Sieving in Function Fields

Ralf Flassenberg and Sachar Paulus

CONTENTS

We present the first implementation of sieving techniques in the context of function fields. More precisely, we compute in class groups of quadratic congruence function fields by combining the algorithm of Hafner and McCurley with sieving ideas known from factoring. We apply our methods to the computation of generators and relations of the Jacobian variety of hyperelliptic curves over finite fields.

The algorithms introduced here were implemented in C++ with the help of LEDA and LiDIA. We provide examples of running times and comparisons with earlier algorithms.

1. INTRODUCTION

Jacobian varieties of hyperelliptic curves over finite fields can (under certain conditions) be interpreted as class groups of imaginary quadratic congruence function fields; the algorithm of Hafner and McCurley [1989] known to compute the class group of imaginary quadratic number fields and having subexponential running time in the size of the discriminant can be applied. This idea is realized (with a slight modification) in [Adleman et al. 1994] by Adleman, DeMarrais and Huang who claim this algorithm to be of subexponential running time in the genus, based on heuristic evidence. An unconditional proof for this statement can probably be ruled out by adapting [Müller et al. 1999] to the case of imaginary quadratic congruence function fields. In that paper, Müller, Stein and Thiel prove that the computation of the regulator and of a fundamental unit in a real quadratic congruence function field is subexponential in the genus of the curve.

Experiments in [Paulus 1996b] showed that the algorithm of [Adleman et al. 1994], though thought to be subexponential, is even slower than the in principle exponential algorithms as baby-step-giant-step and Pollard rho. The second author proposed to apply the sieving principles from factoring and combine them with the above method. Even using trial

1058-6458/1999 \$0.50 per page

division instead of the real sieve for computing factorizations of several polynomials he got a considerable speedup (see [Paulus 1996b]). Based on this work, the first author implemented a sieving procedure to compute these factorizations analogous to the sieving principle used in factoring methods. This again gave drastic speedups. To our knowledge, this is the first implementation of sieving principles in the context of function fields. The objectives of this paper are to present the methods used and to describe the implications from practical experiments.

This work has cryptographic significance in the following sense: Koblitz [1989] proposed the discrete logarithm on the Jacobian variety of a hyperelliptic curve over a finite field as a new cryptographic one-way function. The major advantage in comparison with discrete logarithms on elliptic curves, that is, hyperelliptic curves of genus 1, consists in a considerably smaller underlying finite field, where all basic computations are done, for approximately the same size of the group where the discrete logarithm is defined. To examine the practical security of these cryptosystems it is necessary to compute such discrete logarithms with up-to-date methods.

The paper is organized as follows: first we explain how to do arithmetic in the Jacobian of a hyperelliptic curve. Then we recall the algorithm of Hafner and McCurley to compute imaginary quadratic class groups, together with the principal sieving idea before we show how we implemented the sieving for congruence function fields. All algorithms will be described in a general matter in words and additionally precisely formulated in pseudo code. Finally, we give practical results concerning timings and storage requirements when using different parameters for the sieving procedure and give generators and relations for some class groups.

We formulate theoretical statements in the context of hyperelliptic curves over any field. Theoretically, there is no reason which restricts the algorithm to the finite field case, but the practical realization of some components may be difficult to achieve (for example, computing prime divisors). Furthermore, some modifications should be made taking into account the probable infinity of the Jacobian variety. We restrict our presentation of the algorithm to the case of a finite field.

2. ARITHMETIC IN THE JACOBIAN OF A HYPERELLIPTIC CURVE

Let k be a field and $\mathcal{C}$ a hyperelliptic curve defined over k. Assume for simplicity that $\operatorname{char} k \neq 2$. Denote the function field of $\mathcal{C}$ over k by $k(\mathcal{C})$, the prime divisors of $k(\mathcal{C})$ by $\sum_k$, the divisors of degree 0 by $\mathrm{Div}_k^0(\mathcal{C})$ and the principal divisors by $\mathcal{P}_k(\mathcal{C})$. The factor group

$$\mathrm{Pic}_k^0(\mathcal{C}) = {\mathrm{Div}_k^0(\mathcal{C})}\big/{\mathcal{P}_k(\mathcal{C})}$$

is called the *divisor class group* of $\mathcal{C}$ over k and is equal to the group of k-rational points of the Jacobian of the curve. See [Mumford 1974] for the definitions.

Let $S \subsetneq \sum_k$ be a nonempty set of prime divisors. Let

$$\mathcal{O}_S := \{f \in k(\mathcal{C}) : \nu_{\mathfrak{p}}(f) \geq 0 \text{ for all } \mathfrak{p} \in S\}$$

be the intersection of all valuation rings $\mathcal{O}_{\mathfrak{p}}$ for $\mathfrak{p} \in S$. We have the following exact sequence:

$$1 \to \mathrm{Ker} \to \mathrm{Pic}_k^0(\mathcal{C}) \to \mathrm{Cl}(\mathcal{O}_S) \to 1.$$

Ker is generated by the degree 0 divisors with support in $\sum_k \setminus S$ modulo principal divisors. If $S = \sum_k \setminus \{\mathfrak{p}_0\}$ with $\deg \mathfrak{p}_0 = 1$, then to every ideal $\mathfrak{a} = \prod_{\mathfrak{p} \in S} \mathfrak{p}^{a_{\mathfrak{p}}}$ can be assigned the preimage

$$\sum_{\mathfrak{p} \in S} a_{\mathfrak{p}} \mathfrak{p} - \Bigl(\sum_{\mathfrak{p} \in S} a_{\mathfrak{p}} \deg \mathfrak{p}\Bigr) \mathfrak{p}_0$$

and thus $\mathrm{Pic}_k^0(\mathcal{C})$ is isomorphic to the (ideal) class group of $\mathcal{O}_S$.

The existence of such a prime divisor $\mathfrak{p}_0$ with $\deg \mathfrak{p}_0$ induces the existence of an equation of the curve of the form $Y^2 = \Delta(X)$ with $\Delta(X)$ monic, squarefree and of degree $2g+1$. We will always assume the existence of such a prime divisor in the sequel. The discriminant of $\mathcal{O}_S$ is then generated by $\Delta = \Delta(X)$. Consequently, we will do arithmetic in the Jacobian of the curve described by $Y^2 = \Delta$ by computing in the class group of $k[X]\bigl[\sqrt{\Delta}\,\bigr]$.

We show now how to do arithmetic in this class group. Another formulation of this arithmetic has been proposed by Cantor [1987] and goes back to Artin [1924]. In the special case $g = 1$, we recover the classical representation of points on an elliptic curve together with the geometric chord-tangent addition of points.

We will explain the arithmetic using binary quadratic forms since this notion will be used in the sequel. There exists a one-to-one correspondence between classes of ideals (modulo principal ideals) in $k[X]\left[\sqrt{\Delta}\,\right]$ and classes of primitive binary quadratic forms (a, b, c) with coefficients in $k[X]$ of discriminant $b^2 - 4ac = \Delta$ such that $\gcd(a, b, c) = 1$ (shortly called *forms* in the sequel) (modulo the action of $\mathrm{GL}_2(k[X])$). There exists a reduction theory for forms, that is, an algorithm which computes for a given form a unique equivalent form with a monic, $\deg a \leq g$ and $\deg(b) < \deg(a)$. Such a form is called *reduced.*

A class is represented by the first two coefficients (a, b) of the unique reduced form (a, b, c) of discriminant Δ in this class. Given (a, b), a representation of the inverse class is $(a, -b)$. A class is the identity element if the first coefficient $a = 1$. Two classes (a_1, b_1) and (a_2, b_2) are multiplied in two steps: first, a form (a_3, b_3, c_3) representing the product class of the two forms is computed (this is called *composition of forms*) and second, the unique reduced form (a_4, b_4, c_4) equivalent to (a_3, b_3, c_3) is computed (this is called *Gaussian reduction*). Then the product class is represented by (a_3, b_3).

Several algorithms for composition and reduction can be found in [Artin 1924; Cantor 1987; Koblitz 1989; Paulus and Rück 1999]. Optimized versions for both algorithms including a rigorous complexity analysis can be found in [Paulus and Stein 1998] using the reduction variant of Tanner.

3. THE ALGORITHM

The principal idea to do fast computations in the Jacobian variety consists in computing generating elements of a special kind — called generators — and relations between them. We will explain a variant of the algorithm which computes a minimal set of generators for the Jacobian together with relations in Hermite normal form. We first explain the general method, then we show how to compute relations and finally we discuss the number of needed generators. As discussed in the introduction, we restrict ourselves to the case of a finite field $k = \mathbb{F}_q$ with $\operatorname{char} k \neq 2$.

At the end of this section, we will shortly explain the necessary modifications for computing discrete logarithms on the Jacobian of a hyperelliptic curve using this algorithm.

3.1. The Algorithm of Hafner and McCurley

As shown in [Paulus 1996a], the algorithm is generic in the sense that it is applicable for computations of class groups of quadratic orders over principal ideal domains. We will therefore call irreducible polynomials of $\mathbb{F}_q[X]$ also *primes*; the *order* of Δ, denoted by $\mathcal{O}_\Delta$, for a $\Delta \in \mathbb{F}_q[X]$ is the $\mathbb{F}_q[X]$-module generated by $\left(1, \sqrt{\Delta}\right)$ over $\mathbb{F}_q[X]$. The class group of $\mathcal{O}_\Delta$ is denoted by $\mathrm{Cl}(\mathcal{O}_\Delta)$ and is our representation for the Jacobian of the curve $Y^2 = \Delta$. A form $f_p = (p, b_p, c_p)$ of discriminant Δ with $b_p^2 \equiv \Delta \bmod p$ and $\deg b_p < \deg p$ for a prime $p \in \mathbb{F}_q[X]$ is called *prime form for* p (and corresponds to a prime divisor of the curve). Note that there exists either exactly one prime from for p (if $p \mid \Delta$) or exactly two prime forms for p, namely (p, b_p, c_p) and $(p, -b_p, c'_p)$.

The method of Hafner and McCurley is based on the following observation:

Proposition 1. *Let $\Delta \in \mathbb{F}_q[X]$ be a non-square, $\mathcal{P}$ a set of n primes, for which Δ is a quadratic residue. Suppose that the prime forms f_p for the primes $p \in \mathcal{P}$ generate $\mathrm{Cl}(\mathcal{O}_\Delta)$. Then the map*

$$\begin{array}{cccc} \Omega: & \mathbb{Z}^n & \to & \mathrm{Cl}(\mathcal{O}_\Delta) \\ & (x_p)_{p\in\mathcal{P}} & \mapsto & \left[\prod_{p\in\mathcal{P}} f_p^{x_p}\right] \end{array}$$

is a surjective homomorphism and we have

$$\mathbb{Z}^n\big/\ker\Omega \cong \mathrm{Cl}(\mathcal{O}_\Delta) \quad \textit{and} \quad \det(\ker\Omega) = |\mathrm{Cl}(\mathcal{O}_\Delta)|.$$

The set of primes $\mathcal{P}$ is called the *factor base*. The vectors $(x_p) \in \ker\Omega$ are called *relations*; such relations are collected during the computation. The sublattice Λ spanned by the relations already computed is called *relation lattice* and is represented by the *relation matrix*, whose columns are the computed relations.

Assume that the prime forms for the primes of the factor base generate the whole class group of $\mathcal{O}_\Delta$ and that we know an upper bound L for the class number of the form $L/2 < |\mathrm{Cl}(\mathcal{O}_\Delta)| \leq L$. The method of Hafner and McCurley proceeds in three steps: First, determine prime forms corresponding to the primes in the factor base, that is, compute a set of generators. For a given prime p, this amounts to the decision whether Δ is a quadratic residue mod p

and to the computation of a quadratic root of Δ mod p. Both tasks can be achieved by computing in the multiplicative group $(\mathbb{F}_q[X]/(p))^*$, using a generalization of the RESSOL algorithm of Shanks (see [Buchmann and Paulus 1995]). Second compute relations until the relation matrix is regular and finally compute relations until $\det(\Lambda) \leq L$. The computation of a relation is the object of the next subsection.

A bound L of the shape described above is in most cases given by the theorem of Hasse–Weil. It induces the following

Proposition 2. *Let $\mathcal{C}$ be a hyperelliptic curve of genus g over a finite field $\mathbb{F}_q$. Then the number of points $|\operatorname{Jac}(\mathcal{C})|$ of the Jacobian variety of $\mathcal{C}$ fulfills*

$$(\sqrt{q}+1)^{2g}/2 < \left|\operatorname{Jac}(\mathcal{C})\right| \leq (\sqrt{q}+1)^{2g}$$

exactly when

$$q > \left(\frac{2}{2^{1/(2g)}-1}+1\right)^2.$$

The proof is immediate. Such a bound is, for example, fulfilled for $g = 2$ if $q > 134$, for $g = 4$ if $q > 301$ and for $q \approx 10^{40}$ if $g < 10^{20}$.

We formulate the basic algorithm as follows:

Algorithm 3 (Hafner–McCurley-type class group).

Input: $\Delta \in \mathbb{F}_q[X]$ not a square; the factor base $\mathcal{P} \subset \mathbb{F}_q[X]$ such that the prime ideals lying over the primes of $\mathcal{P}$ generate the class group; $e_{\max} \in \mathbb{N}$ upper bound for the exponents (e_p); $L \in \mathbb{N}$ such that $L/2 < |\mathrm{Cl}(\Delta)| \leq L$

Output: $(\gamma_1, \ldots, \gamma_k)$ such that $\langle\gamma_1, \ldots, \gamma_k\rangle = \mathrm{Cl}(\Delta)$; the relation matrix A in Hermite normal form; $|\mathrm{Cl}(\Delta)|$

1. Compute for each $p \in \mathcal{P}$ a prime form f_p
2. Compute relations until the relation matrix A is regular
3. $A \leftarrow \operatorname{Hermite}(A)$
4. Compute the determinant of A
5. **while** $\det A > L$ **do**
   a. **repeat**
      i. Compute a relation s
      ii. $B \leftarrow \operatorname{Hermite}(A, s)$
      until $\det B < \det A$
   b. $A \leftarrow B$
6. $C \leftarrow$ nontrivial rows and columns of A;
   $\mathcal{P}' \leftarrow$ those $p \in \mathcal{P}$ corresponding to the rows of C
7. **output** $(\mathcal{P}', C, \det A)$

By "taking nontrivial rows and columns" we mean rejecting those columns whose diagonal element is one and removing the corresponding rows by changing the other columns appropriately.

There are several choices for computing a Hermite normal form and it is not clear which algorithm is best in practice. We do not emphasize on that point. We used in our final implementation a non-modular implementation of Havas [Havas and Majewski 1997] and a modular version due to Domich, Kannan and Trotter [Domich et al. 1987], taking account of the additional information we have about the maximal value of the determinant.

3.2. Computing Relations

We first recall the original idea of Seysen [1987] used in the method of Hafner and McCurley for getting a relation. The following proposition is an immediate generalization of in [Seysen 1987, Theorem 3.1] to the function field situation.

Proposition 4. *Let (a, b, c) be a form of discriminant $\Delta \in \mathbb{F}_q[X]$ and let*

$$a = \varepsilon \prod_p p^{v_p}$$

be a prime decomposition of a where $\varepsilon \in \mathbb{F}_q[X]^*$. Then we have

$$(a, b, c) \sim \prod_p f_p^{\varepsilon_p v_p},$$

where $f_p = (p, b_p, c_p)$ are prime forms and $\varepsilon_p = \pm 1$ such that $b \equiv \varepsilon_p b_p \bmod 2p\mathbb{F}_q[X]$.

This induces the following method to compute relations: compute a random form by choosing random exponents $e_p \in \mathbb{Z}$:

$$(a, b, c) = \prod f_p^{e_p}.$$

The reduction of this form yields a form (a', b', c'). If a' factors over the factor base, say

$$a' = \prod p^{g_p},$$

we get a relation $(e_p - \varepsilon_p g_p)_{p \in \mathcal{P}}$, where $\varepsilon_p = \pm 1$ is defined by $b' = \varepsilon_p b_p \bmod 2p$.

Note that the algorithm of Adleman and Huang differs from the algorithm of Hafner and McCurley in that point. They do not compute a random form starting from the factor base elements; instead, they generate a random element $A+B\sqrt{\Delta} \in \mathbb{F}_q[X]\left[\sqrt{\Delta}\,\right]$ and try and factor it over the prime forms corresponding to the primes in the factor base.

Both ways of getting a relation have been proven not to compute enough relations in the congruence function field case. We use a more general idea such that we obtain many forms equivalent to (a,b,c). This is done as follows: any form that is equivalent to (a,b,c) is of the shape $(ax^2+bxy+cy^2,*,*)$ for some relatively prime $x,y \in \mathbb{F}_q[X]$ (see [Paulus 1996a]). For several relatively prime pairs (x,y) we try and factor $ax^2+bxy+cy^2$, the *sieve elements*, over the factor base. Assume that we can do this effectively. If this is successful, we compute u, v such that $ux+vy=1$, that is, we compute in fact the transformation matrix

$$A=\begin{pmatrix} x & -v \\ y & u \end{pmatrix}$$

with $g=Af$ and recover from there the second coefficient of the form which is needed for the determination of the ε_p. The set of pairs (x,y) is called *sieve array* and is denoted by $\mathcal{B}$.

In fact, we do not require (x,y) to be relatively prime. If $d=\gcd(x,y)$, then $(x/d,y/d)$ are relatively prime and d^2 divides $ax^2+bxy+cy^2$, such that we get a relation by taking the decomposition of $(ax^2+bxy+cy^2)/d^2$. In practice, this does not occur, since the pair $(x/d,y/d)$ will be treated before (x,y) and the number of y will be very small.

We formulate our technique to compute a relation as follows:

Algorithm 5 (Relation computation).

Input: $\mathcal{P}$ factor base; $\mathcal{B}\subset\mathbb{F}_q[X]\times\mathbb{F}_q[X]$ finite sieving array; $e_{\max}\in\mathbb{N}$ upper bound for exponents
Output: relations $(n_p)_{p\in\mathcal{P}}$ or "No relation found"

1. Determine random integers $e_p\in[0,e_{\max}-1]$ for each $p\in\mathcal{P}$ and compute $(a,b,c)\sim\prod f_p^{e_p}$
2. **for** all $(x,y)\in\mathcal{B}$ with $\gcd(x,y)=1$
   a. $n\leftarrow\deg(ax^2+bxy+cy^2)$
   b. **for** all $p\in\mathcal{P}$
      i. $v_p\leftarrow v_p(n)$
      ii. $n\leftarrow n-v_p$
      iii. **if** $n=0$ **then**
         compute $u,v\in\mathbb{F}_q[X]$ such that $xu-yv=1$
         $b'\leftarrow 2axv+b(xu+yv)+2cyu$
         for all $v_p\neq 0$
            if $b'\not\equiv b_p \bmod 2p\mathbb{F}_q[X]$ **then** $v_p\leftarrow -v_p$
   c. **output** $\big((v_p-e_p)_{p\in\mathcal{P}}\big)$
3. **output** "No relation found"

Here $v_p(n)$ denotes the valuation of p at n. The algorithm does not describe in which way $v_p(n)$ is computed. In a first implementation we used trial division and got remarkable improvements over the traditional Hafner–McCurley algorithm. A much cleverer method analogous to the sieving procedures known from factoring is explained in the next section.

3.3. The Factor Base

We discuss the size of the factor base. The use of a factor base which provably generates the whole class group is a major problem in practice: using techniques of Bach [1990], it is proved in [Müller et al. 1999] that all primes having degree at most

$$\left\lceil\frac{2\log(4g-1)}{\log q}\right\rceil$$

generate the whole class group. This bound yields even a polynomial bound in g for the size of the factor base. But this number is at least 1; this means that one should always include all primes of degree 1 which split in $\mathcal{O}_\Delta$ in the factor base. Since there are about $q/2$ primes of degree 1 which split, this is unacceptable in practice for large fields. Instead of this, we proceed as follows:

Pick at random a certain number of primes having degree respecting the bound given above and splitting in the quadratic extension. Compute the generated group — ignoring the known bound L for the order of the class group — with a prescribed precision. That is, if generating a random form by choosing exponents at random does not give a new nontrivial relation l times successively, then the probability for the group being generated by the prime forms for the primes in the factor base is greater than $1-1/2^l$. If the result fits the L-bound, then the output is the whole class group with probability greater than $1-1/2^l$.

Much research and many experiments and have to be done to evaluate which method is best used for choosing the members of the factor base. We will propose one method which produced the best practical results in our experiments in the last section.

3.4. Discrete Logarithms

Let g be an element of the class group and $a = g^x$. The *discrete logarithm problem* consists in computing $\log_g(a) = x$ given a and g.

The modifications of the algorithm of Hafner and McCurley to compute a discrete logarithm are as follows: The generator for the discrete logarithm g is added to the factor base. Instead of Hermite reducing the relation matrix, one solves the system such that we have $\log_g[f_p]$ for all classes $[f_p]$ of prime forms f_p lying over the factor base. (The solution of the system can be realized, for example, with a modular variant of Hermite reduction). Finally, an additional relation involving a to the power 1 is computed and the discrete logarithm extracted as a linear combination of the $\log_g p$. See [Lenstra and Lenstra 1990], for example.

4. SIEVING

The factorization of the sieve elements $ax^2 + bxy + cy^2$ is not done sequentially as presented in the previous algorithm, but it is done "in parallel": First, we store the degree of all these polynomials. Now for every prime p in the factor base we find out which elements $ax^2 + bxy + cy^2$ are divisible by p. We use sieving ideas for this step. If a sieve element is divisible by p, we subtract the degree of p of the degree of this sieve element. If this degree is zero, we have a complete factorization of $ax^2 + bxy + cy^2$ over the factor base.

We discuss how to adapt the sieving techniques from factoring to our situation. We recall the principal sieving idea and explain differences to the sieving over $\mathbb{Z}$. We present the sieving procedure in detail and demonstrate our implementation solutions.

4.1. The Idea

Assume that we have a polynomial $g \in \mathbb{Z}[X]$, a finite set of prime numbers $\mathcal{P}$ and an interval $[a, b] \subset \mathbb{Z}$. Assume that we want to know those $z \in [a, b]$ for which $g(z)$ splits completely over $\mathcal{P}$ together with their factorization. Let us call such a z *interesting.* The naive method consists in testing for each p if p divides $g(z)$ for every z. The sieving idea is the following: for every prime $p \in \mathcal{P}$ compute the roots of $g(X) \bmod p$. Then p divides $g(z)$ exactly when $z = r + s * p$, where r is a root of $g(X) \bmod p$ and $s \in \mathbb{Z}$. By this way, one can mark all interesting z by "jumping" through the interval with steps of size p for all primes $p \in \mathcal{P}$. One stores the logarithm of $g(z)$ for each z at the beginning and subtracts $\log p$ every time when a jump hits z. If this number becomes (approximately) 0, then we have completely factored $g(z)$ over $\mathcal{P}$.

This method ignores those z for which $g(z)$ has a square as a factor. Using Hensel lifting one could extend this method to higher exponents and determine all $v_p(g(z))$. But experience shows that exponents greater than 1 occur rarely in a complete factorization over the factor base so that the additional amount of work is not worth it. One discards those numbers which have a square as a factor.

In our situation, the polynomial $g \in \mathbb{Z}[X]$ is replaced by a primitive binary quadratic form

$$g(x, y) = ax^2 + bxy + cy^2$$

with coefficients $a, b, c \in \mathbb{F}_q[X]$ taking as arguments two elements $x, y \in \mathbb{F}_q[X]$. The prime elements are now irreducible polynomials of $\mathbb{F}_q[X]$ and the logarithm of a number is replaced by the degree of a polynomial. The sieving procedure differs in two points from the classical situation:

- We are sieving in two directions, since f takes two arguments. Thus the interval is replaced by a sieving array. The set of solutions mod p has to be computed for two variables as has to be done the "jumping" through the sieving array.
- Jumping through the sieving array is not as immediate as in the classical case, since the steps between two consecutive interesting polynomials vary in size.

As in the classical case, complete factorizations with exponents greater than 1 occur very rarely, so we do not use Hensel lifting and discard numbers which have a square as a factor.

4.2. The Sieving Procedure

We first explain how to represent polynomials for indexing array entries. Every element $\alpha \in \mathbb{F}_q$ can be uniquely represented by a natural number $\nu(\alpha)$ such that $0 \le \nu(\alpha) < q$. E.g. if ξ is a generating element of $\mathbb{F}_q$ over $\mathbb{F}_{q_0}$ where q_0 is the characteristic of $\mathbb{F}_q$, then $\nu(\alpha) = \nu(\sum_{j=0}^{d-1} x_i \xi^i) = \nu(x_i) q_0^i$, where $\nu(x_i)$ is the unique integer in the range $\{0, \dots, q_0 - 1\}$

naturally representing x_i. So there is a one-to-one map $\nu : \mathbb{F}_q[X] \to \mathbb{N}$ sending $a = \sum_{i=0}^{m} \alpha_i X^i$ to $\sum_{i=0}^{m} \nu(\alpha_i) q^i$. If we write $D(x,y)$ for an array entry in the sequel, we really compute with $D(\nu(x), \nu(y))$.

This representation induces a restriction on the form of the sieving array. Since array indices have to be a series of consecutive numbers, there is an implicit ordering of the elements of $\mathbb{F}_q[X]$ and the sieving array has to be, for example, of the form

$$\mathcal{B} = \{x \in F_q[X] : 0 \le \nu(x) \le x\_bound\} \times \{y \in F_q[X] : 0 \le \nu(y) \le y\_bound\}.$$

The idea of the sieving procedure has been presented above. We now formulate this central part of our algorithm in pseudo-code. We will explain the remaining details which differ from sieving in $\mathbb{Z}$ in the following subsection.

Algorithm 6 (Sieving in congruence function fields).

Input: $\mathcal{P}$ factor base; $\mathcal{B}$ sieving array as described; and $g(x,y) = ax^2 + bxy + cy^2$ a primitive binary quadratic form.

Output: all $(x_p)_{p\in\mathcal{P}}$ with $x_p \in \{0,1\}$ such that
$$\prod_{p\in\mathcal{P}} p^{x_p} = g(x,y) \quad \text{for } (x,y) \in \mathcal{B}$$

1. Compute a two-dimensional integer matrix
$$\big(D(x,y)\big)_{(x,y)\in\mathcal{B}}$$
containing the degrees of $g(x,y)$ for all $(x,y) \in \mathcal{B}$
2. Initialize a three-dimensional integer matrix
$$\big(R(x,y,p)\big)_{(x,y)\in\mathcal{B},p\in\mathcal{P}}$$
which codes whether p divides $g(x,y)$ with zeroes
3. **for** all $p \in \mathcal{P}$
   a. Compute a complete set of solutions $\mathcal{S} \subset \{(x,y) : x,y \in \mathbb{F}_q[X],\ \deg x, \deg y < \deg p\}$ of $g(x,y) = 0 \bmod p$.
   b. **for** all $(x,y) \in \mathcal{S}$
      i. $R(x,y,p) \leftarrow 1$
      ii. $D(x,y) \leftarrow D(x,y) - \deg p$
      iii. **if** $D(x,y) = 0$ **output** $(R(x,y,p))_{p\in\mathcal{P}}$
      iv. **for** all $r,s \in \mathbb{F}_q[X]$ with $(x+rp, y+sp) \in \mathcal{B}$ (Jumping through the sieving array)
      $R(x+rp, y+sp, p) \leftarrow 1$;
      $D(x+rp, y+sp) \leftarrow D(x+rp, y+sp) - \deg p$;
      if $D(x+rp, y+sp) = 0$
      output $(R(x+rp, y+sp, p))_{p\in\mathcal{P}}$

Remark: In step 3a, it may occur that $\deg p$ is greater than x_bound or y_bound. One obviously computes only a set of solutions for the existing sieve elements.

4.3. Implementation Details

In the rest of this section we explain how to initialize the matrix D, which are the solutions of $g(x,y) \equiv 0 \bmod p$ and how the jumping through the array is realized in practice.

4.3.1. Initializing In step 1 of Theorem 6, we have to compute for every $(x,y) \in \mathcal{B}$ the degree of $g(x,y) = ax^2+bxy+cy^2$. The naive method would be to really compute $g(x,y)$ and to deduce the degree. This is very slow. Most of the time we can do better. Set

$$\begin{aligned}\deg_a &= \deg(a) + 2\deg(x),\\ \deg_b &= \deg(b) + \deg(x) + \deg(y),\\ \deg_c &= \deg(c) + 2\deg(y),\end{aligned}$$

and denote by m the maximum among these three numbers. The degree is computed as follows:

1. if only one of them equals m, then $\deg g(x,y) = m$.
2. (at least two values equal m). Compute the sum of the leading coefficients of those terms having degree m. If it is different from 0, then $\deg(x,y) = m$.
3. ($\deg g(x,y) \ne m$). Compute $g(x,y)$ explicitely.

Note that if x and y do not change their degree, the values of $\deg_a$, $\deg_b$ and $\deg_c$ are unchanged. This can be taken into account when filling the degree matrix successively.

4.3.2. Solutions "mod p" Let $g(x,y) = ax^2 + bxy + cy^2$ be a primitive binary quadratic form with coefficients $a,b,c \in \mathbb{F}_q[X]$, $p(x) \in \mathbb{F}_q[X]$ an irreducible polynomial and (p, b_p, c_p) a prime form corresponding to p. The set of polynomials with degree less than $\deg p$ form a complete set of representatives of $\mathbb{F}_q[X]/\langle p\rangle$. A complete set of solutions of $g(x,y) \equiv 0 \bmod p$ is given by $\mathcal{S}$, where $\mathcal{S}$ is as follows:

- if $a \equiv 0 \bmod p$ and $b \equiv 0 \bmod p$, then $\mathcal{S} = \{(x,0) : \deg x < \deg p\}$
- if $a \equiv 0 \bmod p$ and $b \not\equiv 0 \bmod p$, then $\mathcal{S} = \{(x,0) : \deg x < \deg p\} \cup \{(-y\frac{c}{b}, y) : \deg y < \deg p\}$,
- if $a \not\equiv 0 \bmod p$ and $b_p \equiv 0 \bmod p$, then $\mathcal{S} = \{(-y\frac{b}{2a}, y) : \deg y < \deg p\}$,

- if $a \not\equiv 0 \bmod p$ and $b_p \not\equiv 0 \bmod p$, then $\mathcal{S} = \{(-y\frac{b \pm b_p}{2a}, y) : \deg y < \deg p\}$.

4.3.3. Jumping The objective of this section is to show how one can compute all $\nu(x + rp)$ for

$$0 \le \deg r < \log_q(x\_bound) - \deg p + 1$$

most effectively, since this part of the algorithm is very time consuming in the congruence function field case. We explain the procedure for the first component (x), the same method is applied to the second component (y).

The polynomials $x + rp$ are computed by subsequent additions of p, p^2, p^3, up to x. The polynomials involved here are treated as vectors of integers; that is, the i-th coefficient of a is denoted by $a[i]$ and interpreted as a number in the range $0, \dots, q-1$. Addition of p^i to x is done by "shifting" the coefficients of p by i before adding them to the corresponding coefficients of x. The complete jumping algorithm is now straightforward and looks like this:

Algorithm 7 (Fast index computation).

Input: $p \in \mathbb{F}_q[X]$ irreducible polynomial; $(x, y) \in \mathcal{S}$ solution of $g(x, y) \equiv 0 \bmod p$; $x\_bound \in \mathbb{N}$ bound for the sieving array

Output: All $0 \le \nu(x + rp) < x\_bound$ such that $g(x + rp, y) \equiv 0 \bmod p$

1. $act\_poly \leftarrow x$; $r \leftarrow 0$
2. **while** $\deg r < \log_q x\_bound - \deg p + 1$
   a. **if** $r[0] \neq q$ **then**
      i. **for** $j \in [0 \,..\, \deg(p)]$ **do**
         $act\_poly[j] \leftarrow act\_poly[j] + p[j] \bmod q$
      ii. **output** $\nu(act\_poly)$
      iii. $r[0] \leftarrow r[0] + 1$
      else
      i. $i \leftarrow \max\{j : r[k] = q \text{ for all } 0 \le k \le j\}$
      ii. **for** $j \in [0 \,..\, \deg(p)]$ **do** $act\_poly[j + i] \leftarrow act\_poly[j + i] + p[j] \bmod q$
      iii. **output** $\nu(act\_poly)$
      iv. **for** $j \in [1 \,..\, i]$ **do** $r[j] \leftarrow 0$
      v. $r[i + 1] \leftarrow r[i + 1] + 1$

Further optimization can be obtained by taking care how to modify $\nu(act\_poly)$ from the previous value instead of recomputing it from scratch every time.

5. PRACTICAL RESULTS AND DISCUSSION

We computed the divisor class group of the curve $Y^2 = X^{2g+1} + 2X + 1$ over $\mathbb{F}_p$, where $X^{2g+1} + 2X + 1$ was squarefree in all examples. The best results — input parameters, timings and storage requirements — for the fastest computation are presented in Table 1.

We now discuss the input values. The possible size of the sieving array is bounded by the maximum memory limit. Furthermore, experiences with larger sieving arrays gave not much more relations compared with the additional time needed. It would have been necessary to increase the size of the factor base to get better results. The size of the factor base will be discussed in the sequel. Another observation is that it seems to be optimal to choose a small value for one dimension of the factor base and and a large value for the other dimension. This covers experience from sieving in the factoring context, although people from factoring suggest to choose the value for one dimension being 1. This had not proven to be optimal in our context.

The factor base is computed from the first two values A and B as follows: take A successive polynomials of degree at most B and compute for all irredicuble polynomials in this set a prime form if it exists. The resulting number of factor base elements is then given by the third value. The examples have not been large enough to observe a dependency of the number of computed relations from B, although the computation of a single relation is faster as B grows. It may be possible to think of a *large prime variant* analogous to the factoring variant.

The possible size of the factor base is governed by the Hermite reduction. Twice the average entry of the matrix being Hermite reduced is given by the last input value, namely the upper bound for the exponents of the factor base elements used in the relation computation. The determinant of the matrix should finally, that is, assuming that enough relations have been computed, be of size approximately q^g. With growing size of the factor base and growing determinant, the Hermite reduction begins to dominate the computation both in time and in space. Since Hermite reduction is difficult to parallelize over several machines, this is the real bottleneck of the computation.

p	g	x	y	F_1	F_2	F_3	M	T	T_r	T_h	S_r	S_h
11	1	2	10	5	1	4	5	$1''$	$1''$	–	1	1284
11	2	5	10	10	1	8	5	$3''$	$3''$	–	2	1292
11	3	5	50	20	2	15	5	$7''$	$7''$	–	17	1368
11	4	20	100	40	2	29	5	$17''$	$15''$	–	243	1708
11	5	20	80	50	2	29	5	$53''$	$50''$	–	194	1656
11	6	20	100	50	2	33	5	$1'1''$	$59''$	–	274	1788
11	7	20	100	50	2	38	5	$2'46''$	$2'44''$	–	313	1932
11	8	20	2000	400	3	232	10	$17'42''$	$11'33''$	$48''$	36578	47868
11	9	20	2000	400	3	231	10	$18'56''$	$14'50''$	$37''$	36421	47164
11	10	20	2000	400	3	242	10	$34'53''$	$32'19''$	$1'29''$	38140	57620
11	11	20	2000	400	3	232	10	1h11$'$	$56'37''$	$13'34''$	36578	88888
11	12	20	2000	400	3	254	10	3h41$'$	1h40$'$	$1h59'$	40015	214928
101	1	20	40	20	1	10	5	$9''$	$9''$	–	37	1348
101	2	30	500	35	1	19	10	$33''$	$32''$	–	1234	2644
101	3	10	5000	60	1	32	10	$2'30''$	$2'28''$	–	6679	8660
101	4	5	5000	100	1	43	10	$3'16''$	$3'21''$	$1''$	4433	6604
101	5	5	5000	150	1	51	10	$20'6''$	$20'1''$	$2''$	5214	7796
101	6	10	5000	150	2	129	10	5h1$'$	4h53$''$	$8'5''$	25625	50404
1009	1	10	1000	100	1	52	10	$1'42''$	$1'37''$	$1''$	2117	4072
1009	2	10	10000	300	1	153	10	$9'3''$	$8'26''$	$18''$	60625	66636
1009	3	10	10000	400	1	205	10	1h40$'$	$46'37''$	$53'13''$	80937	177648
10007	1	20	4000	100	2	55	10	1h5$'$	1h4$'$	$2''$	17843	22040

TABLE 1. Input parameters, timings and storage requirements. The table is indexed by g and p. The columns x and y indicate the size of the sieve array in both dimensions; F_1, F_2, F_3 are three values concerning the construction of the factor base and M is the maximum random exponent for a factor base element used for the relation computation. T is the total time needed, T_r the time needed for computing the relations and T_h that needed for the Hermite reduction. S_r and S_h are the memory sizes needed by the relation computation and the Hermite reduction, both in kilobytes.

To achieve further improvements, it is necessary to tailor the relations for an efficient Hermite reduction. The major modification should be to use *sparse* relations, that is, relations where only a few factor base elements are used to produce a random form together with a much larger factor base. The resulting matrix would consist of a *dense* and a *sparse* part, where the dense part looks like the matrices we produce now and the dense part consists only of some 1 as entries. A drawback is that the computation of a matrix with full rank is much more complicated. This method is more convenient for computing discrete logarithms.

We show in Table 2 the timings of the Hafner–McCurley method used with different sieving strategies.

Finally, in Table 3 we give some examples for the class groups we computed. The class groups are presented in the following format: we give the type of the group and generators for the corresponding cyclic subgroups.

The computations were done using the computer algebra and packages LiDIA [LiDIA n.d.] and LEDA [LEDA n.d.]. Most of the programs implemented here are available on request via ftp as an add-on package for LiDIA.

REFERENCES

[Adleman et al. 1994] 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", pp. 28–40 in *Algorithmic number theory: ANTS-I* (Ithaca, NY, 1994), edited by L. M. Adleman and M.-D. Huang, Lecture Notes in Comp. Sci. **877**, Springer, Berlin, 1994.

[Artin 1924] E. Artin, "Quadratische Körper im Gebiete der höheren Kongruenzen, I", *Math. Zeitschrift* **19**

p	g	T_1	T_2	T_3	T_4
11	1	$6''$	$3''$	$1''$	$3''$
11	2	$49''$	$6''$	$3''$	$10''$
11	3	$13'43''$	$10''$	$7''$	$25''$
11	4	$44'32''$	$59''$	$17''$	$1'12''$
11	5		$1'30''$	$53''$	$2'1''$
11	6		$7'26''$	$1'1''$	$17'2''$
11	7			$2'46''$	
11	8		1h41$'$	$17'42''$	2h8$'$
11	9			$18'56''$	
11	10			$34'53''$	
11	11			1h11$'$	
11	12			3h41$'$	
101	1	$43''$	$16''$	$9''$	$7''$
101	2	1h31$'$	$1'53''$	$33''$	$43''$
101	3		$8'16''$	$2'30''$	$4'3''$
101	4		$22'31''$	$3'16''$	1h55$'$
101	5		23h31$'$	$20'6''$	
101	6			5h1$'$	
1009	1	1h13$'$	$4'31''$	$1'42''$	$11''$
1009	2		7h34$'$	$9'3''$	$3'37''$
1009	3		42h37$'$	1h40$'$	3h20$'$
10007	1		3h33$'$	1h5$'$	$41''$
10007	2		11h37$'$		$44'$
100003	1		5h3$'$		$2'34''$
1000003	1				$9'51''$
100000007	1				1h32$'$

TABLE 2. Timings for different sieving strategies. T_1: no sieving (comparable to the Adleman–Huang method). T_2: using only trial division to determine the factorizations. T_3: same as T in Table 1. For comparision, we also give the time T_4 needed by a baby-step-giant-step implementation.

(1924), 153–206. reprinted in *The collected papers of Emil Artin*, edited by S. Lang and J. Tate, Addison-Wesley, Reading, MA, 1965.

[Bach 1990] E. Bach, "Explicit bounds for primality testing and related problems", *Math. Comp.* **55**:191 (1990), 355–380.

[Buchmann and Paulus 1995] J. Buchmann and S. Paulus, "Algorithms for finite abelian groups", pp. 151–161 in *Number-theoretic and algebraic methods in computer science* (Moscow, 1993), edited by A. van der Poorten et al., World Sci. Publishing, River Edge, NJ, 1995.

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

[Domich et al. 1987] P. D. Domich, R. Kannan, and L. E. Trotter, Jr., "Hermite normal form computation using modulo determinant arithmetic", *Math. Oper. Res.* **12**:1 (1987), 50–59.

[Hafner and McCurley 1989] J. L. Hafner and K. S. McCurley, "A rigorous subexponential algorithm for computation of class groups", *J. Amer. Math. Soc.* **2**:4 (1989), 837–850.

[Havas and Majewski 1997] G. Havas and B. S. Majewski, "Integer matrix diagonalization", *J. Symbolic Comput.* **24**:3-4 (1997), 399–408.

[Koblitz 1989] N. Koblitz, "Hyperelliptic cryptosystems", *J. Cryptology* **1**:3 (1989), 139–150.

[LEDA n.d.] The LEDA Group, "LEDA: a C++ library of data types and algorithms for combinatorial computing", software, Max-Planck-Institute for Computer Science, University of Saarland, Germany. See http://www.mpi-sb.mpg.de/LEDA.

p	g	Type	Generators
11	1	(16)	$(X,\,10)$
11	2	(237)	$(X,\,1)$
11	3	(1130, 2)	$(X+2,\,1)$, $(X^2+10X+8,\,0)$
11	4	(6398, 2, 2)	$(X^4+10X^2+2X+8,\,X^3+8X^2+9X+9)$, $(X+8,\,0)$, $(X^4+10X^3+5X^2+5X+6,\,0)$
11	5	(80526)	$(X^5+6X^4+8X^3+1X^2+9X+7,\,10X^4+4X^3+5X^2)$
101	1	(92)	$(X+16,\,69)$
101	2	(4798, 2)	$(X^2+31+18,\,54X+3)$, $(X+58,\,0)$
101	3	(1079904)	$(X^3+26X^2+87X+74,\,99X2+16X+65)$
1009	1	(1060)	$(X+54,\,1006)$
1009	2	(501356, 2)	$(X^2+211X+561,\,885X+190)$, $(X+797,\,0)$
10007	1	(5014, 2)	$(X+6900,\,4673)$, $(X+2271,\,0)$

TABLE 3. Examples for Jacobians of $Y^2 = X^{2g+1} + 2X + 1$ over $\mathbb{F}_p$.

[Lenstra and Lenstra 1990] A. K. Lenstra and H. W. Lenstra, Jr., "Algorithms in number theory", pp. 673–715 in *Handbook of theoretical computer science*, vol. A, edited by J. van Leeuwen, Elsevier, Amsterdam, 1990.

[LiDIA n.d.] The LiDIA Group, "LiDIA: a C++ library for computational number theory", software, Technische Universität Darmstadt, Germany. See http://www.informatik.tu-darmstadt.de/TI/LiDIA.

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

[Mumford 1974] D. Mumford, *Abelian varieties*, 2nd ed., Tata Inst. Fund. Res. Stud. math. **5**, Oxford Univ. Press, London, 1974.

[Paulus 1996a] S. Paulus, "An algorithm of subexponential type computing the class group of quadratic orders over principal ideal domains", pp. 243–257 in *Algorithmic number theory: ANTS-II* (Talence, 1996), edited by H. Cohen, Lecture Notes in Comp. Sci. **1122**, Springer, Berlin, 1996.

[Paulus 1996b] S. Paulus, *Ein Algorithmus zur Berechnung der Klassengruppe quadratischer Ordnungen über Hauptidealringen*, Dissertation, Universität GH Essen, Essen, 1996.

[Paulus and Rück 1999] S. Paulus and H.-G. Rück, "Real and imaginary quadratic representations of hyperelliptic function fields", *Math. Comp.* **68**:227 (1999), 1233–1241.

[Paulus and Stein 1998] S. Paulus and A. Stein, "Comparing real and imaginary arithmetics for divisor class groups of hyperelliptic curves", pp. 576–591 in *Algorithmic number theory: ANTS-III* (Portland, OR, 1998), edited by J. P. Buhler, Lecture Notes in Comp. Sci. **1423**, Springer, Berlin, 1998.

[Seysen 1987] M. Seysen, "A probabilistic factorization algorithm with quadratic forms of negative discriminant", *Math. Comp.* **48**:178 (1987), 757–780.

Ralf Flassenberg, Institut für Experimentelle Mathematik, Universität — GH Essen, 45326 Essen, Germany. Current address: SECUNET Security Networks AG, Im Teelbruch 116, 45219 Essen, Germany (flassenberg@secunet.de)

Sachar Paulus, Institut für Theoretische Informatik, Technische Hochschule Darmstadt, 64283 Darmstadt, Germany. Current address: SECUDE Sicherheitstechnologie, Informationssysteme GmbH, Landwehrstraße 50a, 64293 Darmstadt, Germany (paulus@secude.com)

Received July 18, 1997; accepted in revised form November 5, 1998