The Function Field Sieve in the Medium Prime Case

Antoine Joux[1,3] and Reynald Lercier[1,2]

[1] DGA
[2] CELAR
Route de Laillé
35170 Bruz, France
Reynald.Lercier@m4x.org
[3] Université de Versailles St-Quentin-en-Yvelines
PRISM
45, avenue des Etats-Unis
78035 Versailles Cedex, France
Antoine.Joux@m4x.org

Abstract. In this paper, we study the application of the function field sieve algorithm for computing discrete logarithms over finite fields of the form $\mathbb{F}_{q^n}$ when q is a medium-sized prime power. This approach is an alternative to a recent paper of Granger and Vercauteren for computing discrete logarithms in tori, using efficient torus representations. We show that when q is not too large, a very efficient $L(1/3)$ variation of the function field sieve can be used. Surprisingly, using this algorithm, discrete logarithms computations over some of these fields are even easier than computations in the prime field and characteristic two field cases. We also show that this new algorithm has security implications on some existing cryptosystems, such as torus based cryptography in T_{30}, short signature schemes in characteristic 3 and cryptosystems based on supersingular abelian varieties. On the other hand, cryptosystems involving larger basefields and smaller extension degrees, typically of degree at most 6, such as LUC, XTR or T_6 torus cryptography, are not affected.

1 Introduction

Computing discrete logarithms is, with integer factorization, one of the two number-theoretical hard problems upon which public-key cryptography is usually based. Two kind of groups are often considered, elliptic curves and multiplicative groups of finite fields. The latter case is further partitioned into several sub-cases, prime fields $\mathbb{F}_p$, characteristic two fields $\mathbb{F}_{2^n}$, where n is usually prime and extensions of medium-sized fields $\mathbb{F}_{q^n}$, where q is a medium-sized prime power[4] p^k. Until recently, the last case was rarely considered in cryptography.

[4] Remark that we use the notation $\mathbb{F}_{q^n}$ to emphasize the fact that for composite extension degrees, viewing the field as $\mathbb{F}_{p^{kn}}$ is not necessarily optimal.

However, two recent developments make use of such fields, pairing-based cryptography [13, 5–7] and torus-based cryptography [19, 8, 21, 27]. For this reason, practical evaluation of the hardness of discrete logarithms in such fields is becoming an important issue. Recently, an approach based on rational torus representation was proposed by Granger and Vercauteren [12], it was applied in [22]. In this paper, we revisit a much older approach, the function field sieve. This algorithm was originally introduced by Adleman [3] as an extension of Coppersmith's algorithm [9]. Its complexity was subsequently improved by Adleman and Huang in [4]. This algorithm is known to be efficient when the base field is fixed and the extension degree grows. Moreover, it was shown to be practical and applied to characteristic 2 in [14]. Later on, it was also used in characteristic 3 in [11]. However, when both p and the total extension degree nk grow, the reference is the approach of Adleman and Demarrais in [2, 1], which makes use of a variation of Coppersmith's algorithm, involving function fields, when $p \leq nk$. As soon as this bound is exceeded they use a different algorithm based on number fields. This approach gives an $L(1/2)$ complexity for medium-sized base fields. In this paper, we describe a new variation of the function field sieve which is dedicated to medium-sized values of q and allows for fast computation of discrete logarithms in $\mathbb{F}_{q^n}$, even when q is much larger than n. For such fields, we show that our approach is faster, both from a theoretical complexity viewpoint with an $L(1/3)$ complexity and as a practical tool. More precisely, this variation of the function field sieve is applicable with $L(1/3)$ complexity whenever $\log q$ remains smaller than $O(\sqrt{n} \log n)$.

The paper is organized as follows, in section 2 we describe the function field sieve variation we are considering, in section 3 we show that the asymptotic complexity is the same as the complexity of the function field sieve with small base fields, in section 4 we describe real sized experimentations with this algorithm, finally in section 5 we discuss the impact of our algorithm on the security of some cryptosystems.

2 A medium sized variation on the function field sieve

The function field sieve algorithm for computing discrete logarithms over $\mathbb{F}_{p^n}$ is quite similar to the number field sieve for computing discrete logarithms over $\mathbb{F}_p$ (see [10, 28]). Both algorithms consider multiplicative identities using smooth objects over well-chosen smoothness bases. With the number field sieve, the objects are numbers in number fields and the smoothness bases contain ideals of small norm. With the function field sieve, the objects are polynomials in function fields and the smoothness bases contain ideals whose norms are polynomials of small degree. The complexity of such algorithms, is usually expressed using a notation, initially introduced for fast integer factorization algorithms [20]. This now classical notation is defined as follows:

$$L_Q(\alpha, c) = \exp((c + o(1))(\log Q)^\alpha (\log \log Q)^{1-\alpha}).$$

For the two extreme cases, prime fields $\mathbb{F}_p$ and extension fields $\mathbb{F}_{p^n}$ with fixed characteristic p, the number field sieve and the function field sieve respectively yield $L(1/3,(64/9)^{1/3})$ and $L(1/3,(32/9)^{1/3})$ algorithms. In the intermediate cases, the best available complexity is $L(1/2)$ as described by Adleman and Demarrais in [1, 2]. We would like to further remark, that using the function field sieve with fixed p, we have a smaller constant in the $L(1/3)$ expression than with the number field sieve. This is due to the fact that $\mathbb{F}_{p^n}$ has a large number of different representations, one for each irreducible polynomial of degree n over $\mathbb{F}_p$. This was discovered in [4] and a practical variant was presented in [14]. Surprisingly, even with medium-sized base fields, a similar construction that makes use of well chosen representations is possible, as shown below. The most important question is how to choose a good smoothness basis. With a medium sized base field $\mathbb{F}_{q^n}$, when q has just the right size, this is in fact very simple. It suffices to choose as the smoothness bases the sets of ideals whose norms are degree one polynomials, no more, no less. When $\log q$ and $\sqrt{n}\log n$ are correctly balanced, this choice yields a very efficient algorithm with complexity $L(1/3,3^{1/3})$. In section 2.2, we discuss different choices of smoothness bases that should be used instead of this simple choice, when the balance between q and n varies.

More precisely, let q be the cardinality of the base field and n the degree of the extension. In order to define $\mathbb{F}_{q^n}$, we proceed as follows. First, choose a minimal pair (d_1,d_2), with $d_2=d_1$ or d_1+1, and with $d_1d_2\geq n$. Then, find two polynomials f_1 and f_2, in two unknowns, X and t, of the form:

$$f_1(X,t)=X-g_1(t),\qquad f_2(X,t)=g_2(X)+t,$$

where g_1 and g_2 are univariate polynomials of degree d_1 and d_2, such that, $g_2(g_1(t))+t$ has an irreducible factor $F(t)$ of degree n over $\mathbb{F}_q$. We claim that such polynomials are easy to find (see section 4 for examples). We use $F(t)$ as our definition polynomial for $\mathbb{F}_{q^n}$. Clearly, f_1 and f_2 have a common root $X=g_1(t)$ in $\mathbb{F}_{q^n}$. As a consequence, f_1 and f_2 define good function fields for the function field sieve algorithm. Using standard vocabulary, we say that f_1 defines the linear side of the sieve.

The next step of the algorithm is to send objects of the form $a(t)X-b(t)$ in the two function fields. At this point, we slightly differ from standard practice and consider only a subset of such objects, by fixing $a(t)=wt+1$ and choosing $b(t)=ut+v$, where u, v and w are elements of the base field $\mathbb{F}_q$. As usual, we then compute the norm of $a(t)X-b(t)$ in the two function fields. This restriction on $a(t)$ comes from the fact that, since we are working with polynomials, all factorizations are defined up to a constant in the base field. This choice of $a(t)$ avoids multiple sieving of the same objects. Note that from a practical point of view, when q is large enough, it is even better to reduce the sieving space and fix $a(t)=1$ only. Then, on the linear side, we find $b(t)-g_1(t)$ a degree d_1 polynomial. On the other side, we find $g_2(b(t))+t$ a degree d_2 polynomial. This contrasts with the general case, where the respective degrees are d_1+1 and d_2+1. It is a well-known fact that among polynomials of degree d over $\mathbb{F}_q$, the proportion of degree d polynomials having d roots quickly tends towards $1/d!$ as q grows. We say that $b(t)$ generates a relation when both sides completely split into degree 1 factors. Using the traditional heuristic and assuming that the sieving process generates random looking polynomials, this occurs with a probability which is very close to either $1/(d_1! \cdot d_2!)$ or $1/((d_1+1)! \cdot (d_2+1)!)$. It remains to see whether we obtain enough relations. On the linear side, our chosen smoothness basis contains the q possible unitary polynomials of degree 1, namely the polynomials $t+u$, with u in $\mathbb{F}_q$. On the other side, due to our particular choice of f_2, the smoothness basis also contains q elements, which are ideals of norm $t+g_2(u)$, with u in $\mathbb{F}_q$. As a consequence, we need $2q$ equations. Since we are sieving over either q^2 or q^3 elements, this particular choice works when either $q \geq 2\ d_1! \cdot d_2!$ with reduced sieving space or $q^2 \geq 2\ (d_1+1)! \cdot (d_2+1)!$ with full sieving space.

After generating the multiplicative identities as above, we transform them into linear equations involving logarithms of polynomials on the linear side and "logarithms of ideals" on the other side[5]. The resulting system of equations is then solved using a sparse linear algebra algorithm such as Lanczos or Wiedeman [18, 23, 30, 17]. This linear algebra step is performed modulo $(q^n-1)/(q-1)$. Indeed, the multiplicative identities are defined up to a multiplicative constant in $\mathbb{F}_q$ and the logarithms are computed in the quotient group of $\mathbb{F}_{q^n}^*$ by $\mathbb{F}_q^*$. It is interesting to note that due to the very specific form of the equations we use, with exactly d_1 (or d_1+1) unknowns (potentially counting multiplicities) on the left-hand side and d_2 (or d_2+1) unknowns on the right-hand side, our system does not have full rank over the rationals. There is a "parasitic" solution with all the left-hand side unknowns set to d_2 and all right-hand side unknowns set to d_1. This means that after the linear algebra, the resulting solution does not contain pure discrete logarithms, the result is masked by some additive constant. However, by considering fractions such as $(t+u)/(t+v)$, the contribution of this constant can be cancelled. Moreover, if we can find even a single equation with a different structure, the masking constant can easily be found. The simplest way to proceed is to find a linear polynomial which completely splits in the function field defined by f_2. This yields a specific kind of equation[6] which nicely breaks the above symmetry and allows us to find and remove the unwanted constant. An example of this technique is given in section 4.

2.1 Individual discrete logarithms

Once the two steps described above, sieving and linear algebra, have been performed, we obtained the logarithms of the elements of the smoothness bases. This is well and good, but does not fully solve the discrete logarithm problem. An additional step is required to compute the logarithms of large elements in the finite field. We propose a classical approach based on "special-$\mathfrak{q}$" descent, which

[5] This notion of logarithms of ideals is described and used in [15, 14]. With the specific choice of f_2 we have given, there is a simpler description, because the function field is principal and all ideals can be represented by a single element in the finite field.

[6] These equations are often used in function field sieve algorithms and are called systematic equations.

is similar to the approach proposed in [9, 14] for the case of logarithms over an extension of a small base field. The idea is the following. Given an element y in the finite field, whose logarithm is wanted, we first build many elements of the form $y^i \cdot t^j$. Each of these elements can be represented as a polynomial in t of degree n. Alternatively, using continued fractions, we can also find representations by rational fractions, whose numerators and denominators have degrees near $n/2$. From an asymptotic viewpoint, both approaches are equivalent. However, in practice, the latter is more efficient. Once we obtain such a representation, we test whether it can be factored in polynomials of degree $\mu\sqrt{n}$ for a constant μ to be determined in the sequel. After testing sufficiently many representations, we find an adequate one and are left with the problem of computing logarithms of polynomials of degree at most $\mu\sqrt{n}$. Let $\mathfrak{q}$ be such a low degree polynomial. We can now find its logarithm by sieving again on elements of the form $a(t)X - b(t)$, where $a(t)$ and $b(t)$ are polynomials of degree at most $\mu\sqrt{n}$ chosen to ensure that $\mathfrak{q}$ divides the linear side (in the function field defined by f_1) of the resulting equations. After finding an element $a(t)X - b(t)$ that factor in both function fields into polynomials of degree smaller than the degree of $\mathfrak{q}$, we iterate the descent down to degree one, where all logarithms are known. This descent alternates between special-$\mathfrak{q}$ on the linear and the high degree function fields. Once the descent reaches degree one, we backtrack and compute the logarithms of each special-$\mathfrak{q}$ and finally the logarithms of $y^i \cdot t^j$ and y. If the special-$\mathfrak{q}$ values occurring at the first level are small enough, then the total degree of the objects to be factored in the next levels are strictly smaller and the bottleneck of this step is the search for a good representation of $y^i \cdot t^j$. In fact, this can be ensured by choosing μ such that $\mu\sqrt{n} \cdot (d_2 + 1) + d_1 < n$. We show that in the complexity analysis of section 3 and prove that choosing a value of μ between 1/2 and 1 ensures a good behavior of the individual logarithm phase.

2.2 Extension to smaller base fields

From a practical point of view, the above case is probably the most interesting. However, it is nice to know whether the approach can be extended to different choices of q and n. We now briefly describe a family of algorithm which neatly cover all the cases where q is smaller than above. Each algorithm depends on a main parameter D, which bounds the degree of norms of elements in the smoothness bases. The previous algorithm corresponds to $D = 1$. The general case of D is very similar to the restricted case $D = 1$. We construct the function fields in the same manner as above, only changing the choices of d_1 and d_2. More precisely, we take $d_1 \approx \sqrt{Dn}$ and $d_2 \approx \sqrt{n/D}$. We sieve over $a(t)X - b(t)$, where $a(t)$ and $b(t)$ are also degree D polynomials and $a(t)$ is unitary. The total size of the sieving space is q^{2D+1}. On the linear side, we need to factor a polynomial of degree at most $d_1 + D$ over the smoothness basis. On the other side, we need to factor a polynomial of degree at most $d_2 D + 1$. Dismissing constants and low order terms, both degrees are near $\sqrt{nD}$. In this context, we need to know the asymptotic smoothness probability of a degree n polynomial into factors of degree at most m. This problem has been widely studied and very precise estimates are given in [24]. However, results are usually given for fixed q, when both n and m grow. Here, m is fixed and both q and n grow. Yet, the logarithm of the probability of smoothness is still equivalent to $n/m \log(m/n)$. Moreover, in order to prove our complexity result, a lower bound on the probability is sufficient. For the sake of completeness, we prove this lower bound in appendix A. For simplicity of exposition, in the rest of this section, we work with equivalent expressions, however, the argument can easily be rewritten to accommodate a lower bound only.

From these estimates, we deduce that the logarithm of the probability of smoothness on each side is approximately $-\sqrt{n/D} \log \sqrt{n/D}$. Adding the two, we obtain a total logarithm of heuristic probability of $-\sqrt{n/D} \log(n/D)$. Moreover, the total size of the two smoothness bases is about $2q^D$. As with the case $D = 1$, we should make sure that we obtain enough equations, this approximately requires:

$$(D + 1) \log(q) \geq \sqrt{n/D} \log(n/D).$$

With this algorithm, the individual logarithms phase remains almost identical. The only needed change is to use polynomials of degree $\mu\sqrt{Dn}$ to represent the element under consideration. We analyze the heuristic complexity of this extension in section 3 and show that, as in the case $D = 1$, the right choice is to take μ between $1/2$ and 1.

2.3 Practical improvements

Large primes variation In the asymptotic analysis below (section 3), we observe that when q decreases below some point, we need to increase the parameter D to successfully compute discrete logarithms. However, this change of algorithm greatly increases the overall complexity. This happens when the sieving space is not large enough to get enough equations. However, right at the boundary, the number of missing equations is quite small. In that case, it is a good idea to use a large prime variation by allowing a small number of higher degree polynomials in each decomposition when splitting polynomials over the smoothness bases. This does not improve the asymptotic complexity, however, in practice, it can make the difference between a feasible and an infeasible computation. We do not further discuss this idea, which is classical in implementations of number field and function field sieves.

Use of Galois action In some specific cases, it is possible to use additional structure of the field $\mathbb{F}_{q^n}$ to improve the practicality of our algorithm. The basic idea is to use the Galois group in order to reduce the size of the smoothness bases. Assume that there exists an element ϕ of the Galois group which acts on both smoothness bases by sending any element to a conjugate also belonging to the smoothness basis. If we further express ϕ as a Frobenius power, we can reduce the number of unknowns in the linear algebra by a factor which is equal to the order of the action of ϕ on $\mathbb{F}_{q^n}$. We can also speed up the sieving process by the same factor, since less equations are needed. However, we should take care and avoid sieving on values for $a(t)X - b(t)$ yielding conjugate equations.

To make this idea more precise, let us discuss the specific case of $\mathbb{F}_{2^{nk}}$, where $q = 2^k$ and n and k are coprime. In that case, we can view $\mathbb{F}_{2^{nk}}$ as a tower of extensions or alternatively as a compositum. With the latter representation, we independently define $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^k}$ and put the two representations together to get $\mathbb{F}_{q^n}$. This means that f_1 and f_2 can both have their coefficients in $\mathbb{F}_2$. In that case, we take for ϕ the n-th Frobenius power, i.e., the mapping which sends x to x^{2^n} in $\mathbb{F}_{q^n}$. Clearly, if t is a root of the irreducible polynomial $F(t)$ defined by f_1 and f_2, it is an element of $\mathbb{F}_{2^n}$ and thus fixed by ϕ. However, the action of ϕ on $a(t)$ and $b(t)$ is not trivial. Indeed, assume that we work with parameter $D = 1$, then $b(t) = ut + v$ with u and v in $\mathbb{F}_{2^k}$. Unless u and v are both in $\mathbb{F}_2$ the image of $b(t)$ by ϕ is a different polynomial. Repeating the application of ϕ, we find yet another polynomial, and so on ... Since k and n are coprime, the order of the action of ϕ on $\mathbb{F}_{2^k}$ is k. As a consequence, the sieving process can be sped up by a factor[7] of k. Moreover, choose $t + u$ an element of the smoothness basis on the linear side. Clearly, $(t + u)^{2^n} = t + \phi(u)$ is another element of the same smoothness basis and the logarithms of the two elements say l_u and $l_{\phi(u)}$ are related by $l_{\phi(u)} = 2^n l_u$. This implies that the number of unknowns on the linear side can be divided by k. A similar argument also applies on the other side. As a consequence, we gain a speed-up by k on the sieving process and a speed-up by k^2 on the linear algebra.

Clearly, the same construction works for any small characteristic. Use of Galois action to speed-up the computation is also possible in other cases. In particular, for all fields of the form $\mathbb{F}_{q^2}$, it is possible to gain a constant speed-up of two. In some cases, it is also possible to have a larger speed-up. However, the details are much more technical and in particular may require to construct f_1 and f_2 in a different manner than the construction given at the beginning of the present section. We only illustrate this by giving an example in section 4.

3 Asymptotic heuristic complexity

In this section, given the respective values of q and n, we give the asymptotic complexity of our algorithm both for $D = 1$, for other fixed values of D and, finally, in the general case. It is convenient to let Q denote q^n and to assume, when the parameter D is fixed, that there exists a parameter α such that:

$$n = \frac{1}{\alpha} \cdot \left(\frac{\log Q}{\log \log Q} \right)^{2/3}, \qquad q = \exp\left(\alpha \cdot \sqrt[3]{\log Q \cdot \log^2 \log Q} \right).$$

Using this notation, we can analyze the complexity of each algorithm in the family, determined by the parameter D. Since there are two main phases, sieving and linear algebra, the total complexity expressed by $L(1/3, c)$ is determined

[7] Disregarding the rare cases where both $a(t)$ and $b(t)$ have all their coefficients in $\mathbb{F}_2$

by the maximum of the complexities of each phase. Let $L(1/3, c_1)$ be the complexity of the sieving and $L(1/3, c_2)$ be the complexity of linear algebra. Then, recalling from the analysis of section 2 that the smoothness basis has $O(q^D)$ elements and that the logarithm of the heuristic probability of finding a relation is $-\sqrt{n/D}\log(n/D)$, we find:

$$c_1 = \frac{2}{3\sqrt{\alpha D}} + \alpha D \quad \text{and} \quad c_2 = 2\alpha D.$$

Moreover, we need to check that we obtain enough equations. We recall that this approximately requires:

$$(D+1)\log(q) \geq \sqrt{n/D}\log(n/D) \quad \text{or} \quad (D+1)\alpha \geq \frac{2}{3\sqrt{\alpha D}}.$$

Whenever this condition is satisfied, we say that the algorithm with parameter D is applicable. Putting all these conditions together, we find that for each value of α we should use the lowest possible parameter D yielding an applicable algorithm. Moreover, in the range of applicability the complexity of each algorithm decreases with α. The optimal case for each algorithm happens when:

$$(D+1)\alpha = \frac{2}{3\sqrt{\alpha D}}.$$

Just below this threshold, we need to use the next algorithm in the family and the complexity jumps up to $L(1/3, c_2(D+1)) = L(1/3, 2\alpha(D+1))$. Thus at each threshold, a discontinuity occurs in the complexity . The largest such gap is between $D = 1$ and $D = 2$, at $\alpha = 3^{-2/3}$. On both sides of the gap, the respective complexities are $L(1/3, \sqrt[3]{3})$ for $D = 1$ and $L(1/3, \sqrt[3]{64/9})$ for $D = 2$. All the other gaps are smaller and the gap size decreases with D and tends to 0 as D grows. Moreover, the complexity tends to $L(1/3, \sqrt[3]{32/9})$. Thus, up to a single exception, the complexity of our family of algorithms is at worst the complexity of the number field sieve, is at best even better than the complexity of the function field sieve with fixed prime and tends to this latter complexity when D grows. The exception happens when α becomes too large even for $D = 1$, more precisely when $\alpha > \sqrt[3]{8/9}$. Indeed, in that case, the complexity $L(1/3, 2\alpha)$ is larger than the complexity of the number field sieve. We summarize this complexity analysis in figure 1. The three horizontal lines on this graph represents the constants $\sqrt[3]{3}$, $\sqrt[3]{32/9}$ and $\sqrt[3]{64/9}$.

General values of q. Another interesting question is to study the complexity of the algorithm when q grows more slowly than $L_Q(1/3, \epsilon)$ for all ϵ. In that case, we no longer use a fixed parameter D, but let it grow slowly with Q. More precisely, we choose for D the nearest integer to the solution d of the following equation:

$$q^d = L_Q(1/3, \sqrt[3]{4/9}).$$

This choice yields complexity $L(1/3, \sqrt[3]{32/9})$ in all the considered cases. Note that this includes the usual function field sieve, for fixed q, as a special case.

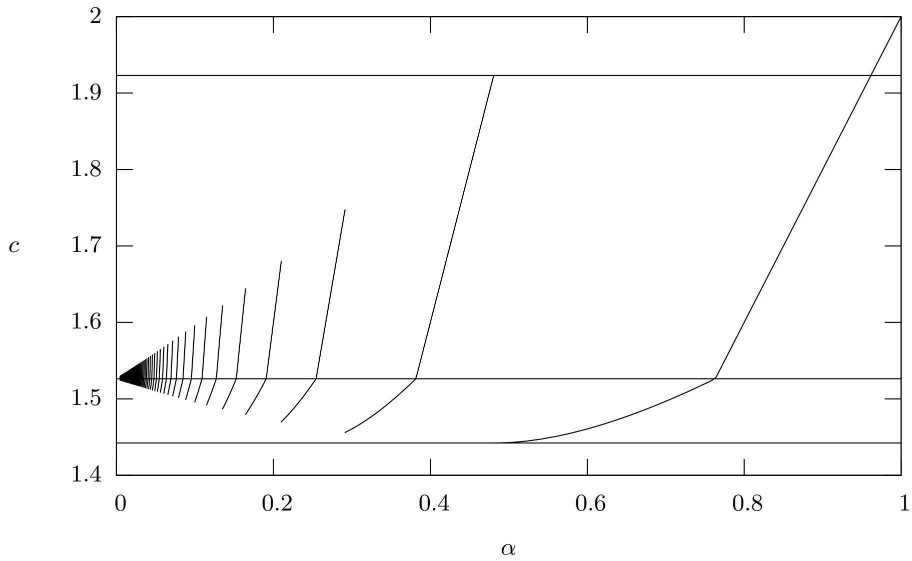

Fig. 1. Complexity $L_Q(1/3, c)$ as a function of $q = L_Q(1/3, \alpha)$

We also remark, that as announced in introduction the $L(1/3)$ boundary on q corresponds to the range where $\log q$ remains smaller than $O(\sqrt{n} \log n)$.

Individual logarithms. Concerning individual logarithms, we should choose a constant μ, both small enough to guarantee that the degrees of polynomials occurring during the descent are strictly decreasing and large enough to ensure that the initial good representation is found efficiently. Let $m = \mu\sqrt{Dn}$ be the maximal degree of the polynomials appearing in a good representation. Once again, we need to use the fact that the logarithm of the probability of smoothness is still equivalent to $n/m \log(m/n)$, even when m, n and q all grow. Moreover, as before, the argument could be rewritten using only the lower bound given in A.

Replacing n by its expression in term of Q we find that the probability is equivalent to

$$1/L_Q\left(1/3, \frac{1}{3\mu\sqrt{\alpha D}}\right).$$

We would like to ensure that the constant in this expression is smaller than the constant in the complexity of the main phase of the algorithm. This implies:

$$\frac{1}{3\mu\sqrt{\alpha D}} < \max(c_1, c_2), \text{ with } c_1 = \frac{2}{3\sqrt{\alpha D}} + \alpha D \text{ and } c_2 = 2\alpha D.$$

It clearly suffices to have: $\frac{1}{3\mu\sqrt{\alpha D}} < \frac{2}{3\sqrt{\alpha D}}$, where the right hand side is the first summand in c_1. This is true whenever $\mu > 1/2$.

Moreover, we need to make sure the special-$\mathfrak{q}$ descent involves polynomials of decreasing degree. Since the degrees of $a(t)$ and $b(t)$ during the descent are at most the degree of the special-$\mathfrak{q}$ itself, substituting in f_1 and f_2, we require: $(d_2\mu\sqrt{Dn}+1)+(d_1+\mu\sqrt{Dn}) < n$. Replacing d_1 and d_2 by their values and disregarding low order terms, we get: $\mu n < n$. This can be ensured by choosing $\mu < 1$. As a consequence, we can choose any value of μ in the range $]1/2; 1[$.

Finally, we need to check that at each step of the special-$\mathfrak{q}$ descent, there are sufficiently many pairs $(a(t), b(t))$ to obtain at least one relation. Potentially, we might expect to encounter problems for a special-$\mathfrak{q}$ of degree two, when trying to relate it to polynomials of degree one. In that case, the natural choice would be to select linear polynomials for $a(t)$ and $b(t)$. Since $a(t)$ has the restricted form $wt+1$, there are only $q^3/q^2 = q$ possible pairs involving the special-$\mathfrak{q}$ value. As a consequence, with such a choice for $a(t)$ and $b(t)$, we cannot guarantee that a relation can be found for this special-$\mathfrak{q}$ value. Thus, we need to use polynomials of degree two for $a(t)$ and $b(t)$. Of course, this lowers the smoothness probability which becomes:

$$\frac{1}{((d_1+2)!\cdot(2d_2+1)!)}$$

instead of $1/((d_1+1)!\cdot(d_2+1)!)$ in the main phase. However, since a single relation is needed, we keep the good asymptotic complexity. Indeed, in the least favorable case with respect to this issue, which happens to be the extreme case of the basic $(D=1)$ algorithm, the main phase probability is almost equal to $1/q^2$ and the main sieving costs q^3. Using the same parameters, the smoothness probability of the individual logarithm phase is asymptocally equivalent to $1/q^3$. Thus, in this worst case, the individual logarithm phase has the same asymptotic cost as the main phase. In all other cases, the main phase dominates the complexity.

4 Numerical examples

4.1 Basic example

Our first example is the computation of discrete logarithms over $\mathbb{F}_{65537^{25}}$. The cardinality Q of this field is a number of about 400 bits or 120 decimal digits. It can be factored as:

$$Q = 65536 \cdot 3571 \cdot 37693451 \cdot 137055701 \cdot 1085370589456396893 7051 \cdot P_{247}$$

Since the largest prime factor has 247 bits, Pollard's rho [25] is not practical for this example. As far as we know, this sets a new record for the computation over medium characteristic fields.

We first choose our function fields, fixing the two definition polynomials f_1 and f_2 as follows:

$$f_1(X,t) = X - t^5 - t - 3, \quad f_2(X,t) = X^5 + X + 1 + t.$$

Taking the resultant of f_1 and f_2, thus eliminating X, we find an irreducible polynomial $F(t)$ over $\mathbb{F}_{65537}$. We let α denote a root of $F(t)$ in the extension field. We also let β denote $\alpha^5+\alpha+3$.

Once this is done, we start the sieving process, using the reduced sieving space $X - (a\,t + b)$, with a and b in $\mathbb{F}_{65537}$. When we find a good pair (a, b) we obtain an equality between smooth objects. Indeed, the two function fields we are using are principal, thus whenever both norms are smooth, we can write an explicit identity between generators. For example, replacing X by $-2\,t + 20496$ in f_1 and f_2 yields smooth polynomials. Writing down explicit generators for the corresponding ideals, this yields the following equality:

$$\begin{aligned}(\alpha + 2445) \cdot (\alpha + 9593) \cdot (\alpha + 31166) \cdot (\alpha + 39260) \cdot (\alpha + 48610) =\\ \lambda(\beta + 43449) \cdot (\beta + 18727) \cdot (\beta + 17129) \cdot (\beta + 1946) \cdot (\beta + 49823),\end{aligned}$$

where $\lambda = -2$ is an element of $\mathbb{F}_{65537}$.

The sieving process itself is extremely fast, we give in appendix B the source C code of the program we used. This program finds all good (a, b) pairs in two minutes on a Pentium laptop at 1.6 GHz. Once the sieving is complete, each good pair yields a linear equation between 5 logarithms of elements $\alpha + u$ and 5 logarithms of $\beta + v$. We converted the output of the C program into linear equations using a short interpreted PARI/GP script. This conversion took an additional two minutes. It was as long as the sieve itself, however, it did not seem necessary to write a faster program for this task.

Solving the linear algebra system was the bottleneck of the algorithm. After some structured gaussian elimination, we had to solve a sparse system of 79 466 equations in 78 465 unknowns and 3.8 million entries. This was done using the Lanczos algorithm. In order to avoid divisions by non-invertible elements, we worked modulo $q_0 = Q/(65536 \cdot 3571)$. This took a little more than two days on the same laptop. The resulting solution gave logarithms, up to an additive constant. As explained in section 2, we determined the constant using the following systematic equation:

$$\beta \cdot (\beta + 16) \cdot (\beta - 16) \cdot (\beta + 4096) \cdot (\beta - 4096) = -(\alpha + 1).$$

After removing this additive constant and renormalizing the result, we had all the logarithms of elements $\alpha + u$ and $\beta + v$ modulo q_0. For example,

$$\begin{aligned} l = {}& 95805410880093234842298898214533393829434304594545362348248\\ & 840375483524017353229706334323184929723853320944439485,\\ m = {}& 4649571275692520918560124050338108397005057301288170051718\\ & 556686238431642289730613529631676496393555258546887691\end{aligned}$$

are the respective logarithms modulo q_0 of $\alpha + 1$ and β in base α. This can be checked by testing that $(\alpha + 1)^{3571 \cdot l}/\alpha^{3571}$ belongs to $\mathbb{F}_{65537}$, and similarly for $\beta^{3571 \cdot m}/\alpha^{3571}$.

The final step was to choose a random looking element of $\mathbb{F}_{65537^{25}}$ and to compute its complete logarithm. Since, α itself does not generate the full multiplicative group, we decided to express the logarithm in basis 3α, which is a generator. We took as challenge the element:

$$\lambda = \sum_{i=0}^{24} (\lfloor \pi \cdot 65537^{i+1} \rfloor \bmod 65537)\alpha^i = 41667\alpha^{24} + \cdots + 9279.$$

After finding a good representation of λ using polynomials of degree at most 3 and completing the special-$\mathfrak{q}$ descent, we added the contribution of the logarithm modulo the powers of 2 and modulo 3571. Finally, we concluded that the logarithm of λ in basis 3α is:

40537369450524407445879885072715457733779105170746399357547363481852609028577772820085371649268383536448936947412841469999.

4.2 Galois action example

We consider here a discrete logarithm challenge that is defined in $\mathbb{F}_{p^{30}}$ where $p = 370801$: such a finite field has got a 556-bit cardinality and it contains a 114-bit multiplicative subgroup. A smaller extension $\mathbb{F}_{p^{18}}$ has been recently performed by Vercauteren and Lercier [22] at the expense of one week over a network of 10 AMD's Athlon(TM) XP 2000+ for the sieving step and 12 hours for the linear algebra step, using the algorithm of [12]. To solve our $\mathrm{T}_{30}(\mathbb{F}_p)$ challenge, we first experimented with the algorithm defined in section 2. It turns out that, with $f_1(X,t) = X - (t^6 + t + 30)$ and $f_2(X,t) = X^5 + X + 1 + t$, a three hours computation for the sieving step and a two days computation for the linear step[8] would have been necessary on a 1.15 GHz 16-processors HP AlphaServer GS1280.

Thus, it was preferable to make use of the Galois action idea and define

$$f_1(X,t) = X - t^5 \text{ and } f_2(X,t) = X^6 + X - 17 - t^5.$$

This yields a definition polynomial for $\mathbb{F}_{p^{30}}$ equal to $F(t) = t^{30} - 17$, the Galois group of which is generated by $\phi : t \mapsto t^p = 172960 \times t$. With such a choice, f_1 and f_2 have a common root $X = t^5$, which is fixed by ϕ^6 and thus lies in the subfield $\mathbb{F}_{p^6}$. As a result, the conjugates by ϕ^6 of places (in both algebraic function fields) in the smoothness basis reduced modulo p are still elements of the smoothness basis. Since discrete logarithms of conjugates differ from each other by a power of p^6, we clearly divide by *five* the size of the smoothness basis: only 74161 places in the linear side[9] and only 74114 places in the other side.

With a sieving program similar to the one given in appendix B, we found 329082 useful divisors of functions $X - (at + b)$ with a and b in $\mathbb{F}_p$, in 45 minutes.

[8] This matrix is twice as big as the one used by Vercauteren and Lercier in $\mathbb{F}_{p^{18}}$. It is also twice as heavy.

[9] In truth, only 12361 places in the linear side are really necessary because conjugates by ϕ itself are again elements of the smoothness basis. Due to the additional coding work that would have been required, we did not take advantage of this speed-up in our experiment.

The supports of these divisors contain only degree one places and we restricted the values of a to avoid conjugate equations. Since, the reduction modulo p of these degree one places is equal to a suitable power of p^6 of one element of the smoothness basis, we clearly have enough equations.

We skipped the structured Gaussian elimination step, since at this time our code is not able to handle matrices with so many large coefficients. Of course, we had to modify our implementation of the Lanczos algorithm to handle this case. Finally, we were able to solve this sparse system of 150270 equations in 148270 variables (with 11 entries by row equal to powers p^{6i}, $i = 0, \ldots 4$) at the cost of a 10 hours computation on 8 processors of a 1.15 GHz HP AlphaServer GS1280. We worked modulo

$$q_0 = 129717983265199170691 \times 3780896193379818021601 \times \\ 27084969683231313608318791573698901.$$

Let us note that the kernel of this matrix has got only one vector (its coefficients are not all equal to one and thus, we do not have any "parasitic" solution). After this step, using the Galois action of ϕ^6, we have the logarithms, modulo q_0, of elements $t + u \in \mathbb{F}_{p^{30}}$ for any $u \in \mathbb{F}_p$.

In the final step, we took as challenge the element

$$\lambda = \sum_{i=0}^{29} (\lfloor \pi \times p^{i+1} \rfloor \bmod p) t^i = 162147t^{29} + \cdots + 52502.$$

We first write this element as a product of elements of degree at most four and using a special-q descent, we finally found that the logarithm of λ in basis $t - 6$ is:

$$\begin{array}{l} 83493475831866903958473832166988064644596198972030791927 \\ 23664325744787878765540875000760439341325398846364432518 \\ 4051550980392237533812685076653542562214928407573371226. \end{array}$$

5 Security implications

In this section, we discuss the applicability of our variation of the function field sieve to cryptosystems that make use of extension fields. First of all, we remark that for some systems, our approach is slower than generic algorithms and does not improve upon known attacks. Let us start by giving examples of such cryptosystems which are immune to our attack. The relevant property is that the systems make use of extensions of quite small degree over prime fields. Typically, the security of systems which use extension degree 6 over prime fields are not affected by this algorithm. In particular, this includes LUC [19], XTR [8, 21], CEILIDH [27], some pairing-based schemes as the complex multiplication variation of the short signature scheme of [6, 7] and also torus-based cryptography in T_6. When the extension degree is larger than that, it is important to reassess the security on a case by case basis. In the rest of this section, we do so for torus-based cryptography in T_{30}, the short signature scheme of [6, 7] in characteristic three and some of the supersingular abelian varieties proposed in [26].

For the case T_{30}, the base field is quite large and the bottleneck of the algorithm is the linear algebra whose complexity is $(d_1 + d_2)p^2$ additions modulo some factor of $p^{30} - 1$. In typical instantiation the relevant factor is a prime q_0 between 160 and 256 bits, according to the expected security level. We should compare our algorithm to a generic algorithm such as Pollard's rho [25], whose complexity is $\sqrt{q_0}$ operations in the finite field. Since additions modulo q_0 are less expensive than operations in the finite field and since $d_1 + d_2$ is small, it seems fair to proceed by comparing p^2 with $\sqrt{q_0}$. We conclude that for 80-bit security, it is necessary to choose for p a prime of 40 bits or more. For 32-bit primes and 160-bit subgroup, as proposed in [29], the expected security level is not reached and the effective security level is around 2^{64}. On the other hand, the security of the 64-bit primes examples with 200-bit subgroups proposed in the same paper is unaffected.

The short signature scheme of [6, 7] can be instantiated in two different ways. Either by using complex multiplication technique to build elliptic curves over $\mathbb{F}_p$ with a pairing that outputs numbers in $\mathbb{F}_{p^6}$. Or by using special supersingular curves over $\mathbb{F}_{3^\ell}$ with a pairing having values in $\mathbb{F}_{3^{6\ell}}$. Note that since the journal version [7], the characteristic three instantiation is no longer recommended. As said above, our algorithm does not change the security of the first case. In the second case, we can restate the problem as discrete logarithm in $\mathbb{F}_{q^\ell}$, where $q = 3^6 = 729$. From a practical point of view, this opens the possibility to use our algorithm with a parameter D equal to 2 or 3. With luck, and depending on the exact value of ℓ, we may fall in a zone where our algorithm is more efficient than the regular function field sieve in characteristic 3. Let us consider some of the usual possibilities. The easiest case is $\ell = 121$, since the extension field can even be viewed as a degree 33 extension of $\mathbb{F}_{3^{22}}$, which can be adressed with parameter $D = 1$, yielding a complexity near 2^{70}, which might be improved using Galois action. The expected Pollard rho complexity is 2^{78}. Similarly for $\ell = 97$ using $D = 2$, we find a complexity around 2^{71} instead of the expected 2^{76} and for $\ell = 149$, using $D = 3$ we find a complexity around 2^{105} instead of 2^{110}.

In fact, when looking at fields of characteristic three, our attack applies even better with the proposal of [26] which is to work with supersingular abelian varieties. Indeed, in the most extreme case, this proposal relies on the security of discrete logarithms in $3^{30\ell}$, which due to large choice of possible subfields is extremely likely to fall in a good case of our algorithm. In the same paper, the use of fields of the form $2^{12\ell}$ is also considered. Our algorithm can again be used here, especially when ℓ is composite (the cases $\ell = 121$ and $\ell = 87$ for example).

6 Conclusion

In this paper, we have presented a new variation of the function field sieve algorithm, which unexpectedly applies to finite field of the form $\mathbb{F}_{q^n}$ when both

q and n are of medium sized. This allows us to compute discrete logarithms in $\mathbb{F}_{q^n}$ faster than for discrete logarithms problems in field of a comparable size of the form $\mathbb{F}_p$ (with p prime) or even $\mathbb{F}_{2^n}$ (with n prime). This shows that despite former belief, discrete logarithms in some fields $\mathbb{F}_{q^n}$ are easier than in $\mathbb{F}_p$ or $\mathbb{F}_{2^n}$. As a consequence, we show that the security of some recent cryptosystems needs to be reassessed to account for this fact. We leave as an open question the problem of finding an efficient $L(1/3)$ algorithm for solving discrete logarithms in $\mathbb{F}_{q^n}$ when q is larger than $L_{q^n}(1/3)$. Up to q of the form $L(1/2)$, our algorithm with parameter $D = 1$ is the fastest known technique and has complexity q^2, beyond that one should turn to the number field based algorithm described in [1, 2] with complexity $L(1/2)$.

Last minute news. A recent preprint [16] describes a generalization of the number field sieve that is applicable to finite finite fields of size $Q = q^n$, whenever q grows faster than $L_Q(1/3)$. Put together with the present paper, this gives asymptotic complexity $L_Q(1/3)$ for discrete logarithms in all finite fields.

References

1. L. Adleman and J. DeMarrais. A subexponential algorithm for discrete logarithms over all finite fields. In D. Stinson, editor, *Proceedings of CRYPTO'93*, volume 773 of *Lecture Notes in Comput. Sci.*, pages 147–158. Springer, 1993.
2. L. Adleman and J. DeMarrais. A subexponential algorithm for discrete logarithms over all finite fields. *Math. Comp.*, 61(203):1–15, 2003.
3. L. M. Adleman. The function field sieve. In *Algorithmic Number Theory, Proceedings of the ANTS-I conference*, volume 877 of *Lecture Notes in Comput. Sci.*, pages 108–121, 1994.
4. L. M. Adleman and M. A. Huang. Function field sieve method for discrete logarithms over finite fields. In *Information and Computation*, volume 151, pages 5–16. Academic Press, 1999.
5. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Crypto '2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, 2001.
6. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Proceedings of ASIACRYPT'2001*, volume 2248 of *Lecture Notes in Comput. Sci.*, pages 514–532. Springer, 2001.
7. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *J. of Cryptology*, 17(4):297–319, 2004.
8. A.E. Brouwer, R. Pellikaan, and E.R. Verheul. Doing More with Fewer Bits. In *Advances in Cryptology — ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 321–332. Springer, 1999.
9. D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE transactions on information theory*, IT-30(4):587–594, July 1984.
10. D. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math*, 6:124–138, 1993.
11. R. Granger, A. Holt, D. Page, N. Smart, and F. Vercauteren. Function field sieve in characteristic three. In D. Buell, editor, *Algorithmic Number Theory, Proceedings of the ANTS-VI conference*, volume 3076 of *Lecture Notes in Comput. Sci.*, pages 223–234. Springer, 2004.

12. R. Granger and F. Vercauteren. On the discrete logarithm problem on algebraic tori. In V. Shoup, editor, *Proceedings of CRYPTO'2005*, volume 3621 of *Lecture Notes in Comput. Sci.*, pages 66–85. Springer, 2005.
13. A. Joux. A one round protocol for tripartite diffie-hellman. In *Fourth Algorithmic Number Theory Symposium*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394, 2000.
14. A. Joux and R. Lercier. The function field sieve is quite special. In C. Fieker and D. Kohel, editors, *Algorithmic Number Theory, Proceedings of the ANTS-V conference*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 431–445. Springer, 2002.
15. A. Joux and R. Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the gaussian integer method. *Math. Comp.*, 72:953–967, 2003.
16. A. Joux, R. Lercier, N. Smart, and F. Vercauteren. The number field sieve in the medium prime case. Preprint.
17. B.A. LaMacchia and A.M. Odlyzko. Solving Large Sparse Linear Systems Over Finite Fields. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer-Verlag, 1991.
18. C. Lanczos. Solutions of systems of linear equations by minimized iterations. In *J. Res. Nat.*, volume 49, pages 33–53. Bureau of Standards, 1952.
19. M.J.J. Lennon and P.J. Smith. LUC: A New Public Key System. In *IFIP TC11 Ninth International Conference on Information Security IFIP/Sec*, pages 103–117, 1993.
20. A. K. Lenstra and H. W. Lenstra, Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer–Verlag, 1993.
21. A.K. Lenstra and E.R. Verheul. The XTR Public Key System. In *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2000.
22. R. Lercier and F. Vercauteren. Discrete logarithms in $\mathbb{F}_{p^{18}}$ - 101 digits. NMBRTHRY mailing list, June 2005.
23. A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology — EUROCRYPT '84*, volume 209 of *Lecture Notes in Computer Science*, pages 224–314. Springer–Verlag, 1985.
24. D. Panario, X. Gourdon, and P. Flajolet. An analytic approach to smooth polynomials over finite fields. In J. Buhler, editor, *Algorithmic Number Theory, Proceedings of the ANTS-III conference*, volume 1423, pages 226–236. Springer, 1998.
25. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Math. Comp.*, 32:918–924, 1978.
26. K. Rubin and A. Silverberg. Supersingular abelian varieties in cryptology. In M. Yung, editor, *Proceedings of CRYPTO'2002*, volume 2442 of *Lecture Notes in Comput. Sci.*, pages 336–353. Springer, 2002.
27. K. Rubin and A. Silverberg. Torus-Based Cryptography. In *Advances in Cryptology — CRYPTO 2003*, volume 2442 of *Lecture Notes in Computer Science*, pages 349–365. Springer, 2003.
28. O. Schirokauer. Discrete logarithms and local units. *Phil. Trans. R. Soc. Lond. A 345*, pages 409–423, 1993.
29. M. van Dijk, R. Granger, D. Page, K. Rubin, A. Silverberg, M. Stam, and D. Woodruff. Practical cryptography in high dimensional tori. In R. Cramer, editor, *Proceedings of EUROCRYPT'2005*, volume 3494 of *Lecture Notes in Comput. Sci.*, pages 234–250. Springer, 2005.

30. D.H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. *IEEE Trans. Information Theory*, 32:54–62, 1986.

A Lower bound on the smoothness probability

In this appendix, we prove the lower bound of the probability of smoothness of polynomials of degree n over the basis of monic irreducible polynomials of degree at most m. As usual, it suffices to work with unitary polynomials of degree n and we denote by $N_q(n,m)$ the number of m-smooth unitary polynomials. Before giving our lower bound on $N_q(n,m)$, we recall that the number of monic irreducible polynomials of degree t is:

$$I_q(t) = \frac{1}{t}\sum_{d|t}\mu(t/d)q^t \geq \frac{1}{t}\left(q^t - \lceil \log_2 t \rceil q^{t/2}\right),$$

where μ denotes the Möbius function. We first show the expected lower bound when n is a multiple of m, $n = \ell m$. In that case, the number of smooth polynomials is greater than the number of possible products of ℓ distinct polynomials of degree m. This number is: $\frac{1}{\ell!}\prod_{i=0}^{\ell-1} I_q(m) - i$. Replacing I_q by its values, letting ℓ and q grow and dividing by q^n to get a probability, we obtain a lower bound of: $\frac{1}{\ell!(m+\epsilon)^\ell}$ for any value of $\epsilon > 0$. Taking the logarithm we find $\ell(\log \ell + m + \epsilon)$ which is asymptotically equivalent to $\ell \log \ell$ as expected.

In the general case, we write $n = \ell m + r$ with $r < m$ and proceed similarly with a product of one irreducible of degree r and ℓ distinct irreducibles of degree m. The lower bounds immediately follows.

B Listing of sieving C code for 65537^{25}

```
#include <stdio.h>
#include <stdlib.h>
#define PRIME 65537
int RootTab[2*PRIME]; int AlphaTab[2*PRIME]; char Count[PRIME];
AddSieveElement(int root, int alpha) { static int count=0;
  RootTab[count]=root; AlphaTab[count]=alpha; count++;
}
InitLinearSide() { /* Polynomial X-(t^5+t+3) */
  int alpha,root; long long tmp;
  for (alpha=0;alpha<PRIME;alpha++) {
    tmp=alpha; tmp*=tmp; tmp%=PRIME; tmp*=tmp; tmp%=PRIME;
    tmp*=alpha; tmp%=PRIME; tmp+=alpha+3; tmp%=PRIME;
    root=tmp; AddSieveElement(root,alpha);
}}
InitOtherSide() { /* Polynomial X^5+X+1+t */
  int alpha,root; long long tmp;
  for (root=0;root<PRIME;root++) {
    tmp=root; tmp*=tmp; tmp%=PRIME; tmp*=tmp; tmp%=PRIME;
    tmp*=root; tmp%=PRIME; tmp+=root+1; tmp%=PRIME;
    alpha=tmp; if (alpha) alpha=PRIME-alpha;
    AddSieveElement(root,alpha);
}}
FindMultiCollisions(int line) { int i,root;
  for (i=0;i<PRIME;i++) Count[i]=0;
  for (i=0;i<2*PRIME;i++) {root=RootTab[i]; Count[root]++;}
  for (i=0;i<PRIME;i++)
    if (Count[i]>=9) printf("b(t)=%d*t+%d;\n",-line,i);
}
UpdateTables() { int i,root;
  for (i=0;i<2*PRIME;i++) { root=RootTab[i]+AlphaTab[i];
    if (root>=PRIME) root-=PRIME;
    RootTab[i]=root;
}}
main() { int line; InitLinearSide(); InitOtherSide();
  for(line=0;line<PRIME;line++) {
    FindMultiCollisions(line); UpdateTables();
}}
```