In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory: 5th International Symposium, ANTS-V, Sydney, Australia, July 7-12, 2002. Proceedings*, volume 2369 of *Lecture Notes in Computer Science*, pages 431–445. Springer Berlin / Heidelberg, July 2002.

The Function Field Sieve is quite special

Antoine Joux[1] and Reynald Lercier[2]

[1] DCSSI Crypto Lab, 51, Bd de Latour Maubourg
F-75700 Paris 07 SP, France
Antoine.Joux@m4x.org

[2] CELAR, Route de Laillé
F-35570 Bruz, France
lercier@celar.fr

Abstract. In this paper, we describe improvements to the function field sieve (FFS) for the discrete logarithm problem in $\mathbb{F}_{p^n}$, when p is small. Our main contribution is a new way to build the algebraic function fields needed in the algorithm. With this new construction, the heuristic complexity is as good as the complexity of the construction proposed by Adleman and Huang [2], i.e $L_{p^n}[1/3, c] = \exp((c + o(1)) \log(p^n)^{\frac{1}{3}} \log(\log(p^n))^{\frac{2}{3}})$ where $c = (32/9)^{\frac{1}{3}}$. With either of these constructions the FFS becomes an equivalent of the special number field sieve used to factor integers of the form $A^N \pm B$. From an asymptotic point of view, this is faster than older algorithm such as Coppersmith's algorithm and Adleman's original FFS. From a practical viewpoint, we argue that our construction has better properties than the construction of Adleman and Huang. We demonstrate the efficiency of the algorithm by successfully computing discrete logarithms in a large finite field of characteristic two, namely $\mathbb{F}_{2^{521}}$.

1 Introduction

Due to their cryptographic significances, the integer factorization problem and the discrete logarithm problem in finite fields have been extensively studied in the last decades. The best methods currently known to solve these problems are index calculus techniques. In the field of integer factorization, the number field sieve (NFS) [15] having surpassed its ancestor, the quadratic sieve [24], is now the faster of the known factoring algorithm. It exists in two flavors, the general number field sieve which can factor any integer and the special number field sieve which is useful for numbers of a special form: many integers of the form $A^N \pm B$ were factored using the special number field sieve. The latest example is the factorization of $2^{773} + 1$ at CWI [21].

For the computation of discrete logarithms in prime fields, the situation is similar. The quadratic sieve has an analog called the gaussian integer method [6, 13]. Similarly a variant of the number field sieve [23] can be used for computing discrete logarithms. Furthermore, as for factorization, we can distinguish between the general number field sieve and the special number field sieve. For example a computation of discrete logarithms modulo $p = (739 \cdot 7^{149} - 736)/3$ can be found in [27].

In this paper, we address the case of discrete logarithm computations in $\mathbb{F}_{p^n}$, when p is small. The best known practical method for the typical case $p = 2$ is due to Coppersmith [5]. It was used in 1992 by Gordon and McCurley [11] to compute discrete logarithms in $\mathbb{F}_{2^{401}}$. In the same paper, the sieving part of the discrete logarithm computation for $\mathbb{F}_{2^{503}}$ was also reported. More recently, the sieving part of a discrete logarithm computation in $\mathbb{F}_{2^{607}}$ using Coppersmith's method has been performed [25]. The linear algebra step of this computation has been finished very recently [26]. From a more theoretical viewpoint, there exists an analog of the general number field sieve called the function field sieve (FFS), which is not restricted to $p = 2$ [1, 2]. From a practical viewpoint, only Coppersmith's algorithm was considered by now in the characteristic two case [7].

Adleman and Huang [2] showed that the asymptotics of the function field sieve can be largely improved. In fact, with these improvements the function field sieve becomes an equivalent of the special number field sieve. In this paper, we propose a different method that achieves the same complexity. Moreover, in most cases, our method is faster. As a consequence, both the asymptotic complexity and the practical implementation turn out to be better than in older works. We finally illustrate this result by a computation in $\mathbb{F}_{2^{521}}$.

2 Algorithmic considerations

The function field sieve was introduced in [1] for computing discrete logarithms in $\mathbb{F}_{p^n}$ with small values of p. It is quite similar to the number field sieve and it has a complexity of the same order $L_{p^n}[1/3]$. However, there are some crucial differences that allow large improvements. Most notably, a field such as $\mathbb{F}_{p^n}$ can be represented in many different ways. In Coppersmith's algorithm [5], as in the work of Adleman and Huang [2], the key idea was to select a "small representation" of the finite field, more precisely, this is done by selecting a polynomial $\lambda(t)$ of degree as low as possible, such that $t^n - \lambda(t)$ is irreducible. Once $\lambda(t)$ is chosen, it is possible to find two good polynomials having a common root in this field. Clearly, these two constructions focus on a small subset of the possible representations of $\mathbb{F}_{p^n}$. In this paper, we propose a construction that allows a much larger varieties of possible representations. This extra degree of freedom reduces the task of choosing good polynomials at the beginning of the function field sieve algorithm. It turns out that this method keeps the good complexity proved by Adleman and Huang (cf. section 3). Moreover, the selected polynomials are somewhat better. In term of the asymptotic complexity, this is hidden in the $o(1)$, however this yields a significant decrease of the practical run times. In this section, we mostly focus on our new polynomial selection phase.

As a foreword, let us recall that the method we use to compute discrete logarithms in a field $\mathbb{F}_{p^n}$ is derived from the well known Pollig–Hellman method. The first step is to factor $p^n - 1$. For any small factor ℓ of $p^n - 1$, discrete logarithms modulo ℓ can be found using the Pollard Rho method. For the remaining prime factors ℓ of $p^n - 1$, we use the index-calculus method described here. Finally, we combine the results thanks to the Chinese Remainder Theorem in order to get a result modulo $p^n - 1$.

Therefore, in the sequel, we will assume that we are computing discrete logarithms modulo a large prime factor ℓ of $p^n - 1$. This is not an issue since computing the prime factors of $p^n - 1$ can be done with the special number field sieve with a complexity of the same order $L_{p^n}[1/3]$.

2.1 Representation of $\mathbb{F}_{p^n}$

One classical way to work with $\mathbb{F}_{p^n}$ consists in handling equivalence classes in the quotient of the commutative ring $\mathbb{F}_p[t]$ by one of its proper maximal ideals $f(t)\mathbb{F}_p[t]$ where $f(t)$ is an irreducible element of $\mathbb{F}_p[t]$ of degree n. Each equivalence class is then uniquely determined by a polynomial of $\mathbb{F}_p[t]$ of degree strictly smaller than n. Consequently, any element of $\mathbb{F}_{p^n}$ can be seen as a polynomial of degree smaller than n. With such a representation, adding two elements of $\mathbb{F}_{p^n}$ is the same as adding two elements of $\mathbb{F}_p[t]$. Multiplying two elements of $\mathbb{F}_{p^n}$ is the same as multiplying two elements of $\mathbb{F}_p[t]$ and reducing the result modulo $f(t)$.

Since there are numerous irreducible elements of degree n in $\mathbb{F}_p[t]$, there are numerous ways to represent $\mathbb{F}_{p^n}$. The computation of the map between two representations consists in computing the roots over one representation of $\mathbb{F}_{p^n}$ of a polynomial of degree n whose coefficients are in $\mathbb{F}_p$. From an algorithmic viewpoint, this is known as the "equal degree factorization" problem. This can be done quite efficiently since there exists algorithms for this task whose complexity is polynomial in $\log p^n$ (a good survey can be found in [16]). As a consequence, if some particular representation of $\mathbb{F}_{p^n}$ is well suited to discrete logarithm computations, it is a simple matter to switch from a given representation to the more adapted one. In the sequel, we take that step for granted and forget about the given initial representation.

2.2 General principle of the FFS

The Function Field Sieve algorithm is an "index-calculus" method. So it can be seen at a high level of abstraction as a two steps algorithm.

Step 1: One fixes a subset $S = \{\gamma_1, \dots, \gamma_{|S|}\}$ of $\mathbb{F}_{p^n} \simeq \mathbb{F}_p[t]$ called the factor base and tries to collect relations between products of elements of S. So, we have equations of the form

$$\sum_{(\epsilon,\gamma)\in\mathbb{Z}\times S} \epsilon \log_x \gamma = 0 \qquad (1)$$

where x is a generator of the multiplicative subgroup of order ℓ in $\mathbb{F}_{p^n}$. When enough such relations are collected, one obtains the quantities $\log_x \gamma$ via the inversion modulo ℓ of the corresponding linear system.

Step 2: To find the discrete logarithm of an element y which is not in S, one tries random integers ν until $x^\nu y$ is a product of elements of S. Then

$$\log_x y = \left(-\nu + \sum_{(\epsilon,\gamma)\in\mathbb{Z}\times S} \epsilon \log_x \gamma \right) \bmod \ell. \tag{2}$$

The way how the factor base S is chosen is specific to each variation. In the original Function Field Sieve as described by Adleman, the factor base is the image by a morphism ϕ in $\mathbb{F}_{p^n}$ of the generators of two sets S_α and S_β. The set S_α is a set of $\mathbb{F}_p$-rational principal places in the rational function field $\mathbb{F}_p(t)$. The set S_β is a set of $\mathbb{F}_p$-rational principal places defined in an algebraic function field.

Once a random polynomial $\mu(t) \in \mathbb{F}_p[t]$ has been chosen, this function field is defined by an absolutely irreducible bivariate polynomial $H(t, X) = \sum_{i=0}^{d} \sum_{j=0}^{d'} h_{i,j} X^i t^j$ such that $H(t, \mu(t)) = 0 \bmod f(t)$. The mapping ϕ from this algebraic function field to $\mathbb{F}_{p^n}$ is then easily defined by $X \longrightarrow \mu(t)$.

The algorithm consists in finding couples $(r(t), s(t)) \in \mathbb{F}_p[t]^2$, where $r(t)$ and $s(t)$ are relatively prime, such that the polynomial $r(t)\mu(t) + s(t)$ can be written as a product of irreducible polynomials in S_α and such that the divisor associated to the function $r(t)X + s(t)$ can be written as a sum of places in S_β. Following Adleman, such a pair $(r(t), s(t))$ is called "doubly smooth" since $r(t)\mu(t) + s(t)$ is smooth and $r(t)X + s(t)$ is smooth in the sense that the norm over $\mathbb{F}_p[t]$ of $r(t)X + s(t)$ is smooth.

Thanks to eight technical conditions given on $H(t, X)$ by Adleman, these equalities in terms of divisors can be seen as equalities in terms of functions, once raised to the order h of the jacobian power. One can apply the morphism ϕ to get a relation in $\mathbb{F}_{p^n}$. Applying also this morphism on the rational side in the same manner finally yields a relation of the form (1).

2.3 Choice of the polynomials

In full generality, as explained in [8] for the NFS case, the function field sieve requires two polynomials $f_\alpha(X)$ and $f_\beta(X)$ with a common root μ in $\mathbb{F}_{p^n}$. For the algorithm to be efficient, these polynomials should have small coefficients.

The method suggested in [1] is an adaptation of the base m technique used in NFS [3] to the function field case. The method works as follows: choose a polynomial $m(t)$ and write the definition polynomial $f(t)$ of $\mathbb{F}_{p^n}$ in base $m(t)$ as $\sum h_i(t)\mu(t)^i$. Then $X - \mu(t)$ and $H(t, X) = \sum h_i(t)X^i$ clearly have the common root $\mu(t)$ in $\mathbb{F}_{p^n}$. Thus, we get a rational side (corresponding to the degree one polynomial) and an algebraic side. For the number field sieve, several techniques for the polynomial construction lead to two polynomials of degree greater than one. In this case, we no longer have a rational side, which leads to technical difficulties in the later phases of the number field sieve [23].

The version of FFS suggested by Adleman and Huang in [2] is asymptotically much faster. It works by selecting $f(t)$, the polynomial describing the field representation, to be of the form $f(t) = t^n + \lambda(t)$ where $\lambda(t)$ is of degree as low as possible. Then, they choose a parameter d, let $e = \lceil n/d \rceil$ and construct two polynomials $H(t, X) = X^d + t^{ed-n}\lambda(t)$ and $X - t^e$, with common root $\mu(t) = t^e$. In fact, Coppersmith's algorithm [5] can be seen as a subcase of the algorithm of Adleman and Huang, when $p = 2$ and d is a power of two.

We present here a new technique to build good polynomials in the function field case. As the version of Adleman and Huang, this technique is specific to $\mathbb{F}_{p^n}$ and in general cannot be applied in the number field sieve.

The basic idea is simple, we do the construction backward. Instead of choosing a definition polynomial $f(t)$ beforehand, we only fix p^n. Then we choose a polynomial $H(t, X) = \sum h_i(t)X^i$ of degree d in X (the exact value of d will be determined during the complexity analysis) whose coefficients $h_i(t)$ are polynomials in $\mathbb{F}_p[t]$ with very small degrees in t. Afterward, we choose random polynomials $\mu_1(t)$ and $\mu_2(t)$ of degree at most $\lfloor n/d \rfloor$ in t and check whether $f(t) = \mu_2(t)^d H(t, -\mu_1(t)/\mu_2(t))$ is an irreducible polynomial of degree n over $\mathbb{F}_p$. If the test is successful, we are done, otherwise, we choose another pair

$(\mu_1(t), \mu_2(t))$ and restart. Of course, it is essential to correctly choose the coefficients of H to guarantee that f can be of degree n. This implies that the degree of at least one coefficient in H should be the remainder of the division of n by d. Thus the coefficients of H cannot be of arbitrary small degree, however their degrees can be smaller than d in all cases. Moreover, some care should be taken when choosing H. We discuss this point in the next section.

To compare our construction with that of Adleman and Huang, we need to compare the size (degree in t) of the coefficients involved in the two polynomials $H(t, X)$ and $\mu_2(t)X + \mu_1(t)$ (resp. $X - \mu(t)$). A simple way to perform this comparison is to compute the resultant of the two polynomials and compare the respective degrees. With our method, the degree is exactly n as explained above. With the method of Adleman and Huang, the degree is ed and varies from n to $n+d-1$. Unless d divides n, our construction leads to smaller polynomials and thus to a faster algorithm.

In practice, Coppersmith's algorithm is the only one which has been considered for computing discrete logarithms in large finite fields of small characteristic. When writing its complexity as

$$L_{p^n}[1/3, c] = \exp((c + o(1)) \log(p^n)^{\frac{1}{3}} \log(\log(p^n))^{\frac{2}{3}}),$$

we get a value of c between $(32/9)^{\frac{1}{3}}$ and $c = 4^{\frac{1}{3}}$. More precisely, with Coppersmith's algorithm the value of c is not a constant, since there are good cases and bad cases. At best, we have $c = (32/9)^{\frac{1}{3}}$ and at worst $c = 4^{\frac{1}{3}}$, this is always better than Adleman's FFS where $c = (64/9)^{\frac{1}{3}}$. With our construction or that of Adleman and Huang, we have $c = (32/9)^{\frac{1}{3}}$ in all cases. Thus, from a theoretical viewpoint, our algorithm has a larger scope and is faster than Coppersmith's. Indeed, it can be used with a characteristic different from 2. In practice, our algorithm is faster in characteristic two than Coppersmith's whenever the optimal choice of degree for H does not turn out to be a power of 2.

2.4 Number theoretical conditions on the chosen polynomial

When writing down the equation associated to a smooth pair, we must be careful and be sure that these equations really make sense. This involves two technical difficulties.

The first difficulty is that we should not forget any valuation on the algebraic side of the equation. However, when factoring the norm of an ideal, we miss the valuations at infinity. Thus, we need to add these valuations when writing down the equation. In [1], this was done by choosing an algebraic field with several valuations at infinity and by using dehomogenization techniques to compute these valuations. However, this approach is quite cumbersome, specially when writing down the equation. Ideally, we would like to ignore the valuations at infinity. This is possible with the use of so-called "$C_{a,b}$ curves".

Theorem: [18] Let K be a perfect field, $\overline{K}$ the algebraic closure of K, $C_{a,b} \subset \overline{K}$ be a possibly reducible affine algebraic set defined over K, t, X be the coordinates of the affine space, and a, b relatively prime positive integers. Then the following conditions are equivalent.

- $C_{a,b}$ is an absolutely irreducible affine algebraic curve with exactly one K-rational place P_∞ at infinity and the pole divisor of t and X are aP_∞ and bP_∞ respectively.
- $C_{a,b}$ is defined by a bivariate polynomial of the form

$$H(t, X) = h_{a,0}X^a + h_{0,b}t^b + \sum_{ib+ja<ab} h_{i,j}X^i t^j \tag{3}$$

  where $h_{i,j} \in K$ for all i, j and $h_{0,b}$, $h_{a,0}$ are nonzero.

As outlined in [18], any bivariate polynomial $H(t, X)$ of the form (3) is absolutely irreducible. So, only two conditions on H among the eight conditions initially given by Adleman must be satisfied.

1. $f(t)$ divides $\mu_2(t)^d H(t, -\mu_1(t)/\mu_2(t))$.
2. The order of the jacobian of the curve defined by $H(t, X)$ is relatively prime to $(p^n - 1)/(p - 1)$.

The second technical difficulty is the existence of an obstruction group that voids the validity of the equations in certain cases. In the case of the number field sieve, the obstruction group is discussed in [23] and [12]. This obstruction group has two components. The first one comes from the group of units and the second one from the class group of this field. Dealing with the first component is quite difficult and relies on certain maps introduced by Schirokauer. However, dealing with the class group is extremely easy as long as its order is relatively prime with ℓ (the cardinality of the subgroup in which we are computing discrete logarithms). Indeed, in that case, we can simply forget the existence of the class group and everything falls out correctly. For a detailed explanation see [12].

With the FFS, units are handled by Adleman by the valuations at infinity. With $C_{a,b}$ curves, we only have one valuation at infinity, and so, the only units are the elements of $\mathbb{F}_p$. Thus the only obstruction stems from the class group. Since H has a small degree and small coefficients, the class number h is always small. As recalled in [22], $h \leq (\sqrt{p}+1)^{2g}$ where g is the genus of the function field defined by $H(t,X)$. Since in our construction, the polynomial $H(t,X)$ has very small degrees in X and t, g is always small and it is feasible to compute the order of the jacobian to check condition 2. So, since ℓ is supposed to be a large prime because part of the logarithm in small multiplicative subgroup can be determined by other techniques, ℓ and the class number are always relatively prime. Thus, the obstruction group when using this variation of the FFS completely vanishes and the conditions given above can always be considered as satisfied.

2.5 Linear algebra

The linear algebra consists of two sub-steps, the structured gaussian elimination and an iterative solver based on Lanczos' algorithm.

The way we implement the structured gaussian elimination is completely described in [12]. Lanczos' algorithm is described in [14].

2.6 Computing individual logarithms

When computing discrete logarithm with the number field sieve or the function field sieve, finding logarithms of individual numbers is not a negligible task. Indeed, in the theoretical studies of these algorithms, both O. Schirokauer for the number field sieve [23] and L. Adleman for the function field sieve [1] suggest methods where the whole computation essentially needs to be redone for each new logarithm. From a computational viewpoint, this is not acceptable.

However, in [5] a different method was suggested by Coppersmith. A similar method also exists in the large characteristic case [12]. From a theoretical point of view, the complexity of computing an individual logarithm is once again $L_{p^n}[1/3]$, thus it is comparable to Schirokauer's and Adleman's methods. A first attempt at analyzing this approach can be found in [22] (some insights about the complexities are given). In practice, it turns out to be quite efficient.

We now describe Coppersmith's method and adapt it to our construction. The method consists in two steps. In the first step, the individual logarithm computation is split into the logarithm computation of several smaller polynomials, dubbed medium-sized [5]. More precisely, starting from a polynomial $y(t)$, we randomize it by computing $z(t) = x(t)^\nu y(t)$, where $x(t)$ is an element of the factor base. We further write $z(t) = z_1(t)/z_2(t)$, where $z_1(t)$ and $z_2(t)$ have degrees around $n/2$, using the extended Euclidean algorithm. Then we check whether $z_1(t)$ and $z_2(t)$ are smooth with respect to a smoothness bound $L_{p^n}[2/3]$. We now need to compute the logarithm of many polynomials of degree $L_{p^n}[2/3]$. In the second step, each of these logarithms is further splitted into the logarithm of even smaller polynomials. We stop when all polynomials are below the smoothness bound used in the preprocessing stage.

3 Heuristic complexity analysis for small, fixed p

In order to give an heuristic analysis of the complexity of an index calculus method, the traditional approach is to assume that the objects we are trying to factor over the chosen factor base in order to find equations behave like random objects. Under such an heuristic assumption, it is quite easy to quantify the probability of smoothness. This leads in turn to a precise evaluation of the number of couples $(r(t), s(t))$ we need to try before getting sufficiently many equations. With our variant of the function field sieve, candidate equations have two sides which need to be factored. In order to simplify the analysis, the classical approach [3, 1] that we now follow is to assume that the factor bases on the left and the right hand sides contain the same irreducible polynomials. In that case, we can group the two sides in a single polynomial by multiplying together the polynomials coming from the left and the right hand sides[3].

In order to evaluate the probability of smoothness, we use the following result, which can be found in [19, 17]. Let $\mathcal{P}(k, m)$ denotes the probability for a random polynomial of degree k to factor into irreducible polynomials of degree lower than or equal to m. Then, when $k^{1/100} \le m \le k^{99/100}$, i.e. in all the range of interest in our case, we have:

$$\mathcal{P}(k, m) = \exp\left((1 + o(1))\frac{k}{m}\log\frac{m}{k}\right).$$

Assuming that $r(t)$ and $s(t)$ are polynomials of degree lower or equal than l, we find that the degree of the linear side of a candidate equation is at most $l + \lfloor n/d \rfloor$. Similarly, on the other side, testing the function $r(t)X + s(t)$ for smoothness yields polynomials $r(t)^d H(t, s(t)/r(t))$ the degrees of which are bounded by $dl + d$. Indeed, as seen in section 2.3, $H(t, X)$ is polynomial of degree d in X whose coefficients have degree lower than d in t. When multiplying the two sides together, we get a polynomial of degree $dl + d + l + \lfloor n/d \rfloor$. Going through all the possible pairs $(r(t), s(t))$ such that $\gcd(r(t), s(t)) = 1$, we need to find enough smooth pairs. In fact, the number of pairs $(r(t), s(t))$ such that $\gcd(r(t), s(t)) = 1$ is a constant fraction of all pairs. In the sequel, we estimate this number by p^{2l}. We need as many smooth pairs as the number of elements in the factor base. Each factor base contains about p^{m+1}/m elements, since this number counts all the unitary polynomials of degree up to m. Thus, counting both factor bases, we can give an upper bound of $4p^{m+1}/m$ smooth pairs needed. This can be approximated to p^m in an asymptotic approach. Assuming that m and l are already fixed, we can find the optimal value of d by minimizing the total degree $dl + d + l + \lfloor n/d \rfloor$. Asymptotically, we can forget about rounding to the nearest integer and minimize $dl + d + l + n/d = n/d + (d + 1)(l + 1) - 1$. Of course, d should be rounded to the nearest integer, which leads to

$$d = \left\lceil \sqrt{\frac{n}{l+1}} \right\rfloor.$$

Replacing d by its value, we find that the total degree is approximately $2\sqrt{n(l + 1)}$. Moreover, in order to balance the complexity of the sieving phase, which is quadratic in p^l, and of the linear algebra phase, which is quadratic in p^m when using sparse techniques, we need to choose $l = m$. In order to get enough smooth pairs, the smoothness probability should be of the order of p^{-m}. Taking logarithm, we need to satisfy the following equation:

$$\log \mathcal{P}(2\sqrt{n(l+1)}, m) \approx -m \log p, \text{ i.e,}$$
$$\frac{2\sqrt{n(m+1)}}{m} \log\left(\frac{2\sqrt{n(m+1)}}{m}\right) \approx m \log p.$$

Expressing p^m as $L_{p^n}[1/3, c] = \exp((c + o(1)) \log(p^n)^{\frac{1}{3}} \log(\log(p^n))^{\frac{2}{3}})$, we can write $m = (c + o(1))n^{\frac{1}{3}} \log_p(n)^{\frac{2}{3}}$ and get the following equation on c,

$$\frac{2}{3\sqrt{c}} = c.$$

Thus, we find $c = (4/9)^{\frac{1}{3}}$. Since the complexity of the algorithm is quadratic p^m, it can be written as $L_{p^n}[1/3, 2c] = L_{p^n}[1/3, (32/9)^{\frac{1}{3}}]$.

We conclude that the complexity of discrete logarithm computations in $\mathbb{F}_{p^n}$, when p is small and fixed, is in fact the same as the complexity of factoring special integers with the special number field sieve.

[3] Because of this approximation, the complexity stated at the end of this section is clearly an upper bound of the heuristic complexity of the algorithm. However, as far as we known, a more precise analysis would not yield a better complexity.

4 Implementation choices

4.1 Sieving

Sieving is done in $\mathbb{F}_{p^n}$ in a similar way as this is done in $\mathbb{F}_p$ following a now traditional "sieving by vectors" with special–q technique as introduced by Odlyzko for Coppersmith's algorithm [19] or by Pollard for factorizing integers [20]. In fact, we have not implemented any efficient line sieving as proposed in [5] or as generalized in [9].

Using special–q means here that in order to get sufficiently many relations, we sieve many independent sets of values for $(r(t), s(t))$. Each set is defined by an irreducible polynomial $q(t)$ called the special–q, and contains pairs $(r(t), s(t))$ such that $q(t)$ divides $r(t)\mu_1(t) + s(t)\mu_2(t)$. If $\boldsymbol{u}$ and $\boldsymbol{v}$ form a basis of the corresponding lattice, then $(r(t), s(t))$ can be written as $k_u(t)\boldsymbol{u} + k_v(t)\boldsymbol{v}$. Then we can with simple linear algebra send the lattice corresponding to any small prime ideal from the $(r(t), s(t))$ representation to the $(k_u(t), k_v(t))$ representation. Sieving is then done in a big rectangle in the $(k_u(t), k_v(t))$ space by successively marking the points on each of the small prime lattices. A basis for these small prime ideals can be easily obtained from the factorization of $H(t, X)$ modulo the norm of the ideal but the resulting coordinates have size close to the norm of the ideal. We improve this by combining the vectors of this basis in order to get a new basis with coordinates of degree approximately twice as small. This is done in full generality in our implementation by using an adaptation to the ring $\mathbb{F}_2[t]$ of the well-known algorithm of Gauss for reducing lattices in dimension two. We give pseudo-code for this reduction algorithm in figure 1.

```
– **Input:** A basis of the lattice $(\boldsymbol{u}, \boldsymbol{v})$ with $\boldsymbol{u} = (u_1, u_2)$ and $\boldsymbol{v} = (v_1, v_2)$.
– **Output:** A reduced basis.
– **Step 1:** Let $d_u = \max(\deg(u_1), \deg(u_2))$ and $d_v = \max(\deg(v_1), \deg(v_2))$. If $d_u > d_v$, exchange $\boldsymbol{u}$ and $\boldsymbol{v}$.
– **Main loop:** Do
    • Let $\delta_1 = \deg(v_1) - \deg(u_1)$, $\delta_2 = \deg(v_2) - \deg(u_2)$.
    • Let $\boldsymbol{w^{(1)}} = \boldsymbol{v} - t_1^{\delta} \cdot \boldsymbol{u}$.
    • Let $\boldsymbol{w^{(2)}} = \boldsymbol{v} - t_2^{\delta} \cdot \boldsymbol{u}$.
    • If $\max(\deg(w_1^{(1)}), \deg(w_2^{(1)})) < \max(\deg(w_1^{(2)}), \deg(w_2^{(2)}))$, let $\boldsymbol{w} = \boldsymbol{w^{(1)}}$ else let $\boldsymbol{w} = \boldsymbol{w^{(2)}}$.
    • If $\max(\deg(v_1), \deg(v_2)) > \max(\deg(w_1), \deg(w_2))$, let $\boldsymbol{v} = \boldsymbol{w}$ and declare the loop as active.
    • If $\max(\deg(u_1), \deg(u_2)) \geq \max(\deg(v_1), \deg(v_2))$, exchange $\boldsymbol{u}$ and $\boldsymbol{v}$.
– Until the loop is not declared active for two consecutive executions.
– Output $(\boldsymbol{u}, \boldsymbol{v})$.
```

Fig. 1. Algorithm for reducing lattices in dimension 2 over $\mathbb{F}_2[t]$.

In order to consider pairs $(r(t), s(t))$ such that $\gcd(r(t), s(t)) = 1$, a necessary condition is that $\gcd(k_u(t), k_v(t)) = 1$. So, in the rectangle we allocate for the sieving, positions corresponding to a pair $(k_u(t), k_v(t))$ with two coordinates divisible by t can be omitted [10]. This is a quick shortcut to avoid 25% of the gcd computations. Similarly, we avoid the positions where both coordinates are divisible by $t + 1$.

Depending on the problem we have to handle, it can be computationally interesting to perform such a sieve on the algebraic side too. In this case, each point on the rectangle are marked if they are points of the small prime ideals on the linear side or on the algebraic side. This was for instance the case for the computation described in section 5. This is done by representing the prime ideals on the algebraic side as lattices and handling them as on the linear side.

After selecting good $(k_u(t), k_v(t))$ candidates in this way, we can check efficiently that the corresponding values $r(t)\mu_1(t) + s(t)\mu_2(t)$ are indeed smooth using Berlekamp's algorithm. Then, if for some of the remaining couples $(r(t), s(t))$, the divisor of the function $r(t)X + s(t)$ is smooth too, this produces an algebraic relation between elements of the factor bases.

Remark: Berlekamp's algorithm has got roughly two phases; the construction of a set of "f-reducing polynomials" thanks to the kernel computation of the Berlekamp's matrix and the separation step itself involving the computation of gcds between the list of factors and the reducible polynomials [16]. Let us note that the last phase can be speeded up using the fact that the polynomials that we want to factor have their potential factors stored in the factor bases. Thanks to this table of irreducible polynomials, one can test early in the process whether the partial factors are irreducible. When they are, we remove them from the list of factors. Thus we can spare many of the gcd computations that are necessary when no table of irreducible polynomials is available.

4.2 Linear algebra

As explained in section 2.5, it is straightforward to apply to the case $\mathbb{F}_{p^n}$, p small, the ideas developed for $\mathbb{F}_p$, p large. Simply, this step is done modulo each large prime factor ℓ of $(p^n-1)/(p-1)$.

The only small improvement we are aware of concerns the characteristic 2 case. When 2^n-1 is a prime, the arithmetic involved in Lanczos' algorithm can be slightly speeded up. This consists in using the classical fact that the reductions modulo 2^n-1 can be done by a single subtraction on the binary representation of the integers involved. When 2^n-1 is not a prime, it is usually better to perform Lanczos' algorithm modulo each large prime factor ℓ of 2^n-1, instead of modulo 2^n-1.

5 Example

Let σ be the mapping defined from the set of integers to $\mathbb{F}_2[t]$ which sends an integer ν (written in an hexadecimal way) to a polynomial $\sigma(\nu)$ such that substituting t by 2 in $\sigma(\nu)$ yields ν (for instance, $\sigma(\mathtt{b}) = t^3+t+1$), we now describe a discrete logarithm computation in $\mathbb{F}_{2^{521}}$.

Precisely, we were able to compute the discrete logarithm of $e(t)$, $\pi(t)$ and $e(t)+\pi(t)$ where

$$\begin{aligned} e(t) &= \sigma(\lfloor 2^{519}e \rfloor) = t^{520}+t^{518}+\ldots+t^6+t^3, \\ \pi(t) &= \sigma(\lfloor 2^{519}\pi \rfloor) = t^{520}+t^{519}+\ldots+t^6+t^3+1. \end{aligned}$$

At first, we fixed a representation of $\mathbb{F}_{2^{521}}$ by choosing a $C_{1,5}$ algebraic curve over $\mathbb{F}_2$ given by

$$H(t,X) = X^5+X+t+1,$$

and checking that the resultant of $H(t,X)$ with the bivariate polynomial

$$\mu_2(t)X \quad + \quad \mu_1(t) \qquad = \qquad \sigma(\mathtt{1b92c17dec4c4cf4f5ab9c1e86f})X \quad + \quad \sigma(\mathtt{d0e134790925d9e08})$$

yields an irreducible polynomial $f(t)$ of degree 521.

Of course, there exists many $C_{a,b}$ curves which could have been used here. However, following an idea developed for factoring integers, we select a polynomial $H(t,X)$ whose number of roots, modulo irreducible polynomials of small degree over $\mathbb{F}_2$, is slightly larger than usual.

The factor base contains the 300 000-th first irreducible polynomials over $\mathbb{F}_2$ once ordered by their σ values and contains the places with norms of degree smaller than 22 in the function field defined by $H(t,X)$. After a three weeks computation on a quadri-processors alpha server 8400 computer, we obtained 472 121 equations in 450 940 unknowns with 9 235 383 nonzero entries.

So we had 300 000 special-q. For each special-q, we marked points in a rectangle $(k_u(t),k_v(t))$ of size $2^{14}\times 2^{14}$ such that the corresponding pairs $(r(t),s(t))$ are candidates for smoothness (cf. section 4.1). This yielded around 2 000 candidates. Testing them with Berlekamp's algorithm, both in the linear and the algebraic side, gave in average 2 equations.

We then applied a structured Gaussian elimination to reduce our system to 197 039 equations in 196 939 unknowns with 12 220 108 nonzero entries [13, 12] (249277 entries were different from ± 1, the largest was 29). Time needed for this on only one processor was about one hour.

Then, our parallelized version of Lanczos' algorithm took 10 days over 4 processors to finish the linear inversion modulo $2^{521}-1$. At the end, we had "logarithms for ideals" of small norms. As a consequence, we had logarithms for small irreducible polynomials:

$$\begin{aligned} \log_t(t+1) = {} & 94681577152122294076175173598650324606 21 \\ & 888852201905263910801487998985884345864952201320754968 8251 \\ & 3361552641792316365389142863458255063795516109214621940159, \end{aligned}$$

$$\begin{aligned}\log_t(t^2+t+1) = {}& 4099453203357757668284443933632134015543\\ & 7560387960711214880627918982361730130023913248564073810794\\ & 9052818943078142206215533143595141990328387727782201876189 1,\end{aligned}$$

$$\vdots$$

Afterwards, we found in few hours, two polynomials,

$$\begin{aligned}z_1(t) = {}& \sigma(\mathtt{17acf35dc9215}) \times \sigma(\mathtt{33cab5311}) \times \sigma(\mathtt{83b6db37}) \times \sigma(\mathtt{88af29f}) \times \sigma(\mathtt{4c99eb3})\\ & \times \sigma(\mathtt{1a22cdd}) \times \sigma(\mathtt{debb79}) \times \sigma(\mathtt{6358f}) \times \sigma(\mathtt{304f}) \times \sigma(\mathtt{6b5}) \times \sigma(\mathtt{41b}) \times \sigma(\mathtt{75}) \times \sigma(\mathtt{2})^4\end{aligned}$$

and

$$\begin{aligned}z_2(t) = {}& \sigma(\mathtt{41edc78c5127}) \times \sigma(\mathtt{75a6c0fe253}) \times \sigma(\mathtt{b66ac13d5}) \times \sigma(\mathtt{d422507}) \times \sigma(\mathtt{b0b0e11})\\ & \times \sigma(\mathtt{d2c45}) \times \sigma(\mathtt{81869}) \times \sigma(\mathtt{54e1}) \times \sigma(\mathtt{a85}) \times \sigma(\mathtt{409}) \times \sigma(\mathtt{25f}) \times \sigma(\mathtt{fd}) \times \sigma(\mathtt{3b}) \times \sigma(\mathtt{7}) \times \sigma(\mathtt{3})\end{aligned}$$

such that

$$\sigma(\mathtt{3fffcd})^{43} \times e(t) = z_1(t)/z_2(t) \bmod f(t).$$

Then, using at most 6 levels of special–q descents, computing discrete "logarithms for the ideals" of norms larger than $\sigma(\mathtt{5df401})$ in the left algebraic field was (thanks to a one hour computation for each ideal, on a unique processor) equivalent to compute discrete "logarithms for ideals" of norms larger than those of the factor base in the algebraic function field. Time needed for computing the corresponding discrete logarithms was at most one hour for each on a unique processor. At the end, we obtained

$$\begin{aligned}\log_t e(t) = {}& 2632477621938341298849947024285383 6\\ & 0289317407093273177190025600958418025325465481707648375864292\\ & 8456502454746890820252043876734626779920800953806109457874358.\end{aligned}$$

Similarly, we found

$$\begin{aligned}\log_t \pi(t) = {}& 5475291480121133585788840019440488 3\\ & 4396041695431692261837322543793760717339868611259553398016090\\ & 4708790051138588209091739455561530487613513767198209433496844,\end{aligned}$$

and

$$\begin{aligned}\log_t(e(t)+\pi(t)) = {}& 4159201401120253179205437750401930 7\\ & 6439975376714991661072042543367168030386736581168078966485150\\ & 6272465239307862846898957189950632165222399100568185398518167.\end{aligned}$$

So, as a conclusion, time that we need for computing discrete logarithms in $\mathbb{F}_{2^{521}}$ on a 525 MHz quadri-processor alpha server 8400 computer is approximatively 12 hours for each, once the sieving step (21 days) and the linear algebra step (10 days) is performed.

The software we used is an adaptation to the characteristic two of a $\mathbb{F}_p$ implementation [12] taking advantage of a generic software based on a finite field `C` library called `ZEN` [4].

Remark: Since the current record in this field of research is a computation in $\mathbb{F}_{2^{607}}$ obtained with Coppersmith's algorithm after one year over 100 PCs [26], it is natural to estimate on the basis of our computation over $\mathbb{F}_{2^{521}}$ what would be the time needed by this FFS implementation to handle $\mathbb{F}_{2^{607}}$. This can be easily done by computing $L_{2^{607}}[1/3,(32/9)^{\frac{1}{3}}]\ /\ L_{2^{521}}[1/3,(32/9)^{\frac{1}{3}}]$. This yields a factor of 12 and means a one year computation on a single 525 MHz quadri-processor alpha server 8400 computer. We have performed some experiment in this range, they corroborate this rough estimate.

6 Conclusion

In this paper, we described improvements to the function field sieve for the discrete logarithm problem. With these improvements, we computed discrete logarithms in $\mathbb{F}_{2^{521}}$ and showed that the function field sieve can be considered as an equivalent of the special number field sieve, giving the confirmation that it is faster, both from an asymptotic and from a computational viewpoint, than Coppersmith's algorithm and Adleman's original FFS.

7 Acknowledgments

The authors would like to thank J.M. Couveignes for some helpful discussions about $C_{a,b}$ curves. They would also like to thank the anonymous referees for valuable comments and for pointing out to them that Adleman and Huang [2] already proved that the Function Field Sieve has the same complexity as the special Number Field Sieve.

References

[1] L. M. Adleman. The function field sieve. In *Proceedings of the ANTS-I conference*, volume 877 of *Lecture Notes in Computer Science*, pages 108–121, 1994.

[2] L. M. Adleman and M. A. Huang. Function field sieve method for discrete logarithms over finite fields. In *Information and Computation*, volume 151, pages 5–16. Academic Press, 1999.

[3] J. P. Buhler, H. W. Lenstra, Jr., and C. Pomerance. Factoring integers with the number field sieve. Pages 50–94 in [15].

[4] F. Chabaud and R. Lercier. *ZEN, User Manual*. Available at `http://www.di.ens.fr/~zen/`.

[5] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE transactions on information theory*, IT-30(4):587–594, July 1984.

[6] D. Coppersmith, A. Odlyzko, and R. Schroppel. Discrete logarithms in $\mathbb{F}_p$. *Algorithmica*, 1:1–15, 1986.

[7] T. Denny, O. Schirokauer, and D. Weber. Discrete Logarithms: The effectiveness of the Index Calculus Method. In *Proceedings of the ANTS-II conference*, volume 1122 of *Lecture Notes in Computer Science*, pages 337–361, 1996.

[8] M. Elkenbracht-Huizing. An implementation of the number field sieve. *Experimental Mathematics*, 5(3):231–253, 1996.

[9] S. Gao and J. Howell. A general polynomial sieve. *Designs, Codes and Cryptography*, 18:149–157, 1999.

[10] R. Golliver, A. K. Lenstra, and K. McCurley. Lattice sieving and trial division. In *Proceedings of the ANTS-I conference*, volume 877 of *Lecture Notes in Computer Science*, pages 18–27. Springer-Verlag, 1994.

[11] D. Gordon and K. McCurley. Massively parallel computation of discrete logarithms. In *Advances in Cryptology — CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 312–323. Springer-Verlag, 1993.

[12] A. Joux and R. Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. *Math. Comp.*, 2000. To appear. Preprint available at `http://www.medicis.polytechnique.fr/~lercier/`.

[13] B. A. LaMacchia and A. M. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography*, 1:47–62, 1991.

[14] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse systems over finite fields. In *Advances in Cryptology — CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer-Verlag, 1991.

[15] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer–Verlag, 1993.

[16] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison–Wesley, 1983.

[17] R. Lovorn. *Rigorous Subexponential Algorithms for Discrete Logarithms Over Finite Fields*. PhD thesis, Univ. of Georgia, 1992.

[18] R. Matsumoto. Using C_{ab} curves in the function field sieve. *IEICE Trans. Fundamentals*, E82-A(3):551–552, march 1999.

[19] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology — EUROCRYP'84*, volume 209 of *Lecture Notes in Computer Science*, pages 224–314. Springer–Verlag, 1985. Available at `http:/www.dtc.umn.edu/~odlyzko`.

[20] J.M. Pollard. The lattice sieve. Pages 43–49 in [15].

[21] P. Montgomery S. Cavallar and H. te Riele. New record SNFS factorization. Available at `http://-listserv.nodak.edu/archives/nmbrthry.html`, november 2000. Factorization of $2^{773} + 1$.

[22] O. Schirokauer. The special function field sieve. Preprint.

[23] O. Schirokauer. Discrete logarithms and local units. *Phil. Trans. R. Soc. Lond. A 345*, pages 409–423, 1993.

[24] R. D. Silverman. The Multiple Polynomial Quadratic Sieve. *Math. Comp.*, 48:329–339, 1987.

[25] E. Thomé. Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$. In *Advances in Cryptology — ASIACRYPT'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 107–124. Springer–Verlag, 2001.

[26] E. Thomé. Discrete logarithms in $\mathbb{F}_{2^{607}}$. Available at `http://listserv.nodak.edu/archives/-nmbrthry.html`, february 2002.

[27] D. Weber and Th. Denny. The solution of mccurley's discrete log challenge. In H. Krawczyk, editor, *Advances in Cryptology — CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 458–471. Springer–Verlag, 1998.