create M, we use sieving based techniques to find relations of the form

$$(\alpha) = \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n},$$

where $\alpha \in \mathbb{K}$, and the $\mathfrak{p}_i$ are the prime ideals belonging to the set $\mathcal{B}$ of prime ideals of norm bounded by a certain bound B. Every time such a relation is found, we add the vector $[e_1, \dots, e_n]$ as a row of M. The most efficient way to do this is to use an adapation of the multiple polynomial quadratic sieve due to Jacobson [11], which was improved by Biasse [1] who used the large prime variants. Under the generalized Riemann hypothesis (GRH), if $B \geq 6 \log^2 |\Delta|$, the lattice Λ generated by all the possible relations satisfies

$$Cl(\Delta) \simeq \mathbb{Z}^n / \Lambda.$$

A linear algebra phase consisting in the computation of the Smith Normal Form (SNF) of M yields the group structure of $Cl(\Delta)$.

The units of $\mathcal{O}_\Delta$ form a multiplicative group

$$U_\Delta \simeq \mu_\Delta \times \{\varepsilon_\Delta\} \text{ (real) or } U_\Delta \simeq \mu_\Delta \text{ (imaginary)},$$

where μ_Δ are the roots of unity, and ε_Δ is the *fundamental unit*. In the real case, we compute the regulator $R_\Delta := \log |\varepsilon_\Delta|$ by finding kernel vectors of M during the linear algebra phase. Then, we give them as input to an algorithm due to Maurer [16] along with the generators α_i, $i \leq n$ of the relations. For cryptographic applications, we focus on solving the discrete logarithm problem (infrastructure DLP in the real case). Given two ideals $\mathfrak{a}$ and $\mathfrak{b}$ such that there exists $x \in \mathbb{Z}$ with $[\mathfrak{b}] = [\mathfrak{a}]^x \in Cl(\Delta)$, we aim at finding x and $\log |\alpha| \mod R_\Delta$ where $\mathfrak{b} = (\alpha)\mathfrak{a}^x$. In imaginary fields ($\Delta < 0$), α is trivial, and we only compute x.

3 Structured Gaussian elimination

Before applying the linear algebra algorithms we mentioned, we perform a Gaussian elimination step to reduce the dimensions of M. The main drawback of this strategy is that the density and the size of the coefficients of the matrix increase after each recombination of rows. We used a graph-based elimination strategy first described by Cavallar [8] for factorization, and then adapted by Biasse [1] to the context of number fields. Given a column involving N rows, this algorithm finds an optimal recombination strategy between the rows with respect to a cost function

$$C(r) := \sum_{1 \leq |e_i| \leq 8} 1 + 100 \sum_{|e_j| > 8} |e_j|,$$

where $r = [e_1, \dots, e_n]$ is a row. This cost function penalizes rows with large entries and high density. The first step of the algorithm is to build the complete graph $\mathcal{G}$ having N edges, and whose vertices (i, j) are weighted by the cost of the

recombination involving the rows i and j according to C. Then, we compute the minimum spanning tree $\mathcal{T}$ of $\mathcal{G}$. Finally, we recombine the rows starting from those corresponding to the leaves of $\mathcal{T}$ and finishing with its root. At the end, we verify that the resulting matrix M_{red} has full rank with Linbox `rank` function. If not, we add more rows and repeat the process.

To illustrate the impact of this structured Gaussian elimination strategy over the naive Gaussian elimination, we monitored in Table 1 the evolution of the dimensions of the matrix, the average Hamming weight of its rows, the extremal values of its coefficients and the time taken for computing its HNF in the case of a relation matrix corresponding to $\Delta = 4(10^{60} + 3)$. We kept track of these values after all i-way merges for some values of i between 5 and 170. The original dimensions of the matrix were 2000×1700, and the timings were obtained on a 2.4 Ghz Opteron with 32GB of memory.

Table 1. Comparative table of elimination strategies

Naive Gauss						
i	Row Nb	Col Nb	Average weight	max coeff	min coeff	HNF time
5	1189	1067	27.9	14	-17	357.9
10	921	799	49.3	22	-19	184.8
30	757	635	112.7	51	-50	106.6
50	718	596	160.1	81	-91	93.7
70	699	577	186.3	116	-104	85.6
90	684	562	205.5	137	-90	79.0
125	664	542	249.0	140	-146	73.8
160	655	533	282.4	167	-155	72.0
170	654	532	286.4	167	-155	222.4
With dedicated elimination strategy						
i	Row Nb	Col Nb	Average weight	max coeff	min coeff	HNF time
5	1200	1078	26.8	13	-12	368.0
10	928	806	42.6	20	-15	187.2
30	746	624	82.5	33	-27	100.8
50	702	580	107.6	64	-37	84.3
70	672	550	136.6	304	-676	73.4
90	656	534	157.6	1278	-1088	67.5
125	637	515	187.1	3360	-2942	63.4
160	619	497	214.6	5324	-3560	56.9
170	615	493	247.1	36761280	-22009088	192.6

Table 1 shows that the use of our elimination strategy led to a matrix with smaller dimension (493 rows with our method, 533 with the naive elimination) and lower density (the average weight of its rows is of 214 with our method and 282 with the naive elimination). These differences result in an improvement of

the time taken by the HNF computation: 56.9 with our method against 72.0 with the naive Gaussian elimination.

4 Regulator computation

To solve the infrastructure discrete logarithm problem, we first need to compute an approximation of the regulator. For this purpose, we used an improved version of Vollmer's system solving based algorithm [24] described by Biasse and Jacobson [2]. In order to find elements of the kernel, the algorithm creates extra relations r_i, $0 \leq i \leq k$ for some small integer k (in our experiments, we always have $k \leq 10$). Then, we solve the k linear systems $X_i M = r_i$ using the function `certSolveRedLong` from the IML library [22]. We augment M by adding the r_i as extra rows, and augment the vectors X_i with $k-1$ zero coefficients and -1 at index $n+i$, yielding

$$M' := \begin{pmatrix} M \\ \cdots\cdots \\ r_i \end{pmatrix}, \quad X_i' := \left(\begin{array}{c|c} X_i & 0 \ldots 0 \ -1 \ 0 \ldots 0 \end{array} \right) .$$

The X_i' are kernel vectors of M', which can be used along with the vector $\boldsymbol{v}$ containing the real parts of the relations, to compute a multiple of the regulator with Maurer's algorithm [17, Sec 12.1]. As shown in Vollmer [24], this multiple is equal to the regulator with high probability. In [2], it is shown that this method is faster than the one requiring a kernel basis because it only requires the solution to a few linear systems, and it can be adapted in such a way that the linear system involves M_{red}.

To illustrate the impact of this algorithm, we used the relation matrix obtained in the base case for discriminants of the form $4(10^n+3)$ for n between 40 and 70. The timings are obtained on a 2.4GHz Opteron with 16GB of memory. In Table 2, the timings corresponding to our system solving approach are taken

Table 2. Comparative table of regulator computation time

n	Kernel Computation	System Solving
40	15.0	6.2
45	18.0	8.3
50	38.0	20.0
55	257.0	49.0
60	286.0	103.0
65	5009.0	336.0
70	10030.0	643.0

with seven kernel vectors. However, in most cases only two or three vectors are

required to compute the regulator. As most of the time taken by our approach is spent on system solving, we see that computing fewer kernel vectors would result in an improvement of the timings, at the risk of obtaining a multiple of the regulator.

5 Class group computation

The class group structure is obtained with the diagonal coefficients of the SNF of M. Unfortunatelly, even after the Gaussian elimination, the dimensions of M_{red} are too large to allow a direct SNF computation. We thus have to compute the Hermite normal form (HNF) H of M first, and then to find the SNF of the essential part of H. A matrix H is said to be in HNF if with $\forall j < i : 0 \leq h_{ij} < h_{ii}$ and $\forall j > i : h_{ij} = 0$. For the imaginary case, we can use an algorithm due to Vollmer [25] which requires solutions to linear systems. For each $i \leq n$, we define two matrices

$$M_i = \begin{pmatrix} a_{1,1} & \dots & a_{m,1} \\ \vdots & & \vdots \\ a_{1,i} & \dots & a_{m,i} \end{pmatrix} \quad \text{and} \quad e_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

For each i, let h_i be the minimal denominator of a rational solution of the system $M_i x = e_i$ solved using the function `MinCertifiedSol` of IML [22]. We have $h_\Delta = \prod_i h_i$, and an extra computation involving modular reductions yields the essential part of the HNF of M. In most cases only a limited number of systems are to be soved. In the real case, we used a modular HNF algorithm [20]. It needs a multiple of h_Δ in input. To compute this multiple, we took the GCD of the determinants h_1 and h_2 of two $n \times n$ submatrices of M. We used the `determinant` function of Linbox for this purpose, which is why we refer to this strategy as NTL/Linbox in the following. Several implementations of an HNF algorithm are available today. In this section, we compare the most efficient ones: Magma, Sage, Kash, Pari to the methods we used in our computations. We used Magma V2.11-2 whereas a new algorithm is used since V.2.14. According to the developers' webpage, this algorithm should be more efficient on random dense matrices than the one we used, but we were not able to have it run on the same plateform. Sage Version 4.1.1, which is open-source, has an HNF algorithm based on the heuristic idea of Micciancio and Warinschi [18], which was analyzed and implemented By Pernet and Stein [21]. We used Kash Version 4 and Pari-2.3.5 whose HNF algorithm is due to Batut. Note here that Kash and Pari's algorithm provide the unimodular transformation matrix corresponding to the operations on the rows resulting in the HNF of the relation matrix. This step is not necessary for the ideal class group computation and can be time consuming.

Algorithm 1 Essential part of the HNF

Input: Δ, relation matrix $A \in \mathbb{Z}^{m \times n}$ of full rank and h_* such that $h^* \leq h < 2h^*$.
Output: The essential part of the HNF of A.

$h \leftarrow 1$, $i \leftarrow n$, $l \leftarrow 1$.
while $h < h_*$ **do**
  Compute the minimal denominator h_i of a solution $\overrightarrow{v_i}$ of $A_i \cdot x = e_i$.
  $h \leftarrow h \cdot h_i$.
  $i \leftarrow i - 1$, $l \leftarrow l + 1$.
end while
Let U be the $l \times m$ matrix whose rows are the $\overrightarrow{v_i}$ for $i \leq l$ and $H = (h_{ij})$ be the $l \times l$ submatrix of UA containing its last l rows.
for $2 \leq i \leq l$ **do**
  $h_{ij} \leftarrow h_{ij} \mod h_{ii}$ for all $j > i$.
end for
return H.

We also assessed the performances of Vollmer's algorithm on a single node, and on a several nodes. We noted between brackets the minimum number of nodes that is required to obtain the best performances in the parallelized version.

Table 3. Comparative timings of the HNF algorithms in the imaginary case

size	$\lvert\mathcal{B}\rvert$	algorithm	HNF time	stdev
130	320	Pari	16,64	9,01
		Kash	3,4	0,5
		Sage	5,6	0,68
		Magma	0,98	0,12
		NTL/Linbox	2,37	0,34
		Vollmer	2,2	1,04
		Vollmer Par (6)	0,61	0,05
150	400	Pari	30,56	4,09
		Kash	13,09	2,57
		Sage	12,69	8,06
		Magma	3,63	0,91
		NTL/Linbox	6,61	0,74
		Vollmer	7,12	3,54
		Vollmer Par (7)	1,62	0,11
160	450	Pari	54.41	25.55
		Kash	19,57	6,25
		Sage	19,96	5,26
		Magma	5,55	2,25
		NTL/Linbox	8,67	2,39
		Vollmer	8,02	2,62
		Vollmer Par (6)	1,91	0,38

In Table 3, we compared the time for computing the HNF of a relation matrix corresponding to negative discriminants of size ranging between 130 and 160 bits. For each discriminant size, we drew five random fundamental discriminants and computed a relation matrix on a 2.4 GHz Opteron with 8GB of memory. Unlike for the other benchmarks, we did not draw random prime discriminants because Vollmer's algorithm tends to be faster when working with relation matrices corresponding to cyclic ideal class groups. We notice that the best performances on a single are still optainded by Magma, but that the parallelized version of Vollmer's algorithm allows a significant speed-up if several nodes are available for this computation. This opens the way to fully parallelized algorithms since the relation collection phase is trivially parallelizable on as many nodes as we want.

6 DLP solving

For solving the discrete logarithm problem, we implemented an algorithm due to Vollmer [23] which also involves system solving. Given two ideals $\mathfrak{a}$ and $\mathfrak{b}$ such that $\mathfrak{b} = \mathfrak{a}^x$ for some integer x, it consists of finding two extra relations $\mathrm{r}_a : (\alpha_\mathfrak{a}) = \mathfrak{a} * \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n}$ and $\mathrm{r}_b : (\alpha_\mathfrak{b}) = \mathfrak{b} * \mathfrak{p}_1^{f_1} \dots \mathfrak{p}_n^{f_n}$ and extending the factor base with two extra elements: $\mathcal{B}' = \mathcal{B} \cup \{\mathfrak{a}, \mathfrak{b}\}$. The extra relations are obtained by multiplying $\mathfrak{a}$ and $\mathfrak{b}$ by random power products of primes in $\mathcal{B}$ and sieving with the resulting ideal. Then, we construct the matrix

$$A' := \left(\begin{array}{c:c} A & (0) \\ \hdashline r_\mathfrak{b} & 1 \\ r_\mathfrak{a} & 0 \end{array}\right),$$

and solve the system $XA' = (0, \dots, 0, 1)$. The last coordinate of X necessarily equals $-x$. For each system, we used `certSolveRedLong` from the IML library [22]. It appeared that it was faster than both kernel computation and HNF computation. Testing the principality of an ideal I and finding α such that $(\alpha) = I$ can be done by finding a power product satisfying $I = \prod_i \mathfrak{p}_i^{e_i}$. Then, we need to solve the system $XM = b$ where $b = [e_1, \dots, e_n]$. If this system has a solution, then I is principal and its generator is $\alpha = \prod_i \alpha_i^{x_i}$ where the α_i are the generators of the relations used for constructing M and the x_i are the coefficients of X. An algorithm of Maurer [16] computes $\log |\alpha| \mod R_\Delta$ given R_Δ, $(\alpha_i)_{i \leq n}$ and X.

To study the impact of Vollmer's algorithm for solving the discrete logarithm problem without computing the structure of $Cl(\Delta)$, we provided numerical data in Table 4 for discriminants of size between 140 and 220 bits. The timings, given in CPU seconds, are averages of three different random prime discriminants, obtained with 2.4 GHz Opterons with 8GB or memory. We denote by "DL" the discrete logarithm computation using Vollmer's method and by "CL" the

Table 4. Comparison between class group computation and Vollmer's Algorithm

Size	Strategy	$\lvert\mathcal{B}\rvert$	Sieving	Elimination	Linear algebra	Total
140	CL	200	2.66	0.63	1.79	5.08
	DL	200	2.57	0.44	0.8	3.81
160	CL	300	11.77	1.04	8.20	21.01
	DL	350	10.17	0.73	2.75	13.65
180	CL	400	17.47	0.98	12.83	31.28
	DL	500	15.00	1.40	4.93	21.33
200	CL	800	158.27	7.82	81.84	247.93
	DL	1000	126.61	9.9	21.45	157.96
220	CL	1500	619.99	20.99	457.45	1098.43
	DL	1700	567.56	27.77	86.38	681.71

class group computation. We list the optimal factor base size for each algorithm and discriminant size (obtained via additional numerical experiments), the time for each of the main parts of the algorithm, and the total time. In all cases we allowed two large primes and took enough relations to ensure that M_{red} have full rank. Our results show that using Vollmer's algorithm for computing discrete logarithms is faster than the approach of [12] that also requires the class group.

7 Security estimates

As the relation collection clearly influences the overall time of the algorithm, we classified the quadratic discriminants with respect to the difficulty to create the relation matrix. During the sieving phase, we essentially test the smoothness of ideals with respect to $\mathcal{B}$. This boils down to testing the smoothness of norms with respect to primes p such that there is a $\mathfrak{p} \in \mathcal{B}$ satisfying $\mathcal{N}(\mathfrak{p}) = p$. Therefore, the hardest discriminants will be those satisfying $\left(\frac{\Delta}{p}\right) = -1$ for the small primes. When we choose discriminants at random, we cannot control this property, and we thus observe a high standard deviation in the performances of the DLP algorithms at a fixed discriminant size. To provide security estimates, we want to choose our instances of the discrete logarithm problem amongst the easiest ones, and ensure that the performances of the algorithm for solving the DLP are regular. In our experiments, we studied the performances of Vollmer's algorithm for solving the discrete logarithm problem on imaginary discriminants of three classes.

1. The *easy discrimiants*, satisfying $\left(\frac{\Delta}{p}\right) = 1$ for $p = 2, 3, 5, 7, 11$.
2. The *intermediate discriminants*, satisfying $\left(\frac{\Delta}{p}\right) = -1$ for $p = 2, 3, 5, 7, 11$ and $\left(\frac{\Delta}{p}\right) = 1$ for $p = 13, 17, 19, 23, 31$.
3. The *hard discrinants*, satisfying satisfying $\left(\frac{\Delta}{p}\right) = -1$ for $p \leq 31$.

In Table 5, we randomly drew 10 negative prime discriminants of size 170, 190 and 210 bits for each class of discriminant, and computed the time to solve an instance of the DLP. We used a 2,4 GHz Opteron with 32GB of memory and counted the time in CPU seconds.

Table 5. Comparative table of DLP time for $\Delta < 0$

	Easy		**Intermediate**		**Hard**	
Size	Average	Stdev	Average	Stdev	Average	Stdev
170	22.1	4.7	65.5	18.7	103.0	26.5
190	70.3	13.3	162.7	25.5	224.8	32.26
210	257.7	26.0	655.7	99.7	885.5	152.1

We observe in Table 5 that the time taken to solve the discrete logarithm problem corroborates the hypothesis we made on the difficulty of solving the discrete logarithm problem on the classes of discriminants we described. For our security estimates, we will take random discriminants belonging to the easy class, unlike in [3], where we took random discriminants and thus observed timings with large standard deviations. The rest of the methodology remains the same. We first provide timings allowing ourselves to decide if the run time of our algorithm follows the proven complexity $O(L_{|\Delta|}[1/2, 3\sqrt{2}/4 + o(1)]$, or the heuristic one $O(L_{|\Delta|}[1/2, 1 + o(1)])$. Then, we give the discriminant size required to provide an equivalent level of security as the RSA moduli recommended by NIST [19]. We assume that the run time of factoring algorithms follow the heuristic complexity of the generalized number field sieve $L_N[1/3, \sqrt[3]{64/9} + o(1)]$, and follow the approach of Hamdy and Möller [10] who used the equation

$$\frac{L_{N_1}[e, c]}{L_{N_2}[e, c]} = \frac{t_1}{t_2}, \tag{1}$$

to compute the run time t_2 on input size N_2, knowing the run time t_1 on input size N_1. To date, the largest RSA number factored is RSA-768, a 768 bit integer [15]. It is estimated in [15] that the total computation required 2000 2.2 GHz AMD Opteron years. As our computations were performed on a different architecture, we follow Hamdy and Möller and use the MIPS-year measurement to provide an architecture-neutral measurement. In this case, assuming that a 2.2 GHz AMD Opteron runs at 4400 MIPS, we estimate that this computation took 8.8×10^6 MIPS-years. Using this estimate in conjunction with (1) yields the estimated running times to factor RSA moduli of the sizes recommended by NIST given in Table 6, where we focus on the three classes of discriminants and compare them to random discriminants.

The results of our experiments for the imaginary case are given in Table 7, and for the real case in Table 8. They were obtained on 2.4 GHz Xeon with 2GB of memory. For each bit length of Δ, denoted by "size(Δ)," we list the

Table 6. Security Parameter Estimates

RSA	$\Delta < 0$ (rnd.)	$\Delta < 0$ (easy)	$\Delta < 0$ (int.)	$\Delta < 0$ (hard)	Est. time (MIPS-years)
768	640	661	640	631	8.80×10^6
1024	798	821	798	788	1.07×10^{10}
2048	1348	1378	1349	1337	1.25×10^{19}
3072	1827	1860	1827	1813	4.74×10^{25}
7680	3598	3643	3599	3579	1.06×10^{45}
15360	5971	6028	5972	5948	1.01×10^{65}
RSA	$\Delta > 0$ (rnd.)	$\Delta > 0$ (easy)	$\Delta > 0$ (int.)	$\Delta > 0$ (hard)	Est. time (MIPS-years)
768	634	638	632	629	8.80×10^6
1024	792	796	789	786	1.07×10^{10}
2048	1341	1346	1337	1334	1.25×10^{19}
3072	1818	1824	1814	1810	4.74×10^{25}
7680	3586	3594	3580	3575	1.06×10^{45}
15360	5957	5966	5949	5942	1.01×10^{65}

average time in seconds required to solve an instance of the appropriate discrete logarithm problem ($\overline{t_\Delta}$) and standard deviation (std). For each size, we solved 10 instances of both problems. In both cases, we concluded that the run timed was in $O(L_{|\Delta|}[1/2, 1 + o(1)])$.

References

1. J-F. Biasse, *Improvements in the computation of ideal class groups of imaginary quadratic number fields*, To appear in *Advances in Mathematics of Communications*.
2. J-F. Biasse and M. J. Jacobson, Jr., *Practical improvements to class group and regulator computation of real quadratic fields*, 2010, To appear in ANTS 9.
3. J-F. Biasse, M. J. Jacobson, Jr., and A. K. Silverster, *Security estimates for quadratic field based cryptosystems*, 2010, To appear in ACISP 2010.
4. J. Buchmann and H. C. Williams, *A key-exchange system based on imaginary quadratic fields*, Journal of Cryptology **1** (1988), 107–118.
5. ———, *A key-exchange system based on real quadratic fields*, CRYPTO '89, Lecture Notes in Computer Science, vol. 435, 1989, pp. 335–343.
6. G. Castagnos, A. Joux, F. Laguillaumie, and P. Q. Nguyen, *Factoring pq^2 with quadratic forms: Nice cryptanalyses*, ASIACRYPT '09: Proceedings of the 15th annual international conference on the theory and applications of cryptology and information security (Berlin, Heidelberg), Lecture Notes in Computer Science, vol. 5912, Springer-Verlag, 2009, pp. 469–486.
7. G. Castagnos and F. Laguillaumie, *On the security of cryptosystems with quadratic decryption: The nicest cryptanalysis*, EUROCRYPT '09: Proceedings of the 28th annual international conference on Advances in Cryptology (Berlin, Heidelberg), Lecture Notes in Computer Science, vol. 5479, Springer-Verlag, 2009, pp. 260–277.
8. S. Cavallar, *Strategies in filtering in the number field sieve*, ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory, Lecture Note in Computer Science, vol. 1838, Springer-Verlag, 2000, pp. 209–232.

Table 7. Average run times for the discrete logarithm problem in Cl_Δ, $\Delta < 0$

size(Δ)	$\overline{t_\Delta}$ (sec)	std	$L_{\lvert\Delta\rvert}[1/2, \sqrt{2}]/\overline{t_\Delta}$	$L_{\lvert\Delta\rvert}[1/2, 1]/\overline{t_\Delta}$
140	3.73	0.61	2.33×10^{12}	3.78×10^{8}
142	4.75	0.86	2.37×10^{12}	3.57×10^{8}
144	4.92	0.92	2.97×10^{12}	4.14×10^{8}
146	5.21	0.64	3.62×10^{12}	4.68×10^{8}
148	5.92	0.69	4.10×10^{12}	4.93×10^{8}
150	6.36	1.47	4.92×10^{12}	5.49×10^{8}
152	6.65	0.89	6.05×10^{12}	6.26×10^{8}
154	8.20	1.27	6.30×10^{12}	6.06×10^{8}
156	10.68	4.11	6.20×10^{12}	5.55×10^{8}
158	10.36	2.23	8.18×10^{12}	6.81×10^{8}
160	12.11	2.51	8.95×10^{12}	6.93×10^{8}
162	18.03	3.64	7.67×10^{12}	5.53×10^{8}
164	22.78	8.03	7.74×10^{12}	5.20×10^{8}
166	21.23	4.84	10.58×10^{12}	6.62×10^{8}
168	26.59	8.32	10.75×10^{12}	6.27×10^{8}
170	29.15	8.15	12.45×10^{12}	6.77×10^{8}
172	32.24	6.78	14.28×10^{12}	7.24×10^{8}
174	49.71	18.65	11.74×10^{12}	5.55×10^{8}
176	52.08	12.57	14.18×10^{12}	6.26×10^{8}
178	51.99	9.79	17.96×10^{12}	7.40×10^{8}
180	75.10	18.75	15.70×10^{12}	6.04×10^{8}
182	73.34	4.76	2.02×10^{13}	7.29×10^{8}
184	80.66	14.19	2.32×10^{13}	7.81×10^{8}
186	79.69	16.25	2.96×10^{13}	9.30×10^{8}
188	93.73	11.09	3.16×10^{13}	9.30×10^{8}
190	100.89	13.93	3.69×10^{13}	10.15×10^{8}
192	117.18	14.71	3.98×10^{13}	10.26×10^{8}
194	133.77	16.43	4.37×10^{13}	10.54×10^{8}
196	167.70	21.11	4.37×10^{13}	9.86×10^{8}
198	162.21	13.59	5.65×10^{13}	11.94×10^{8}
200	195.29	24.69	5.87×10^{13}	11.61×10^{8}
202	291.58	27.96	4.90×10^{13}	9.10×10^{8}
204	292.70	42.55	6.09×10^{13}	10.59×10^{8}
206	335.39	39.38	6.63×10^{13}	10.80×10^{8}
208	360.00	51.24	7.69×10^{13}	11.75×10^{8}
210	396.10	82.10	8.69×10^{13}	12.46×10^{8}
212	448.85	72.62	9.53×10^{13}	12.82×10^{8}
214	535.67	123.40	9.92×10^{13}	12.52×10^{8}
216	595.56	109.94	11.07×10^{13}	13.12×10^{8}
218	641.99	89.52	12.73×10^{13}	14.16×10^{8}
220	829.98	151.75	12.19×10^{13}	12.74×10^{8}
230	1564.74	226.924	18.60×10^{13}	14.27×10^{8}
240	1564.74	226.924	52.48×10^{13}	29.71×10^{8}
250	5552.59	953.788	40.94×10^{13}	17.20×10^{8}

Table 8. Average run times for the infrastructure discrete logarithm problem.

size(Δ)	$\overline{t_\Delta}$ (sec)	std	$L_{\lvert\Delta\rvert}[1/2,\sqrt{2}]/\overline{t_\Delta}$	$L_{\lvert\Delta\rvert}[1/2,1]/\overline{t_\Delta}$
140	3.66	3.00	2.38×10^{12}	3.86×10^{8}
142	9.33	0.85	1.21×10^{12}	1.82×10^{8}
144	10.49	1.00	1.39×10^{12}	1.94×10^{8}
146	10.78	0.92	1.75×10^{12}	2.26×10^{8}
148	10.21	1.32	2.38×10^{12}	2.86×10^{8}
150	11.14	1.70	2.81×10^{12}	3.13×10^{8}
152	12.29	1.39	3.27×10^{12}	3.39×10^{8}
154	11.11	0.97	4.65×10^{12}	4.47×10^{8}
156	14.58	2.59	4.54×10^{12}	4.06×10^{8}
158	15.46	2.35	5.48×10^{12}	4.56×10^{8}
160	15.72	2.21	6.89×10^{12}	5.34×10^{8}
162	29.48	6.72	4.69×10^{12}	3.38×10^{8}
164	31.71	3.49	5.56×10^{12}	3.73×10^{8}
166	33.82	4.54	6.64×10^{12}	4.15×10^{8}
168	37.61	4.95	7.60×10^{12}	4.43×10^{8}
170	40.06	5.43	9.06×10^{12}	4.92×10^{8}
172	42.63	5.80	10.80×10^{12}	5.48×10^{8}
174	47.45	8.81	12.30×10^{12}	5.82×10^{8}
176	50.73	8.92	14.56×10^{12}	6.43×10^{8}
178	55.09	14.07	16.95×10^{12}	6.99×10^{8}
180	65.12	25.86	18.11×10^{12}	6.97×10^{8}
182	218.06	23.48	6.82×10^{12}	2.45×10^{8}
184	204.61	18.27	9.16×10^{12}	3.08×10^{8}
186	222.69	21.26	10.59×10^{12}	3.33×10^{8}
188	220.46	22.92	13.45×10^{12}	3.95×10^{8}
190	221.67	24.60	16.80×10^{12}	4.62×10^{8}
192	232.10	27.68	2.01×10^{13}	5.18×10^{8}
194	239.50	29.81	2.44×10^{13}	5.89×10^{8}
196	307.33	38.90	2.38×10^{13}	5.38×10^{8}
198	298.28	55.29	3.07×10^{13}	6.49×10^{8}
200	337.96	73.80	3.39×10^{13}	6.71×10^{8}
202	791.08	113.13	1.80×10^{13}	3.35×10^{8}
204	888.10	95.55	2.01×10^{13}	3.49×10^{8}
206	900.51	61.40	2.47×10^{13}	4.02×10^{8}
208	871.15	80.96	3.17×10^{13}	4.85×10^{8}
210	948.95	114.40	3.63×10^{13}	5.20×10^{8}
212	1021.10	79.65	4.19×10^{13}	5.63×10^{8}
214	1091.83	160.53	4.86×10^{13}	6.14×10^{8}
216	1110.52	146.59	5.93×10^{13}	7.03×10^{8}
218	1250.34	194.58	6.53×10^{13}	7.27×10^{8}
220	1415.05	237.89	7.15×10^{13}	7.47×10^{8}
230	4196.60	812.71	6.93×10^{13}	5.32×10^{8}
240	6409.90	1097.76	12.81×10^{13}	7.25×10^{8}
250	16253.60	2653.25	13.98×10^{13}	5.87×10^{8}

9. H. Cohen and H. W. Lenstra, Jr., *Heuristics on class groups of number fields*, Number Theory, Lecture notes in Math., vol. 1068, Springer-Verlag, New York, 1983, pp. 33–62.
10. S. Hamdy and B. Möller, *Security of cryptosystems based on class groups of imaginary quadratic orders*, Advances in Cryptology - ASIACRYPT 2000, Lecture Notes in Computer Science, vol. 1976, 2000, pp. 234–247.
11. M. J. Jacobson, Jr., *Subexponential class group computation in quadratic orders*, Ph.D. thesis, Technische Universitt Darmstadt, Darmstadt, Germany, 1999.
12. ———, *Computing discrete logarithms in quadratic orders*, Journal of Cryptology **13** (2000), 473–492.
13. M. J. Jacobson, Jr., R. Scheidler, and H. C. Williams, *The efficiency and security of a real quadratic field based key exchange protocol*, Public-Key Cryptography and Computational Number Theory (Warsaw, Poland), de Gruyter, 2001, pp. 89–112.
14. M. J. Jacobson, Jr. and H. C. Williams, *Solving the Pell equation*, CMS Books in Mathematics, Springer-Verlag, 2009, ISBN 978-0-387-84922-5.
15. T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmerman, *Factorization of a 768-bit RSA modulus*, Eprint archive no. 2010/006, 2010.
16. M. Maurer, *Regulator approximation and fundamental unit computation for real quadratic orders*, Ph.D. thesis, Technische Universitt Darmstadt, Darmstadt, Germany, 1999.
17. ———, *Regulator approximation and fundamental unit computation for real-quadratic orders*, Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2000.
18. D. Micciancio and B. Warinschi, *A linear space algorithm for computing the hermite normal form*, ISSAC '01: Proceedings of the 2001 international symposium on Symbolic and algebraic computation (New York, NY, USA), ACM, 2001, pp. 231–236.
19. National Institute of Standards and Technology (NIST), *Recommendation for key management - part 1: General (revised)*, NIST Special Publication 800-57, March, 2007, See: `http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf`.
20. R. Kannan P. Domich and L. Trotter, *Hermite normal form computation using modulo determinant arithmetic*, Math. Oper. Research **12** (1987), 50–59.
21. C. Pernet and W. Stein, *Fast computation of hermite normal forms of random integer matrices*, Journal of Number Theory **In Press, Corrected Proof** (2010), –.
22. A. Storjohann, *Iml*, http://www.cs.uwaterloo.ca/∼astorjoh/iml.html.
23. U. Vollmer, *Asymptotically fast discrete logarithms in quadratic number fields*, Algorithmic Number Theory — ANTS-IV, Lecture Notes in Computer Science, vol. 1838, 2000, pp. 581–594.
24. ———, *An accelerated Buchmann algorithm for regulator computation in real quadratic fields*, Algorithmic Number Theory — ANTS-V, Lecture Notes in Computer Science, vol. 2369, 2002, pp. 148–162.
25. Ulrich Vollmer, *A note on the Hermite basis computation of large integer matrices*, International Symposium on Symbolic and Algebraic Computation, ISSAC '03 (J. Rafael Sendra, ed.), ACM Press, 2003, pp. 255–257.