An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves

Pierrick Gaudry*

LIX, École Polytechnique,
91128 Palaiseau Cedex, France
gaudry@lix.polytechnique.fr

Abstract. We present an index-calculus algorithm for the computation of discrete logarithms in the Jacobian of hyperelliptic curves defined over finite fields. The complexity predicts that it is faster than the Rho method for genus greater than 4. To demonstrate the efficiency of our approach, we describe our breaking of a cryptosystem based on a curve of genus 6 recently proposed by Koblitz.

1 Introduction

The use of hyperelliptic curves in public-key cryptography was first proposed by Koblitz in 1989 [24]. It appears as an alternative to the use of elliptic curves [23] [31], with the advantage that it uses a smaller base field for the same level of security. Several authors have given ways to build hyperelliptic cryptosystems efficiently. The security of such systems relies on the difficulty of solving 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 [39] that the complexity is at least $\Omega(\sqrt{n})$, 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. The Pollard Rho method and its parallel variants are the most important examples [34], [46], [17].

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 [30], [38], [40], [37]. Similar cases were discovered for hyperelliptic curves [14], [35]. However they are very particular and can be easily avoided when designing a cryptosystem.

In 1994, Adleman, DeMarrais and Huang [1] published the first algorithm (ADH for short) to compute discrete logs 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 to it were done. Flassenberg and Paulus [13] implemented a sieve version of this algorithm, but

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

the consequences for cryptographical applications is not clear. Enge [11] improved the original algorithm and gave a precise evaluation of the running time, but did not implement his ideas. Müller, Stein and Thiel [32] 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, following ideas of Frey; they dealt with general curves (not hyperelliptic).

Our purpose is to present a variant of existing index-calculus algorithms like ADH or Hafner-McCurley [19], which allowed us to break a cryptosystem based on a curve of genus 6 recently proposed by Koblitz. The main improvement is due to the fact that the costly HNF computation in classical algorithms is replaced by that of the kernel of a sparse matrix. A drawback is that we have to assume that the order of the group in which we are working is known. This is not a constraint in a cryptographical context, because the knowledge of this order is preferable to build protocols. But from a theoretical point of view it differs from ADH or Hafner-McCurley algorithm where the order of the group was a byproduct of the discrete logarithm computation (in fact the aim of the HNF computation was to find the group structure).

We will analyse our method for small genus and show that it is faster than the Pollard Rho method as soon as the genus is strictly greater than 4. Indeed its complexity is $O(q^2)$ where q is the cardinality of the base field. We will explain below some consequences for the choice of the parameters, curve and base field, when building a cryptosystem.

Moreover, the presence of an automorphism of order m on the curve can be used to speed up the computation, just as in the Rho method [9] [17] [48]. This is the case in almost all the examples in the literature. The gain in the Rho method is a factor $\sqrt{m}$, but the gain obtained here is a factor m^2, which is very significant in practice.

The organization of the paper is as follows: in section 2 after some generalities on hyperelliptic curves, our algorithm is described. It is analyzed in section 3, and in section 4 we explain how the presence of an automorphism can help. Finally the section 5 gives some details on our implementation and the results of our experiments with Koblitz's curve.

2 Description of the Algorithm

2.1 Hyperelliptic Curves

We give an overview of the theory of hyperelliptic curves. More precise statements can be found in [24], [4], [15]. We will restrict ourselves to the so-called imaginary quadratic case.

A *hyperelliptic curve* $\mathcal{C}$ of genus g over a field $\mathbb{K}$ is a smooth plane 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, both with coefficients in $\mathbb{K}$.

A *divisor* on the curve $\mathcal{C}$ 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. It can be shown that the sum is finite, and moreover that the degree of such a divisor is always zero. The set of all divisors built from a function a subgroup of $Div^0(\mathcal{C})$ denoted by $\mathcal{P}(\mathcal{C})$ and we call these divisors *principal*. The *Jacobian* of the curve $\mathcal{C}$ is then defined by the quotient group $Jac(\mathcal{C}) = Div(\mathcal{C})^0/\mathcal{P}(\mathcal{C})$.

If the base field of the curve is a finite field with cardinality q, then the Jacobian of the curve is a finite abelian group of order around q^g. The Hasse-Weil bound gives a precise interval for this order: $(\sqrt{q}-1)^{2g} \leq \#Jac(\mathcal{C}) \leq (\sqrt{q}+1)^{2g}$.

In [4], Cantor gave an efficient algorithm for the computation of the group law. We do not recall his method, but we recall the representation of the elements.

Proposition 1 *In every class of divisors in $Jac(\mathcal{C})$, there exists an unique divisor $D = P_1 + \cdots + P_g - g\infty$, such that for all $i \neq j$, P_i and P_j are not symmetric points. Such a divisor is called* reduced*, and there is a unique representation of D by two polynomials $[u, v]$, such that* $\deg v < \deg u \leq g$*, and u divides $v^2 + hv - f$.*

In this representation, the roots of the polynomial u are exactly the abscissae of the points which occur in the reduced divisor.

The group $Jac(\mathcal{C})$ can now be used in cryptographical protocols based on the discrete logarithm problem, for example Diffie-Hellman or ElGamal's protocols. The security relies on the difficulty of the following problem.

Definition 1 *The* hyperelliptic discrete logarithm problem *takes on input a hyperelliptic curve of given genus, an element D_1 of the Jacobian, its order n, and another element D_2 in the subgroup generated by D_1. The problem is to find an integer λ modulo n such that $D_2 = \lambda.D_1$.*

2.2 Smooth Divisors

Like any index-calculus method, our algorithm is based on the notions of smoothness, and prime elements. We will recall these notions for divisors on hyperelliptic curves, which were first defined in ADH.

Definition 2 *With the polynomial representation $D = [u, v]$, a divisor will be said to be* prime *if the polynomial u is irreducible over $\mathbb{F}_q$.*

For a prime divisor D, when there is no possible confusion with the degree of D as a divisor (which is always zero), we will talk about the degree of D instead of the degree of u.

Proposition 2 *A divisor D of $Jac(\mathcal{C})$ represented by the polynomials $[u, v]$ is equal to the sum of prime divisors $[u_i, v_i]$, where the u_i are the prime factors of u.*

Now we can give the smoothness definition. Let S be an integer called the smoothness bound.

Definition 3 *A divisor is said to be S-*smooth *if all its prime divisors are of degree at most S. When $S = 1$, a 1-smooth divisor will be a divisor for which the polynomial u splits completely over $\mathbb{F}_q$.*

The case $S = 1$ is the most important for two reasons: the first one is that for a relatively small genus (say at most 9), and a reasonable field size, this choice is the best in practice. The second one is that if we want to analyze our algorithm for a fixed g and a q tending to infinity, this is also the good choice.

The definition of a smooth divisor can be seen directly on the expression of D as a sum of points of the curve. Note that a divisor defined over $\mathbb{F}_q$ is defined by being invariant under the Galois action. But it does not imply that the points occuring in it are defined over $\mathbb{F}_q$; they can be exchanged by Galois. Hence an equivalent definition of smoothness is given by the following proposition.

Proposition 3 *A divisor $D = P_1 + \cdots + P_g - g\infty$ is S-smooth if and only if each point P_i is defined over an extension $\mathbb{F}_{q^k}$ with $k \leq S$.*

We define also a factor basis, similar to the one used for classical discrete log problem over $\mathbb{F}_p^*$.

Definition 4 *The* factor basis*, denoted by G_S, is the set of all the prime divisors of degree at most S. For $S = 1$ we simply write G.*

In the following, we will always take $S = 1$ and we will say 'smooth divisor' for 1-smooth divisor.

2.3 Overview of the Algorithm

For the sake of simplicity, we will suppose that the Jacobian of the curve has an order which is almost prime and that we have to compute a discrete log in the subgroup of large prime order (this is always the case in cryptography). Let $n = \mathrm{ord}(D_1)$ be this prime order, and D_2 be the element for which we search the log.

We introduce a pseudo-random walk (as in [45]) 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, we 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 $\lambda = -(\alpha_{i_1} - \alpha_{i_2})/(\beta_{i_1} - \beta_{i_2}) \bmod n$. We can however make use of the smooth divisors. For each R_i of the random walk, test its smoothness. If it is smooth, express it on the factor 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 factor 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 $1/n$).

We summarize this algorithm in the figure 1.

2.4 Details on Critical Phases

In the first step, we have to build the factor basis, and for that, we have to find, if it exists, a polynomial v corresponding to a given irreducible u. This can be rewritten in solving an equation of degree 2 over $\mathbb{F}_q$, which can be done quickly.

The initialization of the random walk is only a matter of operations in the group; after that, computing each random divisor R_i requires a single operation in the group.

One crucial point is to test the smoothness of a divisor, i.e. to decide if a polynomial of degree g (the u of the divisor) splits completely on $\mathbb{F}_q$. A way to do that is to perform the beginning of the factorization of u, which is called DDF (stands for distinct degree factorization). By computing $\gcd(X^q - X, u(X))$, we get the product of all the prime factors of u of degree 1. Thus if the degree of this product is equal to the degree of u, it proves that u splits completely on $\mathbb{F}_q$.

In the case where a smooth divisor is detected, the factorization can be completed, or a trial division with the elements of the basis can be performed.

The linear algebra is the last crucial point. The matrix obtained is sparse, and we have at most g terms in each row. Then sparse technique like Lanczos's [27] or Wiedemann's [47] algorithm can be used, in order to get a solution in time quadratic in the number of rows (instead of cubic by Gaussian elimination).

Some other optimizations can be done to speed up the computation. They will be described in section 5.

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

1. /* Build the factor basis G */
   For each monic irreducible polynomial u_i over F_q of degree 1, try to find v_i such that $[u_i, v_i]$ is a divisor of the curve. If there is a solution, store $g_i = [u_i, v_i]$ in G (we only put one of the two opposite divisors in the basis).
2. /* 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 $R_0 := \alpha_0 D_1 + \beta_0 D_2$.
   Set k to 1.
3. /* 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 smooth divisor.
   (b) /* Express R_0 on the basis G */
   Factor $u_0(z)$ over $\mathbb{F}_q$, and determine the positions 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 3.a.
4. /* 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 $\mathbb{Z}/n\mathbb{Z}$.
5. /* Solution */
   Return $\lambda = -(\sum \alpha_k \gamma_k)/(\sum \beta_k \gamma_k) \mod n$. (If the denominator is zero, return to step 2.)

Fig. 1. Discrete log algorithm

3 Analysis

3.1 Probability for a Divisor to Be Smooth

The following proposition gives the proportion of smooth divisors and then the probability of smoothness in a random walk. This is a key tool for the complexity analysis.

Proposition 4 *The proportion of smooth divisors in the Jacobian of a curve of genus g over $\mathbb{F}_q$ tends to $1/g!$ when q tends to infinity.*

Proof: This proposition is based on the Hasse-Weil bound for algebraic curves: 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 the applications considered.

To evaluate the proportion of smooth divisors, we consider the number of points of the curve over $\mathbb{F}_q$ which is approximatively q. Now, the smooth divisors of the Jacobian are in bijection with the g-multiset of points of the curve: we have $q^g/g!$ smooth divisors, and the searched proportion is $1/g!$. □

3.2 Complexity

The complexity of the algorithm will be exponential in the size of q, so we will count the number of operations which can be done in polynomial time. These operations are of four types: we denote by c_J the cost of a group operation in the Jacobian, c_q the cost of an operation in the base field, $c_{q,g}$ the cost of an operation on polynomials of degree g over the base field, and c_n the cost of an operation in $\mathbb{Z}/n\mathbb{Z}$, where $n \approx q^g$ is the order of the Jacobian. We consider the enumeration of steps in figure 1.

Step 1. For the building of the factor basis, we have to perform q times (i.e. the number of monic irreducible polynomial of degree 1) a resolution of an equation of degree 2 over $\mathbb{F}_q$. Hence the complexity of this phase is $O(qc_q)$.

Step 2. The initialization of the random walk is only a polynomial number of simple operations. Hence we have $O((\log n)c_J)$ for this step.

Step 3. We have to repeat $\#G = O(q)$ times the steps 3.a. and 3.b.
Step 3.a. The computation of a new element of the random walk costs an addition in the Jacobian and two additions modulo n, and the test for its smoothness costs a first step of DDF. By proposition 4, we have to compute $g!$ divisors on average before getting a smooth one and going away from step 3.a. Hence the cost of this step is $O(g!(c_J + c_n + c_{q,g}))$.
Step 3.b. The final splitting of the polynomial in order to express the divisor on the factor basis can not be proved to be deterministic polynomial (though it is very fast in practice). For the analysis, we can then suppose that we do a trial division with all the elements of the basis. This leads to a complexity of $O(qc_{q,g})$.

Hence the complexity of step 3. is $O(qg!(c_J + c_n + c_{q,g})) + O(q^2 c_{q,g})$.

Step 4. This linear algebra step consists in finding a vector of the kernel in a sparse matrix of size $O(q)$, and of weight $O(gq)$; the coefficient are in $\mathbb{Z}/n\mathbb{Z}$. Hence Lanczos's algorithm provides a solution with cost $O(gq^2 c_n)$.

Step 5. This last step requires only $O(q)$ multiplications modulo n, and one inversion. Hence the complexity is $O(qc_n)$.

Finally, the overall complexity of the algorithm is $O(g!qc_J)+O((g!q+gq^2)(c_n+c_{q,g}))+O(qc_q)$. Now, by Cantor's algorithm c_J is polynomial in $g\log q$, and classical algorithm on finite fields and polynomials give c_n polynomial in $n = g\log q$,

c_q polynomial in $\log q$ and $c_{q,g}$ polynomial in $g \log q$. Hence all these operations can be done in time bounded by a polynomial in $g \log q$.

Theorem 1 *The algorithm requires $O(q^2 + g!q)$ polynomial time operations in $g \log q$ and if one considers a fixed genus g, the algorithm takes time $O(q^2 \log^\gamma q)$.*

4 Using Automorphisms on the Curve

4.1 Curves with Automorphisms in the Literature

When buiding a cryptosystem based on a hyperelliptic curve, it is preferable to know the order of the Jacobian of this curve. Indeed, some protocols use the group order; moreover it is necessary to be sure that it is not smooth. For elliptic curves, the Schoof-Elkies-Atkin algorithm allows to compute quickly this order for random curves (see [29] [28] [22]). For random hyperelliptic curves, a similar polynomial time algorithm exists [33], however it is still unusable in practice (see recent progress on this subject [21] [43]). That is the reason why the curves that we can find in the literature are very particular: they are built in such a way that the order of their Jacobian is easy to compute.

A first way to build such curves is to take a curve defined over a small finite field $\mathbb{F}_q$. It is then possible to deduce the Zeta function (and hence the order) of the Jacobian on the large field $\mathbb{F}_{q^n}$ from the Zeta function of the Jacobian on the small field. This construction provides then the so-called Frobenius automorphism defined by $x \mapsto x^q$, which can be applied to each coordinate of a point of the curve and gives therefore an automorphism of order n.

Another construction, which is a bit harder than the previous (see [42] [7] [3], comes from the theory of complex multiplication. This theory allows to build a curve starting from its ring of endomorphisms. In some cases, this ring contains units of finite order, and then there is an automorphism on the curve corresponding to this unit.

In table 1 we give some examples of curves found in the literature with non trivial automorphisms, and the order obtained by combining them together with the hyperelliptic involution.

Author	Equation of curve	Field	Automorphisms	Order
Koblitz [24], [25]	$Y^2 + Y = X^{2g+1} + X$ $Y^2 + Y = X^{2g+1}$	$\mathbb{F}_{2^n}$ $\mathbb{F}_{2^n}$	Frobenius Frobenius	$2n$ $2n$
Buhler Koblitz [3] Chao et al. [7]	$Y^2 + Y = X^{2g+1}$ (and twists)	$\mathbb{F}_p$ with $p \equiv 1\ (2g+1)$	mult by ζ_{2g+1}	$2(2g+1)$
Sakai Sakurai [36] Smart [41]	$Y^2 + Y = X^{13} + X^{11} + X^9 + X^5 + 1$	$\mathbb{F}_{2^{29}}$	Frobenius and $\begin{cases} X \mapsto X+1 \\ Y \mapsto Y + X^6 + X^5 \\ \quad + X^4 + X^3 + X^2 \end{cases}$	4×29
Duursma Sakurai [10]	$Y^2 = X^p - X + 1$	$\mathbb{F}_{p^n}$	Frobenius and $\begin{cases} X \mapsto X+1 \\ Y \mapsto Y \end{cases}$	$2np$

Table 1. Examples of curves

4.2 Reducing the Factor Basis with an Automorphism

In the context of the Pollard's rho algorithm, the existence of an automorphism of order m that can be quickly evaluated can be used to divide the expected running time by a factor $\sqrt{m}$, see [9]. With our algorithm, the automorphism can be used to reduce the basis and leads to a speed-up by a factor m^2, which can be very significant in practice. Moreover, the automorphism does not need to be so quickly evaluated as in the rho method. A polynomial time evaluation is enough.

The idea is to keep in the factor 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, assume that the Jacobian is cyclic of prime order $n = \mathrm{ord}(D_1)$, and denote by σ an automorphism of order m on $\mathcal{C}$ extended by linearity to an automorphism of $Jac(\mathcal{C})$. Then $\sigma(D_1)$ belongs to $Jac(\mathcal{C}) = \langle D_1 \rangle$, and there exists an integer θ such that $\sigma(D_1) = \theta D_1$. Moreover, σ being a group automorphism, for all $D \in Jac(\mathcal{C})$, $D = kD_1$ and we have $\sigma(D) = \sigma(kD_1) = k\sigma(D_1) = k\theta D_1 = \theta D$.

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 1. 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 reduced factor 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.

For the general case where the Jacobian is not cyclic and where we work in a subgroup of prime order n, we have to work a little to justify the computations, but in practice we do essentially the same.

5 Implementation and Results

We have implemented the algorithm in two distinct parts. The first one deals with the building of the matrix and is written in the computer algebra system Magma [2], which is a very good compromise between high level programming and efficiency. The second part is our optimized implementation of the Lanczos algorithm written in C.

5.1 Implementation of the Search for Relations

This part of the implementation was not optimized: it can be done in parallel and it is not the limiting phase. However an interesting optimization suggested by François Morain has been tested. It is based on a paper by Swan [44], where a theorem is given which relates the parity of the number of irreducible factors of a polynomial over a finite field and the fact that its discriminant is a square or not in the corresponding local field. In the context of smoothness testing, a first computation can be done that tests if the discriminant is a square, and then in half the cases we know that the polynomial cannot split completely and we reject it. If the first test is passed, we do the classical smoothness test by DDF.

This technique provides a gain if and only if Swan's test costs less than half the time of the classical one. In odd characteristic, this is always the case (for large q), but in characteristic 2, the running time estimation is harder because some computations have to be done over an extension of $\mathbb{Z}/8\mathbb{Z}$ and no package exists that provides optimized code for this ring. Note that the complications for the even characteristic is not surprising because in the finite field $\mathbb{F}_{2^n}$ every element is a quadratic residue and it is not simple to have a practical translation of Swan's theorem .

In our implementation, the use of Swan's theorem gave us a speed-up of 30 to 40% for the smoothness test in odd characteristic, but no improvement for characteristic 2.

5.2 Implementation of the Linear Algebra

A critical step in the algorithm is the search of a vector in the kernel of a sparse matrix. We chose Lanczos's algorithm in preference to Wiedemann's, because it needs only $2n$ products of the matrix by a vector, to be compared to $3n$ with Wiedemann's technique. The drawback is a non negligible amount of time spent in computing some scalar products. We refer to [27] for a precise comparison of these two algorithms.

We wrote our program in the C language, using the ZEN library [6] for things which were not critical (i.e. operations that are called a linear number of times), and for others (i.e. operations in the matrix-vector multiplication and scalar products), we used direct calls to some assembly routines taken from the GMP [18] and BigNum [20] packages. Indeed our compact representation of the matrix led to an overcost when using the ZEN functions. We used a classical representation (we could probably obtain a better efficiency with Montgomery representation), with the lazy reduction technique explained in [8].

Before running Lanczos's algorithm, a preprocessing can be done on the matrix (see [8] [5]). This filtering step (also called structured Gaussian elimination) consists in the following tasks:

- Delete the empty columns.
- Delete the columns with exactly one term and the corresponding row.
- If the number of rows is greater than the number of columns plus one, delete one row (randomly chosen, or via an heuristic method).
- Try the beginning of a Gaussian elimination, where the pivot is chosen as to minimize the augmentation of the weight of the matrix, and stopping when it increases the cost of Lanczos's algorithm.

For the examples below, we have run only the first three tasks, our implementation of the last one being unsatisfactory. Therefore there is still some place for further optimizations.

5.3 Timings for Real Life Curves

The first example is a cryptosystem recently proposed by Buhler and Koblitz [3]. We took the values recommended by Koblitz in his book [26], i.e. we have worked on the curve $y^2 + y = x^{13}$, with a prime base field of order p greater than $5,000,000$, with $p \equiv 1 \mod 13$. This curve has an automorphism of order 13 coming from complex multiplication, which helps in the computation of the order of the Jacobian, but helps also our attack.

The following table gives precise information on that curve.

field	$\mathbb{F}_{5026243}$
equation	$y^2 + y = x^{13}$
genus	6
$\#J$	$13^3 \times 7345240503856807663632202049344834001 \approx 10^{40}$

We give the measured timings for the computation of a discrete logarithm in the following table. These timings are on a Pentium II 450 MHz with 128 Mb. During the Lanczos's step (the most space consuming part of the algorithm), the memory used was around 60Mb.

cardinal of factor basis	$193,485$
time for building the basis	1638 sec
number of random steps	$201,426,284$
number of early abort by Swan	$100,721,873$
number of relations collected	$281,200$
proportion of smooths ($g!$)	716.3 (720)
total time for collecting the relations	$513,870$ sec = **6 days**
time for writing relations on the basis	$8,822$ sec
time for preprocessing the matrix	1218 sec
size of the matrix	$165,778 \times 165,779$
total time for Lanczos	$780,268$ sec = **9 days**

Our algorithm is not dependent on the characteristic of the base field. We have tested our implementation on a genus 6 curve over $\mathbb{F}_{2^{23}}$. This curve was obtained by extending the scalars of a curve defined over $\mathbb{F}_2$. Therefore the Frobenius automorphism can be used for accelerating the attack. The size of the Jacobian is around 10^{41}. Such a curve is not breakable by a parallel collision search based on the birthday paradox (variants of Rho); indeed even using the automorphism, we should compute about 2^{63} operations in the Jacobian.

We give the same indications as for the previous curve.

field	$\mathbb{F}_{2^{23}}$
equation	$y^2 + (x+1)y = x^{13} + x^{11} + x^8 + x^7 + x^5 + x^4 + x + 1$
genus	6
$\#J$	$2^3 \times 7 \times 622571845211703438355012489904899949517 7 \approx 10^{41}$

cardinal of factor basis	$182,462$
time for building the basis	6575 sec
number of random steps	$165,732,450$
number of relations collected	$231,000$
proportion of smooths ($g!$)	717.5 (720)
total time for collecting the relations	$797,073$ sec = **9 days**
time for writing relations on the basis	$12,057$ sec
time for preprocessing the matrix	880 sec
size of the matrix	$162,873 \times 162,874$
total time for Lanczos	$1,038,534$ sec = **12 days**

6 Conclusion

We have proposed an algorithm for the hyperelliptic discrete log problem, which is simpler to implement and to analyze than the previous ones. It is specially well suited for practical cryptosystems where the genus is not too large (say less than 9), and the base field is relatively small. Indeed the expected running time is $O(q^2)$ for curves of small genus and therefore it is faster than Pollard Rho as soon as the genus is greater than 4, as explained in the following table:

g	1	2	3	4	5	6	7
Rho	$q^{1/2}$	q	$q^{3/2}$	q^2	$q^{5/2}$	q^3	$q^{7/2}$
Index	q^2	q^2	q^2	q^2	q^2	q^2	q^2

Practical experiments have shown that this algorithm is efficient in practice, and a genus 6 example was broken by this technique. Hence it seems that there is no point in using hyperelliptic cryptosystem with genus other than 2, 3 or 4, because for a higher genus, the size of the key has to be chosen larger in order to guarantee a given level of security. Indeed, assume that we want to have a key of size 2^{160}, i.e. a group of order $\approx 2^{160}$, then we have to choose $g \log q \approx 160$. Increasing g implies decreasing $\log q$ and helps the attack. Hence one of the interests of the use of hyperelliptic curves, which was to decrease the size of q (for example to avoid multiprecision) becomes a weakness.

The special case of genus 4 has to be further studied. In a first approximation the complexity of Rho and our algorithm seem similar, but one trick can be played. We can decide to keep only a fraction of the divisors in the factor basis. Assume that we reduce the basis by a factor n. Then the probability to get a good divisor in the random walk is reduced by a factor n^g, and the cost of the first phase of the algorithm increases by a factor n^{g-1}, whereas the linear algebra is reduced by a factor n^2. In this context, Robert Harley pointed out to us that if we assume that the factorization of polynomials can be done in polynomial time (true in practice), we can balance both phases and choose n in order to get an overall complexity of $O(q^{\frac{2g}{g+1}})$. For $g = 4$, it becomes $O(q^{8/5})$, which is better than the complexity of the Rho method. We are going to do practical comparisons between the two approaches in a near future.

From a theoretical point of view, we can also analyse our algorithm in the same model as for ADH algorithm, i.e. we assume that the genus grows with q and is always large enough. More precisely, if we have $g > \log q$, we can let vary the smoothness bound S (instead of have it fixed to one), and we obtain a subexponential algorithm with expected running time $L_{q^g}[1/2, \sqrt{2}]$. This result is part of a work with Andreas Enge, where a general framework for this kind of attack is given [12].

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, 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.
2. W. Bosma and J. Cannon. *Handbook of Magma functions*, 1997. Sydney, `http://www.maths.usyd.edu.au:8000/u/magma/`.
3. J. Buhler and N. Koblitz. Lattice basis reduction, Jacobi sums and hyperellitic cryptosystems. *Bull. Austral. Math. Soc.*, 58:147–154, 1998.
4. D. G. Cantor. Computing in the Jacobian of an hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.
5. S. Cavallar. Strategies in filtering in the Number Field Sieve. Extended abstract, conference MPKC, Toronto, June 1999.
6. F. Chabaud and R. Lercier. *ZEN, A new toolbox for computing in finite extensions of finite rings*, February 1998. distributed with the ZEN package at `http://www.dmi.ens.fr/~zen`.
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. T. Denny and D. Weber. The solution of McCurley's discrete log challenge. In H. Krawczyk, editor, *Proc. of CRYPTO'98"*, volume 1462 of *Lecture Notes in Comput. Sci.*, pages 458–471, 1998.
9. I. Duursma, P. Gaudry, and F. Morain. Speeding up the discrete log computation on curves with automorphisms. In K.Y. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology – ASIACRYPT '99*, volume 1716 of *Lecture Notes in Comput. Sci.*, pages 103–121. Springer-Verlag, 1999. International Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 1999, Proceedings.

10. 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 *Proceedings of the "International Conference on Coding Theory, Cryptography and Related Areas"*, Lecture Notes in Comput. Sci., 1999. Guanajuato, Mexico on April, 1998.
11. 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.
12. A. Enge and P. Gaudry. A general framework for subexponential discrete logarithm algorithms. In preparation, 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. Preprint HP-LABS Technical Report (Number HPL-1999-70)., 1999.
17. R. Gallant, R. Lambert, and S. Vanstone. Improving the parallelized Pollard lambda search on binary anomalous curves. `http://www.certicom.com/chal/download/paper.ps`, 1998.
18. T. Granlund. *The GNU Multiple Precision arithmetic library – 2.0.2.* GNU, 1996. distributed with the gmp package at `ftp://prep.ai.mit.edu/pub/gnu/gmp-M.N.tar.gz`.
19. J. L. Haffner and K. S. McCurley. A rigorous subexponential algorithm for computation of class groups. *J. Amer. Math. Soc.*, 2(4):837–850, 1989.
20. J.-C. Hervé, B. Serpette, and J. Vuillemin. BigNum: A portable and efficient package for arbitrary-precision arithmetic. Technical Report 2, Digital Paris Research Laboratory, May 1989.
21. M.-D. Huang and D. Ierardi. Counting points on curves over finite fields. *J. Symbolic Comput.*, 25:1–21, 1998.
22. T. Izu, J. Kogure, M. Noro, and K. Yokoyama. Efficient implementation of Schoof's algorithm. In K. Ohta and D. Pei, editors, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *Lecture Notes in Comput. Sci.*, pages 66–79. Springer-Verlag, 1998. International Conference on the theory and application of cryptology and information security, Beijing, China, October 1998.
23. N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, January 1987.
24. N. Koblitz. Hyperelliptic cryptosystems. *J. of Cryptology*, 1:139–150, 1989.
25. 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.
26. N. Koblitz. *Algebraic aspects of cryptography*, volume 3 of *Algorithms and Computation in Mathematics.* Springer–Verlag, 1998.
27. 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.

28. R. Lercier. *Algorithmique des courbes elliptiques dans les corps finis.* Thèse, École polytechnique, June 1997.
29. R. Lercier and F. Morain. Counting the number of points on elliptic curves over finite fields: strategies and performances. In L. C. Guillou and J.-J. Quisquater, editors, *Advances in Cryptology – EUROCRYPT '95*, volume 921 of *Lecture Notes in Comput. Sci.*, pages 79–94, 1995. Saint-Malo, France, May 1995, Proceedings.
30. 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.
31. V. Miller. Use of elliptic curves in cryptography. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Comput. Sci.*, pages 417–426. Springer-Verlag, 1987. Proceedings, Santa Barbara (USA), August 11–15, 1986.
32. 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.
33. J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comp.*, 55(192):745–763, October 1990.
34. J. M. Pollard. Monte Carlo methods for index computation mod p. *Math. Comp.*, 32(143):918–924, July 1978.
35. H. G. Rück. On the discrete logarithm in the divisor class group of curves. *Math. Comp.*, 68(226):805–806, 1999.
36. 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.
37. 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.
38. 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.
39. 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.
40. N. Smart. The discrete logarithm problem on elliptic curves of trace one. *J. of Cryptology*, 12(3):193–196, 1999.
41. 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.
42. A.-M. Spallek. *Kurven vom Geschlecht 2 und ihre Anwendung in Public-Key-Kryptosystemen.* PhD thesis, Universität Gesamthochschule Essen, July 1994.
43. A. Stein and E. Teske. Catching kangaroos in function fields. Preprint, March 1999.
44. R. G. Swan. Factorization of polynomials over finite fields. *Pacific J. Math.*, 12:1099–1106, 1962.
45. 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.

46. P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *J. of Cryptology*, 12:1–28, 1999.
47. D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inform. Theory*, IT–32(1):54–62, 1986.
48. M. J. Wiener and R. J. Zuccherato. Faster attacks on elliptic curve cryptosystems. In S. Tavares and H. Meijer, editors, *Selected Areas in Cryptography '98*, volume 1556 of *Lecture Notes in Comput. Sci.* Springer-Verlag, 1999. 5th Annual International Workshop, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings.