Fine Tuning the Function Field Sieve Algorithm for the Medium Prime Case

Palash Sarkar
Indian Statistical Institute
palash@isical.ac.in

Shashank Singh
Indian Statistical Institute
sha2nk.singh@gmail.com

January 29, 2014

Abstract

This work builds on the variant of the function field sieve (FFS) algorithm for the medium prime case introduced by Joux and Lercier in 2006. We make two contributions. The first contribution introduces a divisility and smoothness technique which is similar to that of the special-q technique used in integer factorisation algorithms. Such a technique, though, has not been earlier used in the context of discrete log computations and provides concrete speed-ups in the practical run-time of the relation collection and the descent phases of the FFS algorithm. The second contribution is to improve the descent phase of the algorithm. The improvements are based on increasing the degree of freedom and the use of a walk technique. As a consequence, we show that it is feasible to carry out discrete log computations for certain fields which are excluded by the analysis of Joux and Lercier. In concrete terms, we present record computations of discrete logs for fields with 16 and 18-bit prime characteristic. Further, we provide concrete analysis of the effectiveness of the FFS algorithm for certain fields with medium sized prime characteristic.

1 Introduction

Let $\mathbb{F}_Q$ be a finite field and α a generator of the multiplicative group. Given an element β of $\mathbb{F}_Q$, the discrete log of β to base α, denoted as $\log_\alpha \beta$ (or simply as $\log \beta$ when α is implicit), is the integer $i \in [0, Q-2]$ such that $\beta = \alpha^i$. The discrete log problem (DLP) is the following: Given $\mathbb{F}_Q$, α and β, compute $\log_\alpha \beta$.

The function field sieve (FFS) [4, 5, 16] is an index calculus algorithm for solving the discrete log problem over finite fields. For small characteristic fields, there has been a good deal of recent research [11, 10, 9, 13, 6]. Barbulescu, Gaudry, Joux and Thome [6] have shown a quasi-polynomial time algorithm for extension fields with small characteristic. Further, Joux [14] has computed the discrete log in the binary extension field $\mathbb{F}_{2^{6168}}$. Concrete analysis and discrete log computation for certain characteristic 3 fields have been done in [1, 2, 3]. The ideas used for small characteristic fields, however, do not apply when the characteristic of the field is larger. In this work, we will be interested in values of Q of the type p^n where p is a medium sized prime (16-bit or larger).

A variant of the FFS applicable for medium sized primes has been proposed by Joux and Lercier in [17]. Recent progress for this variant has been reported by Joux in [15] where the important technique of pinpointing has been introduced.

The general form of the Joux-Lercier algorithm suggests the use of a factor base consisting of irreducible polynomials of maximum degree D. For solving discrete log over $\mathbb{F}_{p^n}$, the values p and n are to be balanced in the following sense.

$$n = n(Q) = \frac{1}{\alpha}\left(\frac{\ln Q}{\ln\ln Q}\right)^{2/3} \text{ and } q = \exp\left(\alpha\sqrt[3]{\ln Q \cdot \ln^2 \ln Q}\right) \tag{1}$$

where $Q = p^n$ and α is such that

$$\alpha^{3/2} \geq \frac{2}{3(D+1)\sqrt{D}}. \tag{2}$$

We will use ln to denote natural logarithms and lg to denote logarithms to base 2.

The algorithm is parameterised by D and as mentioned in [17], the optimal case for each algorithm happens when (2) holds with equality. The size of the factor base is about $2p^D$. It is suggested in [17] that by varying D it is possible to have algorithms to solve discrete log problem for a large range of Q. Let us consider the problem of computing discrete log for fields $\mathbb{F}_{p^n}$ where p is a 16 or 18-bit prime. The numerical examples given in [17] considered primes of these sizes, so, we are not the first to consider such primes.

Suppose the value of D is chosen to be 2. Then the factor base consists of about $2p^2$ elements. To find logarithms of the elements of the factor base, the number of relations should be a little more than the size of the factor base. Each relation will involve a constant number, say c, of elements of the factor base. For the linear algebra step, the resulting matrix will have about $2p^2$ rows. The complexity of the block Wiedemann or the Lanczos algorithm is $O(RW)$ where R is the number of rows and W is the number of non-zero elements in the matrix. Applied to the matrix obtained after the relation collection step, the complexity will be about $4cp^4$. If p is a 16-bit prime, the complexity is about 2^{66} ring operations; if p is larger, then the difficulty of the computation grows. Also, the complexity grows if D is chosen to be greater than 2. This shows that for primes of the size 16-bit or larger, taking D to be greater than 1 makes the resulting computation infeasible. In view of this fact, in this paper we work only with $D = 1$.

For $D = 1$, the optimal value of α is $3^{-2/3}$ and the asymptotic complexity of the Joux-Lercier algorithm is $L_Q(1/3, 3^{1/3})$, where $L_Q(a, c)$ with $0 < a < 1$ and $c > 0$ denotes the sub-exponential expression

$$\exp\left((c + o(1))(\ln Q)^a(\ln\ln Q)^{1-a}\right).$$

The use of the pinpointing technique in [15] reduces the value of c in $L_Q(a, c)$ without affecting the value of a.

Suppose α is fixed to $(1/3)^{2/3}$ (corresponding to $D = 1$). Then (1) provides n as a function of Q. Let us denote this value of n as $n(Q)$. Table 1 shows the values of $\lg Q$, $n(Q)$ along with the values of n and p that have been actually been tackled in [17, 15] and here. For the previous computations shown in Table 1, the extension degree n that has been tackled is at most the value of $n(Q)$ which tallies with the analysis done in [17]. In contrast to the previous works, we perform DLP computations for $\mathbb{F}_Q$ where the extension degree n is greater than $n(Q)$. The differences are small, but, in concrete terms suggest that the asymptotic analysis in [17] may not cover all the possible tunings of the parameters that are involved. Apart from the fields for which we have actually computed discrete log, we also perform a concrete analysis of other fields. For such fields, the gap between the n that can be tackled and $n(Q)$ turns out to be significantly more. More details are provided in Section 6.

Table 1: Values of $\lg Q$, $n(Q)$ along with the values of n and p for which actual discrete log computations have been reported.

Ref.	p	$\lg p$	n	$\lg Q$	$n(Q))$	cond
[17]	65537	16	25	400	28	–
	370801	18.5	30	556	34	$n\|p-1$
[15]	33553771	25	47	1175	52	$n\|p-1$
	33341353	24.99	57	1425	57	$n\|p-1$
Here	64373	15.97	37	592	35	–
	297079	18.18	40	728	39	–

Our contributions: We revisit the FFS algorithm as described in [17, 12, 15]. One of our contributions is to introduce a divisibility and smoothness technique which reduces the practical run-time of the relation collection and the individual logarithm phases of the FFS algorithm. This technique can be considered to be the discrete log counterpart of the special-q technique used in integer factorisation algorithms. To the best of our knowledge, the technique has not been earlier reported in the context of the discrete log problem. We work out the details of the idea and the concrete gains in run-time that can be obtained.

Detailed consideration of the descent phase of the FFS algorithm shows that descending from degree-2 polynomials to degree-1 polynomials is the most problematic step. The computations used in [17, 15] for such descent (which we call 2-1 descent) use one degree of freedom, resulting in at most p trials. This upper bounds the value of n for which the 2-1 descent can be carried out. To tackle this problem, the works [17, 12] briefly mention the possibility of using 3 degrees of freedom resulting in p^3 trials, but, lower smoothness probability. This idea though is not fully explored.

A second contribution of this work is to explore in details the effect of increasing the degree of freedom for the 2-1 descent step of the algorithm. We provide a systematic framework for increasing the degree of freedom. In particular, the case of 2 degrees of freedom (not considered in [17, 12, 15]) is important. We are able to use 2 degrees of freedom to compute discrete log over a field with 18-bit characteristic and extension degree 40.

There is another issue regarding a 2-1 descent which has been briefly mentioned in [15]. Suppose that the 2-1 descent fails for a certain quadratic polynomial. Then [15] suggests that one should move to another quadratic polynomial and attempt the descent. No details, however, are provided as to whether such a strategy will always succeed and the number of times it would be required to move from one quadratic polynomial to another which would be the length of the walk.

We develop the details of the walk technique by considering several options and the associated probabilities. It turns out that the walk technique will not always work and we provide explanations of why it may not work. We have conducted experiments with the walk technique for 3 fields. For the 25-bit prime, extension degree 57 field considered in [15], the walk lengths are a few steps. Apart from this, we consider two other fields. The first one has 16-bit characteristic and extension degree 37 and the walk technique works for this field. The average walk length turns out to be 17. The other field has an 18-bit characteristic and extension degree 40. For this field, the walk technique does not work due to a branching effect which we explain. Accordingly, we used a higher degree of freedom for this field.

In terms of actual computations, we report two record discrete log computations for the above mentioned fields, i.e., 16-bit characteristic, extension degree 37 (592-bit field); and, 18-bit characteristic and extension degree 40 (768-bit). Prime characteristic of these sizes were earlier considered in [17]. The extension degrees that we have tackled are currently the highest known.

In [15] discrete log computations have been reported over larger fields $\mathbb{F}_{p^n}$ where $n|p-1$. The last condition allows reducing the size of the factor base by a factor of about n considerably easing the computation of the linear algebra step. While this is useful, the property $n|p-1$ is restrictive. In the examples that we consider, this property does not hold. As a consequence, for *general* medium-characteristic fields, the 768-bit field that we consider is currently the largest over which discrete log computations have been carried out. On the other hand, the techniques that we develop are general in nature and can also be used with the special fields considered in [15].

We go beyond the actual computation of discrete log and perform a concrete analysis of the application of the algorithm to certain specific fields. For 16, 18, 20 and 25-bit primes, we indicate the maximum extension degrees that can be feasibly handled. We believe that access to a supercomputer will make these computations possible. For 32-bit primes, we show that tackling extension degree 100 can be done in about 2^{80} steps.

The values of the parameters of FFS that we consider lead to a situation where the individual descent phase (more specifically the 2-1 descent step) requires more time than the relation collection phase. It is perhaps due to this reason that we are able to tackle extension degrees which cross the value of $n(Q)$. The asymptotic analysis in [17] considered the situation where the time for the individual descent phase is at most that of the main phase.

2 A Description of the Function Field Sieve Algorithm for the Medium Prime Case

The function field sieve algorithm [4, 5] is an index calculus technique used for computing discrete logarithm on finite fields. Our description of the algorithm is based on the two papers [17, 15] and the book [12]. The focus of our work is the medium prime case and so we do not discuss the improvements that can be made when the characteristic of the field is a small prime. There are three phases of the algorithm, namely, relation collection, linear algebra and the individual discrete log phase. There are two auxiliary phases called the preparatory phase and the final phase. Descriptions of these phases are given below.

2.1 Preparatory Phase

In the preparatory phase, a suitable representation of the finite field is chosen. Given a prime p and a positive integer n, upto isomorphism, there is exactly one finite field of order p^n. Given two different representations of the field of order p^n, it is easy to compute the isomorphism between them. Due to this, one is free to choose any convenient representation to solve the discrete log problem. The solution can later be transferred to any other given representation.

Field representation: The field $\mathbb{F}_{p^n}$ is realised as $\frac{\mathbb{F}_p[x]}{\langle f(x)\rangle}$ where $f(x)$ is an irreducible polynomial of degree n and is determined as follows: Choose two polynomials $g_1(x)$ and $g_2(x)$ in $\mathbb{F}_p[x]$

of degrees n_1 and n_2 respectively such that $f(x)$ is a factor of $x - g_2(g_1(x))$. The polynomial $x - g_2(g_1(x))$ itself may be irreducible and can be taken to be $f(x)$, otherwise, one chooses as $f(x)$ the highest degree irreducible factor of this polynomial. The values of n_1 and n_2 are crucial to determining the complexity of the algorithm.

The benefit of choosing such a representation is the following. Since $f(x)$ divides $x - g_2(g_1(x))$, if we set $y = g_1(x)$, then $x = g_2(y) \bmod f(x)$. So, the equalities

$$y = g_1(x) \text{ and } x = g_2(y) \tag{3}$$

provide two basic relations between two elements of $\mathbb{F}_{p^n}$. These relations are at the heart of the algorithm and was first suggested in [17].

The $n_1 n_2 + 1$ variant: In the above representation, we obtain $n \leq n_1 n_2$. Joux [15] has further suggested a way to increase the extension degree while keeping n_1 and n_2 the same. If we take $g_2(y) = ty^{-n_2}$, we can get the extension degree $n = n_1 n_2 + 1$ with minor changes in the relation collection and the individual logarithm phases. In the examples considered in [15], $g_1(x)$ was taken to be x^{n_1} which requires working with a Kummer extension and the condition $n|p-1$.

It is not necessary, however, to take both $g_1(x)$ and $g_2(y)$ to be of the special form. Setting $g_2(y) = y^{-n_2}$ one can allow $g_1(x)$ to be arbitrary. The relation collection and the individual logarithm phases can be made to go through. Further, one may choose $g_1(x) = x^{-n_1}$ and $g_2(y)$ to be of general form and still obtain an appropriate field representation. Our example (later) of 16-bit prime and extension degree 37 illustrates these points.

Generator of the field: The generator of the finite field (and the base of the discrete logarithms) is fixed as follows. If $f(x)$ is primitive polynomial, then x itself is taken as a generator and hence $\log(x) = 1$. More generally, we can take as the generator any primitive element which is smooth over the factor base. In practice it turns out that for some $a_j \in \mathbb{F}_p$, $x + a_j$ generates field $\mathbb{F}_{p^n}$. In that case, we have $\log(x + a_j) = 1$.

Factor base: The factor base is the following.

$$\mathbb{B} = \{(x + a_i), (y + b_j) : a_i, b_j \in \mathbb{F}_p\}.$$

In other words, the factor base consists of all polynomials of degree one in x and y. So the size of the factor base is $2p$. More generally, one can define the factor base to consist of all irreducible polynomials in x and y of degrees at most D. This leads to a factor base of size about $2p^D$. As explained earlier, for a medium-sized p (16-bit or more in our case), having $D = 2$ makes the resulting linear algebra phase infeasible. So, we do not consider this option.

It is possible to reduce the size of the factor base by suitably choosing n and p. As explained in [17] in terms of Galois action and in [15] in terms of Kummer extensions, for certain choices of n and p, the size of the factor base can be reduced by a factor of n. This reduction can allow the computation of the linear algebra for a suitably larger prime p.

While this is interesting, achieving this loses the generality of the algorithm. A requirement for such reduction is that $\mathbb{F}_p$ contains all the n roots of unity. This in turn requires the condition that n divides $p-1$. So, to apply the reduction technique to the factor base, one has to choose n and p such that $n|p-1$. In general, one would not be allowed to choose the values of n and

p; these would be provided and in such a case, it is quite unlikely that the condition $n|p-1$ will hold. In view of this, we have decided not to work with this option.

Modulus of discrete log: Since we are working over $\mathbb{F}_{p^n}$, discrete logs are defined modulo p^n-1. Suppose α is a generator of $\mathbb{F}_{p^n}$ so that the order of α is p^n-1. An element α^i is in $\mathbb{F}$ if and only if $\alpha^{ip}=\alpha^i$, i.e., if and only if, $(p^n-1)|i(p-1)$. Let $M=(p^n-1)/(p-1)$. Then α^i is in $\mathbb{F}$ if and only if i is a multiple of M.

Suppose $c\in\mathbb{F}_p$ and $\beta\in\mathbb{F}_{p^n}$. Then the discrete log of $c\beta$ is $\log c+\log\beta$. By the above observation, $\log c\equiv 0 \bmod M$ and so the discrete log of $c\beta$ modulo M is simply $\log\beta \bmod M$. In other words, by working modulo M, we can ignore constants arising in the intermediate stages.

In practice, one does not even work modulo M. Instead, first the discrete log is computed modulo the large prime factors of M and then combined using the Chinese remainder theorem. At a later stage, the Pollard's rho and/or the Pohlig-Hellman algorithms are applied to compute the discrete log modulo the small factors of p^n-1 to get the actual discrete log.

2.2 Relation Collection Phase

In the relation collection phase, the aim is to compute multiplicative relations amongst the elements of factor base. This is achieved as follows. Consider a field element of the following form:

$$(x+a)y+(bx+c)=xy+ay+bx+c \text{ where } a,b,c\in\mathbb{F}_p. \tag{4}$$

Expression (4) can be considered to have two equivalent representations, one in terms of x and another in terms of y, i.e.,

$$h_1(x)\stackrel{\Delta}{=}xg_1(x)+ag_1(x)+bx+c \text{ and } h_2(y)\stackrel{\Delta}{=}g_2(y)y+ay+bg_2(y)+c. \tag{5}$$

Suppose both $h_1(x)$ and $h_2(y)$ are $\mathbb{B}$-smooth, i.e., they factor into linear terms as $h_1(x)=c_1\prod_{\alpha_i}(x+\alpha_i)$ and $h_2(y)=c_2\prod_{\beta_j}(y+\beta_j)$, where $c_1,c_2\in\mathbb{F}_p$. Then a relation of the following form is obtained.

$$c_1\prod_{\alpha_i}(x+\alpha_i)=c_2\prod_{\beta_j}(y+\beta_j). \tag{6}$$

As explained above, the constants c_1 and c_2 are ignored by computing discrete log modulo M. The relation (6) provides the following relation between the discrete logs of some of the elements of the factor base.

$$\sum_{\alpha_i}\log(x+\alpha_i)=\sum_{\beta_j}\log(y+\beta_j) \mod M \tag{7}$$

In the relation collection phase, more than $2p$ such relations are collected so that there are $2p$ linearly independent equations involving the discrete logs of the factor base elements.

Obtaining a relation of the form (6) is dependent on the degree n_1+1 of $h_1(x)$ and the degree n_2+1 of $h_2(y)$. The usual heuristic is to assume that the two polynomials behave like independent random polynomials. It is known that the probability that a random polynomial

of degree m factors into linear terms is about $\frac{1}{m!}$. Under the assumption that $h_1(x)$ and $h_2(y)$ behave like independent random polynomials, the probability of obtaining a single relation is about $1/((n_1+1)!(n_2+1)!)$. The quantities a, b and c in (4) provide three degrees of freedom leading to a total of p^3 choices. Among these, about $p^3/((n_1+1)!(n_2+1)!)$ relations are to be expected. The relation collection phase succeeds if

$$\frac{p^3}{(n_1+1)!(n_2+1)!} > 2p. \tag{8}$$

The quantity on the left side is maximised when n_1 and n_2 are roughly equal.

Pinpointing: Joux [15] introduced the pinpointing technique to speed up the relation collection phase. The technique works for any given p and n. The idea is to choose $g_1(x) = x^{n_1}$. This choice of $g_1(x)$ is not restrictive. Experiments show that it is possible to set $g_1(x) = x^{n_1}$ and then obtain a suitable $g_2(y)$ such that the polynomial $x - g_2(g_1(x))$ is irreducible and can be taken to be the field defining polynomial $f(x)$.

It is required to factor $xg_1(x) + ag_1(x) + bx + c$ into linear terms. With $g_1(x) = x^{n_1}$, the polynomial $h_1(x) = xg_1(x) + ag_1(x) + bx + c$ becomes $x^{n_1+1} + ax^{n_1} + bx + c$. Suppose that $x^{n_1+1} + ax^{n_1} + bx + c$ factors into linear terms, i.e.,

$$x^{n_1+1} + ax^{n_1} + bx + c = \prod(x + \alpha_i). \tag{9}$$

The nice idea of Joux is to observe that using the transformation $x \to tx$, $t \in \mathbb{F}_p$, ensures that the right side of (9) remains smooth while the left side transforms into a polynomial of a similar form with t^{n_1+1} as the leading coefficient. By working modulo M, it is possible to divide the whole of left side by t^{n_1+1} to get a monic polynomial of the same form with different a, b and c. The polynomial $h_2(y)$ is then still of degree $n_2 + 1$.

As a result, once after $(n_1+1)!$ trials, one obtains a single set of values for a, b and c with x-side smooth, then by varying t over all non-zero elements of $\mathbb{F}_p$, it is possible to obtain $p-1$ x-side smooth polynomials at very little extra cost. Using pinpointing, the amortised cost of obtaining one relation which has both sides smooth, is [15]

$$\frac{(n_1+1)! + (p-1)}{(p-1)/(n_2+1)!} = \frac{(n_1+1)!(n_2+1)!}{p-1} + (n_2+1)! \tag{10}$$

The choice of $g_1(x) = x^{n_1}$ helps in speeding up the x-side computation of the relation collection phase. This corresponds to applying pinpointing from the x-side. Similarly, choosing $g_2(y)$ to be y^{n_2} one can apply pinpointing from the y-side. The choice of which side to apply pinpointing from depends on the relative values of n_1 and n_2. If $n_1 \geq n_2$, then it is advantageous to apply pinpointing from the x-side and if not, then it is advantageous to apply the technique from the y-side.

The question arises as to whether it is possible to simultaneously choose $g_1(x) = x^{n_1}$ and $g_2(y) = ty^{n_2}$ for some $t \in \mathbb{F}_p$ and whether such a choice indeed speeds up both sides of the computation of the relation collection phase. The answer to both questions is yes and the technique has been called advanced pinpointing [15]. However, advanced pinpointing does not apply in general; to apply this technique, one has to carefully choose n and p. Due to this reason, we do not discuss the technique any further in this work.

The size of the factor base is $2p$ and so a little more than $2p$ relations are collected between the discrete logs of the elements of the factor base. The linear algebra phase computes the discrete logs (modulo large prime factors of $p^n - 1$) of the elements of the factor base.

2.3 Linear Algebra

Once sufficient number of relations is obtained, the algorithm proceeds to the linear algebra phase. The system of linear equations that is obtained is highly sparse. The works [17, 15] first reduce the number of unknowns using structured Gaussian elimination and the apply either the Lanczos algorithm or the block Wiedemann algorithm to solve the resulting system of linear equations. The computations are separately done modulo the large prime factors of $p^n - 1$ and then combined using the CRT. In [15], the linear algebra step is carried out using a super-computer.

For our computations, we have used MAGMA [7] to perform the linear algebra step. MAGMA provides options for using either the structured Gaussian elimination or the Lanczos algorithm. There does not appear to be a method by which the structured Gaussian elimination is used up to some extent and then the Lanczos algorithm is applied. As a result, we used the Lanczos option. The computation was done on a single core without any parallelisation.

2.4 Individual logarithm phase

The aim of the individual discrete log phase is to express a given element $\phi(x)$ of the field as a ratio of products of elements of the factor base thereby expressing the discrete log of $\phi(x)$ as a sum and difference of the discrete logs of the elements of the factor base. This is achieved recursively. First express $\phi(x)$ as a ratio of products of polynomials of degrees less than the degree of $\phi(x)$. Next, the same algorithm is applied to each factor of $\phi(x)$ recursively until the process descends to linear factors. This phase is also known as the descent phase as in every step the degree keeps on descending until linear terms are reached.

The initial descent is quite fast and is achieved as follows. Randomly choose a few linear and quadratic polynomials and call their product $\Delta(x)$. If $\Delta(x)\phi(x)$ factors into significantly lower degree polynomials, then $\Delta(x)\phi(x) = N(x)$ where $N(x)$ is product of low degree polynomials . The following algorithm is then applied to each factor of $\Delta(x)$ and $N(x)$. Suppose $\phi(x)$ is a factor of degree d and it is desired to reduce the problem of finding the log of $\phi(x)$ to that of finding logs of lower degree polynomials. Let $T(x, y)$ $(= A(x)y + B(x))$ be a bi-variate polynomial where the degree of x is t_1 and the degree of y is t_2. Then $T(x, y)$ has $(t_1 + 1)(t_2 + 1)$ monomials of the form $x^i y^j$ with $0 \leq i \leq t_1$ and $0 \leq j \leq t_2$. Write $T(x, y)$ into two equivalent forms $F(x) \triangleq T(x, g_1(x))$ and $C(y) \triangleq T(g_2(y), y)$ in the variables x and y respectively. Three things are to be ensured: $\phi(x)$ divides $F(x)$; $G(x) \triangleq F(x)/\phi(x)$ is $(d-1)$-smooth; and $C(y)$ is also $(d-1)$-smooth. Then computing the logs of the factors of $G(x)$ and $C(y)$ will provide the log of $\phi(x)$.

This defines the descent from a degree d polynomial in x to degree $d-1$ polynomials in x and also polynomials in y. As a result, in subsequent steps, it is also required to apply the descent method to the polynomials in y. This is a one-step descent. In fact for higher values of d, it is possible to get several steps of descent in one round, but for the lower values of d, the descent proceeds one step at a time. It is the lower side of descent, particularly the 2-1 descent, which takes most of the time.

The idea of the descent from a polynomial in y is the same, except that one starts with $A(y)x + B(y)$. The descent from y-side will also involve lower degree polynomials in y as well as polynomials in x. The method is continued until descent to degree 1 is achieved for both polynomials in x and polynomials in y whence the elements of $\mathbb{B}$ are involved. Since the logs

of these elements have been computed in Phase 2, it is possible to use these values and retrace the descent steps to compute the log of $r(x)$.

2.5 Restrictions on the Extension Degree

For a given p, Equation (8) which determines the possibility of relation collection also provides an upper bound on the values of n_1 and n_2 (and so on the value of n) that can be tackled using the FFS algorithms in [15, 17].

The values of n_1 and n_2 are further restricted by the 2-1 descent. For this descent, [17, 15] work with the form $T(x, y) = A(x)y + B(x)$ where the degrees of $A(x)$ and $B(x)$ are d_1 and d_2 respectively. The degree of $\phi(x)$ for which descent is attempted has degree $d = 2$. As a result, the degree of freedom is $(d_1 + 1)(d_2 + 1) - d - 1 = 1$. So, there is a single degree of freedom which allows for p trials. In these many trials, it is desired to obtain $G(x)$ and $C(y)$ to be smooth. The degrees of $G(x)$ and $C(y)$ are $n_1 - 1$ and $n_2 + 1$ respectively. Heuristically, the probability that both are smooth is $1/((n_1 - 1)!(n_2 + 1)!)$. So, for the descent to be possible, it is required that

$$\frac{p}{(n_1 - 1)!(n_2 + 1)!} \geq 1. \tag{11}$$

It is clear that if p is such that (11) holds, then certainly (8) is also satisfied. So, given p, (11) determines the maximum values of n_1 and n_2 and hence of n for which the method can be successful. What happens if the condition in (11) does not hold? Two methods are briefly indicated in [17, 15, 12].

Walk: Suppose $\phi(x)$ is a quadratic polynomial for which the algorithm is unable to descend to linear polynomials. To tackle such a scenario, the following has been mentioned in [15]. "When not possible, we use a relation that also includes another degree 2 polynomial and restart from that polynomial." This is very brief and does not address the questions about whether the walk technique will always succceed and the number of steps in the walk that would be required before a descent is obtained.

Additional degrees of freedom: Both [17, 12] mention that one way to tackle the problem is to increase the degrees of $A(x)$ and $B(x)$ to 2. This increases the degree of freedom to 3 and lowers the simultaneous smoothness probability of the resulting $G(x)$ and $C(y)$ to $1/((n_1 + 2)!(2n_2 + 1)!)$. Note that the degree of freedom jumps from 1 to 3 and the intermediate two degrees of freedom is not considered in [17, 12]. These works also do not report any computations using additional degrees of freedom.

3 Divisibility and Smoothness Technique

We start by considering the special-q technique due to Davis and Holdridge [8]. The technique was originally proposed in the context of the quadratic sieve (QS) algorithm and was later suggested for use with the number field sieve algorithm by Pollard. We briefly describe the technique as it is used in the QS algorithm. Let N be the number to be factored. In the QS algorithm, one starts with a factor base FB of "small" primes. Let $g(x) = x^2 - N$. The basic idea is to obtain values of x close to $\sqrt{N}$ such that $g(x)$ factors over FB. A modification of

this idea was suggested in [8]. Let q be a "large" prime which is outside FB and suppose it is possible to obtain an r such that $r^2 \equiv N \bmod q$. Then q divides $g(tq+r)$ for all $t \geq 0$. Instead of looking at all integers close to $\sqrt{N}$, this modified method looks at only integers of the form $tq+r$ such that $g(tq+r)$ factors over $FB \cup \{q\}$. This leads to practical savings. The core idea is to ensure that $g(tq+r)$ has q as a factor and then try out choices for t in an attempt to factor $g(tq+r)/q$ over FB.

We suggest a similar technique for speeding up discrete log computation[1]. For one thing, this requires us to work with polynomials. Also, there are some differences from the special-q method. In our method there is no "special" q. This role is played by products of elements of the factor base. The second difference arises from this one. In the special-q method, only integers of the form $g(tq+r)$ are considered and so, some integers which may be smooth over FB may be omitted. For our method, since there is no "special" q, we do not omit any polynomial which may be smooth over the factor base. There are other differences in how the technique is actually used.

From the description of FFS in Section 2, we note that the basic computation required in the relation collection and the descent phases have a similarity. The similarity is observed most clearly when we consider the 2-1 descent. In both the phases, the idea is to factorise two equivalent forms of the expression $A(x)y+B(x)$. For relation collection, all the factors have to be linear while for the descent phase, there is one fixed quadratic factor $\phi(x)$ (whose descent is to be worked out) and rest to be linear. So, if we consider $\phi(x)$ to be 1 in the relation collection step, then the two phases are almost the same.

"Good" $T(x,y)$: Given $\phi(x)$, we say that a $T(x,y)$ is *good for* $\phi(x)$ if the following conditions hold.

1. $\phi(x)$ divides $F(x) = T(x, g_1(x))$ (**Divisibility**);
2. Both $G(x) = F(x)/\phi(x)$ and $C(y) = T(g_2(y), y)$ factor into linear terms (**Smoothness**).

When $\phi(x)$ is clear from the context, we will simply write $T(x,y)$ is good.

3.1 A Special Case

First consider the *special* case when $g_1(x) = x^{n_1}$. The key observation is that certain powers of x are missing in the expression $A(x)g_1(x) + B(x)$. Since $g_1(x) = x^{n_1}$, the lowest degree term in $A(x)g_1(x)$ is x^{n_1} while the highest degree term in $B(x)$ is d_2. If $n_1 > d_2 + 1$, then the coefficients of $x^{d_2+1}, \ldots, x^{n_1-1}$ in $A(x)g_1(x) + B(x)$ are zero. These missing powers of x create a gap and the number of such missing powers is the size of the gap which is equal to $n_1 - d_2 - 1$.

Supposing $\phi(x)$ divides $A(x)g_1(x) + B(x)$, the degree of the quotient is $d_1 + n_1 - d$. Let e be the difference between the degree of this quotient and the gap in $A(x)g_1(x) + B(x)$, i.e., $e = (d_1 + n_1 - d) - (n_1 - d_2 - 1) = d_1 + d_2 - d + 1$. Write

$$A(x)g_1(x) + B(x) \quad = \quad (x - a_1) \cdots (x - a_e)H(x)\phi(x) \tag{12}$$

[1]We had obtained our technique without reference to the special-q technique. Reviewers of the previous version of this paper had pointed out the connection.

where $H(x) = b_0 + \cdots + b_{m-1}x^{m-1} + x^m$ is a monic polynomial of degree $m = n_1 - d_2 - 1$. There are m undetermined coefficients of $H(x)$ and the size of the gap in $A(x)g_1(x) + B(x)$ is also m. This leads to a system of m linear equations in these many undetermined coefficients of $H(x)$.

By *symbolically* solving this system of linear equations, we obtain $b_i = h_i(a_1, \ldots, a_e)$ for some easily computed functions h_i, where $0 \leq i \leq m-1$. Note that solutions for b_i are symbolically expressed in terms of $a_1, \ldots, a_e$ using the function h_i. By independently choosing values for $a_1, \ldots, a_e$, we obtain solutions for $H(x)$ and hence for $A(x)$ and $B(x)$. As a result, after symbolically solving the small linear system once, by varying $a_1, \ldots, a_e$, we obtain the different choices of $H(x)$ leading to the different choices of $A(x)$ and $B(x)$ such that the following three conditions hold.

1. $\phi(x)$ divides $A(x)g_1(x) + B(x)$;
2. the quotient $G(x)$ when $F(x) = A(x)g_1(x) + B(x)$ is divided by $\phi(x)$ is $(x - a_1) \cdots (x - a_e)H(x)\phi(x)$ and is directly obtained without any further computation;
3. $G(x)$ has e linear factors and hence satisfies a partial smoothness condition.

The cost for this method is to symbolically solve once a system of m linear equations in m variables and to obtain the $H(x)$ for a particular choice of $a_1, \ldots, a_e$, one needs to evaluate the functions $h_0, \ldots, h_{m-1}$.

Let us consider the conditions under which the method is applicable. First, we need m to be positive to ensure that there is a non-empty system of linear equations and second, we need e to be positive to ensure that there is at least one degree of freedom. So, a set of necessary and sufficient conditions for the method to be applicable is the following.

$$\left.\begin{array}{rcl} e & = & d_1 + d_2 - d + 1 > 0 \\ m & = & n_1 - d_2 - 1 > 0 \end{array}\right\} \tag{13}$$

Thus, every random choice of $a_1, \ldots, a_e$ ensures divisibility and at the same time yields e linear factors, providing partial smoothness. We provide examples to illustrate the method.

Example 1: Suppose $A(x) = x + b$, $B(x) = ax + c$, $n_1 = 4$ and so $g_1(x) = x^4$. Further, suppose $\phi(x) = 1$, i.e., we are only interested in ensuring partial smoothness. Then we can write $A(x)x^4 + B(x) = (x - a_1)(x - a_2)(x - a_3)(x^2 + b_1x + b_0)$. Equating the coefficients of x^2 and x^3 of the right side to 0, we obtain the following two equations.

$$\begin{array}{rcl} -a_1a_2a_3 - b_0a_1 - b_0a_2 - a_3b_0 + a_1a_2b_1 + a_1a_3b_1 + a_2a_3b_1 & = & 0 \\ a_1a_2 + a_1a_3 + a_2a_3 + b_0 - a_2b_1 - a_3b_1 & = & 0. \end{array}$$

Solving for b_0 and b_1 in terms of a_1, a_2 and a_3 gives the following expressions.

$$\begin{array}{rcl} b_0 & = & h_0(a_1, a_2, a_3) = \dfrac{-a_1a_2(-a_1 - a_2 - a_3)a_3 - (a_1a_2 + a_1a_3 + a_2a_3)^2}{a_1^2 + a_1a_2 + a_2^2 + a_1a_3 + a_2a_3 + a_3^2} \\ b_1 & = & h_1(a_1, a_2, a_3) = \dfrac{-a_1^2a_2 - a_1a_2^2 - a_1^2a_3 - 2a_1a_2a_3 - a_2^2a_3 - a_1a_3^2 - a_2a_3^2}{a_1^2 + a_1a_2 + a_2^2 + a_1a_3 + a_2a_3 + a_3^2}. \end{array}$$

Example 2: With $A(x)$, $B(x)$ and $g_1(x)$ as above, suppose that $\phi(x) = x^2 + x + 1$. Then we can write $A(x)x^4 + B(x) = \phi(x)(x - a_1)(x^2 + b_1x + b_0)$. Again equating the coefficients of x^2

and x^3 of the right side to 0, we obtain the following two equations.

$$\begin{aligned} -a_1 + b_0 - a_1 b_0 + b_1 - a_1 b_1 &= 0 \\ 1 - a_1 + b_0 + b_1 - a_1 b_1 &= 0. \end{aligned}$$

Solving for b_0 and b_1 in terms of a_1 gives the following expressions.

$$\begin{aligned} b_0 &= h_0(a_1) = -\frac{1}{a_1} \\ b_1 &= h_1(a_1) = \frac{1 - a_1 + a_1^2}{(a_1 - 1)a_1}. \end{aligned}$$

3.2 A General Description

The above discussion is for $T(x, y)$ to be of the form $T(x, y) = A(x)y + B(x)$ and $g_1(x)$ to be of the form x^{n_1}. We consider the more general case. Suppose $\phi(x)$ is a monic polynomial of degree d and $T(x, y)$ is any bi-variate polynomial having a total of $\rho + 1$ monomials out of which the coefficient of any one monomial (usually the leading monomial) is 1 and the other ρ monomials are undetermined. We will assume $d < \rho$. Further, suppose that the degree of $F(x) = T(x, g_1(x))$ is ρ_1 and the degree of $C(y) = T(g_2(y), y)$ is ρ_2. Our description of the divisibility technique starts with $\phi(x)$ which is a polynomial in x. A similar description can be provided if one wishes to start with a polynomial in y.

We assume that $F(x)$ is monic (which is easy to ensure since we are interested in discrete log modulo M). The degree of $F(x)$ is ρ_1 and since $F(x)$ is monic, there are ρ_1 coefficients of $F(x)$. The polynomial $F(x)$ is obtained as $T(x, g_1(x))$. Since $T(x, y)$ has ρ monomials whose coefficients are unknown, the coefficients of $F(x)$ can be written as linear functions of these ρ unknowns. The degree of $G(x) = F(x)/\phi(x)$ is $\rho_1 - d$. Then $G(x)$ is also monic and there are $\rho_1 - d$ unknown coefficients of $G(x)$. Let $e = \rho - d$ and write $G(x)$ as $G(x) = (x - a_1) \cdots (x - a_e)H(x)$. So, the degree of $H(x)$ is $(\rho_1 - d) - (\rho - d) = \rho_1 - \rho$. Again, $H(x)$ is monic and so there are a total of $\rho_1 - \rho$ unknown coefficients of $H(x)$. We now consider the identity

$$F(x) = \phi(x)(x - a_1) \cdots (x - a_e)H(x). \tag{14}$$

The left side of (14) is a monic polynomial of degree ρ_1 and the coefficients are linear functions of ρ unknowns. The right side is also a monic polynomial of degree ρ_1. Treating $a_1, \ldots, a_e$ as constants, the unknowns on the right side consist of the $\rho_1 - \rho$ coefficients of $H(x)$. These together with the ρ unknowns on the left side of (14) give a total of ρ_1 unknowns. Equating the coefficients of both sides give a system of ρ_1 linear equations in the ρ_1 unknowns. Symbolically solving this equations gives the coefficients of $H(x)$ and the unknown coefficients of $T(x, y)$ as functions of $a_1, \ldots, a_e$. As a result, the coefficients of $C(y)$ can also be determined as functions of $a_1, \ldots, a_e$.

At this point, it has been ensured that $\phi(x)$ divides $F(x)$ and that the quotient has e linear factors. Since the coefficients of $H(x)$ and $C(y)$ are expressed as functions of $a_1, \ldots, a_e$, it is possible to obtain different $H(x)$ and $C(y)$ by varying $a_1, \ldots, a_e$ over the elements of $\mathbb{F}$. These can then be checked for smoothness. Before proceeding, we summarise the different quantities

below.

$$\left.\begin{array}{lcl} \rho & : & \text{the number of undetermined coefficients in } T(x,y); \\ \rho_1 & : & \text{the degree of } F(x) = T(x, g_1(x)); \\ \rho_2 & : & \text{the degree of } C(y) = T(g_2(y), y); \\ d & : & \text{the degree of } \phi(x); \\ \rho_1 - d & : & \text{the degree of } G(x) = F(x)/\phi(x); \\ e & : & \text{equals } \rho - d \text{ which is the degree of freedom.} \end{array}\right\} \tag{15}$$

Working with a smaller system of linear equations: In the above description, we have a system of ρ_1 linear equations in ρ_1 variables. Depending upon the structure of $T(x,y)$, it may be possible to reduce the size of the linear system. This can be useful in practice, since the linear system is solved symbolically. We illustrate the idea in the following example.

Let $T(x,y) = A'(y)x + B'(y)$ and consider $\phi(y)$ to be the polynomial for which we need to ensure that $\phi(y)$ divides $T(g_2(y), y)$. Let the degrees of $A'(y)$ and $B'(y)$ be d_1 and d_2 respectively. Further, let $d_1 = d_2 = 2, d = 3$ and $g_2(y) = x^7 + c_6y^6 + c_5y^5 + c_4y^4 + c_3y^3 + c_2y^2 + c_1y + c_0$ be given. Here, $c_0, \ldots, c_6$ are known constants. Suppose $\phi(y) = l_0 + l_1y + l_2y^2 + y^3$ (so that $d = 3$), where l_0, l_1 and l_2 are known. Write

$$A'(y)g_2(y) + B'(y) = (y^2 + ay + b)g_2(y) + (\alpha y^2 + \beta y + \gamma) \tag{16}$$

as

$$(y + a_1)(y + a_2)\underbrace{(b_0 + b_1y + b_2y^2 + b_3y^3 + y^4)}_{H'(y)}\underbrace{(l_0 + l_1y + l_2y^2 + y^3)}_{\phi(y)} \tag{17}$$

As per the general description, we need to compare the coefficients of y^0 to y^8 of (16) and (17). Symbolically solving the resulting system of equations will provide all the unknowns, i.e., $b_0, \ldots, b_3, a, b, \alpha, \beta, \gamma$ as functions of a_1 and a_2. Due to the particular form of $T(x,y)$, we can work with a smaller system of equations. Note that α, β and γ are involved only in determining the coefficients of y^2, y and the constant term. If we leave out these 3 powers of y, then we get the following 6 linear equations in b_0, b_1, b_2, b_3, a, b by comparing the coefficients of $y^3, \ldots, y^8$ of (16) and (17). Note that in these equations a_1 and a_2 are to be regarded as constants.

$$\begin{aligned}
0 &= l_2a_1a_2b_1 + l_1a_1a_2b_2 + l_0a_1a_2b_3 + l_2a_1b_0 + l_2a_2b_0 + a_1a_2b_0 + l_1a_1b_1 + l_1a_2b_1 \\
&\quad + l_0a_1b_2 + l_0a_2b_2 - ac_2 - bc_3 + l_1b_0 + l_0b_1 - c_1; \\
0 &= l_2a_1a_2b_2 + l_1a_1a_2b_3 + l_0a_1a_2 + l_2a_1b_1 + l_2a_2b_1 + a_1a_2b_1 + l_1a_1b_2 + l_1a_2b_2 \\
&\quad + l_0a_1b_3 + l_0a_2b_3 - ac_3 - bc_4 + l_2b_0 + a_1b_0 + a_2b_0 + l_1b_1 + l_0b_2 - c_2; \\
0 &= l_2a_1a_2b_3 + l_1a_1a_2 + l_2a_1b_2 + l_2a_2b_2 + a_1a_2b_2 + l_1a_1b_3 + l_1a_2b_3 - ac_4 - bc_5 \\
&\quad + l_0a_1 + l_0a_2 + l_2b_1 + a_1b_1 + a_2b_1 + l_1b_2 + l_0b_3 - c_3 + b_0; \\
0 &= l_2a_1a_2 + l_2a_1b_3 + l_2a_2b_3 + a_1a_2b_3 - ac_5 - bc_6 + l_1a_1 + l_1a_2 + l_2b_2 + a_1b_2 \\
&\quad + a_2b_2 + l_1b_3 - c_4 + l_0 + b_1; \\
0 &= -ac_6 + l_2a_1 + l_2a_2 + a_1a_2 + l_2b_3 + a_1b_3 + a_2b_3 - b - c_5 + l_1 + b_2; \\
0 &= -a - c_6 + l_2 + a_1 + a_2 + b_3.
\end{aligned}$$

This system can be symbolically solved to obtain solutions for b_0, b_1, b_2, b_3, a, b in terms of a_1 and a_2. Once b_0, b_1, b_2, b_3, a, b there is no more freedom left and α, β and γ are determined.

So, given $g_2(y)$ and $\phi(y)$, by varying a_1 and a_2, it is possible to generate $A'(y)$ and $B'(y)$ such that $A'(y)g_2(y) + B'(y) = (y + a_1)(y + a_2)H'(y)\phi(y)$. So, we have ensured that $\phi(y)$ divides $A'(y)g_2(y) + B'(y)$ and that the quotient has two linear factors, namely $(y + a_1)$ and $(y + a_2)$.

Since $H'(y)$ is of degree 4, we can expect a factorisation into linear terms in about 4! trials. Without the use of the divisibility technique (as in the kernel method), one would not have the linear terms $(y + a_1)$ and $(y + a_2)$ and it would be required to contend with ensuring the smoothness of a polynomial of degree 6 which would require about 6! trials. We note that the divisibility technique does not improve the degree of freedom. It only ensures a few linear factors leading to a lesser number of trials to obtain smoothness.

Completeness: Suppose there is a choice of $A'(y)$ and $B'(y)$ such that $A'(y)g_2(y) + B'(y)$ is divisible by $\phi(y)$ and the quotient $Q(y) = (A'(y)g_2(y) + B'(y))/\phi(y)$ factors into linear terms. The divisibility technique will not miss any such $Q(y)$. To see this, note that since $Q(y)$ is smooth, we can write it as $Q(y) = (y + \alpha_1) \cdots (y + \alpha_6)$. In the above set up, if we choose $a_1 = \alpha_1$ and $a_2 = \alpha_2$, then the resulting $H'(y)$ will turn out to be $(y + \alpha_3) \cdots (y + \alpha_6)$.

Repetition: Suppose in the above example, for some choice of values β_1 and β_2 for a_1 and a_2 respectively, we have obtained $H'(y)$ to be smooth, i.e., $H'(y) = (y + \beta_3) \cdots (y + \beta_6)$ and so $A'(y)g_2(y) + B'(y) = (y + \beta_1)(y + \beta_2)H'(y)\phi(y) = (y + \beta_1) \cdots (y + \beta_6)\phi(y)$. Then, for any $i \neq j$, choosing β_i and β_j to be values of a_1 and a_2 leads to another $H'(y)$ which is equal to $((y + \beta_1) \cdots (y + \beta_6))/((y + \beta_i)(y + \beta_j))$ and so once more we have $A'(y)g_2(y) + B'(y) = (y + \beta_1) \cdots (y + \beta_6)\phi(y)$. This means that we do not get any new polynomial for this choice of a_1 and a_2. As a result, if we try out all the possible p^2 choices of a_1 and a_2, then each possible *smooth* polynomial $(A'(y)g_2(y)+B'(y))/\phi(y)$ will occur exactly $\binom{6}{2}$ times. More generally, if we are assured of χ linear factors and the degree of H' is υ, then each possible smooth polynomial $(A'(y)g_2(y) + B'(y))/\phi(y)$ will occur exactly $\binom{\chi+\upsilon}{\chi}$ times.

To avoid considering the same polynomial twice, one may use a list to store the polynomials already obtained. But searching and inserting into this list will be a bottleneck for the computation. So, we did not implement this strategy. Instead we implemented the simpler strategy of ensuring that the choices of a_1 and a_2 are made such that $a_1 \leq a_2$. (More generally, the χ roots are chosen in an ordered fashion.) This mitigates the repetition issue to some extent.

Note that repetition becomes an issue when the number of trials are significantly large. If the number of trials is not too large, then repetitions do not occur. This, however, is not always possible to control. The number of trials depends on the probability of obtaining a good $T(x, y)$.

3.3 Probability of Obtaining Good $T(x, y)$

Given $\phi(x)$ of degree d, we wish to obtain an estimate of the probability of obtaining a $T(x, y)$ which is good for $\phi(x)$. This probability does not depend on the divisibility technique, i.e., the divisibility technique neither increases nor lowers this probability. Its effect is to reduce the number of trials of obtaining a good $T(x, y)$ *assuming* that the probability is sufficiently high.

Suppose as before that $T(x, y)$ has ρ undetermined coefficients and so the degree of freedom of $T(x, y)$ is ρ. Also, let the degree of $\phi(x)$ be d. In general (i.e., without the divisibility technique), ensuring that $\phi(x)$ divides $T(x, g_1(x))$ uses up d degrees of freedom and we are left with $\rho - d$ degrees of freedom. So, a maximum of $p^{\rho-d}$ trials can be carried out to ensure

that both $G(x) = T(x, g_1(x))/\phi(x) = F(x)/\phi(x)$ and $C(y) = T(g_2(y), y)$ factor into linear terms. Suppose as before that the degrees of $F(x)$ and $C(y)$ are ρ_1 and ρ_2 respectively. Then heuristically, the probability of getting both these polynomials to be smooth in a single trial is $1/((\rho_1 - d)!\rho_2!)$.

Let $\mathbf{E}$ be the event of obtaining in $p^{\rho-d}$ trials a $T(x, y)$ such that $\phi(x)|T(x, y)$ and both $F(x)$ and $C(y)$ are smooth. Again, heuristically,

$$\Pr[\mathbf{E}] \;=\; 1 - \left(1 - \frac{1}{(\rho_1 - d)!\rho_2!}\right)^{p^{\rho-d}}. \tag{18}$$

Note that the above is a heuristic expression, since it may turn out that $\mathbf{E}$ is an impossible event and so its probability is 0. Nevertheless, (18) turns out to be a good approximation in practice. In $p^{\rho-d}$ trials, the expected number of good $T(x, y)$ is

$$\frac{p^{\rho-d}}{(((\rho_1 - d))!(\rho_2)!)}. \tag{19}$$

The value of this expression turns out to be good indicator of the probability of $\mathbf{E}$. If the value of (19) is at least 1, then $\Pr[E]$ is high and it is likely that we will obtain a desirable $T(x, y)$. On the other hand, if the value of (19) is very low, then $\Pr[\mathbf{E}]$ is close to zero and it is unlikely that a suitable $T(x, y)$ will be obtained. This issue is discussed later in connection with the walk technique.

Suppose that the probability of $\mathbf{E}$ is high and assume that we are assured of obtaining a suitable $T(x, y)$. It is under this condition that the divisibility technique helps in reducing the number of trials required to obtain $T(x, y)$. The divisibility technique ensures $e = \rho - d$ linear factors for $F(x)$. Then the heuristic probability of obtaining a suitable $T(x, y)$ in a single trial becomes $1/((\rho_1 - \rho)!\rho_2!)$. As a result one can expect to obtain a suitable $T(x, y)$ in about

$$(\rho_1 - \rho)!\rho_2! \tag{20}$$

trials. Without the divisibility technique, the number of trials will be $(\rho_1 - d)!\rho_2!$. Since $\rho > d$, use of the divisibility technique will lead to an improvement.

4 Application to Relation Collection and 2-1 Descent

Below, we separately discuss the application of the divisibility and smoothness technique to the relation collection phase and the 2-1 descent step.

Relation collection: Recall from (8) that the relation collection phase succeeds if $p^3/((n_1 + 1)!(n_2 + 1)!) > 2p$. Using the divisibility technique does not improve this condition. Assuming that this condition holds, using the divisibility technique results in faster collection of the relations.

For the relation collection phase, $T(x, y) = A(x)y + B(x)$ where both $A(x)$ and $B(x)$ are linear polynomials with $A(x)$ being monic. (The degrees d_1 and d_2 of $A(x)$ and $B(x)$ are both 1.) So, there are 3 undetermined coefficients in $T(x, y)$ leading to 3 degrees of freedom, i.e., $\rho = 3$. Here $\phi(x) = 1$ and so $d = 0$. The degree of $F(x)$ is $\rho_1 = n_1 + 1$ and that of $C(y)$ is $\rho_2 = n_2 + 1$. A single random choice of $F(x)$ will be smooth with probability $1/(n_1 + 1)!$ and such an $F(x)$ can be obtained in about $(n_1 + 1)!$ trials.

On the other hand, the divisibility technique can be applied with $\phi(x) = 1$ and then $e = d_1 + d_2 - d + 1 = 3$. So, $F(x) = \phi(x)(x - a_1)(x - a_2)(x - a_3)H(x)$ and the degree of $H(x)$ is $\rho_1 - \rho = n_1 - 2$. The probability that $H(x)$ is smooth is $1/(n_1 - 2)!$ and such an $H(x)$ can be obtained in about $(n_1 - 2)!$ trials. So, using the divisibility technique reduces the number of trials by a factor of $n_1(n_1^2 - 1)$.

Let us now consider this in combination with the pinpointing technique. Assume that $g_1(x) = x^{n_1}$ and pinpointing is applied from the x-side. In about $(n_1 - 2)!$ trials a single smooth polynomial on the x-side is obtained. From this, the pinpointing technique generates $p - 1$ x-side smooth polynomials. Out of these, about $(p - 1)/(n_2 + 1)!$ polynomials are also smooth on the y-side. So, the amortised number of trials for obtaining a single relation with both sides smooth is about

$$(n_2 + 1)! + \frac{(n_1 - 2)!(n_2 + 1)!}{p - 1}. \tag{21}$$

This expression should be compared to (10) which gives the amortised number of trials for obtaining a single relation when the divisibility technique is not used. In a similar manner, if $g_2(y) = y^{n_2}$ and pinpointing is applied from the y-side, the amortised number of trials for obtaining a relation is about

$$(n_1 + 1)! + \frac{(n_2 - 2)!(n_1 + 1)!}{p - 1}. \tag{22}$$

Depending on the values of n_1 and n_2, one of the above will be smaller and it will be preferrable to apply pinpointing from the side for which the number of trials is expected to be lesser.

2-1 descent: Suppose we wish to descend from a quadratic polynomial $\phi(x)$ so that $d = 2$. Let $T(x, y) = A(y)x + B(y)$ and the degrees of $A(y)$ and $B(y)$ are d_1 and d_2 respectively with $A(y)$ being monic. The polynomials $A(y)$ and $B(y)$ have $d_1 + 1$ and $d_2 + 1$ terms respectively and so the total number of monomials in $T(x, y)$ is $d_1 + d_2 + 2$. Since $A(y)$ is constrained to be monic the number of undetermined coefficients in $T(x, y)$ is $\rho = d_1 + d_2 + 1$.

The degree of $F(x) = T(x, g_1(x))$ is $\rho_1 = \max(n_1 d_1 + 1, n_1 d_2)$ and the degree of $C(y) = F(g_2(y), y)$ is $\rho_2 = \max(n_2 + d_1, d_2)$. Typically, the choice of d_1 and d_2 will be such that $\rho_2 = n_2 + d_1$. The degree of freedom $e = \rho - d = d_1 + d_2 - 1$. For different values of d_1 and d_2, Table (2) provides the values of ρ, the expected number of suitable $T(x, y)$ predicted by (19) and the number of trials required using the divisibility technique to obtain one such $T(x, y)$.

A table similar to (2) can be derived if one wishes to descend from a quadratic polynomial in y. If n_1 and n_2 are not equal or if the degree of freedom is more than 1, then the number of trials required for descending from the x-side and the y-side are not equal. Hence, it is advisable to make the 2-1 descend either from x-side or from the y-side but, not both. If the descent from the y-side is faster, then from the quadratic x-polynomials one should move to quadratic y-polynomials and then descend from the y-side. The converse strategy should be applied if the descent from the x-side is faster.

As mentioned earlier, degrees of freedom one and three have been considered in [17, 12]. The degree of freedom two (corresponding to $d_1 = 1$ and $d_2 = 2$) does not seem to have been considered earlier although it turns out to be important in practice.

If the divisibility technique is not used then the number of trials for one 2-1 descent would be $(n_1 - 1)!(n_2 + 1)!$, $(2n_1 - 2)!(n_2 + 1)!$ and $(2n_1 - 1)!(n_2 + 2)!$ corresponding to one, two and

Table 2: Estimates of the expected number of good $T(x, y)$ given by (19) and the number of trials for a single 2-1 descent from the x-side given by (20).

d_1	d_2	ρ	e	ρ_1	ρ_2	# good $T(x,y)$ from (19)	# trials from (20)
1	1	3	1	n_1+1	n_2+1	$p/((n_1-1)!(n_2+1)!)$	$(n_1-2)!(n_2+1)!$
1	2	4	2	$2n_1$	n_2+1	$p^2/((2n_1-2)!(n_2+1)!)$	$(2n_1-4)!(n_2+1)!$
2	2	5	3	$2n_1+1$	n_2+2	$p^3/((2n_1-1)!(n_2+2)!)$	$(2n_1-4)!(n_2+2)!$

three degrees of freedom respectively. So, using the divisibility technique lowers the number of trials by a factor of $n_1 - 1$ when the degree of freedom is 1; by a factor of $(2n_1 - 2)(2n_1 - 3)$ when the degree of freedom is 2; and by a factor of $(2n_1-1)(2n_1-2)(2n_1-3)$ when the degree of freedom is 3. This shows that the gain in using the divisibility technique increases as the degrees of freedom increase.

5 The Walk Technique for a 2-1 Descent

Suppose that we wish to descend from a quadratic polynomial $\phi(x)$ and there are e degrees of freedom. With e degrees of freedom, there can be p^e trials. The probability of obtaining a good $T(x, y)$ in these number of trials is given by (19) and the expected number of good $T(x, y)$ is given by (20).

Let us assume that in none of these trials it is possible to obtain a $T(x, y)$ which is good for $\phi(x)$. Typically, the divisibility of $F(x) = T(x, g_1(x))$ by $\phi(x)$ can be ensured, but, the simultaneous smoothness for $G(x) = T(x, g_1(x))/\phi(x)$ and $C(y) = T(g_2(y), y)$ will not be achieved. At this point, the descent can get stuck. The way around is to try to move to another quadratic polynomial and try to descend from that. There are two ways to do this.

1. Given $\phi(x)$, try to obtain $T(x, y)$ such that $G(x)$ has one quadratic factor $\phi_1(x)$ and the other factors are linear and $C(y)$ has only linear factors. Then $\phi(x) = C(y)/G(x)$. If a descent from $\phi_1(x)$ is possible, then this descent can be combined with the factorisation of $G(x)/\phi_1(x)$ and $C(y)$ to get a descent for $\phi(x)$.

2. Alternatively, given $\phi(x)$, one can try to obtain $T(x, y)$ such that $G(x)$ is smooth and $C(y)$ has one quadratic factor $\psi(y)$ and the others are linear factors. If a descent from $\psi(y)$ is possible, then so is a descent from $\phi(x)$.

In the descent phase we will get quadratic polynomials in both x and y. So there are 4 kinds of step one may consider during the 2-1 descent viz. x-x, x-y, y-y and y-x. Let ζ_{ab} represent the probability of an a-b step. This probability depends on the available degree of freedom for the 2-1 descent. We provide heuristics estimates of ζ_{ab} when the degree of freedom is 1. Similar estimates can be obtained for higher degrees of freedom.

Consider ζ_{xy} which is the probability of an x-y step. The x-side is a polynomial of degree n_1+1 of which $\phi(x)$ is a quadratic factor and it is required for the other factor of degree n_1-1 to factorise into linear terms. The probability of this happening is $1/(n_1-1)!$. The y-side is a

polynomial of degree $n_2 + 1$. For a complete 2-1 descent, it is required for this polynomial to factorise completely which occurs with probability $1/(n_2 + 1)!$. On the other hand, for an x-y step, the requirement on the y-side is a quadratic factor and the other factor of degree $n_2 - 1$ to completely factorise into linear terms. This probability is clearly at most $1/(n_2 - 1)!$ and is greater than the probability $1/n_2!$ that a random polynomial of degree n_2 factors completely into linear terms. Using the heuristic assumption of independence of factorisation on the x and the y sides, the probability of an x-y step lies between $1/((n_1 - 1)!(n_2)!)$ and $1/((n_1 - 1)!(n_2 - 1)!)$. The probabilities of the other a-b steps can be similarly worked out and these are given below.

$$\left.\begin{array}{rcccl} \frac{1}{(n_1-2)!(n_2+1)!} & < & \zeta_{xx} & < & \frac{1}{(n_1-3)!(n_2+1)!} \\ \frac{1}{(n_1-1)!(n_2)!} & < & \zeta_{xy} & < & \frac{1}{(n_1-1)!(n_2-1)!} \\ \frac{1}{(n_1+1)!(n_2-2)!} & < & \zeta_{yy} & < & \frac{1}{(n_1+1)!(n_2-3)!} \\ \frac{1}{(n_1)!(n_2-1)!} & < & \zeta_{yx} & < & \frac{1}{(n_1-1)!(n_2-1)!} \end{array}\right\} \tag{23}$$

Experimentally, we have found that the probabilities ζ_{ab} are close to their upper bound. As a consequence, $\zeta_{xy} \approx \zeta_{yx}$ and this value is greater than ζ_{xx} or ζ_{yy}. So, for one degree of freedom, it is advisable to have an alternating walk, i.e., if the descent fails for $\phi_1(x)$, then take an x-y step to obtain $\psi_1(y)$; if the descent fails for $\psi_1(y)$, then take a y-x step to $\phi_2(x)$; and so on. If the degree of freedom is more than 1, then the values of ζ_{ab} will change and the walk will have to appropriately designed.

Branching: As described above, the walk technique converts a quadratic polynomial to another quadratic polynomial. This, however, may not always be possible. In such a situation, one has to try and move from a quadratic polynomial to a pair of quadratic polynomials. This creates a branching and increases the number of quadratic polynomials for which descent is required. If the number of such branchings is too much, then there will be an exponential increase in the number of polynomials for which descent is required. In such a scenario, the method will not succeed. On the other hand, a few branchings do not affect convergence.

For the actual implementation, there can be several ways of realising branchings. Suppose, we are trying to descend from an x-polynomial $\phi(x)$ and it turns out that for this polynomial descent is not possible and neither are x-x or x-y steps of the walk possible. Then, from $\phi(x)$ one needs to move to two quadratic polynomials. There are three options for this:

- from $\phi(x)$ move to $\phi_1(x), \phi_2(x)$;
- from $\phi(x)$ move to $\phi_1(x), \psi_1(y)$;
- from $\phi(x)$ move to $\psi_1(y), \psi_2(y)$.

These options have different probabilities which also depend on the degree of freedom. For the actual implementation, one should choose the option with the highest probability first.

Cycling: There is another way in which the walk may fail. Suppose the walk starts from $\phi(x)$ and alternates between x and y-polynomials. If one of the x-polynomials turn out to be $\phi(x)$, then the walk may cycle. Appropriately using randomisation can usually avoid this possibility.

Probability of making a successful step: The probability that an a-b step is successful in a single trial is ζ_{ab} and so the probability that such a step is not possible in a single trial is $(1-\zeta)$. Due to the single degree of freedom, it is possible to make p trials. The probability that all the trials will fail is $(1-\zeta)^p$ (under the heuristic assumption that the trials are independent) and so the probability that an a-b step is successful is $1-(1-\zeta_{ab})^p$. If this probability is close to one, i.e., if $(1-\zeta_{ab})^p \approx 0$, then the a-b step will go through.

Length of the walk: Let η_x (resp. η_y) be the probability that given a fixed quadratic irreducible x-polynomial (resp. y-polynomial) it is possible to descend into linear factors using one degree of freedom. So, $\eta_x = 1/((n_1-1)!(n_2+1)!)$ (resp. $\eta_y = 1/((n_1+1)!(n_2-1)!)$) and with probability $1-\eta_x$ (resp. $1-\eta_y$) the descent fails at the x-polynomial (resp. y-polynomial). Due to one degree of freedom, we can make about p trials and the probability that the descent fails in all the trials is $(1-\eta_x)^p$ (resp. $(1-\eta_y)^p$). This is the probability that the walk at an x-polynomial (resp. y-polynomial) has to continue further. The walk alternates between x and y-polynomials. Suppose there are t_1 x-polynomials and t_2 y-polynomials in the walk. We make the simplifiying assumption that $t_1 = t_2 = t$.

The probability that descent is not successful at a particular x-polynomial is $(1-\eta_x)^p$. Heuristically assuming independence, the probability that descent is not successful at all the t x-polynomials is $(1-\eta_x)^{pt}$. Similarly, the probability that descent is not successful at all the t y-polynomials is $(1-\eta_y)^{pt}$. So, the total probability that the descent is not successful is $((1-\eta_x)(1-\eta_y))^{pt}$. From this we get that for the descent to be successful with high probability, we must have $((1-\eta_x)(1-\eta_y))^{pt} \approx 0$. In this expression, the quantities η_x, η_y and p are fixed. So, the value of t must be chosen large enough so that $((1-\eta_x)(1-\eta_y))^{pt}$ becomes close to zero. Then the number of steps in the walk is about $2t$.

Examples: First, let us consider the 25-bit prime and extension degree 57 example used in [15]. In this case, $p = 33341353$, $n_1 = 8$ and $n_2 = 7$. Using (23) we have $0.0001 \leq (1-\zeta_{xy})^p \leq 0.27$. The lower bound corresponds to the upper bound in (23) and vice versa for the upper bound. As mentioned earlier, experiments show that the actual value of ζ_{xy} is close to the upper bound in (23) and so $(1-\zeta_{xy})$ should be close to $0.0001 \approx 0$. This suggests that the walk technique works well in this case. The work [15] does not mention the walk length. To get an idea, we have run some experiments with the 2-1 descent for this example and have observed the the average walk length is about 5. This indicates that the application of the walk technique for this example is not very difficult.

We now report on the walk technique for two examples that we have considered. For $p = 64373$, $n_1 = 6$ and $n_2 = 6$, using (23) we have $0.01 \leq (1-\zeta_{xy})^p \leq 0.47$. As mentioned above, the probability should be close to $0.01 \approx 0$. This suggests that the walk technique should work in this case and as we report later, it indeed does. There were no branchings and the average walk length turned out to be 17. Note that this walk length is more than the walk length for the previous example which matches the difference in the lower bounds for ζ_{xy} in the two cases.

For $p = 297079$, $n_1 = 8$ and $n_2 = 5$, using (23), we have $0.09 \leq (1-\zeta_{xy})^p \leq 0.61$. The lower bound is not sufficiently close to 0 for the walk technique to work. This is what we observed in our experiments. For this case we are not always able to move from one quadratic polynomial to another. The number of times when we have to move from one quadratic polynomial to two quadratic polynomials turns out to be significantly high. As a result of such branching, the

walk technique does not converge. We provide some results for the experiment that we have run. For the element given by (27) in Section 7.2, we have carried out the descent up to degree 2 polynomials. This resulted in 59 quadratic polynomials. Out of these for 28 polynomials, the walk terminated and it was possible to make the descent. In these cases, the average walk length was 12. Some very few branchings also take place, but, these terminate very quickly.

The walks for the other 31 polynomials did not terminate. For each polynomial, we ran the walk for about 600 steps and observed that on an average there were about 80 branchings. These branchings create a tree like structure resulting in more and more quadratic polynomials from which descents are to be made. As a result, the walks for these polynomials do not terminate. Due to this reason, we decided to switch to a different representation of the field where we can make use of 2 degrees of freedom to perform the 2-1 descent.

6 Concrete Analysis

We perform the following analysis. Fix the size of the underlying prime p to be δ bits. Then we consider different possible extension degrees and the associated costs for the different phases of the FFS algorithm.

The extension degree n is taken to be $n_1 n_2$ or $n_1 n_2 + 1$. Any relation obtained in the relation collection phase involves $n_1 + n_2$ elements of the factor base. The number of relations required is slightly more than $2p$. Consider the matrix which is provided as input to the linear algebra phase. The number of rows R in the matrix is about $2p$ and each row has $n_1 + n_2$ non-zero entries. So, the total weight W of the matrix is around $2p(n_1 + n_2)$. As mentioned earlier, the cost of the Lanczos or the block Wiedemann algorithm is proportional to $R \times W$ which in the present case is about $4(n_1 + n_2)p^2$. This determines the time taken by the linear algebra step.

For the relation collection phase, we need to consider two things. The first is whether the relation collection step is indeed feasible which is determined by the inequality in (8). The second consideration is the number of trials required to obtain a single relation which is determined by either (21) or (22) depending on whether pinpointing is applied from the x or the y-side. The total number of trials for relation collection is obtained by multiplying the expression in (21) or (22) by $2p$.

In the descent phase, the most time consuming step is the 2-1 descent. Again, there are two considerations for this – feasiblity and the number of trials. This depends on the values of d_1 and d_2 and the degree of freedom e as indicated in Table 2. The number of trials is for a single 2-1 descent. Typically, the descent to degree 2 polynomials will result in several polynomials. In the cases for which we have done the complete discrete log computations, the number of such polynomials is less than 100. For larger fields, there may be few hundred quadratic polynomials. To get the total number of trials for the complete 2-1 descent it is required to multiply the number of trials for a single descent with the number of polynomials for which descent is required.

Concrete estimates of the above quantities for different values of δ are provided in Tables 3 to 7. In these tables, the value of $Q = p^n$ and $n(Q)$ is given by (1) with $\alpha = (1/3)^{2/3}$ for $D = 1$ (i.e., the factor base consists only of linear polynomials). In later sections, we report on the actual computation of discrete log for 16-bit and 18-bit primes with extension degrees 37 and 40 respectively. Here we note the following points.

1. In all cases (with the exception of $\delta = 25$ and $n = 57$), we find $n > n(Q)$. This shows that it is feasible to compute discrete log on $\mathbb{F}_Q$ for $n > n(Q)$.

2. For $\delta = 16$ and $n = 35$, the expected number of good $T(x, y)$ is 0.13 (with one degree of freedom). Since this value is less than 1, the walk technique is required and it works in this case. Similarly, for $\delta = 25$ and $n = 57$, the expected number of good $T(x, y)$ is 0.17. Computation of the discrete log for this field carried out in [15] would have required the walk technique. On the other hand, for $\delta = 18$ and $n = 40$, using a single degree of freedom, the expected number of good $T(x, y)$ is 0.07. We had attempted the walk technique for this example. It turned out that in many cases there were too many branchings and it was not possible to complete the descent. So, we used two degrees of freedom for this example.

3. As n increases, the expected number of good $T(x, y)$ for the 2-1 descent step decreases. If this number goes below 1, then the walk technique is required. Further, if this number becomes very low, then the walk technique leads to a lot of branchings and does not converge. At this point, it is required to move to a higher degree of freedom.

4. As the number of degrees of freedom increases, the number of trials required for a 2-1 descent also increases. So, the shift to a higher degree of freedom is to be made only after ascertaining that descent is not possible for a lower degree of freedom even with the walk technique.

5. As n increases beyond $n(Q)$, the time for 2-1 descent becomes more than the time for the relation collection phase and also grows at a faster rate. To a certain extent this is to be expected, since the bound $n(Q)$ in [17] was obtained assuming that the individual logarithm phase takes at most as much time as the relation collection phase.

6. The number of trials for obtaining one relation is based on the pinpointing technique. We have not considered the faster two-sided (or advanced) pinpointing since it is not generally applicable.

7. The estimates for the number of trials for both relation collection and 2-1 descent is based on the divisibility technique of Section 3. If this technique is not used, then the number of trials required will increase. The extent of the increase will depend on the degree of freedom and the values of n_1 and n_2. For the values of n_1 and n_2 considered in the tables, the increase will be from about 5 times to about 50 times or more.

8. For certain cases ($(\delta, n) = (16, 40), (18, 45), (20, 64)$), the expected number of good $T(x, y)$ is less than 1 even though the degree of freedom is greater than 1. This suggests that for the corresponding computations, the walk technique will be required. Our probability estimates and the alternating x-y walk is for a single degree of freedom. For higher degrees of freedom, appropriate estimates can be developed and a suitable walk designed.

Based on the tables, we can conclude that access to a sufficiently powerful super-computer will enable the computation of discrete log for the following parameter choices:

$$(\delta, n) = (16, 49), (18, 56), (20, 64), (25, 64).$$

The linear algebra step for the 25-bit prime may be difficult to carry out. Suitably choosing p can reduce the size of the factor base by a factor of n. Though this loses generality, the technique has been used in [15] for extension degrees 47 and 57. The same technique may be used for extension degree 64 which will make the linear algebra step feasible. The 2-1 descent, however, will not be affected. This computation being highly parallelisable can be carried out on a super-computer.

The concrete analysis for 32-bit primes shows that it is unlikely that these computations will be carried out anytime soon. For this size primes, the most striking point is perhaps the fact that for $n = 100$ (corresponding to $\log Q = 3200$), the discrete log computation should be possible in about 2^{80} time.

7 Actual Discrete Log Computations

We report the actual computations of discrete logs over some fields. For these computations, we have used Magma [7] and SAGE [18] computer algebra systems.

7.1 A Record on a Field of Size p^{37}, $p = 64373$

We have carried out the discrete log computation over a field $Q = p^n$ for a 16-bit prime $p = 64373$ and extension degree $n = 37 = 6 \times 6 + 1$ with $n_1 = 6$ and $n_2 = 6$. This uses the $n_1 n_1 + 1$ variant of the FFS algorithm as mentioned in Section 2.

The defining polynomial $f(x)$ of this field is generated by $g_1(x) = x^{-6}$ and $g_2(x) = x^6 + 14833x^5 + 50952x^4 + 62125x^3 + 6269x^2 + 35223x + 53172$ and $f(x)$ is

$$f(x) = x^{37} + 11201x^{36} + 29150x^{30} + 58104x^{24} + 2248x^{18} + 13421x^{12} + 49540x^6 + 64372.$$

The polynomial $f(x)$ generates the field $\mathbb{F}_{p^n}$ (which is a 592-bit field). This polynomial is not primitive. So x will not generate the multiplicative group of the field. We found that $x + 4$ is primitive, and so, we took this to be the generator of the multiplicative group and hence the base of the discrete log problem.

We have used Magma Computer Algebra Software for entire computation. The hardware platform consists of Intel Xeon(R) CPU X5675 @ 3.07GHz and Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz. We have used the divisibility technique along with pinpointing in the relation collection phase. In around 24 CPU (@ 3.07GHz) hours we have generated the matrix M for the linear algebra step. This phase provides us a sparse matrix with 138837 rows and 128746 columns. We have

$$\begin{aligned} \frac{p^n - 1}{p - 1} &= 149 \times 71 \times 295683503 \times 1389373163 \times 423185284367 \\ &\quad \times \underbrace{6802546264932254537532269582565076 1}_{p_2} \times p_1 \end{aligned} \tag{24}$$

$$\begin{aligned} p_1 &= 736728358367366362807088786077000329142734778435579 81 \\ &\quad 284948280092019110924138717360099475451194738739184563 \end{aligned}$$

where p_1 is a 356-bit prime number. So clearly Pollard's Rho and Pohlig-Hellman algorithms will not work here. We have done the linear algebra using Lanczos Algorithm on the Magma

Table 3: A concrete analysis for a 16-bit prime p.

parameters			2-1 descent (from Tab 2)		relation collection		
$(n, \lg Q, n(Q))$	(n_1, n_2)	(d_1, d_2, e)	good $T(x,y)$	no of trials	lhs of (8)	pinpt side	no of trials (21) or (22)
(35, 560, 34)	(7, 5)	(1, 1, 1)	0.13	$2^{16.40}$	$2^{23.21}$	x	$2^{9.5}$
(37, 592, 35)	(6, 6)	(1, 1, 1)	0.11	$2^{16.88}$	$2^{23.40}$	x or y	$2^{12.3}$
(40, 640, 37)	(5, 8)	(1, 2, 2)	0.29	$2^{27.96}$	$2^{20.04}$	y	$2^{9.51}$
	(5, 8)	(2, 2, 3)	213.75	$2^{31.28}$	$2^{20.04}$	y	$2^{9.51}$
(42, 672, 38)	(6, 7)	(2, 2, 3)	19.43	$2^{33.77}$	$2^{20.40}$	y	$2^{12.3}$
(45, 720, 39)	(5, 9)	(2, 2, 3)	19.43	$2^{34.74}$	$2^{16.72}$	y	$2^{9.6}$
(48, 768, 41)	(6, 8)	(2, 2, 3)	1.94	$2^{37.09}$	$2^{17.23}$	y	$2^{12.31}$
(49, 784, 41)	(7, 7)	(2, 2, 3)	0.12	$2^{40.26}$	$2^{17.40}$	x or y	$2^{15.30}$

Table 4: A concrete analysis for an 18-bit prime p.

parameters			2-1 descent (from Tab 2)		relation collection		
$(n, \lg Q, n(Q))$	(n_1, n_2)	(d_1, d_2, e)	good $T(x,y)$	no of trials	lhs of (8)	pinpt side	no of trials (21) or (22)
(40, 720, 39)	(8, 5)	(1, 1, 1)	0.07	$2^{18.98}$	$2^{26.04}$	x	$2^{9.5}$
	(5, 8)	(1, 2, 2)	4.70	$2^{27.96}$	$2^{26.04}$	y	$2^{9.5}$
(42, 756, 40)	(6, 7)	(1, 2, 2)	0.50	$2^{30.60}$	$2^{26.40}$	y	$2^{12.3}$
	(6, 7)	(2, 2, 3)	1243.66	$2^{33.80}$	$2^{26.40}$	y	$2^{12.3}$
(45, 810, 42)	(5, 9)	(1, 2, 2)	0.50	$2^{31.28}$	$2^{22.72}$	y	$2^{9.52}$
	(5, 9)	(2, 2, 3)	1243.66	$2^{34.74}$	$2^{22.72}$	y	$2^{9.52}$
(48, 864, 43)	(6, 8)	(2, 2, 3)	124.37	$2^{37.09}$	$2^{23.23}$	y	$2^{12.3}$
(49, 882, 44)	(7, 7)	(2, 2, 3)	7.97	$2^{40.26}$	$2^{23.40}$	x or y	$2^{15.3}$
(56, 1008, 47)	(7, 8)	(2, 2, 3)	0.80	$2^{43.58}$	$2^{20.23}$	y	$2^{15.3}$

Table 5: A concrete analysis for a 20-bit prime p.

parameters			2-1 descent (from Tab 2)		relation collection		
$(n, \lg Q, n(Q))$	(n_1, n_2)	(d_1, d_2, e)	good $T(x,y)$	no of trials	lhs of (8)	pinpt side	no of trials (21) or (22)
(45, 900, 44)	(5, 9)	(1, 2, 2)	7.51	$2^{31.28}$	$2^{28.72}$	y	$2^{9.5}$
(48, 960, 46)	(6, 8)	(1, 2, 2)	0.83	$2^{33.77}$	$2^{29.23}$	y	$2^{12.3}$
(49, 980, 47)	(7, 7)	(2, 2, 3)	510.21	$2^{40.26}$	$2^{29.40}$	x or y	$2^{15.2}$
(56, 1120, 50)	(7, 8)	(2, 2, 3)	51.02	$2^{43.58}$	$2^{26.23}$	y	$2^{15.3}$
(64, 1280, 54)	(8, 8)	(2, 2, 3)	0.24	$2^{50.62}$	$2^{23.06}$	x or y	$2^{18.47}$

Table 6: A concrete analysis for a 25-bit prime p.

parameters			2-1 descent (from Tab 2)		relation collection		
$(n,\ \lg Q,\ n(Q))$	$(n_1,\ n_2)$	$(d_1,\ d_2,\ e)$	good $T(x,y)$	no of trials	lhs of (8)	pinpt side	no of trials (21) or (22)
(57, 1425, 57)	(8, 7)	(1, 1, 1)	0.17	$2^{24.79}$	$2^{41.23}$	x	$2^{15.3}$
(64, 1600, 61)	(8, 8)	(1, 1, 1)	0.02	$2^{27.96}$	$2^{38.06}$	x or y	$2^{18.47}$
	(8, 8)	(1, 2, 2)	0.04	$2^{47.30}$	$2^{38.06}$	x or y	$2^{18.47}$
	(8, 8)	(2, 2, 3)	7961.36	$2^{50.63}$	$2^{38.06}$	x or y	$2^{18.47}$

Table 7: A concrete analysis for a 32-bit prime p.

parameters			2-1 descent (from Tab 2)		relation collection		
$(n,\ \lg Q,\ n(Q))$	$(n_1,\ n_2)$	$(d_1,\ d_2,\ e)$	good $T(x,y)$	no of trials	lhs of (8)	pinpt side	no of trials (21) or (22)
(81, 2592, 81)	(9, 9)	(1, 2, 2)	0.24	$2^{58.13}$	$2^{52.72}$	x or y	$2^{21.79}$
(90, 2880, 86)	(9, 10)	(2, 2, 3)	465023	$2^{65.18}$	$2^{48.96}$	y	$2^{21.79}$
(100, 3200, 91)	(10, 10)	(2, 2, 3)	1359.72	$2^{73.09}$	$2^{45.5}$	x or y	$2^{25.25}$

Computer Algebra Software with respect to the two largest primes p_1 and p_2 in the product (24). It took 17 and 7 CPU (@ 3.07GHz) hours to complete the linear algebra with respect to primes p_1 and p_2 respectively. After completion of linear algebra we have got the discrete log of factor base elements modulo primes p_1 and p_2.

***Individual discrete log*:** As target for computing the discrete log, we have chosen the following field element.

$$\Pi(x) := \text{Normalize}\left(\sum_{i=0}^{n-1} \lfloor \pi p^{i+1} \mod p \rfloor x^i\right), \text{ i.e.,}$$

$$\begin{aligned}\Pi(x) \;=\; & x^{36} + 841x^{35} + 8875x^{34} + 2723x^{33} + 58297x^{32} + 37489x^{31} \\ & +14681x^{30} + 33725x^{29} + 27283x^{28} + 23704x^{27} + 54974x^{26} + 45806x^{25} \\ & +8606x^{24} + 30661x^{23} + 16779x^{22} + 46411x^{21} + 9333x^{20} + 32131x^{19} + \\ & 14681x^{18} + 47095x^{17} + 51019x^{16} + 2810x^{15} + 12343x^{14} + 12447x^{13} \\ & +16645x^{12} + 12487x^{11} + 12856x^{10} + 51853x^{9} + 59500x^{8} + 3681x^{7} + \\ & 4146x^{6} + 56159x^{5} + 9055x^{4} + 57991x^{3} + 1660x^{2} + 48553x + 9983. \qquad (25)\end{aligned}$$

The initial descent was quite fast and in few minutes we got $\Pi(x) = \frac{N(x)}{D(x)}$ as follows.

$$\begin{aligned} N(x) = \quad & (x + 49376) \times (x^2 + 6357x + 10310) \times \\ & (x^2 + 54990x + 10403) \times (x^2 + 58305x + 29395) \times \\ & (x^3 + 53070x^2 + 32425x + 31087) \times \\ & (x^3 + 60334x^2 + 48969x + 34559) \times \\ & (x^4 + 46865x^3 + 13867x^2 + 5909x + 49679) \times \\ & (x^4 + 53913x^3 + 48795x^2 + 9304x + 28222) \times \\ & (x^5 + 13129x^4 + 45787x^3 + 11181x^2 + 14710x + 3003) \times \\ & (x^5 + 33678x^4 + 41480x^3 + 31467x^2 + 42544x + 31415) \times \\ & (x^5 + 53759x^4 + 34226x^3 + 44358x^2 + 772x + 14985) \\ D(x) = \quad & (x^2 + 16528x + 61907) \end{aligned} \tag{26}$$

We have descended all the factors into degree two polynomials in x and in y within 6 minutes. The 2-1 descent for these polynomials were completed using the walk technique with 1 degree of freedom. It took us 100 minutes on a single core of the CPU (@ 3.07GHz) to get the complete $2-1$ descent. The average walk length of the entire $2-1$ descent is 16.

Using this descent and discrete logs of factor base elements we got the discrete log of $\Pi(x)$ modulo p_1 and p_2 respectively. Discrete logs modulo other small prime factors were computed using Pollard's Rho method and the results were combined using the CRT. We finally got the discrete log of $\Pi(x)$ as follows.

$$\begin{aligned} \log(\Pi(x)) \quad = \quad & 770654269744664411364422170900833384393002927833357493 \\ & 898475477568896179995725185460938555074307834593294725 5 \\ & 052244377820217461230522621222197519835055557916394440 1 \\ & 19840511924082. \end{aligned}$$

7.2 A Record on a Field of Size p^{40}, $p = 297079$ (728-bits)

We have further tried with a larger value of $p = 297079$, we have worked out the complete discrete log in the field of extension degree $n = 40$.

For this case we chose $n_1 = 8$ and $n_2 = 5$. The defining polynomial $f(x)$ of this field is generated by $g_1(x) = x^8$ and $g_2(x) = x^5 + 44024x^4 + 224924x^3 + 77320x^2 + 291141x + 80867$ with $f(x) = x^{40} + 44024x^{32} + 224924x^{24} + 77320x^{16} + 291141x^8 + 297078x + 80867$. This $f(x)$ generates a field of size p^{40} and $(x + 3)$ is a primitive element of the field.

Since the value of n_2 is 5, pinpointing fits very well here and we are able to complete the relation collection phase in 20 CPU (@ 2.30Ghz) hours. This provided a sparse matrix A with 624159 rows and 594158 columns. We solved this matrix modulo the largest prime factor p_1 of $p^n - 1$, which is a 286-bit prime number. It took 504 CPU (@3.07 GHz) hours using the in-built Lanczos algorithm of Magma. The discrete logs modulo other small prime factors of $p^n - 1$ were obtained using Pollard's rho and Pohlig Hellman algorithm.

We have chosen our target element $\Pi(x)$ as follows.

$$\Pi(x) = \text{Normalize}\left(\sum_{i=0}^{n-1} \lfloor \pi p^{i+1} \mod p \rfloor x^i\right) \text{ i.e.,}$$

$$\begin{aligned}\Pi(x) &= x^{39} + 154424x^{38} + 219291x^{37} + 2288x^{36} + 290227x^{35} + 295582x^{34} \\ &+27398x^{33} + 200403x^{32} + 6836x^{31} + 123295x^{30} + 94923x^{29} + 89389x^{28} \\ &+239023x^{27} + 115439x^{26} + 249309x^{25} + 196503x^{24} + 87998x^{23} + 240098x^{22} \\ &+136326x^{21} + 191206x^{20} + 9602x^{19} + 53215x^{18} + 25787x^{17} + 17954x^{16} \\ &+880x^{15} + 158602x^{14} + 241303x^{13} + 246920x^{12} + 52944x^{11} + 212605x^{10} \\ &+234395x^{9} + 196868x^{8} + 106113x^{7} + 207883x^{6} + 198491x^{5} + 106250x^{4} \\ &+165294x^{3} + 28548x^{2} + 76555x + 241986. \end{aligned} \tag{27}$$

It is the descent phase which took most of the time. The initial descent was quite fast and in few minutes we got $\Pi(x) = \frac{N(x)}{D(x)}$ as follows.

$$\begin{aligned}N(x) = \quad & (x^2 + 129346x + 178289) \\ & \times(x^2 + 129508x + 284926) \\ & \times(x^3 + 280690x^2 + 73103x + 113966) \\ & \times(x^4 + 45948x^3 + 55848x^2 + 84717x + 225935) \\ & \times(x^4 + 192446x^3 + 209512x^2 + 71975x + 24689) \\ & \times(x^4 + 214578x^3 + 134299x^2 + 75777x + 185981) \\ & \times(x^5 + 99215x^4 + 101352x^3 + 277735x^2 + 249592x + 166624) \\ & \times(x^5 + 149426x^4 + 93110x^3 + 223082x^2 + 29091x + 16179) \\ & \times(x^5 + 171408x^4 + 31726x^3 + 58503x^2 + 190122x + 213530) \\ & \times(x^5 + 209418x^4 + 233661x^3 + 279042x^2 + 64930x + 238441) \\ D(x) = \quad & (x^2 + 164221x + 32560).\end{aligned}$$

In around one hour we further converted them to degree 1 polynomials in x and degree at most 2 polynomials in y. Each degree 2 polynomial in y took on an average 500 CPU (@2.30 GHz) hours to descent to degree 1 polynomials and there are 59 of them. So in around 30000 CPU (@ 2.30 GHz) hours, we were able to get the complete descent of $\Pi(x)$ into degree 1 polynomials in x as well as y. The code was not optimised and with proper optimisation the actual computation time can come down greatly. Finally we used the CRT to get the discrete log of $\Pi(x)$ as follows.

$$\begin{aligned}\log(\Pi(x)) = \quad & 730193702775304384046745947228313596346480 \\ & 803400240950763141174029187190517313409792 5 \\ & 342153702522654039372608184558507369137933 7 \\ & 832616768741252142993539044632260376087765 9 \\ & 740520962963146604000921389665780564632839 \\ & 420364.\end{aligned}$$

8 Conclusion

In this paper, we have described a technique which ensures divisibility and partial smoothness which leads to concrete speed-ups in the relation collection and individual descent phases

of the FFS algorithm. A second contribution has been to systematically develop the walk technique and the technique of using additional degrees of freedom for the 2-1 descent step. As a consequence, we report record computations of discrete logs over fields of 16 and 18-bit characteristic. A concrete analysis of the algorithm shows that with access to a super-computer, tackling fields of greater sizes is feasible.

Acknowledgment:

We would like to thank Allan Steel of the Magma team for helpful discussions on the linear algebra step and the application of Pollard's rho.

References

[1] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodriguez-Henriquez. Weakness of $f_{3^{6*1429}}$ and $f_{2^{4*3041}}$ for discrete logarithm cryptography. Cryptology ePrint Archive, Report 2013/737, 2013. `http://eprint.iacr.org/`.

[2] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. Weakness of $f_{3^{6*509}}$ for discrete logarithm cryptography. Cryptology ePrint Archive, Report 2013/446, 2013. `http://eprint.iacr.org/`.

[3] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodrguez-Henrquez. Computing discrete logarithms in $f_{3^{6*137}}$ using magma. Cryptology ePrint Archive, Report 2014/057, 2014. `http://eprint.iacr.org/`.

[4] Leonard M. Adleman. The function field sieve. In Leonard M. Adleman and Ming-Deh A. Huang, editors, *ANTS*, volume 877 of *Lecture Notes in Computer Science*, pages 108–121. Springer, 1994.

[5] Leonard M. Adleman and Ming-Deh A. Huang. Function Field Sieve Method for Discrete Logarithms over Finite Fields. *Inf. Comput.*, 151(1-2):5–16, 1999.

[6] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *CoRR*, abs/1306.4244, 2013.

[7] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[8] James A. Davis and Diane B. Holdridge. Factorization of large integers on a massively parallel computer. In Christoph G. Günther, editor, *EUROCRYPT*, volume 330 of *Lecture Notes in Computer Science*, pages 235–243. Springer, 1988.

[9] Jérémie Detrey, Pierrick Gaudry, and Marion Videau. Relation Collection for the Function Field Sieve. In Alberto Nannarelli, Peter-Michael Seidel, and Ping Tak Peter Tang, editors, *IEEE Symposium on Computer Arithmetic*, pages 201–210. IEEE Computer Society, 2013.

[10] Faruk Göloglu, Robert Granger, Gary McGuire, and Jens Zumbrägel. On the Function Field Sieve and the Impact of Higher Splitting Probabilities: Application to Discrete Logarithms in $\mathbb{F}_{2^{1971}}$. *IACR Cryptology ePrint Archive*, 2013:74, 2013.

[11] Faruk Göloglu, Robert Granger, Gary McGuire, and Jens Zumbrägel. Solving a 6120-bit DLP on a Desktop Computer. *IACR Cryptology ePrint Archive*, 2013:306, 2013.

[12] Antoine Joux. CHAPMAN & HALL/CRC CRYPTOGRAPHY AND NETWORK SECURITY . June 2009.

[13] Antoine Joux. A new index calculus algorithm with complexity $\mathrm{L}(1/4+o(1))$ in very small characteristic. *IACR Cryptology ePrint Archive*, 2013:95, 2013.

[14] Antoine Joux. Discrete Logarithms in $GF(2^{6168})$ $\left[= GF(2^{257^{24}})\right]$. *NMBRTHRY list*, may 2013.

[15] Antoine Joux. Faster Index Calculus for the Medium Prime Case Application to 1175-bit and 1425-bit Finite Fields. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2013.

[16] Antoine Joux and Reynald Lercier. The Function Field Sieve Is Quite Special. In Claus Fieker and David R. Kohel, editors, *ANTS*, volume 2369 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2002.

[17] Antoine Joux and Reynald Lercier. The Function Field Sieve in the Medium Prime Case. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2006.

[18] W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team, 2013. `http://www.sagemath.org`.