ÉCOLE POLYTECHNIQUE

LIX

A variant of the Adleman–DeMarrais–Huang algorithm and its application to small genera

P. Gaudry

LIX/RR/99/04

11 juin 1999

LABORATOIRE D'INFORMATIQUE
Unité mixte CNRS n°7650
École polytechnique 91128 Palaiseau Cedex FRANCE

A variant of the Adleman–DeMarrais–Huang algorithm and its application to small genera

Pierrick Gaudry *
LIX, École polytechnique
91128 Palaiseau
France
gaudry@lix.polytechnique.fr

June 11, 1999

Abstract

We propose a variant for the discrete logarithm algorithm of Adleman–DeMarrais–Huang. Our attack is faster and its complexity can be more precisely analysed. Some real experiments were done and we can give the expected time for breaking a cryptosystem of a given genus, which is better than the Pollard rho method as soon as the genus is greater than 4. The drawback of our method is that it assumes the cardinality of the jacobian to be known.

Keywords: hyperelliptic curves, discrete logarithm, index calculus.

1 Introduction

The use of hyperelliptic curves in public-key cryptography was first proposed by Koblitz in 1989 [19]. It appears as an alternative to the use of elliptic curves, with the advantage that it requires to work in a smaller base field. In order to perform for example a Diffie–Hellman protocol [9], we have to know the number of elements in the group. Thus in the past years some hyperelliptic curves were proposed with a way to compute the cardinality of their jacobians.

The security of such systems resides in the difficulty to solve the discrete logarithm problem in the jacobian of hyperelliptic curves. If an algorithm tries to solve this problem performing "simple" group operations only, it was shown by Shoup [40] that the complexity is at least $O(\sqrt{n})$, where n is the largest prime dividing the order of the group. Algorithms with such a complexity exist for generic groups and can be applied to hyperelliptic curves, but are still exponential.

For the elliptic curve discrete logarithm problem, there are some particular cases where a solution can be found with a complexity better than $O(\sqrt{n})$. See [24], [37], [41], [34]. A few years later, similar cases were discovered for hyperelliptic curves [14], [32]. However they are very particular and can be easily avoided when building a cryptosystem.

In 1994, Adleman, DeMarrais and Huang [2] published a new algorithm which runs in subexponential time when the genus is sufficiently large compared to the size of the ground field. This algorithm was rather theoretical, and some improvements of it were done: Flassenberg and Paulus [13] implemented a sieve version of this algorithm, but the consequences for cryptographical applications is not clear. Enge [12] improved the original algorithm and gave a precise

*This work was supported by Action COURBES of INRIA (action coopérative de la direction scientifique de l'INRIA).

evaluation of the running time, but did not implement his ideas. Müller, Stein and Thiel [25] extended the results to the real quadratic congruence function fields. Smart and Galbraith [16] also gave some ideas in the context of the Weil descent.

However all this did not give a real knowledge on which cryptosystems were weakened by this attack.

Our purpose is to present a variant of the Adleman–DeMarrais–Huang algorithm which allowed us to break a cryptosystem of genus 8 proposed by Koblitz, and seems to make almost breakable a cryptosystem of genus 6 proposed independently by Sakai–Sakurai and Smart. From a theoretical point of view, our algorithm is faster than the Pollard rho method as soon as the genus is strictly greater than 4. Moreover, in our implementation, we can make an extensive use of the presence of an automorphism acting on the curve. Indeed in almost all the examples of the literature there exists one.

The organization of the paper is as follows: in section 2 we recall some basic facts about the construction of hyperelliptic cryptosystems. In section 3 we give a survey about known attacks of the discrete logarithm problem, followed in section 4 by a brief description of the original Adleman–DeMarrais–Huang algorithm. In section 5 we present our variant which we analyze in section 6. In section 7 we describe how to make use of an automorphism of the curve when it exists, and in section 8 we give some timings for breaking real life cryptosystems.

2 Preliminaries on hyperelliptic cryptosystems

For some cryptographical protocols, we need to have a finite abelian group, in which it is easy to compute the addition of two elements and such that the discrete logarithm problem is difficult. The jacobians of hyperelliptic curves defined over finite fields provide such groups.

2.1 The jacobian of a hyperelliptic curve

Let $\mathbb{K}$ be a field. A hyperelliptic curve $\mathcal{C}$ of genus g over $\mathbb{K}$ is a plane non-singular projective curve which admits an affine equation of the form $y^2 + h(x)y = f(x)$, where f is a polynomial of degree $2g+1$, and h is a polynomial of degree at most g. The most general case allows the polynomial f to be of degree $2g+2$, but then the (desingularized) curve has two points at infinity, so it is less easy to deal with, and these curves are not used in cryptography. Note that if the characteristic of $\mathbb{K}$ is odd, then we can always perform a change of variables in order to obtain a simpler equation with $h(x) = 0$.

For a demonstration of the following results, the reader is referred to Fulton [15] or Hartshorne [18]. For the moment, we assume the field $\mathbb{K}$ to be algebraically closed.

A *divisor* on the curve $\mathcal{C}$ (and more generally on every curve) is a finite formal sum of points of the curve. The set of all divisors yield an abelian group denoted by $Div(\mathcal{C})$. For each divisor $D = \sum_i n_i P_i \in Div(\mathcal{C})$, where the P_i are points on the curve, we define the *degree* of D by $\deg(D) = \sum_i n_i$. The set of all divisors of degree zero is a sub-group of $Div(\mathcal{C})$ denoted by $Div^0(\mathcal{C})$.

For each function $\varphi(x, y)$ on the curve, we can define a divisor denoted by $\mathrm{div}(\varphi)$ by assigning at each point P_i of the curve the value n_i equal to the multiplicity of the zero if $\varphi(P_i) = 0$, or the opposite of the multiplicity of the pole if the function is not defined at P_i. We can summarize that by the formula $\mathrm{div}(\varphi) = \sum_i \nu_{P_i}(\varphi)$. We can show that this is a well-defined divisor, i.e. the number of non-zero coefficients $\nu_{P_i}(\varphi)$ is finite. And moreover, for every function φ, the divisor $\mathrm{div}(\varphi)$ has degree 0. Such a divisor is called a *principal* divisor, and the set of all the principal divisors is denoted by $\mathcal{P}(\mathcal{C})$. It is easy to see that for all functions φ and $\psi \neq 0$ we have $\mathrm{div}(\varphi/\psi) = \mathrm{div}(\varphi) - \mathrm{div}(\psi)$, and then that $\mathcal{P}(\mathcal{C})$ is a subgroup of $Div^0(\mathcal{C})$.

The *jacobian* of the curve $\mathcal{C}$ is defined to be the quotient group $Jac(\mathcal{C}) = Div(\mathcal{C})^0/\mathcal{P}(\mathcal{C})$. If two divisors D_1 and D_2 belong to the same class modulo $\mathcal{P}(\mathcal{C})$, we say that they are *linearly equivalent* and we write $D_1 \equiv D_2$.

If the field $\mathbb{K}$ is not algebraically closed (we still suppose however that $\mathbb{K}$ is perfect), we want to define the jacobian over $\mathbb{K}$, and therefore we define a *rational* divisor over $\mathbb{K}$. The natural way to do that is to say that a divisor $D = \sum_i n_i P^i$ is rational over $\mathbb{K}$ if for each automorphism σ of $\overline{\mathbb{K}}$ over $\mathbb{K}$, we have $D^\sigma = D$. This condition is weaker than to ask that all the P_i be defined over $\mathbb{K}$.

2.2 Semi-reduced divisors and their representation

Let $\mathcal{C}$ be an hyperelliptic curve of genus g with one point at infinity denoted by ∞. For each point $P = (x, y)$ on the curve, one can define another point $P' = (x', y')$ such that the divisor $P + P' - 2(\infty) \equiv 0$. If the equation is of the form $y^2 = f(x)$, the point P' is simply defined by $x' = x$ and $y' = -y$. In characteristic 2, the formulae are slightly different. This involution $P \mapsto P'$ is called the hyperelliptic involution.

A *semi-reduced* divisor is a divisor of degree zero which can be written $\sum_{i=1}^{r} P_i - r(\infty)$, where $P_i \neq P_j'$ for all $i \neq j$. For each point P on the curve, $P + P' - 2(\infty)$ is in $\mathcal{P}(\mathcal{C})$, hence one easily sees that each class of divisor in $Jac(\mathcal{C})$ contains at least one semi-reduced divisor (many in fact).

Those divisors can be represented by a very convenient way (see [26], [5], [19]): Let $D = \sum_{i=1}^{r} P_i - r(\infty)$ be a reduced divisor, where $P_i = (x_i, y_i)$. Then we define two polynomials $u(z)$ and $v(z)$ by $u(z) = \prod_{i=1}^{r}(z - x_i)$, and $v(z)$ is the unique polynomial of degree at most $r - 1$ such that $v(x_i) = y_i$ for all i, and $u(z)$ divides $v^2(z) - f(z)$, if the curve is given by $y^2 = f(x)$ (resp. $u(z)$ divides $v^2(z) + h(z)v(z) - f(z)$, if the equation of the curve is $y^2 + h(x)y = f(x)$). The important fact to note is that these polynomials are defined over the field of definition of D, even if one of the P_i's is defined over an extension field.

Moreover, a consequence of the Riemann-Roch theorem (see [15]) is that for every class of divisor in $Jac(\mathcal{C})$, we can find a unique semi-reduced representative of the form $\sum_{i=1}^{r} P_i - r(\infty)$, with $r \leq g$. This representative is called a *reduced* divisor. Hence each element of the group $Jac(\mathcal{C})$ has a unique polynomial representation $[u(z), v(z)]$, where $\deg v(z) < \deg u(z) \leq g$.

Sometimes we shall speak of the *degree* of a semi-reduced divisor instead of the degree of its first polynomial with the above representation. There is no possible confusion because the "true" degree of a semi-reduced divisor is always zero.

2.3 Cantor's addition algorithm

In [5], Cantor describes an algorithm which performs group operations in $Jac(\mathcal{C})$, and operates directly with the polynomial representation. We give here the general formulae for a curve of equation $y^2 + h(x)y = f(x)$ (if the equation is $y^2 = f(x)$, put $h(x) = 0$ in the following).

Divisors composition:

This algorithm takes as input two semi-reduced divisors $D_1 = [u_1(z), v_1(z)]$ and $D_2 = [u_2(z), v_2(z)]$, and computes a third semi-reduced divisor $D_3 = [u_3(z), v_3(z)]$ such that $D_1 + D_2 \equiv D_3$.

First compute $d(z)$, the GCD of the three polynomials $u_1(z)$, $u_2(z)$ and $v_1(z) + v_2(z) + h(z)$, as well as their Bézout coefficients: $d = l_1 u_1 + l_2 u_2 + l_3(v_1 + v_2 + h)$. Now we have the following formulae for D_3:

$$\left\{ \begin{array}{rcl} u_3 & = & \dfrac{u_1 u_2}{d^2}, \\ v_3 & = & \dfrac{l_1 u_1 v_2 + l_2 u_2 v_1 + l_3(v_1 v_2 + f)}{d} \mod u_3. \end{array} \right.$$

Divisor reduction: This algorithm takes as input a semi-reduced divisor $D = [u(z), v(z)]$ of degree $k > g$ (i.e. $\deg u(z) = k$) and outputs a semi-reduced divisor $D' = [u'(z), v'(z)]$ of degree at most $k - 2$ such that $D \equiv D'$.

The result is given by the formulae:

$$\left\{ \begin{array}{rcl} u' & = & \dfrac{f - v^2 - hv}{u} \\ v' & = & (-h - v) \mod u', \end{array} \right.$$

and we make u' monic by dividing it by its leading coefficient.

Now, if one wants to compute the addition of two reduced divisors, we first compose them, thus obtaining a semi-reduced divisor of degree at most $2g$, and then we repeat the reduction step until the degree of the semi-reduced divisor becomes at most g. The reduced divisor we obtain is then the representative of the sum of the two given reduced divisors. The opposite of a reduced divisor $D = [u(z), v(z)]$ is given by the simple expression $-D = [u(z), (-v(z) - h(z)) \mod u(z)]$.

Hence we can easily compute the group law in the jacobian of an hyperelliptic curve.

2.4 Computing the cardinality of the jacobian over a finite field

If $\mathcal{C}$ is a curve of genus g over the finite field $\mathbb{F}_q$, then its jacobian $Jac(\mathcal{C})$ is a finite abelian group, and there is a bound on its cardinality. We can define a polynomial L of degree $2g$ with integer coefficients, such that $L(1)$ is the cardinality of the jacobian over $\mathbb{F}_q$. The Hasse-Weil theorem (sometimes called Riemann Hypothesis for Algebraic Function Fields) says that all the (complex) roots of the L polynomial have absolute value $q^{1/2}$. We can then deduce that the cardinality of $Jac(\mathcal{C})$ is equal to q^g with an error bounded by approximatively $2gq^{g-1/2}$.

For cryptographical applications, one needs to know the exact cardinality of the group in which we are computing. From a theoretical point of view, there is a polynomial time algorithm doing this job for jacobians of curves; this is an hyperelliptic version of the Schoof algorithm counting points on elliptic curves, see [28], [6], [3], [35].

However this algorithm is still not very practical and a lot of techniques to build jacobians with known number of elements can be found in the literature.

The first technique is as follows: take a curve over a small field $\mathbb{F}_q$, then by the Weil theorem it is easy to compute the cardinality of the jacobian over $\mathbb{F}_q$, and with the help of the Zeta function one gets the cardinality of the jacobian over an extension $\mathbb{F}_{q^k}$. Some of these curves can be found in [19] or [33].

Another way is to use the theory of complex multiplication. There are some curves such that the cardinality of the jacobian is related to some computable character sums, see for example [20], [11]. And for each field with complex multiplication, one can build curves for which the cardinality of the jacobian is known, see [43] and [7].

An interesting point to note is that most of the curves that are built by these techniques come with a non-trivial automorphism, which reduces the cost of breaking the system, see [10] and the section 7 below. The only counter-examples are the genus 2 curves built by Spallek [43], see also [48].

3 Known attacks for the discrete log problem

There are several methods to solve the discrete logarithm problem in an arbitrary abelian group. They can be applied to the jacobians of hyperelliptic curves. There are also a few methods which works only for a small class of curves. In this section we recall these techniques.

3.1 The Pohlig–Hellman algorithm

If G is a finite abelian group with $\#G = \prod_i g_i^{e_i}$, there is an algorithm proposed by Pohlig and Hellman [29] which reduces the discrete logarithm problem in this group to the discrete logarithm problem in subgroups of G of cardinality g_i.

Suppose we are given P_1 and P_2 in G and we search λ such that $P_2 = \lambda P_1$. The algorithm computes λ modulo $g_i^{e_i}$, and then recombine the partial results via the Chinese remainder theorem. Let us explain how to compute λ modulo $g_1^{e_1}$. We write $\lambda \equiv \sum_{j=0}^{e_1-1} \lambda_j g_1^j \mod g_1^{e_1}$, and compute the λ_j successively.

Consider $P_1' = (\#G/g_1)P_1$ and $P_2' = (\#G/g_1)P_2$, these are two elements of order dividing g_1, and they are related by $P_2' = \lambda P_1' = \lambda_0 P_1'$. We can then find λ_0 by solving a discrete logarithm problem in a subgroup of cardinality g_1. Now to compute λ_1, we consider $P_2'' = (\#G/g_1^2)(P_2 - \lambda_0 P_1)$ which is an element of order dividing g_1, and we have $P_2'' = (\lambda - \lambda_0)/g_1 P_1' = \lambda_1 P_1'$. And we can find λ_1 again by solving a discrete logarithm in a subgroup of order g_1. In the same manner we determine all the λ_i, and the problem is solved.

Thus we are now only interested in solving the discrete logarithm problem in subgroup of prime order.

3.2 The Shanks method

Let n be the order of P_1 and $w = \lceil \sqrt{n} \rceil$. Then the so-called "Baby-step Giant-step" method due to Shanks [38] computes the discrete logarithms in the group generated by P_1 in time and space $O(w)$. The technique is as follows: write $\lambda = \lambda_0 + \lambda_1 w$, where λ_0 and λ_1 are bounded by w. Then we have to find their values such that $P_2 - \lambda_1(wP_1) = \lambda_0 P_1$. First compute and store the "Baby steps", i.e. all the points kP_1 for k from 1 to w, and then compute the "Giant steps" and check if we have a match in the table: for l from 1 to w, compute $P_2 - l(wP_1)$ and search if there was a k which gave the same point. When we have a match, we have $\lambda_1 = l$, and $\lambda_0 = k$.

To perform this algorithm we have to compute $O(w) = O(\sqrt{n})$ group operations, and to store a table of $O(w)$ elements of the group. The search in the table has to be very quick, and hash method must be used.

Note there is an improved version of this algorithm due to Terr [45] which does not require any bound on the cardinality of the group.

3.3 The Pollard rho algorithm

Pollard [30] proposed a probabilistic algorithm which has about the same time complexity as Shanks method, but uses $O(1)$ space. The principle is to define a pseudo-random walk on the subgroup generated by P_1 (we suppose it is the whole group G of prime order n), and to wait for a collision.

Let $\mathcal{H}$ be a hash function from G to the set $\{1, 2, 3\}$. Let $R_0 = u_0 P_1 + v_0 P_2$, where u_0 and v_0 are random integers. Then define $R_{i+1} = f(R_i)$ by

$$f(R_i) = \begin{cases} R_i + P_1, & \text{if } \mathcal{H}(R_i) = 1, \\ R_i + P_2, & \text{if } \mathcal{H}(R_i) = 2, \\ 2R_i, & \text{if } \mathcal{H}(R_i) = 3. \end{cases}$$

Simultaneously, we keep u_i and v_i updated such that $R_i = u_i P_1 + v_i P_2$.

Then if we have two indices $i \neq j$ such that $R_i = R_j$, we can determine $\lambda = (u_1 - u_2)/(v_2 - v_1) \mod n$. To avoid storing all the previously computed R_i, u_i, v_i, one can only store $R_i, u_i, v_i, R_{2i}, u_{2i}, v_{2i}$, thus detecting a cycle in the graph of the random mapping with almost no storage.

Provided that the walk is random enough, we can estimate the complexity of this method to $O(\sqrt{n})$ group operations. Teske in [47] has experimentally studied the randomness of the original Pollard's walk, and proposed a new one which seems better. Moreover in [46] she presented an algorithm based on this random walk to compute the group structure with $O(1)$ space.

3.4 Possible reductions for some curves

There are two particular families of hyperelliptic curves for which the discrete logarithm problem can be reduced to an easier problem.

The first ones are curves over $\mathbb{F}_q$ such that the subgroup in which we work has order m, with m dividing $q^k - 1$ for some integer k. Then Frey and Rück [14] showed that the Tate pairing allows to reduce the problem to a discrete logarithm problem in $\mathbb{F}_{q^k}^*$. For examples supersingular curves have always a very small k value.

The second class of curves are those whose jacobian over a field of characteristic p has a cyclic group structure of order p^n. Then Rück [32] proved that the discrete logarithm problem can be solved in time $O(n^2 \log p)$.

4 The Adleman–DeMarrais–Huang algorithm

In this section we are given a hyperelliptic curve $\mathcal{C}$ of genus g over a finite field $\mathbb{K}$ of characteristic p different from 2, and we choose an equation for $\mathcal{C}$ of the form $y^2 = f(x)$, with f a square-free polynomial of degree $2g+1$ over $\mathbb{K}$. Let D_1 be a divisor of (prime) order n, and D_2 is a multiple of D_1. We want to compute $\log_{D_1}(D_2)$.

The Adleman–DeMarrais–Huang algorithm is described very precisely in [2]. We only give here a rapid overview of the method in order to show where our improvements occur. The algorithm is an index-calculus like method, so we first recall some $\mathbb{Z}$-linear algebra.

4.1 The Smith Normal Form and finding the group structure

Let G be a finite abelian group, and suppose that we have a set $\{g_1, \ldots, g_w\}$ of elements of G which generates the whole group. Moreover we suppose that we know a large amount of relations between the generators. Then with some $\mathbb{Z}$-linear algebra we can find an *explicit* isomorphism between G and its canonical form $\mathbb{Z}/d_1\mathbb{Z} \oplus \mathbb{Z}/d_2\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/d_n\mathbb{Z}$, where for each i, d_i divides d_{i+1}.

This technique is well known and widely described in Cohen's book [8]. We recall the principles. If A is the free $\mathbb{Z}$-module of rank w with basis $e_1, \ldots, e_w$, each relation between the g_i can be viewed as an element of the kernel of the morphism $\theta : A \to G$, defined by $\theta(e_i) = g_i$ for all i. If we have enough relations (say m), then they generate the whole kernel K and we have an explicit isomorphism between A/K and G. The $\mathbb{Z}$-module A/K can be expressed as a $m \times w$ matrix M, and the heart of the algorithm is to find the *Smith normal form* (SNF) of the matrix, i.e. matrices P in $\mathrm{GL}_m(\mathbb{Z})$, Q in $\mathrm{GL}_w(\mathbb{Z})$ and D a diagonal $m \times w$ matrix such that $PMQ = D$, the diagonal elements of D being the d_i's. Those matrices can be computed with an algorithm which imitates Gauss elimination where the inversions are replaced by GCD's.

Returning to a discrete logarithm context, if we can express the two elements D_1 and D_2 of the group with the generators g_i, then we can use the explicit isomorphism to express them in $\mathbb{Z}/d_1\mathbb{Z} \oplus \mathbb{Z}/d_2\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/d_n\mathbb{Z}$ in which the discrete logarithm is easy to compute (it is only a matter of solving some congruences). The problem can then be reduced to find a set of generators g_i such that relations between them are easy to find and that every element of the group can be expressed with them.

4.2 The smooth divisors basis

In our case, the group is the Jacobian of the hyperelliptic curve $\mathcal{C}$. The set of generators for this group will be the so-called unramified prime divisors of degree at most S. Let us define these terms.

With the polynomial representation $D = [u(z), v(z)]$, a (semi-reduced) divisor will be said to be *prime* if the polynomial u is irreducible over $\mathbb{K}$. If the divisor is not prime, then $u(z) = u_1(z)u_2(z)$, and we can find v_1 and v_2 such that $D = [u_1(z), v_1(z)] + [u_2(z), v_2(z)]$. Hence each divisor is a sum of prime divisors. Now a prime divisor $[u(z), v(z)]$ is called *ramified* if the polynomial v is zero; or what is equivalent, the prime divisor is of order 2.

A semi-reduced divisor will be called S-smooth if it is the sum of unramified primes divisors of degree at most S. Now the algorithm proceeds in two steps: find some relations between the

elements of this basis (the unramified prime divisors of degree at most S), and secondly, if we want to get the discrete logarithm of D_2 to the base D_1, find a representative in the classes of D_1 and D_2 which is S-smooth. The logarithm is then given by solving the $\mathbb{Z}$-linear algebra and some congruences.

4.2.1 Finding a relation

A relation between elements of the basis can be viewed as a semi-reduced S-smooth divisor which is in the class of zero, i.e. which is principal. The idea of the algorithm is to select "random" functions on the curve and to check whether their divisor is S-smooth.

As a matter of fact, one randomly chooses two polynomials A and B in $\mathbb{K}[x]$ of degree at most S', and checks that B is prime to A and f. Then $\varphi_{A,B}(x,y) = A(x)y + B(x)$ is a function on the curve, and let $D = \mathrm{div}(\varphi_{A,B})$ be its divisor. One can show that it is semi-reduced, and that a sufficient and necessary condition for it to be S-smooth is that the polynomial $N(\varphi_{A,B})(x) = A(x)^2 f(x) - B(x)^2$ factors over $\mathbb{K}[x]$ into polynomials of degree bounded by S. The details on how to recover the exact relation are given in [2].

The important point to note is that we have to factor a polynomial of degree $2g + 1 + 2S'$, and we get a relation if it is S-smooth.

4.2.2 Decomposing a divisor on the basis

The decomposition of the two divisors D_1 and D_2 on the basis is quite complicated: we have to rewrite the ramified part of them into a linear combination of unramified prime divisors, and then, for each prime divisor which is not S-smooth, we try to find an expression linearly equivalent to it with only S-smooth prime divisors. Here again we refer the reader to [2] to get the details. We only retain that the complexity of this part is approximatively the same as the complexity of finding *one* relation in the matrix, so we can forget its cost.

4.3 Analysis of the algorithm

The exact complexity of this algorithm is not easy to evaluate. The parameters S and S' have to be chosen in order to minimize the global time. They have to be small enough to obtain a reasonable matrix for which we can compute a Smith form, and they have to be large enough to find the relations (i.e. the rows of the matrix) quickly.

Under certain assumptions Adleman et al. get the subexponential complexity $L_{p^{2g+1}}[1/2, c]$, with c a constant depending of the linear algebra complexity ($c \leq 2.181$).

For example, let g be equal to 6. We can try the algorithm with the smallest basis and take $S = 1$. Then in order to find a relation, we will have to factor a polynomial of degree $2g + 1 + 2S' = 2S' + 13$, and hope that it splits completely, which occurs with probability approximatively $1/(2S' + 13)! \leq 2^{-40}$. And even for such a small genus we will have to deal with a larger basis.

5 A faster algorithm

We propose some ideas providing a significant speed-up of the computation. The first one allows to find relations between elements of the basis much faster than the original technique. The second one is based on the assumption that we know the cardinality of the group and is a mix between the ρ method and the index calculus method. This algorithm finds a solution to the discrete logarithm faster than the pure ρ method as soon as the genus is greater than 4.

5.1 A first improvement of the algorithm

Suppose we have already fixed the smoothness limit S. Then the basis can be written $\{g_1, \ldots, g_w\}$, and the first step of the algorithm is to find relations between those elements. A better way

to do that is to proceed as follows: For each g_i in the basis, we search the smallest $k > g$ (the genus of the curve) such that kg_i reduced by the Cantor algorithm is a S-smooth divisor.

The advantage comes from the fact that for a reduced divisor $[u(z), v(z)]$, the degree of $u(z)$ is bounded by g. And then the probability of this polynomial to have all its irreducible factors of degree at most S is much higher than for a polynomial of degree $2g + 1 + 2S'$.

If we apply this to each g_i in the basis, we get w rows in the matrix. However the Smith form often requires much more relations; but we can obtain other ones by searching another k (not a multiple of the first one) such that kg_i is S-smooth.

Let us return to the example of genus 6, where we try to take $S = 1$. Then we can estimate that the smallest convenient k is $6! = 720$ on average (see next section for a precise analysis). Thus the average cost for obtaining a relation is the cost of 720 additions of reduced divisors on the jacobian and 720 tests of smoothness of a degree 6 polynomial. This is very feasible in practice, even for thousands of relations.

5.2 New algorithm if the cardinality of J is known

We keep the notations of the previous section, but we do not assume that the characteristic is odd. By the Pohlig-Hellman algorithm we can suppose that the order n of D_1 is a prime. Note that if n is composite, the algorithm still works, but it can happen that we can not invert a nonzero element. Then the algorithm is interrupted, but we get a non trivial factor of n, and we can run again the algorithm for the two subgroups that we have obtained.

Following the ADH algorithm, we define a basis of smooth prime divisors with a maximal degree S. Now we introduce a pseudo-random walk (see [47]) in the subgroup generated by D_1: Let $R_0 = \alpha_0 D_1 + \beta_0 D_2$ be the starting point of the walk, where R_0 is the reduced divisor obtained by Cantor's algorithm, and α_0 and β_0 are random integers. For j from 1 to r, compute random divisors $T^{(j)} = \alpha^{(j)} D_1 + \beta^{(j)} D_2$. The walk will then be given by $R_{i+1} = R_i + T^{(\mathcal{H}(R_i))}$, where $\mathcal{H}$ is a hash function from the subgroup generated by D_1 to the interval $[1, r]$. This hash function is assumed to have good statistical properties; in practice, it can be given by the last bits in the internal representation of the divisors. Once the initialization is finished, we can compute a new pseudo-random element R_{i+1} at the cost of one addition in the jacobian. Moreover at each step we get a representation of R_{i+1} as $\alpha_{i+1} D_1 + \beta_{i+1} D_2$, where α_{i+1} and β_{i+1} are integers modulo n.

The classical ρ method is to wait for a collision $R_{i_1} = R_{i_2}$, which will yield the discrete logarithm $\lambda = -(\alpha_{i_1} - \alpha_{i_2})/(\beta_{i_1} - \beta_{i_2}) \bmod n$ (with high probability the denominator is prime to n). We can however make use of the smooth divisors basis. For each R_i of the random walk, test its S-smoothness. If it is smooth, express it on the basis, else throw it away. Thus we extract a subsequence of the sequence (R_i) where all the divisors are smooth. We denote also by (R_i) this subsequence. Hence we can put the result of this computation in a matrix M, each column representing an element of the basis, and each row being a reduced divisor R_i expressed on the basis: for a row i, we have $R_i = \sum_k m_{ik} g_k$, where $M = (m_{ik})$. We collect $w + 1$ rows in order to have a $(w + 1) \times w$ matrix. Thus the kernel of the transpose of M is of dimension at least 1. Using linear algebra, we find a non-zero vector of this kernel, which corresponds to a relation between the R_i's. Then we have a family (γ_i) such that $\sum_i \gamma_i R_i = 0$. Going back to the expression of R_i in function of D_1 and D_2, we get: $\sum_i \gamma_i(\alpha_i D_1 + \beta_i D_2) = 0$, and then

$$\lambda = -\frac{\sum_i \gamma_i \alpha_i}{\sum_i \gamma_i \beta_i}.$$

The discrete logarithm is now found with high probability (the denominator is zero with probability $1/n$).

The speed-up comes from three points:

- We can take a smaller maximal degree S for the basis (we can take $S = 1$ for $g \leq 8$), because the search of relations is faster.

- We do not need the second phase of the original algorithm, because we keep a trace of D_1 and D_2 in the whole computation.
- The linear algebra is easier, because we only need a vector of the kernel (over $\mathbb{Z}/n\mathbb{Z}$) instead of a SNF (over $\mathbb{Z}$).

We summarize the algorithm in the figure 1.

Input: A divisor D_1 of a curve of genus g over $\mathbb{F}_q$, of prime order $n = \mathrm{ord}(D_1)$, a divisor $D_2 \in < D_1 >$, and a parameter r.
Output: An integer λ such that $D_2 = \lambda D_1$.

1. Fix the smoothness bound S, and set k to 1.
2. /* Build the basis G */
   For each monic irreducible polynomial u_i over F_q of degree $\leq S$, try to find v_i such that $[u_i, v_i]$ is a divisor of the curve. If there is one solution, store $g_i = [u_i, v_i]$ in G.
3. /* Initialization of the random walk */
   For j from 1 to r, select $\alpha^{(j)}$ and $\beta^{(j)}$ at random in $[1..n]$, and compute $T^{(j)} := \alpha^{(j)} D_1 + \beta^{(j)} D_2$.
   Select α_0 and β_0 at random in $[1..n]$ and compute $R_0 := \alpha_0 D_1 + \beta_0 D_2$.
4. /* Main loop */
   (a) /* Look for a smooth divisor */
   Compute $j := \mathcal{H}(R_0)$, $R_0 := R_0 + T^{(j)}$, $\alpha_0 := \alpha_0 + \alpha^{(j)} \mod n$, and $\beta_0 := \beta_0 + \beta^{(j)} \mod n$.
   Repeat this step until $R_0 = [u_0(z), v_0(z)]$ is a S-smooth divisor.
   (b) /* Express R_0 on the basis G */
   Factor $u_0(z)$ over $\mathbb{F}_q$, and determine position of the factors in the basis G.
   Store the result as a row $R_k = \sum m_{ik} g_i$ of a matrix $M = (m_{ik})$.
   Store the coefficients $\alpha_k = \alpha_0$ and $\beta_k = \beta_0$.
   If $k < \#G + 1$, then set $k := k + 1$, and return to step 4.a.
5. /* Linear algebra */
   Find a non zero vector (γ_k) of the kernel of the transpose of the matrix M. The computation can be done in the field $\mathbb{Z}/n\mathbb{Z}$.
6. /* Solution */
   Return $\lambda = -(\sum \alpha_k \gamma_k)/(\sum \beta_k \gamma_k) \mod n$. (If the denominator is zero, return to step 3.)

Figure 1: Discrete log algorithm

5.3 The search for relations

During the first phase of the algorithm most of the divisors in the random walk will not be smooth, and we would like to have a fast way to detect this. The natural method is to start a factorization of the polynomial and to interrupt it as soon as we are sure that there is a big irreducible factor. Therefore we compute first the so-called square-free factorization, and then we pass the result to the DDF algorithm which recognizes the factors of degree 1, then 2, and so on. When we have reached the degree S (the smoothness bound), we know whether or not the divisor is smooth and we do not continue the factorization if the divisor is bad. We refer the reader to [39] for further details and optimizations on polynomial factorization.

The next step consists only in finishing the factorization and some research and storage in the memory. It happens only one time over $g!$, and needs not to be optimized in a first implementation.

When the size of the field and the genus grow, we would like to have a better criterion for stopping when the divisor is not smooth. The use of the discriminant may be fruitful. Indeed Swan [44] showed that the number of irreducible factors of a polynomial is even or odd depending on the degree of the polynomial and whether or not its discriminant is a quadratic residue. This knowledge has the cost of a quadratic residue test, and gives partial information. If this information shows that the divisor is not smooth, we have a speed-up compared to the DDF of degree 1, but if we can not conclude, we lose some time. So we have to be very careful in the implementation. In ours, we obtained a speed up of 30 to 40% for the smoothness test in odd characteristic, but no improvement if the characteristic is 2.

A last point which is worth to be noticed is that the search of relations can be highly parallelized: if we have N computers the time to obtain the required number of rows in the matrix is divided by N. This can be very useful for relatively high genus (greater than 8), where it allows to keep a smoothness bound equal to 1.

5.4 Solving the linear algebra problem

Once we have built the $(w+1) \times w$ matrix M (where w is the size of the basis), we have to find a vector of the kernel of the transpose of M. This can be done by a Gaussian elimination, thus requiring about w^3 field operations. We can accelerate the computation in two ways: compute modulo the prime n and take advantage of the sparse structure of the matrix.

Proposition 1 *We suppose that the integer n which is the order of the divisor D_1 which is the base of the searched discrete logarithm is a prime and that n^2 does not divide the cardinality of the jacobian. Then all the operations on the coefficients of the matrix can be done in the field $\mathbb{Z}/n\mathbb{Z}$. (Even if some elements of the basis are not of order n).*

Proof: Let $\#Jac(\mathcal{C}) = nn_2$, with n_2 prime to n. We take the same notations as in the previous section. The matrix $M = (m_{ik})$ is defined by $R_k = \sum_i m_{ik} g_i$ for all k. Suppose that we have solved the linear algebra modulo n. Then let $(\gamma_1, \ldots, \gamma_{w+1})$ be a non-zero vector of the kernel of the transpose of M modulo n, i.e. for all i we have $\sum_k \gamma_k m_{ik} \equiv 0 \mod n$. The γ_k have value in $\mathbb{Z}/n\mathbb{Z}$, and we consider that they are integers between 0 and $n-1$. Then

$$\sum_k \gamma_k R_k = \sum_k \gamma_k \sum_i m_{ik} g_i = \sum_i \left(\sum_k \gamma_k m_{ik} \right) g_i.$$

Now multiplying both part of this equation by n_2, we obtain $n_2(\sum_k \gamma_k R_k) = \sum_i n_2(\sum_k \gamma_k m_{ik}) g_i$. For each i, the coefficient before g_i is zero modulo $n_2 n$, which is the order of the jacobian, thus the right member is zero. We deduce that $\sum_k \gamma_k R_k$ is a divisor of order dividing n_2. However, each R_k belong to the subgroup generated by D_1, then it is of order dividing n. Then $\sum_k \gamma_k R_k = 0$ in $Jac(\mathcal{C})$, and the solution modulo n gives a solution to our problem. □

Proposition 2 *The matrix built by the above algorithm has at most g non-zero terms in each row. Then the computation of a vector of the kernel can be performed with a probabilistic algorithm requiring approximatively gw^2 fields operations.*

Proof: Each row of the matrix comes from a reduced divisor. It has therefore a degree bounded by g, and is the sum of at most g prime divisors. Hence the total number of non-zero terms in the matrix is gw and we can use a sparse linear algebra algorithm such that Wiedemann's [49] or Lanczos's [22], thus obtaining a solution in approximatively gw^2 fields operations. □

The problem of sparse linear algebra occurs in some algorithms of number theory, mainly the factorization of integers, and the discrete logarithm over finite fields. In factorization, we have to find a relation between rows over $\mathbb{F}_2$, and in the discrete logarithm, we want to find the kernel of a matrix over a prime field. In the first case, to have too many rows increases the size of the kernel of the transpose, and helps to find one element of this kernel, and in the second case, it reduces the dimension of the kernel to 1, and eliminates the possible errors.

In our case, we are in the context of factorization where we want a relation between the rows, but we work over $\mathbb{F}_n$, which is a new problem. Moreover the non zero terms are equidistributed in each row, and we can not use some tricks which were conceived for the matrix coming for factorization or discrete log (very dense in a part and very sparse on the other). We refer the reader to [22] for an overview of the techniques.

However we can hope to obtain a gain by using the so-called "Structured Gaussian Elimination" (SGE) before performing Lanczos's algorithm. It consists in eliminating some rows and columns just like in a classical Gaussian elimination, but with a good choice of the pivot (see [27], [31]). The criterion for a pivot to be good is to minimize the augmentation of the weight of the matrix after elimination. After each step of this technique, we can evaluate the cost of the Lanczos algorithm as if it were run on the matrix. We stop for a value of this cost which seams minimal.

At the beginning of the SGE, we can eliminate all the columns with zero terms. Then we eliminate the columns with exactly one term as well as the corresponding rows. In order to evaluate the efficiency of the SGE, we give an estimation of the number of these nice columns.

Proposition 3 *Let M be a sparse matrix with w columns and $w+1$ rows, with exactly k terms in each rows. Then the average number of columns with exactly l terms is asymptotically*

$$n_l(k,w) = \frac{wk^l e^{-k}}{l!}.$$

6 Analysis of the complexity for small genus

We analyse our algorithm for a *fixed* genus g, and we let q tend to infinity. It does not depend on the characteristic of the base field.

6.1 Probability for a divisor to be S-smooth

We first have to analyse the probability for a random reduced divisor to be S-smooth (for small genera, the S parameters will be 1 or 2).

For a curve $\mathcal{C}$ of genus g over $\mathbb{F}_q$, a reduced divisor on the jacobian can be written $D = P_1 + P_2 + \cdots + P_g - g(\infty)$, where the P_i are points of $\mathcal{C}$ defined over $\mathbb{F}_{q^k}$, with $k \leq g$. It is easy to see that D is S-smooth if and only if all the P_i are defined over $\mathbb{F}_{q^S}$.

Proposition 4 *Let C_k denote the limit when q tends to infinity of the average proportion of*

k-smooth reduced divisors for a curve $\mathcal{C}$ of genus g over $\mathbb{F}_q$. Then

$$\begin{aligned} C_1 &= 1/g!, \\ C_2 &= \sum_{i=0}^{[g/2]} \frac{1}{2^i i!(g-2i)!}, \\ C_{g-1} &= 1 - 1/g, \\ C_g &= 1. \end{aligned}$$

Proof: This proposition is based on the Hasse-Weil bound for algebraic curve: the number of points of a curve of genus g over a finite field with q elements is equal to $q+1$ with an error of at most $2g\sqrt{q}$, i.e. for large enough q we can neglect it. Moreover the cardinality of its jacobian is equal to q^g with an error bounded by approximatively $2gq^{g-\frac{1}{2}}$. Here the approximation holds when q is sufficiently large compared to $4g^2$, which is the case in cryptographical applications.

With these approximations we can prove the proposition. The last equality $C_g = 1$ is trivial. To compute C_{g-1}, we evaluate the number of points of the curve defined over $\mathbb{F}_{q^g}$ and not on $\mathbb{F}_{q^{g-1}}$: they are about $q^g - q^{g-1} \sim q^g$. For each of these points one can form a divisor on the jacobian by adding all its conjugates by the Frobenius map. Thus we obtain about q^g/g divisors which are not g-smooth. Hence $C_{g-1} = \frac{q^g - q^g/g}{q^g} = 1 - 1/g$.

The computation of C_1 is obtained directly from the number of points of the curve over $\mathbb{F}_q$ which is approximatively q. Then the 1-smooth divisors of the jacobian are in bijection with the g-multiset of points of the curve: we have $q^g/g!$ 1-smooth divisors, and $C_1 = 1/g!$. For C_2, we count for each i the number of divisors having i pairs of conjugate points defined over $\mathbb{F}_{q^2}$ and $g-2i$ points defined over $\mathbb{F}_q$: they are $\frac{1}{i!}\left(\frac{q^2}{2}\right)^i \frac{q^{g-2i}}{(g-2i)!}$, and we obtained the announced formula for C_2. □

We could get similar formulae for C_3, C_4, ... but we will not need them for small genera, since we always take $S = 1$ or $S = 2$. We give some values of the inverse of C_1 and C_2 for small genera. We take the inverse because it is the average number of iterations before finding a smooth element.

g	2	3	4	5	6	7	8
C_1^{-1}	2	6	24	120	720	5,040	40,320
C_2^{-1}	1	1.5	2.4	4.6	9.5	22	53
g	9	10	11	12	13	14	15
C_1^{-1}	$3.6\ 10^5$	$3.6\ 10^6$	$4\ 10^7$	$5\ 10^8$	$6\ 10^9$	$9\ 10^{10}$	$1.3\ 10^{12}$
C_2^{-1}	139	382	1,118	3,418	10,953	36,469	126,351

For a cryptanalysis of a genus g cryptosystem, we can then evaluate the timings for filling the matrix. If $S = 1$, then the number of columns is about q, so we need to compute about q rows each of them requiring on average C_1^{-1} additions on the jacobian and tests for smoothness. If C_1^{-1} is too high (probably for $g \geq 9$ or 10 depending on the size q), we can take $S = 2$, thus dealing with a matrix of size about q^2, and requiring $q^2 C_2^{-1}$ additions on the jacobian and tests for smoothness.

6.2 Complexity and comparison with the rho method

We consider that the smoothness bound is set to $S = 1$, because for a fixed value of g and a sufficiently large value of q, this is always the best choice. The complexity decomposes as follows:

1. Building the basis: $O(q)$ square root computations in $\mathbb{F}_q$.
2. Computing of the relations: $O(g!q)$ additions in the jacobian and tests for smoothness, and $O(q)$ factorizations of a polynomial of degree g over $\mathbb{F}_q$.

3. Computation of one vector of the kernel: $O(gq^2)$ operations in a field of size $n \approx q^g$.

Square root computation in $\mathbb{F}_q$ can be done in random time polynomial in $\log q$, and addition in the jacobian requires about g^3 operations in the base field which is $\mathbb{F}_q$. In the case $S = 1$, the test for smoothness is the first phase of DDF (more details are given below), thus requires a time $O(g^2 \log^\alpha q)$. And the full factorization of a smooth polynomial can be done by evaluating it at all the elements of $\mathbb{F}_q$, thus requiring $O(gq \log^\beta q)$ operations.

Thus the complexity of the first phase of the algorithm is $O(g^3 g! q \log^\gamma q) + O(g^2 q^2 \log^\gamma q)$. The linear algebra requires $O(g^3 q^2 \log^\gamma q)$, then the whole algorithm runs in time $O(g^3 g! q \log^\gamma q) + O(g^3 q^2 \log^\gamma q)$. If the genus g is *fixed*, then we can put away the functions in g out of the $O()$, and get the following complexity.

Theorem 1 *For fixed g, our algorithm takes time $O(q^2 \log^\gamma q)$.*

In order to compare this result to the rho method, we denote by $N = q^g$ the cardinality of the group (we suppose that N is almost prime). Then our algorithm has a complexity $O(N^{2/g} \log^\gamma N)$. The complexity of the rho method is $O(N^{1/2} \log^\delta N)$. Thus our algorithm is asymptotically faster as soon as g is greater than 4. Note that this theoretical result is not only asymptotic: as we will see below, even for genus 5 we can compute discrete logarithms for bigger cryptosystems than could be done with the ρ method.

7 Using automorphisms on the curve

In this section we study the speed up obtained by using an automorphism of the curve. For the ρ method, we can have a gain by a factor $\sqrt{m}$, where m is the order of the automorphism (see [10]). In the smooth-divisors attack, there is a way to gain a speed-up by a factor m, and even m^2. This becomes very significant in practice.

The first idea is simple: each time we found a relation, we can get $m-1$ others by applying the automorphism. But this does not reduce the time for the linear algebra, which is the limiting phase. A better way to use the automorphism is to reduce the size of the basis. Instead of considering the set of S-smooth divisors, we only keep in the basis one representative for each orbit under the action of the automorphism. Thus the size of the basis is reduced by a factor m, so the necessary number of relations is reduced by the same factor, and the linear algebra phase is speeded up by a factor m^2. Let us explain how it works.

For the moment, suppose that the jacobian is cyclic of prime order $n = \mathrm{ord}(D_1)$, and denote by σ an automorphism of order m on $Jac(\mathcal{C})$. Then $\sigma(D_1)$ belongs to $Jac(\mathcal{C}) =< D_1 >$, and there exists an integer θ such that $\sigma(D_1) = \theta D_1$. Moreover, σ being an automorphism, for all $D \in Jac(\mathcal{C})$, $D = kD_1$ and we have $\sigma(D) = \sigma(kD_1) = k\sigma(D_1) = k\theta D_1 = \theta D$.

Another useful fact is that a reduced prime divisor is sent by σ on another prime divisor *of the same degree*. Hence an element of the basis G is sent to another element of the basis.

Suppose now that we have only kept in the basis one element for each orbit under σ. Let $R = P_1 + P_2 + \cdots + P_k = \alpha D_1 + \beta D_2$ be the decomposition of a smooth divisor into prime divisors of degree bounded by S. For each i, there is a power of σ such that the prime divisor P_i is equal to $\sigma^{l_i}(g_i)$, where g_i is an element of the basis. Then we can write $R = \theta^{l_1}(g_1) + \cdots + \theta^{l_k}(g_k)$, and we have a relation in a matrix with m times less columns than the original one. The expression of a divisor on the reduced basis requires to compute the action of the automorphism on a few divisors, and we have to suppose that this can be done quickly, which is the case for practical examples.

For the general case where the jacobian is not cyclic and where we work in a subgroup of prime order n, we can do exactly the same computations, provided that n^2 does not divides the cardinality of the jacobian. In fact a proposition similar to proposition 1 would prove that the result is not disturbed by the apparently wrong definition of θ.

8 Timings and prospects

We have implemented our algorithm in the **Magma** computer algebra system version 2.3, which allowed us to program quickly and efficiently. It is probable that we can obtain a faster implementation by using the C language and a number theory package, but our timings are significant enough to illustrate our purpose. The linear algebra is far from being optimized, in particular the timings for the SGE are hoped to be improved.

First of all we give the size of cryptosystems breakable with a basis of given size. We suppose that we use only the hyperelliptic involution, and then $\#G = q/2$. We have now $\#J = q^g = (2\#G)^g$, and get the following values of breakable jacobians.

$\#G \backslash g$	4	5	6	7	8
$100,000$	2^{70}	2^{88}	2^{105}	2^{123}	2^{140}
$1,000,000$	2^{83}	2^{104}	2^{125}	2^{146}	2^{167}

8.1 Some examples

The timings are given on a Pentium II 450 MHz with 128 Mb. In the tables 1, 2,3 we give some examples which were built for testing our algorithm. They are binary curves over $\mathbb{F}_{2^{19}}$ of various genera, and we made use of the Frobenius automorphism. We gave in brackets the size of the basis that we would have to use without automorphism.

field	$\mathbb{F}_{2^{19}}$
equation	$y^2 + (x+1)y = x^9 + x^7 + x^6 + x^5 + x^4 + x^3 + 1$
genus	4
auto	2×19
$\#J$	$2 \times 3 \times 126286575304949459871 91 \approx 2^{76}$
basis	13837 (262885)
matrix	30000×13837

build basis	145 s
search relations	2 h
average wait	23.88
SGE	14 h
Lanczos	2 h
Total	18 h

Table 1: Genus 4 cryptosystem

In table 4, we give a *real life* example: this cryptosystem was proposed by Koblitz in [20] and more recently in [21]. For the computation of the relations, we used several computers in parallel.

8.2 Perspectives

There are some other cryptosystems that fall under our attack.

The curve $y^2 + y = x^{13}$, proposed by Buhler and Koblitz in [4] seems to be feasible: in order to have a jacobian of size about 10^{40} (recommended by the authors), we need a prime field with $p \approx 5,000,000$, which leads to a basis of size $185,000$, due to the complex multiplication automorphism $(x, y) \mapsto (\zeta_{13}x, -y)$.

A more difficult example is given by Sakai–Sakurai [33] and Smart [42]: the curve $y^2 + y = x^{13} + x^{11} + x^9 + x^5 + 1$ over $\mathbb{F}_{2^{29}}$, which has an automorphism of order 4×29. The jacobian is

field	$\mathbb{F}_{2^{19}}$
equation	$y^2 + (x+1)y = x^{11} + x^9 + x^8 + x^7 + x^4 + x + 1$
genus	5
auto	2×19
$\#J$	$2^5 \times 1236507505106065358922109709 \approx 2^{95}$
basis	13782 (261840)
matrix	30000×13782

build basis	145 s
search relations	3.5 h
average wait	119.7
SGE	15.5 h
Lanczos	7 h
Total	26 h

Table 2: Genus 5 cryptosystem

field	$\mathbb{F}_{2^{19}}$
equation	$y^2 + (x+1)y = x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$
genus	6
auto	2×19
$\#J$	$2 \times 23 \times 45660353558951618050075673976938 3 \approx 2^{114}$
basis	13953 (265089)
matrix	30000×13953

build basis	145 s
search relations	9 h
average wait	686.9
SGE	16 h
Lanczos	17 h
Total	42 h

Table 3: Genus 6 cryptosystem

of size $\approx 2^{174}$, and breaking this system requires to deal with a basis of size $\approx 4,500,000$. Note that the search for relations is not a problem, because we can use lots of computers. But the resolution of the huge resulting linear system is still problematic.

9 Some final remarks and open questions

In the analysis, we said that we can perform the EDF by checking every elements in the field, which leads to a complexity $O(q)$. Of course, in practice we use a modern factorization technique, and the complexity looks like polynomial in $\log q$. Hence for real computations, we can consider that the search of relations is in time $O(q)$, and the linear algebra in $O(q^2)$.

There is still some work to do to find a good balance between the amount of relations, and the gain obtained in the linear algebra. The first idea is to restrict the basis by keeping in it only an arbitrary fraction of the elements. More precisely, if we reduce the size of the basis by a factor n, then the probability to find a smooth divisor on this basis is reduced by a factor n^g. This seams to be rather drastic, but as Robert Harley pointed out to me, for $g = 4$, we can take

field	$\mathbb{F}_{133349}$
equation	$y^2 + y = x^{17}$
genus	8
auto	2×17
$\#J$	$17^3 \times 203971815893327943087423276587833460 51 \approx 2^{136}$
basis	3932 (66828)
matrix	4012×3932

build basis	50 s
search relations	8.5 d
SGE	30 m
Lanczos	5 h
Total	9 d

Table 4: Koblitz's cryptosystem

$n = q^{1/5}$, thus obtaining a (unproven) global complexity of $O(q^{8/5})$, which is better than the rho method.

Another way is to keep the whole basis, collect a large amount of relations, and then select the best elements that we want to keep in the basis. By this way we will probably improve the n^g factor, and get a faster algorithm, with an unproven better complexity.

Another open question about speeding up the linear algebra: the terms in the matrix are plus or minus 1 (and plus or minus a power of m-th root of unity modulo n if we use automorphisms). Can we take advantage of this very strange fact during the search of an element of the kernel?

In the case of genus 2, the complexity of the Pollard's rho method is $O(q)$. If we use our algorithm, we get a complexity of $O(q^2)$, which can be reduced to $O(q^{4/3})$ by reducing the basis by a factor $n = q^{1/3}$. More tricks can be played and must be tested. We will explain this elsewhere.

In order to increase the practical efficiency of our algorithm, standard techniques of large prime variation may be applicable. And even the double large prime variation [23] could be fruitful. Likely this will lead to significant speed-up for curves of rather large genus (about 10 or 11), where it will allow to keep the smoothness bound to one. Here again many tests have to be done.

The general machinery described in this paper, i.e. mixing a random walk and a smoothness test, will certainly be applicable for other discrete logarithm problems based on index calculus. The case of $\mathbb{F}_{q^g}^*$ is currently being studied [17]; it is still not clear whether the expected running time will be better than Adleman and DeMarrais's algorithm [1] or Semaev's [36].

Acknowledgements

I am most indebted to François Morain for many fruitful discussions and comments. I would like to thank Emmanuel Thomé, particularly for his help for linear algebra. I am also grateful to Robert Harley and people from the Action Courbes (particularly Daniel Augot) for many discussions concerning this work.

References

[1] L. M. Adleman and J. DeMarrais. A subexponential algorithm for discrete logaritmhs over all finite fields. *Math. Comp.*, 61(203):1–15, July 1993.

[2] L. M. Adleman, J. DeMarrais, and M.-D. Huang. A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields. In L. Adleman and M.-D. Huang, editors, *ANTS-I*, volume 877 of *Lecture Notes in Comput. Sci.*, pages 28–40. Springer–Verlag, 1994. 1st Algorithmic Number Theory Symposium - Cornell University, May 6–9, 1994.

[3] L. M. Adleman and M.-D. A. Huang. Recognizing primes in random polynomial time. In *Proceedings 19th STOC*, pages 462–469. ACM Press, 1986. New-York City, May 25–27, 1987.

[4] J. Buhler and N. Koblitz. Lattice basis reduction, Jacobi sums and hyperellitic cryptosystems. *Bull. Austral. Math. Soc.*, 58:147–154, 1998.

[5] D. G. Cantor. Computing in the Jacobian of an hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.

[6] D. G. Cantor. On the analogue of the division polynomials for hyperelliptic curves. *J. Reine Angew. Math.*, 447:91–145, 1994.

[7] J. Chao, N. Matsuda, O. Nakamura, and S. Tsujii. Cryptosystems based on cm abelian variety. In *Proc. Symposium on Cryptography and Information Security*, 1997.

[8] H. Cohen. *A course in algorithmic algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer–Verlag, 1993.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22-6, November 1976.

[10] I. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. Preprint.

[11] I. Duursma and K. Sakurai. Efficient algorithms for the jacobian variety of hyperelliptic curves $y^2 = x^p - x + 1$ over a finite field of odd characteristic p. In xxx, editor, *Proceedings of the "International Conference on Coding Theory, Cryptography and Related Areas"*, volume yyy of *Lecture Notes in Comput. Sci.*, page zzz, 1999. Guanajuato, Mexico on April, 1998.

[12] A. Enge. Computing discrete logarithms in high-genus hyperelliptic jacobians in provably subexponential time. Preprint; available at `http://www.math.uwaterloo.ca/CandO_Dept/CORR/corr99.html`, 1999.

[13] R. Flassenberg and S. Paulus. Sieving in function fields. Preprint; available at `ftp://ftp.informatik.tu-darmstadt.de/pub/TI/TR/TI-97-13.rafla.ps.gz`, 1997.

[14] G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Math. Comp.*, 62(206):865–874, April 1994.

[15] W. Fulton. *Algebraic curves*. Math. Lec. Note Series. W. A. Benjamin Inc, 1969.

[16] S. D. Galbraith and N. Smart. A cryptographic application of Weil descent, 1999. Preprint.

[17] P. Gaudry. On the discrete log problem over $\mathbb{F}_{p^n}^*$, 1999. Work in progress.

[18] R. Hartshorne. *Algebraic Geometry*, volume 52 of *Graduate Texts in Mathematics*. Springer–Verlag, 1977.

[19] N. Koblitz. Hyperelliptic cryptosystems. *J. of Cryptology*, 1:139–150, 1989.

[20] N. Koblitz. A family of jacobians suitable for discrete log cryptosystems. In S. Goldwasser, editor, *Advances in Cryptology – CRYPTO '88*, volume 403 of *Lecture Notes in Comput. Sci.*, pages 94–99. Springer–Verlag, 1990. Proceedings of a conference on the theory and application of cryptography held at the University of California, Santa Barbara, August 21–25, 1988.

[21] N. Koblitz. A very easy way to generate curves over prime fields for hyperelliptic cryptosystems. Rump-session Crypto'97, 1997.

[22] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology*, volume 537 of *Lecture Notes in Comput. Sci.*, pages 109–133. Springer–Verlag, 1990. Proc. Crypto '90, Santa Barbara, August 11–15, 1988.

[23] A. K. Lenstra and M. S. Manasse. Factoring with two large primes. *Math. Comp.*, 63(208):785–798, October 1994.

[24] A. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curves logarithms to logarithms in a finite field. *IEEE Trans. Inform. Theory*, 39(5):1639–1646, September 1993.

[25] V. Müller, A. Stein, and C. Thiel. Computing discrete logarithms in real quadratic congruence function fields of large genus. *Math. Comp.*, 68(226):807–822, 1999.

[26] D. Mumford. *Tata lectures on theta II*, volume 43 of *Progress in Mathematics*. Birkhauser, 1984.

[27] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In Thomas Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Comput. Sci.*, pages 224–314. Springer–Verlag, 1985. Proceedings Eurocrypt '84, Paris (France), April 9–11, 1984.

[28] J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comp.*, 55(192):745–763, October 1990.

[29] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inform. Theory*, IT–24:106–110, 1978.

[30] J. M. Pollard. Monte Carlo methods for index computation mod p. *Math. Comp.*, 32(143):918–924, July 1978.

[31] C. Pomerance and J. W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experiment. Math.*, 1(2):89–94, 1992.

[32] H. G. Rück. On the discrete logarithm in the divisor class group of curves. *Math. Comp.*, 68(226):805–806, 1999.

[33] Y. Sakai and K. Sakurai. Design of hyperelliptic cryptosystems in small charatcteristic and a software implementation over $\mathbb{F}_{2^n}$. In K. Ohta and D. Pei, editors, *Advances in Cryptology*, volume 1514 of *Lecture Notes in Comput. Sci.*, pages 80–94. Springer–Verlag, 1998. Proc. Asiacrypt '98, Beijing, October, 1998.

[34] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Comment. Math. Helv.*, 47(1):81–92, 1998.

[35] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp.*, 44:483–494, 1985.

[36] I. A. Semaev. An algorithm for evaluation of discrete logarithms in some nonprime finite fields. *Math. Comp.*, 67(224):1679–1689, October 1998.

[37] I. A. Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curves in characteristic p. *Math. Comp.*, 67(221):353–356, January 1998.

[38] D. Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math. vol. 20*, pages 415–440. AMS, 1971.

[39] V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symbolic Comput.*, 20:363–397, 1995.

[40] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Comput. Sci.*, pages 256–266. Springer–Verlag, 1997. International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 1997, Proceedings.

[41] N. Smart. The discrete logarithm problem on elliptic curves of trace one. Preprint HP-LABS Technical Report (Number HPL-97-128). To appear in *J. Cryptology*, 1997.

[42] N. Smart. On the performance of hyperelliptic cryptosystems. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Comput. Sci.*, pages 165–175. Springer–Verlag, 1999. International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 1999, Proceedings.

[43] A.-M. Spallek. *Kurven vom Geschlecht 2 und ihre Anwendung in Public-Key-Kryptosystemen*. PhD thesis, Universität Gesamthochschule Essen, July 1994.

[44] R. G. Swan. Factorization of polynomials over finite fields. *Pacific J. Math.*, 12:1099–1106, 1962.

[45] D. C. Terr. A modification of Shanks' baby-step giant-step algorithm. To appear in Math. Comp., 1999.

[46] E. Teske. A space efficient algorithm for group structure computation. *Math. Comp.*, 67(224):1637–1663, 1998.

[47] E. Teske. Speeding up Pollard's rho method for computing discrete logarithms. In J. P. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 541–554. Springer–Verlag, 1998. Third International Symposium, ANTS-III, Portland, Oregon, june 1998, Proceedings.

[48] P. van Wamelen. Examples of genus two CM curves defined over the rationals. *Math. Comp.*, 68(225):307–320, January 1999.

[49] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inform. Theory*, IT–32(1):54–62, 1986.