

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/237456201>

Improvements to the general number field sieve for discrete logarithms in prime fields

Article · April 2003

CITATIONS

31

READS

248

2 authors, including:



[Reynald Lercier](#)

Université de Rennes 1

68 PUBLICATIONS 1,197 CITATIONS

[SEE PROFILE](#)

IMPROVEMENTS TO THE GENERAL NUMBER FIELD SIEVE FOR DISCRETE LOGARITHMS IN PRIME FIELDS. A COMPARISON WITH THE GAUSSIAN INTEGER METHOD

ANTOINE JOUX AND REYNALD LERCIER

ABSTRACT. In this paper, we describe many improvements to the number field sieve. Our main contribution consists of a new way to compute individual logarithms with the number field sieve without solving a very large linear system for each logarithm. We show that, with these improvements, the number field sieve outperforms the gaussian integer method in the hundred digit range. We also illustrate our results by successfully computing discrete logarithms with GNFS in a large prime field.

1. INTRODUCTION

Since their introduction, index calculus techniques have become the methods of choice for factoring large integers and for computing discrete logarithms in finite fields. In the field of integer factorization, the progress have been steady and in recent years the number field sieve (see [26]) has become the leading technique, surpassing its ancestor, the quadratic sieve (see [38]).

For the computation of discrete logarithm in prime fields, where the analog of the quadratic sieve is the gaussian integer method (see [10, 24]), the situation is less clear. Indeed, the number field sieve has already been well studied (see [19, 35, 36, 40]), and thanks to the major theoretical breakthrough of O. Schirokauer, it seems promising. However, a comparison of the two methods on an 85-digit number by D. Weber in 1998 [41] concluded that the gaussian integer method was still leading the way. Several explanations were given for the failure of the number field sieve method, sieving and linear algebra took longer and, worst of all, it was not possible to efficiently compute individual logarithms. Our work in 1998 on the gaussian integer method, which allowed us to compute discrete logarithms in a prime field defined by a 90-digit number (see [23] or section 4.1), seemed to corroborate this result.

In this paper, we show how to overcome all these difficulties, and demonstrate that in the hundred digit range, the number field sieve becomes more efficient than the gaussian integer method.

2. NUMBER THEORETICAL CONSIDERATIONS

Even though the number field sieve is now a well known algorithm (for details, see [28, 29, 5, 3, 41, 19, 35, 4]), some freedom is still possible. Using the degrees of

Received by the editor July 27, 1999 and, in revised form, October 18, 2000.
2000 *Mathematics Subject Classification.* Primary 11T99, Secondary 11Y05.

freedom that were available, we found some innovations that were especially suited to discrete logarithm computations. In this section, we describe two theoretical improvements, concerning the choice of polynomials and the final computation of discrete logarithms through linear algebra, that are essential to the success of the computation described in section 4.2.

As a foreword, let us recall that in order to compute discrete logarithms in a prime field \mathbb{F}_p , the first step is to factor $p - 1$. For any small factor ℓ of $p - 1$, the discrete logarithm modulo ℓ can be found using the Pollig–Hellman and Rho methods. Therefore, in the sequel, we will assume that we are computing the discrete logarithm modulo a large factor q of $p - 1$.

2.1. Choice of the polynomials. As explained in [16], the number field sieve requires two polynomials f_α and f_β with a common root μ . For the algorithm to be efficient, these polynomials should have small coefficients. Several techniques are known to choose them. For numbers having a special form, very good polynomials can be constructed by playing with the special form (for example, see [42, 33, 28, 15]). For general numbers, choosing good polynomials is a difficult and important step. For example, in the recent factorization record [8] of a 140-digit number from the RSA challenge [37], special care was given to the selection of polynomials.

The most commonly known method for building polynomials for the number field sieve is the base m technique [5], where one simply chooses a number m and writes the modulus (n when factoring, p when computing discrete logarithms) in base m as $\sum a_i m^i$. Then $X - m$ and $\sum a_i X^i$ clearly have a common root $\mu = m$. In that case, we get a rational side (corresponding to the degree 1 polynomial) and an algebraic side. Another approach was suggested by Montgomery for factorization (see [16] for a detailed description). His idea was to use lattice reduction to build two quadratic polynomials with a common root. For discrete logarithms, the gaussian integer method [10] uses continued fractions (which can be seen as a special case of lattice reduction), and gives one linear and one quadratic polynomial with a common root. When the gaussian integer method was compared with the number field sieve in [41], the polynomials used in the latter sieve were constructed according to Montgomery’s technique.

We present here a new technique to build good polynomials in the discrete logarithm case. This technique, which does not seem to apply to factoring, produces two polynomials of degree d and $d + 1$. It is in fact a generalization of the gaussian integer method. We choose a polynomial of degree $d + 1$ with extremely small coefficients, and try to find a root of this polynomial modulo p . If no root exists, we discard the polynomial and pick another. Otherwise, we let μ denotes this root, and we search for a polynomial of degree d also having μ as a root. This can be done easily by reducing the following lattice (the basis vectors are in columns and K is some arbitrarily large constant):

$$\begin{pmatrix} K & K(\mu \bmod p) & K(\mu^2 \bmod p) & \cdots & K(\mu^d \bmod p) & Kp \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

Note that this lattice can be somewhat reduced by subtracting multiples of the first columns from the others and by removing the first line and column. Once this is done, we can move the last column into first position, and we are left with:

$$\begin{pmatrix} p & -\mu & -\mu^2 & \cdots & -\mu^d \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

The lattice reduction can then be done using the LLL algorithm (see [27] or [21, 22] for a tutorial). The first vector of the reduced lattice is of the form

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{pmatrix}.$$

It gives us a second polynomial, $\sum_{i=0}^d a_i X^i$, with coefficients of the order of $p^{1/(d+1)}$. When $d = 1$, this is equivalent to the gaussian integer method. In our example, we choose $d = 2$ and thus obtain two polynomials, of degrees 2 and 3 respectively. In particular, this means that there is no rational side in our computation. In the sequel, some technicalities will arise due to this lack (see sections 3.3 and 3.5).

Remark 1. In an information theoretic sense, this choice is optimal since we code p (a $\log p$ bits number) with $d + 1$ numbers of size $\log p/(d + 1)$ plus a couple of extremely small numbers. This can for example be compared with the base m method, which requires $d + 2$ numbers of size $\log p/(d + 1)$, or with Montgomery's method, which requires 6 numbers of size $\log p/4$.

2.2. Completing the equations and finding a logarithm. With the number field sieve, the relations collected while sieving cannot be used as simply as with the gaussian integer method. When factoring, one needs to add what is called characters in order to ensure the feasibility of the square root computation step [11]. In the discrete logarithm, completing the equations is trickier; however, in [35], O. Schirokauer defined easily computed “linear maps” which can efficiently replace characters.

As far as we know, the method applied until now [40, 41, 42] is as follows: after completing the relations coming from the sieving phase on both sides with Schirokauer's maps, we get a linear system whose elements of the kernel modulo q correspond to q -th powers on both sides. Then, one adds in the system an equation related to x , and another related to y . At the end of the linear algebra, one can find a relation of the form $x^{e_x} = y^{e_y} \bmod p$, thus giving $\log_x(y) \bmod q$. Combining this with the results of the Pollig–Hellman and Rho methods, we find the complete value of $\log_x(y)$.

This algorithm has a major drawback outlined in [41]: whenever we want to recompute the logarithm of a new number, we need to restart the linear algebra step from scratch. For this reason, it is still believed that the gaussian integer method is much better suited to discrete logarithm computation, since it is possible to take advantage of a major precomputation in order to compute an individual logarithm in a relatively short time.

We claim here that such an efficient computation of individual logarithms is also possible with the number field sieve, even if the number fields involved in the computation have structures much more complicated than the quadratic number fields involved in the gaussian integer method.

This algorithm has classically two main phases. In the first phase, we precompute logarithms of numerous specific elements of \mathbb{F}_p (cf. section 2.2.1). In the second phase, we use these precomputations to compute the logarithm of any element of \mathbb{F}_p (cf. section 2.2.2).

2.2.1. Linear algebra. In order to clarify the description, we now recall (maybe in a nonclassical manner) some theoretical basic facts about the Number Field Sieve.

Sieving algorithms enable us to find a set \mathcal{L} of “small integer” couples (a, b) such that the principal ideals generated by the algebraic integers $(a + b\alpha)$ and $(a + b\beta)$ are respectively smooth in the ring of integers \mathcal{O}_α and \mathcal{O}_β of two number fields $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$. Of course, as explained at the top of section 2.1, their defining polynomials $f_\alpha(X)$ and $f_\beta(X)$ have a common root μ in \mathbb{F}_p . In other words, we have at the end of the sieving process a number $|S_\alpha| + |S_\beta| + O(1)$ of couples (a, b) such that $a + b\alpha$ (resp. $a + b\beta$) is completely factored over a set S_α (resp. S_β) of prime ideals in \mathcal{O}_α (resp. \mathcal{O}_β) whose norms are smaller than a bound fixed in advance. Moreover, for the sake of simplicity, we assume that q does not divide the class numbers of $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$. This is not really a restriction, since the contrary is extremely improbable. Moreover, this condition can be tested very early during the polynomial selection process, and thus it can be easily avoided.

At first, from the set of $2|S_\alpha| + 2|S_\beta| + O(1)$ equations $(a + b\alpha) = \prod_{\mathfrak{p} \in S_\alpha} \mathfrak{p}^{e_{\mathfrak{p}}}$ and $(a + b\beta) = \prod_{\mathfrak{p} \in S_\beta} \mathfrak{p}^{e_{\mathfrak{p}}}$, we are able to express, thanks to a classical linear algebra step done modulo q on the indexes $e_{\mathfrak{p}}$, each prime ideal \mathfrak{p} as a function of the principal ideals $(a + b\alpha)$ and $(a + b\beta)$. This gives

$$(1) \quad \forall \mathfrak{p} \in S_\alpha, \mathfrak{p} = I_{\mathfrak{p}}^q \prod_{(a,b) \in \mathcal{L}} (a + b\alpha)^{e_{a,b}^{(\mathfrak{p})}} \text{ and } \forall \mathfrak{p} \in S_\beta, \mathfrak{p} = I_{\mathfrak{p}}^q \prod_{(a,b) \in \mathcal{L}} (a + b\beta)^{e_{a,b}^{(\mathfrak{p})}},$$

where the notation $I_{\mathfrak{p}}$ (or later the notation I) means uncomputed ideals of \mathcal{O}_α or \mathcal{O}_β .

Then, there remain $|S_\beta| + O(1)$ and $|S_\alpha| + O(1)$ equations given by

$$(2) \quad I_i^q = \prod_{(a,b) \in \mathcal{L}} (a + b\alpha)^{e_{a,b}^{(i)}} \text{ and } I_j^q = \prod_{(a,b) \in \mathcal{L}} (a + b\beta)^{e_{a,b}^{(j)}},$$

from which we are going to explain how it is possible to compute the discrete logarithm of elements $a + b\mu$ in \mathbb{F}_p .

Classical ideas. For the sake of comprehension, we start with the analysis of an easy case. We mention that this case is interesting for its own sake, since it happens in the right-hand-side number field used in the gaussian integer method and in our 100-digit example (cf. section 4).

Let's assume that the number field $\mathbb{Q}(\beta)$ has a class group of cardinality h equal to 1 and that the structure of the unit group U of $\mathbb{Q}(\beta)$ is known (i.e., a set of fundamental units can be computed). Following [33], it appears that we have no need of Schirokauer's ideas. Since $h = 1$, any ideal is principal; that is to say, for any ideal I in \mathcal{O}_β there exists some algebraic integer δ_I which generates I (i.e., $I = (\delta_I)$). Thus, the $|S_\alpha| + O(1)$ right-hand-side equations of (2) can be rewritten

in terms of ideals as $(\delta_{I_i})^q = \prod_{(a,b) \in \mathcal{L}} (a + b\beta)^{e_{a,b}^{(i)}}$. Then, one has the following algebraic integers equality:

$$(3) \quad \delta_{I_i}^q = \prod_{u \in U} u^{e_u^{(i)}} \prod_{(a,b) \in \mathcal{L}} (a + b\beta)^{e_{a,b}^{(i)}}.$$

Schirokauer's ideas. The number field $\mathbb{Q}(\alpha)$ does not usually satisfy the previous conditions. One can apply Schirokauer's ideas instead. In our case, Schirokauer's ideas consist in computing $T = O(1)$ linear maps Λ defined from $\mathbb{Q}(\alpha)$ to $\mathbb{Z}/q\mathbb{Z}$ for any equations given by the left-hand side of (2).

Precisely, the $|S_\beta| + O(1)$ equations $I_j^q = \prod_{(a,b) \in \mathcal{L}} (a + b\alpha)^{e_{a,b}^{(j)}}$ yield $|S_\beta| + T$ vectors $(\lambda_1, \dots, \lambda_T)$ defined over $\mathbb{Z}/q\mathbb{Z}$. With a little linear arithmetic modulo q , it is easy to compute some suitable combinations of the last T equations with any of the first $|S_\beta|$ equations to obtain $|S_\beta|$ equations whose “map vectors” are equal to $(0, \dots, 0)$. Then, Schirokauer's results state that, if Leopoldt's conjecture is true, these equations also hold for algebraic integers, thus leading to combined equations of the form

$$(4) \quad \Delta_k^q = \prod_{(a,b) \in \mathcal{L}} (a + b\beta)^{E_{a,b}^{(k)}}.$$

Linear inversion. At this step, it is now licit to classically apply the maps ϕ_α (resp. ϕ_β) defined from \mathbb{Q}_α (resp. \mathbb{Q}_β) to \mathbb{F}_p by $\alpha \longrightarrow \mu$ (resp. $\beta \longrightarrow \mu$), to obtain from equations (3) and (4), $|S_\alpha| + |S_\beta| + O(1)$ equations given by

$$\begin{aligned} 0 &= \sum_{u \in U} e_u^{(i)} \log_x \phi_\alpha(u) + \sum_{(a,b) \in \mathcal{L}} e_{a,b}^{(i)} \log_x (a + b\mu) \bmod q, \\ 0 &= \sum_{(a,b) \in \mathcal{L}} E_{a,b}^{(k)} \log_x (a + b\mu) \bmod q. \end{aligned}$$

A classical (but huge) linear inversion finally enables us to compute $\log_x(a + b\mu)$ for any $(a, b) \in \mathcal{L}$.

2.2.2. Finding a logarithm. To compute the discrete logarithm of any integer y , it is classical with the gaussian integer method to search for two sets \mathcal{N} and \mathcal{D} of “small” integers such that $y = \prod_{\ell \in \mathcal{N}} \ell^{e_\ell} / \prod_{\ell \in \mathcal{D}} \ell^{e_\ell} \bmod p$. In our case, we slightly restricted the choice for \mathcal{N} and \mathcal{D} , because we only admit primes ℓ which completely split in \mathbb{Q}_α or \mathbb{Q}_β . We explain how we can efficiently find such an equality in section 3.5.

Let's assume without loss of generality that ℓ splits in \mathbb{Q}_α . One can substitute the ideals \mathfrak{p} which divide (p) as an expression in $(a + b\alpha)$ thanks to equation (1). This yields

$$(\ell) = \prod_{\mathfrak{p} | (\ell)} I_{\mathfrak{p}}^q \prod_{(a,b) \in \mathcal{L}} (a + b\alpha)^{e_{a,b}}.$$

As was previously done, it is still possible to compute a map vector associated to this equation from which one can combine $O(1)$ equations written on the left-hand side of (2) in order to finally obtain $\ell = \prod_{(a,b) \in \mathcal{L}} (a + b\mu)^{e_{a,b}}$. Finding $\log_x \ell$ from the precomputed values $\log_x(a + b\mu)$ for any $(a, b) \in \mathcal{L}$ is now obvious.

3. ALGORITHMIC AND IMPLEMENTATION CHOICES

3.1. Choice and precomputation of the factor bases. In many recent papers on the number field sieve, the double large prime and even the quadruple large prime (see [14, 41, 42, 8, 6, 7]) variations are used. Here we chose a contrary approach and did not use large primes at all. Instead we preferred to use large factor bases. According to previous articles, this leads to longer sieving times; however, this choice has several advantages. First, with this variation, the bookkeeping required to count the number of usable relations and predict the remaining runtime is much easier. Second, since all relations are full, the initial linear system is less dense and structured gaussian elimination can be applied more efficiently. The anonymous referee did remark that in certain cases using large primes leads to a smaller linear system after these primes have been removed. However, for our choice of parameters, we did not encounter this phenomenon. Third, computing individual logarithms is easier when the factor bases are large.

In truth, choosing the optimal sizes of the factor bases and the right number of large primes is very difficult, and our choice is certainly not optimal. Many more experiments will be needed for the discrete logarithm case before finding a rule giving the optimal choices.

For both polynomials f_α and f_β we choose a smoothness bound, respectively B_α and B_β . Then for each prime under the smoothness bound B_α , we factor it into prime ideals in the ring of integers of $\mathbb{Q}(\alpha)$, where α is a complex root of f_α . We keep the prime ideals of degree 1 and discard the rest. Likewise we factor the primes under B_β in $\mathbb{Q}(\beta)$, where β is a complex root of f_β . Finally, we store the representation of each ideal as a reduced basis of the two-dimensional lattice containing pairs (a, b) such that the prime ideal divides either $(a + b\alpha)$ (for ideals in $\mathbb{Q}(\alpha)$) or $(a + b\beta)$.

3.2. Sieving. Sieving is done here following a now traditional “sieving by vectors” with special- \mathfrak{q} technique [34]. Using special- \mathfrak{q} means that in order to get sufficiently many relations, we sieve many independent sets of values for (a, b) . Each set is defined by a prime ideal \mathfrak{q} called the special- \mathfrak{q} , and contains pairs (a, b) such that \mathfrak{q} divides $(a + b\alpha)$ (assuming that \mathfrak{q} is on the f_α side). If \vec{u} and \vec{v} form a basis of the lattice corresponding to \mathfrak{q} , then (a, b) can be written as $k_u\vec{u} + k_v\vec{v}$. Then we can with simple linear algebra send the lattice corresponding to any small prime ideal from the (a, b) representation to the (k_u, k_v) representation. Sieving is then done in a big rectangle in the (k_u, k_v) space by successively marking the points on each of the small prime lattices. In this rectangle positions corresponding to a pair (k_u, k_v) with two even coordinates can be omitted (see [18]).

After selecting good (k_u, k_v) candidates, we can check efficiently that the corresponding values $(a + b\alpha)$ and $(a + b\beta)$ are indeed smooth by combining trial division and a second sieving step (see [18]). This produces an algebraic relation between our prime ideals.

3.3. Adding Schirokauer’s maps. As explained in section 2.2, we cannot directly use the relations found in section 3.2. The reason for this is that the ring of integer of $\mathbb{Q}(\alpha)$ may have uncomputable units or a class number different from one. Assuming the truth of Leopoldt’s conjecture, the maps described in [35] can remove these obstructions. For a degree d polynomial, d such maps are required. With two polynomials of respective degree d and $d+1$, $2d+1$ such maps are needed. However,

the polynomial of degree $d + 1$ has extremely small coefficients and the resulting number field may have a much simpler structure. In particular, in the example of section 4.2, we have $f_\beta(X) = X^3 + 2$, which leads to a number field $\mathbb{Q}(\beta)$ with a simple unit group generated by $u = \beta + 1$. In that case, it is preferable to work with integers in $\mathbb{Q}(\beta)$ generating the prime ideals, and to split $a + b\beta$ into a product of such numbers times a power of u . This approach was first proposed by Pollard in [33] (cf. section 2.2.1).

In section 2.2, for theoretical clarity, we first compute any prime ideal \mathfrak{p} defined in $\mathbb{Q}(\alpha)$ or in $\mathbb{Q}(\beta)$ as a function of principal ideals $(a + b\alpha)$ or $(a + b\beta)$ (cf. equations (1)) before solving a huge system whose unknowns are logarithms of elements $a + b\mu$ in \mathbb{F}_p . In practice, one can bypass most of this stuff by defining “virtual” logarithms of prime ideals \mathfrak{p} , denoted $\log_x \mathfrak{p}$. Such quantities $\log_x \mathfrak{p}$ can be seen as the logarithm of the algebraic integer part in the right-hand sides of equations (2). With this rough definition, it turns out that to solve the linear systems defined in section 2.2, it is enough to solve the system composed of the following $|S_\alpha| + |S_\beta| + O(1)$ equations:

$$(5) \quad \sum_{\mathfrak{p} \in S_\alpha} e_{\mathfrak{p}} \log_x \mathfrak{p} + \sum_{O(1) \text{ maps } \Lambda} \lambda_\Lambda \log_x \Lambda = \sum_{\mathfrak{p} \in S_\beta} e_{\mathfrak{p}} \log_x \mathfrak{p} + \sum_{\text{units } u} e_u \log_x u.$$

Remark. In equation (5), the coefficients $e_{\mathfrak{p}}$ and e_u are very small, since most of the time they are equal to ± 1 . The coefficients λ are of size p . Fortunately, the number of maps is small.

3.4. Linear algebra. The linear algebra consists of two substeps,¹ the structured gaussian elimination and an iterative solver based on the Lanczos algorithm. The original linear system we need to solve is very sparse and contains mostly small entries (many of which are 1 and -1); it also contains a couple of columns of big numbers corresponding to Schirokauer maps. We first reduce it using the structured gaussian elimination, and then use the Lanczos algorithm. Note that, when factoring a large integer, the linear algebra is done modulo 2, and block Lanczos algorithms are so efficient [31] that gaussian elimination is no longer used (e.g., see [8]). Moreover, since the block Lanczos algorithm returns several vectors in the kernel of the linear system, it is even possible to omit the smallest primes and the characters from the matrix to reduce its weight. For example, after deleting about 50 rows, Block Lanczos yields almost 64 vectors on a 64-bit machine. Then the missing primes and the characters are added back and can be taken care of by gaussian elimination on a small matrix (see [7]).

However, to compute a discrete logarithm, the linear algebra is done modulo q , and a reduction of the linear system is absolutely needed. Moreover, the additional columns corresponding to Schirokauer’s maps contribute a lot to the running time of the Lanczos algorithm. This is due to the simple fact that during the Lanczos algorithm we perform many matrix/vector multiplications modulo q . In these multiplications, columns of large numbers require us to multiply large numbers modulo q , instead of simply adding large numbers or multiplying a large number by a small one. These multiplications between large numbers add themselves to other multiplications of large numbers already present in the Lanczos computation

¹In previous work there often was a preliminary step to remove large primes from the equations using cycles (see [13]). However, since we do not use large primes, we can skip this step.

(in scalar products and scalar/vector multiplications), which were already nonnegligible. For this reason, when using the number field sieve, it is more important to decrease the size of the linear system than when working with the gaussian integer method, even if the system becomes less sparse.

Thus, structured gaussian elimination, first introduced by B. Lamacchia and A. Odlyzko in 1991 [25], is very important. Its goal is to reduce a large and sparse linear system with many more equations than unknowns to a smaller and still quite sparse system. We won't recall here the original method of gaussian elimination, but we will now describe a new technique that leads to an even better reduction in the case of very large systems. This new technique, combined with our choice of having no large prime (and thus a sparser initial system), accounts for the extreme size reduction of our system during the gaussian elimination step.

3.4.1. Description of the gaussian elimination. The basic idea of our method consists, for every potential pivoting number (any 1 or -1 entry in the linear system), in estimating the weight change in the matrix if this pivoting is done. We then select the best possible way of pivoting (according to this estimate) and apply it. Afterward, we greedily repeat the operation again and again. If at some point our allocated memory is full, we remove some of the extra equations, starting with the heaviest ones. When there are only a few extra equations left, we terminate the algorithm.

The difficult point in this algorithm is to estimate the weight changes and to quickly select the best pivot. For any 1 or -1 entry in the system, we denote by l the weight of the line it is in, and by c the weight of the column. Then the weight change when pivoting this entry can be estimated as $(l-2)(c-2)-2$. This estimate is computed under the assumption that when the pivoting line is combined with another one, they have no common variable apart from the pivot. Thus $l-1$ variables are added to each line, and of course 1 is removed. Moreover, the pivoting line can be deleted once used, thus removing l entries. The total change is thus

$$(c-1)(l-2)-l = (l-2)(c-2)-2.$$

The assumption we made is very inaccurate, but it gives a sufficiently good estimate for our purpose. We are now faced with the problem of efficiently finding the entry with the smallest possible value of $(l-2)(c-2)$. Of course, recomputing the estimate from scratch for each entry after every pivoting operation is not an efficient solution. It would be better to proceed incrementally. Luckily, during any pivoting, only a few variables and a few equations are modified, and we can hope to devise a technique for recomputing only the few values that need an update.

3.4.2. Algorithmic details. The algorithm we came up with works by operating on two copies of the linear system. The first one is a sparse representation by line and the second a sparse representation by column. Having these two representations, we can quickly get l and c for any given entry in the matrix. We also keep for each unknown a pointer to the best occurrence of the unknown, i.e., the number of the smallest equation containing this unknown (with a 1 or -1 coefficient), together with the corresponding value of $(l-2)(c-2)$. A balanced binary tree is then built to keep the unknowns sorted according to these values.

Thus, selecting the best candidate for pivoting is done by extracting the first element in the binary tree; the pivoting itself and the bookkeeping needed to update the values is straightforward. However, when updating the value associated with

a given unknown, we should carefully remove it from the binary tree, update the value and put it back in the tree (usually in a different position). To improve the efficiency of the implementation, we don't keep variables appearing too often (say more than 1000 times), neither in the tree, nor in the column representation of the matrix. This reduces both the time needed for the bookkeeping and the memory needed for storing the matrix.

In order to be able to quickly remove the heaviest equations when the memory is almost full, we built a second balanced binary tree to keep the equations sorted by weight.

Without this new structured gaussian elimination technique, we would not have been able to solve the linear systems in the discrete logarithm computations presented in section 4. Moreover, using a careful heuristic, it is possible to further improve this new gaussian elimination technique (see [39]).

Remark 2. For ease of implementation, it might be feasible to replace the balanced trees by two families of linked lists respectively indexed by the values of $(l-2)(c-2)$ and by the weight of the equations. However, we have not tested this alternative data structures.

3.5. Computing individual logarithms. We claimed in section 2.2 that it is possible with the number field sieve to compute individual logarithms in a final step after the rest of the computation is performed. We now explain precisely how this can be done. We proceed in two parts. In a first part, we describe our original idea, since this method was used for the computations given in section 4 (cf. section 3.5.1). Then, in a second part, we show how this can be slightly improved, following valuable ideas due to an anonymous referee (cf. section 3.5.2).

3.5.1. A first attempt. In this section, the main idea is to search for a representation of the number whose logarithm is required, involving only small and medium primes whose logarithms are known or can be found. Clearly the logarithm of a small prime is known if and only if this prime splits into a product of prime ideals of degree one in either of our number fields. About half of the small primes split in our quadratic field; thus, when taking in account the degree 3 polynomial, slightly more than half of the small primes can be used in our representation. A classical technique to find a good representation of a number x (see [24]) was to write $x \equiv A/B \pmod{p}$ with A and B of size \sqrt{p} , to trial divide A and B by small primes and then to check if the representation was good enough. In our case, this approach is completely impractical, because bad small primes will appear most of the time and the number of trials needed will be much too large.

To avoid this pitfall, we propose a new, faster, technique to find good representations. The main idea is that, since we are searching for a smooth representation (involving good primes only), it seems natural to try a sieving algorithm. Thus, we use some kind of sieving to write x (the number whose logarithm is wanted) as A/B , where both A and B are smooth. Since not all values of x admit a good representation, we also use the now classical trick (also in [24]) of replacing x by $z = s^i x$, where s is the largest small prime whose logarithms can be computed from the factor bases, and where i is incremented whenever we need a new value for z . Since the logarithm of s is known at this step, computing the logarithm of z clearly gives the logarithm of x .

Before sieving, we start by reducing the following lattice:

$$\begin{pmatrix} z & p \\ 1 & 0 \end{pmatrix}.$$

After the lattice reduction, we have a new basis

$$\begin{pmatrix} A_1 & A_2 \\ B_1 & B_2 \end{pmatrix},$$

where A_1 , B_1 , A_2 and B_2 are usually all of size \sqrt{p} . Sometimes A_2 and B_2 can be too large; this occurs when A_1 and B_1 are extremely small. In that case, we simply change to the next value of z (e.g., $s^{i+1}x$). It is easy to check that

$$z \equiv \frac{A_1}{B_1} \equiv \frac{A_2}{B_2} \pmod{p}.$$

Thus for any integers k_1 and k_2 , we have

$$z \equiv \frac{k_1 A_1 + k_2 A_2}{k_1 B_1 + k_2 B_2} \pmod{p}.$$

Finding pairs (k_1, k_2) with both $k_1 A_1 + k_2 A_2$ and $k_1 B_1 + k_2 B_2$ smooth (using good primes only) can now be done by sieving. In fact, we can use the same “sieving by vectors” method as described in section 3.2, and sieve on both $k_1 A_1 + k_2 A_2$ and $k_1 B_1 + k_2 B_2$. We use the factor base made of good primes and allow two large primes on each side. Moreover, the large primes must also be good primes. For practical reasons, we limited ourselves to large primes which split on the quadratic side. Then the large primes can be split in prime ideals which can be treated as special- \mathfrak{q} as described in section 3.2. This leads to a relation giving the “logarithm” of this prime ideal. Collecting all these values, we can then compute the logarithms of z and x .

Of course, the larger the factor base, the higher our probability of finding a good relation. However, we do not know how to make a precise theoretical analysis of this individual logarithm computation. In order to enhance the probability of success and speed up the computation, we simply took a large factor base while sieving.

3.5.2. An improvement. One drawback of the method described in section 3.5.1 is that we can only use primes which split completely in an at least one factor base.

Since the way primes split in a number field mainly depends on the Galois group of the defining polynomial, a first attempt to improve things is to deliberately set $f_\beta(X)$ such that its Galois group is cyclic. Then, the only thing that we can expect about $f_\alpha(X)$ is that its Galois group is symmetric. So, for $d = 1$, any prime splits; for $d = 2$, a prime splits with probability $1/3$ in \mathbb{Q}_α and probability $1/2$ in \mathbb{Q}_β ; the combined probability is $2/3$. For $d = 3$, a prime splits with probability $1/6$ in \mathbb{Q}_α and probability $1/4$ in \mathbb{Q}_β ; the combined probability is $3/8$. And so on. Clearly, as d grows, the combined probability of splitting decreases.

Nevertheless, one can slightly improve this construction as follows, when $d + 1$ is prime. Let $f_\beta(X)$ be a polynomial with small coefficients and with a root modulo p such that its Galois group is of order $d + 1$ (i.e., \mathbb{Q}_β is a normal extension of \mathbb{Q}). Looking for such a polynomial $f_\beta(X)$ shouldn't be hard. For instance, $f_\beta(X) = X^3 + 2X^2 - 5X + 1$ works. More generally, any degree-3 polynomial with a discriminant equal to a square might be a candidate.

Given z as in section 3.5.1, one may try to find a quotient

$$z = \frac{a_0 + a_1\mu + \cdots + a_d\mu^d}{b_0 + b_1\mu + \cdots + b_d\mu^d} \bmod p,$$

where, using lattice basis reduction, a_0, a_1, \dots, a_d and b_0, b_1, \dots, b_d are integers of size $O(p^{1/(2d+2)})$. In this case, it is not difficult to show that the principal ideals $(a_0 + a_1\beta + \cdots + a_d\beta^d)$ and $(b_0 + b_1\beta + \cdots + b_d\beta^d)$ split into first-degree ideals in \mathbb{Q}_β , possibly including ideals whose norm divides the discriminant of $f_\beta(X)$, except for rational integers dividing $\gcd(a_0, a_1, \dots, a_d)$ or $\gcd(b_0, b_1, \dots, b_d)$.

The benefit to sieving over this, rather than as described in section 3.5.1, is that all factors dividing a norm, except those dividing $\gcd(a_0, a_1, \dots, a_d)$ or $\gcd(b_0, b_1, \dots, b_d)$, will be in the factor base of \mathbb{Q}_β .

4. EXAMPLES

We now describe two discrete logarithm computations, one with the number field sieve on a 100-digit number, the other with the gaussian integer method on a 90-digit number. Extrapolating this second computation to 100-digit, we show that for numbers of this size, the number field sieve becomes the most efficient method known.

We chose our example primes by considering numbers of the form $p = \lfloor 10^{D-1}\pi \rfloor + O$, where D is the number of digits we want and O is the smallest offset such that p and $q = (p-1)/2$ are both primes. For $D = 90$ we have $O = 156137$ and for $D = 100$, $O = 3932$. We then let $y = \lfloor 10^{D-1}e \rfloor$ and try to compute the logarithms of $y, y+1, y+2$ and $y+3$. The basis of the logarithm is chosen among the small primes. We thus had:

$$\begin{aligned} p_{90} &= \lfloor 10^{89}\pi \rfloor + 156137, & p_{100} &= \lfloor 10^{99}\pi \rfloor + 3932, \\ y_{90} &= \lfloor 10^{89}e \rfloor, & \text{and } y_{100} &= \lfloor 10^{99}e \rfloor. \end{aligned}$$

4.1. Logarithms in $\mathbb{F}_{p_{90}}$. In this case, we chose the two following polynomials with the gaussian integer method:

$$\begin{aligned} f_\beta(X) &= X^2 + 2, \\ f_\alpha(X) &= 248748997517243504330966956058992943413697827X \\ &\quad + 436356663553236108573801834571320046705446681. \end{aligned}$$

The smoothness bounds were $B_\alpha = 1299709$ and $B_\beta = 350377$, and one large prime was permitted on each side. The large prime limit was $B = 10^9$. According to the presence of large primes, the equations are divided in four classes, FF, FP, PF and PP, where F stands for Full and P for Partial.

After a total CPU time of 4 months on one Pentium Pro 180MHz (we had a network of 4 such PCs running for one month) or roughly 60 MIPS-year, we had the following numbers of equations :

FF	FP	PF	PP	Total
83519	235921	1377187	4966470	6663097

In order to solve this linear system produced by the sieving phase, we first removed all useless equations, i.e., equations involving a variable not seen in another equation. This step was applied repeatedly until each variable was seen at least twice. After that, 976062 equations involving 674564 unknowns remained. Then, we did not try to combine partial equations into full or to do any other preprocessing,

but directly started the structured gaussian elimination phase. In this case, it reduced the linear system to 61136 equations in 61036 unknowns with 5683954 nonnull entries (i.e., an average weight per equation of 93).

Finally, we solved the remaining system using Lanczos' algorithm. It took three weeks on a single 180MHz machine, and yielded the logarithms of our factor bases elements, for instance

$$\begin{aligned}\log_2(3) &= 19748300961167147793248409109777288916563744188 \\ &\quad 3405533161063698387711654453959032453019844, \\ \log_2(5) &= 14911991564240187116623990082296104898497606799 \\ &\quad 3569475849547058535625211933991916749097892, \\ \log_2(7) &= 20534713616438695604297290160284536202786764235 \\ &\quad 9069620025634227501709422532443486030114180.\end{aligned}$$

The final step of the computation was to find the logarithm of our chosen integers. For instance, for y we found that

$$\begin{aligned}1299709^{92}y &\equiv \frac{A}{B} \pmod{p}, \text{ where} \\ A &= 7 \cdot 167 \cdot 347 \cdot 421 \cdot 1049 \cdot 3067 \cdot 3079 \cdot 3319 \cdot 174077 \cdot \\ &\quad 293263 \cdot 38174761 \cdot 82044163, \\ B &= 5^2 \cdot 17 \cdot 1187 \cdot 1877 \cdot 2039 \cdot 3019 \cdot 110863 \cdot 207589 \cdot \\ &\quad 2903639 \cdot 5397737 \cdot 1064068253.\end{aligned}$$

Most of the primes involved in A and B are in the rational factor base; thus their logarithms are known. For the few factors outside of the factor base, we applied the lattice sieve using these factors as special- q . After a few additions and subtractions we found that

$$\begin{aligned}\log_2(y) &= 17671380721142169627320482340716202723020579 \\ &\quad 52449914157493844716677918658538374188101093.\end{aligned}$$

For the remaining numbers we had

$$\begin{aligned}\log_2(y+1) &= 3116041987058269748820788091978682382044912 \\ &\quad 0001421617617058468654271221802926927230033421, \\ \log_2(y+2) &= 3089883293350445253338277649145014072371680 \\ &\quad 34577534227927033783999866774252739278678837301, \\ \log_2(y+3) &= 6580688800278838010371298688366325318718350 \\ &\quad 5405451188935055113209887949364255134815297846.\end{aligned}$$

4.2. Logarithms in $\mathbb{F}_{p_{100}}$. In this case, we chose the two following polynomials as in section 2.1:

$$\begin{aligned}f_\beta(X) &= X^3 + 2, \\ f_\alpha(X) &= 289054697525386277139299623663612X^2 + \\ &\quad 347505963979820999503620050239250X + \\ &\quad 1325706851630023730078724403330583.\end{aligned}$$

The smoothness bounds were $B_\beta = 2750161$ and $B_\alpha = 8960467$.

After an estimated CPU time of 8 months on one Pentium II 450MHz (we had a mixed network of PCs), or roughly 300 MIPS-year, we had obtained 2900000 equations. Some trial sieving showed that with the gaussian integer method, sieving would have take at least 12 months.

We added two Schirokauer maps on the f_α side and completed the f_β side taking the unit $1 + \beta$ into account. After removing the useless equations, we had 2592659 equations in 816761 unknowns. The gaussian elimination resulted in a 172049 equations in 171061 unknowns with 21747682 nonnull entries (about 126 entries per line) in less than a day.

Finding a solution of this linear system with Lanczos took 20 days with a parallelized implementation on a quadri-processor DEC alpha 500MHz. With this we had logarithms for good small primes:

$$\begin{aligned}\log_{67}(2) &= 15231699951693893676192894808039900459829546515301 \\ &\quad 17535489480512018192128533920122634532977309738960, \\ \log_{67}(5) &= 19377553728171823105413539921382679272133888091020 \\ &\quad 12017702986213046313481150182975740143065448645802, \\ \log_{67}(11) &= 5992829699536965027860245797180613267248544297257 \\ &\quad 60682869489773252521889486038350714114278971866216.\end{aligned}$$

Finding the logarithm of y was done by writing $8960453^{48}y \equiv -A/B \pmod{p}$ with

$$\begin{aligned}A &= 2^4 \cdot 3361 \cdot 27107 \cdot 3182657 \cdot \\ &\quad 4467119 \cdot 11770463 \cdot 32918983 \cdot 6211385489629777 \\ B &= 613 \cdot 11317 \cdot 16427 \cdot 257473 \cdot \\ &\quad 1188587 \cdot 95157851 \cdot 124086068723 \cdot 293844018211.\end{aligned}$$

This allowed us to compute

$$\begin{aligned}\log_{67}(y) &= 12067222770482298529755135197701403621450790321155 \\ &\quad 79252791899015195684040750444570690623611190553071.\end{aligned}$$

Similarly, we found

$$\begin{aligned}\log_{67}(y+1) &= 13668896526425908269754660718655208207513197302740 \\ &\quad 77643314745334860483809586311739213019685404555432, \\ \log_{67}(y+2) &= 64208948866172239171801490564368810607474232613210 \\ &\quad 9138842932332557858509054461341184976767489934161, \\ \log_{67}(y+3) &= 21151393143412908847405305480189192355314404184028 \\ &\quad 59877706713640788528101895760590714258670606466323.\end{aligned}$$

On average, this final step took about a day on a single machine.

5. CONCLUSION

In this paper, we described improvements to the number field sieve for the discrete logarithm problem. With these improvements, we computed discrete logarithms in a prime field defined by a 100-digit number with no special form, and showed that GNFS now beats the gaussian integer method for discrete logarithms.

We believe that some of the improvements described here could also be used for factoring integers or computing discrete logarithms in \mathbb{F}_{2^n} [20].

REFERENCES

1. L.M. Adleman, *Factoring numbers using singular integers*, Proceedings 23rd Annual ACM Symposium on Theory of Computing (STOC), 1991, pp. 64–71.
2. D. Atkins, M. Graff, A. K. Lenstra, and P. C. Leyland, *The magic words are squeamish ossifrage*, Advances in Cryptology — ASIACRYPT'94, Lecture Notes in Computer Science, vol. 917, Springer-Verlag, 1995, pp. 265–277. MR **97b**:94019
3. D. J. Bernstein and A.K. Lenstra, *A general number field sieve implementation*, Springer-Verlag, 1993, pp. 103–126. CMP 95:09
4. J. Buchmann, J. Loho, and J. Zayer, *An implementation of the general number field sieve*, Advances in Cryptology — CRYPTO'93, Lecture Notes in Computer Science, vol. 773, 1994, pp. 159–165. MR **95e**:11132
5. J.P. Buhler, Jr. H.W. Lenstra, and Carl Pomerance, *Factoring integer with the number field sieve*, Springer-Verlag, 1993, pp. 50–94 (see [26]). CMP 95:09
6. S. Cavallar, *Strategies in filtering in the number field sieve*, Proceedings of the ANTS-IV conference (W. Bosma, ed.), Lecture Notes in Computer Science, vol. 1838, Springer-Verlag, 2000, pp. 209–231. MR **2002h**:11139
7. S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, , P.L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillermin, P. Leyland, J. Marchand, F. Morain, A. Muffet, C. Putman, C. Putman, and P. Zimmerman, *Factorization of a 512-bit RSA modulus*, Advances in Cryptology — EUROCRYPT'2000 (B. Preneel, ed.), Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, 2000, pp. 1–18.
8. Stefania Cavallar, Bruce Dodson, Arjen Lenstra, Paul Leyland, Walter Lioen, Peter Montgomery, Brian Murphy, Herman te Riele, and Paul Zimmerman, *Factorization of RSA-140*, <http://listserv.nodak.edu/archives/nmbrthry.html> — February, 1999.
9. D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two*, IEEE transactions on information theory **IT-30** (1984), no. 4, 587–594. MR **85h**:65041
10. D. Coppersmith, A. Odlyzko, and R. Schroppel, *Discrete logarithms in \mathbb{F}_p* , Algorithmica **1** (1986), 1–15. MR **87g**:11167
11. J.-M. Couveignes, *Computing a square root for the number field sieve*, Springer-Verlag, 1993, pp. 95–102 (see [26]). CMP 95:09
12. J. Cowie, B. Dodson, R. M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer, *A world wide number field sieve factoring record: On to 512 bits*, Advances in Cryptology — ASIACRYPT'96 (K. Kim and T. Matsumoto, eds.), Lecture Notes in Computer Science, vol. 1163, Springer-Verlag, 1996, pp. 382–394. CMP 98:05
13. Th. Denny and V. Müller, *On the reduction of composed relations from the number field sieve*, Proceedings of the ANTS-II conference (H. Cohen, ed.), Lecture Notes in Computer Science, vol. 1122, Springer-Verlag, 1996, pp. 75–90. MR **98k**:11184
14. B. Dodson and A. K. Lenstra, *NFS with four large primes: an explosive experiment*, Advances in Cryptology — CRYPTO'95, Lecture Notes in Computer Science, vol. 963, Springer-Verlag, 1995, pp. 372–385. MR **98d**:11156
15. R. M. Elkenbracht-Huizing, P. Montgomery, R. Silverman, R. K. Wackerbarth, and Jr. S. S. Wagstaff, *The number field sieve on many computers*, Fifth Conference of the Canadian Number Theory Association (Providence RI), CRM Proc. Lecture Notes, no. 19, Amer. Math. Soc., 1999. MR **2000e**:11157
16. R.M. Elkenbracht-Huizing, *An implementation of the number field sieve*, Tech. Report NM-R9511, CWI, 1995. Experiment. Math. **5** (1996), 231–253. MR **98a**:11182
17. ———, *A multiple polynomial general number field sieve*, Proceedings of the ANTS-II conference (H. Cohen, ed.), Lecture Notes in Computer Science, vol. 1122, Springer-Verlag, 1996. MR **98g**:11142
18. R. Golliver, A. K. Lenstra, and K. McCurley, *Lattice sieving and trial division*, Proceedings of the ANTS-I conference, Lecture Notes in Computer Science, vol. 877, Springer-Verlag, 1994, pp. 18–27. MR **96a**:11142
19. D. Gordon, *Discrete logarithms in \mathbb{F}_p using the number field sieve*, SIAM J. Discrete Math **6** (1993), 124–138. MR **94d**:11104

20. D. Gordon and K. McCurley, *Massively parallel computation of discrete logarithms*, Advances in Cryptology — CRYPTO'92, Lecture Notes in Computer Science, vol. 740, Springer-Verlag, 1993, pp. 312–323.
21. A. Joux, *La réduction de réseaux en cryptographie*, Ph.D. thesis, Ecole Polytechnique, Palaiseau, France, 1993.
22. A. Joux and J. Stern, *Lattice reduction: A toolbox for the cryptanalyst*, J. Cryptology **11** (1998), 161–185. MR **99c**:94031
23. Antoine Joux and Reynald Lercier, *Discrete logarithms in $GF(p)$* , <http://listerv.nodak.edu/archives/nmbrthry.html> — May, 1998.
24. B. A. LaMacchia and A. M. Odlyzko, *Computation of discrete logarithms in prime fields*, Designs, Codes and Cryptography **1** (1991), 47–62. MR **92j**:11148
25. ———, *Solving large sparse linear systems over finite fields*, Advances in Cryptology — CRYPTO'90, Lecture Notes in Computer Science, vol. 537, Springer-Verlag, 1991, pp. 109–133.
26. A. K. Lenstra and H. W. Lenstra, Jr. (eds.), *The development of the number field sieve*, Lecture Notes in Mathematics, vol. 1554, Springer-Verlag, 1993. MR **96m**:11116
27. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Mathematische Ann. **261** (1982), 513–534. MR **84a**:12002
28. A.K. Lenstra, Jr. H.W. Lenstra, M.S. Manasse, and J.M. Pollard, *The factorization of the ninth fermat number*, Math. Comp. **61** (1993). MR **93k**:11116
29. ———, *The number field sieve*, Springer-Verlag, 1993, pp. 11–42 (see [26]). CMP 95:09
30. K. McCurley, *The discrete logarithm problem*, Proc. Symp. in Applied Mathematics, Cryptology and Computational Number Theory, no. 42, 1990, pp. 49–74. MR **92d**:11133
31. P. L. Montgomery, *A block Lanczos algorithm for finding dependencies over \mathbb{F}_2* , Advances in Cryptology — EUROCRYPT'95, Lecture Notes in Computer Science, vol. 921, Springer-Verlag, 1995, pp. 106–120. MR **97c**:11115
32. A. M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in Cryptology — EUROCRYPT'84 (T. Beth, N. Cot, and I. Ingemarsson, eds.), Lecture Notes in Computer Science, vol. 209, Springer-Verlag, 1985, pp. 224–314. MR **87g**:11022
33. J.M. Pollard, *Factoring with cubic integer*, Springer-Verlag, 1993, pp. 4–10 (see [26]). CMP 95:09
34. ———, *The lattice sieve*, Springer-Verlag, 1993, pp. 43–49. CMP 95:09
35. O. Schirokauer, *Discrete logarithms and local units*, Phil. Trans. R. Soc. Lond. A **345** (1993), 409–423. MR **95c**:11156
36. O. Schirokauer, D. Weber, and Th. Denny, *Discrete logarithms: the effectiveness of the index calculus method*, Proceedings of the ANTS-II conference (H. Cohen, ed.), Lecture Notes in Computer Science, vol. 1122, Springer-Verlag, 1996. MR **98i**:11109
37. RSA Data Security, *The RSA factoring challenge*, <http://www.rsa.com/rsalabs/html/factoring.html>.
38. R. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987), 329–340. MR **88c**:11079
39. F. Valette, *Algèbre linéaire pour le logarithme discret*, Master's thesis, ENSTA, 1999, Stage de fin d'étude et de DEA.
40. D. Weber, *Computing discrete logarithms with the number field sieve*, Proceedings of the ANTS-II conference, Lecture Notes in Computer Science, vol. 1122, Springer-Verlag, 1996. MR **98k**:11186
41. ———, *Computing discrete logarithms with quadratic number rings*, Advances in Cryptology — EUROCRYPT'98 (K. Nyberg, ed.), Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, 1998, pp. 171–183. CMP 2000:07
42. D. Weber and Th. Denny, *The solution of McCurley's discrete log challenge*, Advances in Cryptology — CRYPTO'98 (H. Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1462, Springer-Verlag, 1998, pp. 458–471. MR **99i**:94057

DCSSI, 18 RUE DU DR. ZAMENHOFF, F-92131 ISSY-LES-MOULINEAUX, FRANCE
E-mail address: Antoine.Joux@m4x.org

CELAR, ROUTE DE LAILLÉ, F-35998 RENNES ARMÉES, FRANCE
E-mail address: lercier@celar.fr