# Faster index calculus for the medium prime case Application to 1175-bit and 1425-bit finite fields

Antoine Joux

CryptoExperts and
Université de Versailles Saint-Quentin-en-Yvelines, Laboratoire PRISM,
45 avenue des États-Unis, F-78035 Versailles Cedex, France
antoine.joux@m4x.org

**Abstract.** Many index calculus algorithms generate multiplicative relations between smoothness basis elements by using a process called *Sieving*. This process allows to filter potential candidate relations very quickly, without spending too much time to consider bad candidates. However, from an asymptotic point of view, there is not much difference between sieving and straightforward testing of candidates. The reason is that even when sieving, some small amount time is spend for each bad candidates. Thus, asymptotically, the total number of candidates contributes to the complexity.
In this paper, we introduce a new technique: *Pinpointing*, which allows us to construct multiplicate relations much faster, thus reducing the asymptotic complexity of relations' construction. Unfortunately, we only know how to implement this technique for finite fields which contain a medium-sized subfield. When applicable, this method improves the asymptotic complexity of the index calculus algorithm in the cases where the sieving phase dominates. In practice, it gives a very interesting boost to the performance of state-of-the-art algorithms. We illustrate the feasability of the method with a discrete logarithm record in medium prime finite fields of sizes 1175 bits and 1425 bits.

## 1 Introduction

Index calculus algorithms form a large class of algorithms for solving hard number theoretic problems which are often used as a basis for public key cryptosystems. They can be used for factoring large integers [14] and for computing discrete logarithms in finite fields [2, 10, 1] and in some elliptic or hyperelliptic curve groups [6, 8, 5, 9, 7].

All index calculus algorithms have in common two main algorithmic phase. The first of these phases is the generation of multiplicative[1] relations, which are converted into linear or affine equalities involving the logarithms of the elements which appears in the multiplicative relations. The second phase is the linear algebra phase, which solves the resulting system of equations. For factoring, the linear algebra is performed modulo 2. For discrete logarithms, it is done modulo the order of the relevant group. In addition to these two common phases, several other phases also appear: these extra phases heavily depend on the exact algorithm being considered. They can be further classified as preparatory or final phases. The preparatory phases search for a good representation of the structure being considered in order to speed-up the main phases. For example,

[1] In the case of curves, the relation are denoted additively, but the principle remains.

polynomial selection is a typical preparatory phase which appears when factoring with the number field sieve [15]. The final phases transform the raw output of the linear algebra phase into a solution of the considered problem. Typically, this includes the so-called square root phase of factoring algorithms and the individual logarithm phase encountered in many discrete logarithm algorithms. It should be noted that the computational cost of these prepatory and final phases is usually much smaller than the cost of the main phases.

In most cases, the designers of index calculus algorithms aim at balancing the theoretical complexity of the two main phases, since this usually yields the best global effectiveness. However, this is not always possible as illustrated by the function field sieve for the medium prime case introduced in [13]. In this specific case, the exact asymptotic complexity varies depending on the relative contribution of the base field and of the extension degree to the total size of the finite field being considered (see Section 2). In practice, the two main phases are usually much less balanced. This is due to the fact that the generation of relations phase can, in general, be distributed among machines in a straightforward way. On the contrary, the linear algebra requires a tightly coordinated computation and is generally performed on a centralized super-computer (or sometimes on a few super-computers). Since centralized computations that require tight communications are more expensive than distributed computations, implementers usually generate an extremely large number of linear equations compared to the number of unknowns. Using various techniques such as filtering, rebalancing or structured Gaussian elimination, this allows to greatly reduce the size of the linear system which is eventually solved and thus the cost of the linear algebra phase. This may increase the total computing power used for the computation, but trading expensive centralized computations for cheaper distributed computations is usually worthwhile.

As a consequence of these considerations, we see that the generation of relations is a very important phase of index calculus algorithms. Up to now, two main techniques are usually used. The simplest approach is direct trial where one simply checks whether a potential candidate turns into an effective relation by testing whether an integer or a polynomial splits into a product of “small” elements. In theory, the parameters of index calculus are selected to make sure that the cost of testing a candidate has a negligible contribution to the overall complexity. However, in practice, factoring these objects has a non-neglibible cost. Thus, the other approach called *sieving* is usually prefered. The basic idea of sieving is to proceed backward and mark all multiples of small elements. Clearly, an object which receives many marks is much more likely to generate a useful multiplicative relation that an object which receives few marks. Note that, from a theoretic point of view, sieving does not change the complexity of the sieving phase. Indeed, all the potential candidates still need to be considered and even reducing the cost of considering a candidate to a unit cost would not be enough to lower the overall asymptotic complexity.

In this paper, we introduce a new technique to generate relations which is much faster that sieving. In the some cases, the cost of relation generation becomes essentially optimal: we only require a small number of arithmetic operations per *generated relation.* To indicate that this technique sometimes allow direct access the relations, we name it *Pinpointing.* Unfortunately, we only know how to achieve this for a limited number of index calculus algorithms. More precisely, we show how to use pinpointing for the medium prime case as described in [13].

## 2 A refresher on the medium prime case

The medium prime discrete logarithms proposed in [13] works as follows. In order to compute discrete logarithms in $\mathbb{F}_{q^n}$, a degree $n$ extension of the base field $\mathbb{F}_q$, it starts by defining the extension field implicitly from two bivariate polynomials in $X$ and $Y$:

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

where $g_1$ and $g_2$ are univariate polynomials of degree $d_1$ and $d_2$. In order to define the expected extension, this requires that the polynomial $-g_2(g_1(Y)) + Y$ has an irreducible factor $F(Y)$ of degree $n$ over $\mathbb{F}_q$. As explained in [13], it is easy to find polynomials $g_1$ and $g_2$ that satisfy this requirement.

The relative degrees of $d_1$ and $d_2$ in this case are controlled by an extra parameter $D$, whose choice is determined by the size of $q$ compared to $q^n$. More precisely, we have $d_1 \approx \sqrt{Dn}$ and $d_2 \approx \sqrt{n/D}$.

Starting from this definition of the finite field, the medium prime field algorithms consider objects of the form $\mathcal{A}(Y)\,X + \mathcal{B}(Y)$, where $\mathcal{A}$ and $\mathcal{B}$ are univariate polynomials of degree $D$ and $\mathcal{A}$ is unitary. Substituting $X$ by $g_1(Y)$ on one side and $Y$ by $g_2(X)$ on the other, we obtain an equation:

$$\mathcal{A}(Y)\,g_1(Y) + \mathcal{B}(Y) = \mathcal{A}(g_2(X))\,X + \mathcal{B}(g_2(X)).$$

This relates a polynomial of degree $d_1 + D$ in $Y$ and a polynomial of degree $Dd_2 + 1$ in $X$.

To use the equations as index calculus relations, the algorithm of [13] selects the set of all unitary polynomials of degree at most $D$ in $X$ or $Y$, with coefficients in $\mathbb{F}_q$ as its smoothness basis and keeps pairs of polynomials $(a, b)$ such that the two polynomials $a(Y)\,g_1(Y) + b(Y)$ and $a(g_2(X))\,X + b(g_2(X))$ both factor into terms of degree at most $D$. These good pairs are found using a classical sieving approach.

Writing $Q = q^n$, to analyze the complexity of the medium prime discrete logarithms, [13] chooses to write $q = L_Q(1/3, \alpha\, D)$, where as usual:

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

In this setting, the (heuristic) asymptotic complexity of the sieving phase is $L_Q(1/3, c_1)$ and the complexity of the linear algebra is $L_Q(1/3, c_2)$, with:

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

Note that the algorithm with parameter $D$ only works under the condition:

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

Otherwise, the number of expected relations is too small to relate all elements of the smoothness basis. For a finite field $\mathbb{F}_{q^n}$, [13] indicates that the best complexity is obtained choosing the smallest acceptable value for the parameter $D$.

### 2.1 Individual discrete logarithms phase

Another very important phase that appears in many index calculus based algorithms is the individual discrete logarithms phase which allows to compute the logarithm of an arbitrary field element by finding a multiplicative relation which relates this element to the elements of the smoothness basis whose logarithms have already been computed.

In [13], this is done by first expressing the desired element as a product of elements which can be represented as low degree polynomials in $X$ or $t$. These polynomials can in turn be related to polynomials of a lower degree and so on, until hitting degree one, i.e. elements of the smoothness basis. For this reason, the individual logarithm phase is also called the descent phase.

As analyzed in [13], the asymptotic complexity of the descent phase is

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

where $\mu < 1$ is an arbitrary parameter. Moreover, any choice of $\mu$ in the interval $]1/2;1[$ ensures that the complexity of the descent phase is asymptotically negligible compared to (at least one of) the main phases.

## 3 Pinpointing

### 3.1 Basic framework

In order to improve the generation of relations, we first consider the simple case with parameter $D = 1$ and we construct our finite field extension using two polynomials that have the following restricted form:

$$X = Y^{d_1} \quad \text{and}$$
$$Y = g_2(X),$$

where $g_2$ is a polynomial of degree $d_2$. To generate relations, since $D = 1$, we consider the space spanned by $XY$, $X$, $Y$ and 1, i.e., after renormalization we are thus considering the following candidates:

$$Y^{d_1+1} + aY^{d_1} + bY + c = X\,g_2(X) + a\,X + b\,g_2(X) + c,$$

where $a$, $b$ and $c$ are arbitrary coefficients in $\mathbb{F}_q$.

A candidate yields a valid multiplicative relation when both sides factor into linear polynomials. We remark that the left-hand side $Y^{d_1+1} + aY^{d_1} + bY + c$ splits into linear terms, if and only if, $U^{d_1+1} + U^{d_1} + b\,a^{-d_1}\,U + c\,a^{-d_1-1}$ factors into linear terms. This can be seen by performing the change of variable $Y = aU$ and dividing by $a^{d_1+1}$.

### 3.2 One-sided pinpointing

Using this remark, we obtain a first form of pinpointing which only focuses on the $Y$ side. This form searches for polynomials in $U$ of the form $U^{d_1+1}+U^{d_1}+B\,U+C$. This can be done either by directly testing candidates or by sieving. We need to consider approximately $(d_1+1)!$ candidates to find a good polynomial.

Once we have obtained one such smooth polynomial, we can amplify it (using the change of variable $U=Y/a$) into many polynomials $Y^{d_1+1}+aY^{d_1}+bY+c$, where $a$ is an arbitrary non-zero element in $\mathbb{F}_q$, $b=Ba^{d_1}$ and $c=Ca^{d_1+1}$. This amortizes the cost of finding the initial polynomial, distributing this cost among many candidates. Indeed, we expect to obtain approximately $q/(d_2+1)!$ relations by testing the right-hand sides corresponding to $q-1$ different values of $a$. Adding to this the cost of finding the initial smooth polynomial, we find an amortized cost per relation close to:

$$\frac{(d_1+1)!+(q-1)}{(q-1)/(d_2+1)!}=\frac{(d_1+1)!\,(d_2+1)!}{q-1}+(d_2+1)!$$

This is clearly better than the cost of classical sieving which, in this case, amounts to $(d_2+1)!\,(d_1+1)!$ operations per relation. More precisely, this improves the cost of the relation by a factor of, at least, $\min(q-1,(d_1+1)!)/2$.

### 3.3 Kummer extensions, Frobenius and advanced pinpointing

With some specific extension fields, it is possible to achieve an even better improvement over sieving, using a two-side approach to pinpointing. Moreover, this can be done while taking into account the action of Frobenius which allows us to reduce the size of the linear system.

We illustrate this using Kummer extensions of degree $n=d_1d_2-1$. We recall that a Kummer extension of degree $n$ is defined over a finite field $\mathbb{F}_q$ which contains $n$-mi roots of unity by a polynomial $P(X)=X^n-\alpha$, where $\alpha$ has no root of prime order $m|n$ in $\mathbb{F}_q$. Let $\mu$ denote a primitive $n$-th root of unity in $\mathbb{F}_q$ and $x$ denote a $n$-th root of $\alpha$ in $\mathbb{F}_{q^n}$, then we have:

$$P(X)=\prod_{i=0}^{n-1}(X-\mu^i x).$$

As a consequence, there exists a $i_0$ prime to $n$ such that $x^q=\mu^{i_0}x$. By changing our choice of primitive root $\mu$, we can ensure that $i_0=1$. Thus, throughout the sequel, we have $x^q=\mu\,x$.

Such a Kummer extension can be obtained in our framework by defining:

$$\begin{aligned} X &= Y^{d_1}/\alpha \quad \text{and} \\ Y &= X^{d_2} \end{aligned} \tag{2}$$

Substituting one equation in the other, we find $X^{d_1d_2}-\alpha\,X=0$. Thus dividing by $X$ we obtain the desired Kummer extension. If $x$ denotes as above the image of $X$ in $\mathbb{F}_{q^n}$, the image of $Y$ is $y=x^{d_2}$. Once again, since we are considering

$D = 1$, our smoothness basis contains all the linear polynomials $x + a$ and $y + a$ with $a$ in $\mathbb{F}_q$.

The Frobenius acts on the smoothness basis as follows:

$$(x + a)^q = x^q + a = \mu\, x + a = \mu(x + a/\mu) \quad \text{and}$$
$$(y + a)^q = y^q + a = \mu^{d_1}\, y + a = \mu^{d_1}(y + a/\mu^{d_1}).$$

As a consequence, in the quotient group $\mathbb{F}_(q^n)^*/\mathbb{F}_q^*$, we have:

$$\log(x + a/\mu) = q \log(x + a) \quad \text{and}$$
$$\log(y + a/\mu^{d_1}) = q \log(y + a).$$

These relations allow us to divide the number of unknowns in the linear system that we need to solve by a factor essentialy equal to $n$. Indeed, all elements in the factor base except $x$ and $y$ have precisely $n$ conjuguates (including themselves). Moreover, since $x^{n(q-1)} = 1$ and $y^{n(q-1)} = 1$, the logarithms of $x$ and $y$ are equal to 0 modulo any large prime dividing the order of the quotient group.

**Advanced pinpointing: Generating equations in Kummer extensions**
As in the one-sided case, we consider the space of candidates generated by $XY$, $X$, $Y$ and 1. Due to our specific choices, the renormalized candidates can be rewritten in a slightly simpler form:

$$XY + aY + bX + c =$$
$$X^{d_2+1} + aX^{d_2} + bX + c = Y^{d_1+1}/\alpha + b\,Y^{d_1}/\alpha + aY + c.$$

We now remark that the polynomial on the $X$ side splits, if and only if, $U^{d_2+1} + U^{d_2} + b\,a^{-d_2}\,U + c\,a^{-d_2-1}$ splits. Moreover, the polynomial on the $Y$ side splits, if and only if, $V^{d_1+1}/\alpha + V^{d_1}/\alpha + a\,b^{-d_1}\,V + c\,b^{-d_1-1}$ splits.

Let $\lambda = c/(ab)$, then the polynomials in $U$ and $V$ can respectively be rewritten as:

$$U^{d_2+1} + U^{d_2} + b\,a^{-d_2}\,(U + \lambda) \quad \text{and} \quad (V^{d_1+1} + V^{d_1})/\alpha + a\,b^{-d_1}(V + \lambda).$$

Converserly, choose a triple $(A, B, \lambda)$, with $A \neq 0$ and $B \neq 0$ and $AB^{d_2}$ a $n$-th power in $\mathbb{F}_q$ such that:

$$U^{d_2+1} + U^{d_2} + A\,(U + \lambda) \quad \text{and} \quad (V^{d_1+1} + V^{d_1})/\alpha + B(V + \lambda)$$

both split. Then, we can recover a unique (up to Frobenius action) triple $(a, b, c)$ corresponding to a candidate that yields an equation in the finite field. We first recover $a$ and $b$. Putting together the two equations $A = ba^{-d_2}$ and $B = ab^{-d_1}$, we find $b^n = b^{d_1 d_2 - 1} = 1/(AB^{d_2})$. Since, by hypothesis, $AB^{d_2}$ is a $n$-th power this equation has $n$ distinct solutions. Choose one arbitrary solution for $b$, then we necessarily have $a = Bb^{d_1}$ and $c = \lambda ab$. We thus obtain a valid candidate $(a, b, c)$. To show the unicity up to Frobenius action, we start from another solution $\mu^i b$ and obtain the triple $(\mu^{d_1 i}, \mu^i b, \mu^{(d_1+1)i} c)$. Now, let the Frobenius act $j$ times on:

$$X^{d_2+1} + aX^{d_2} + bX + c$$

and renormalize to obtain:

$$X^{d_2+1} + a\mu^{-jd_2}X^{d_2} + b\mu^{-jd_1d_2}X + c\mu^{-jd_1(d_2+1)}.$$

Since $d_1d_2 \equiv 1 \pmod{n}$, we see that for $j \equiv -i \pmod{n}$, the action of Frobenius yields that same equation as the new choice for $b$.

*Notes.* Once $\lambda$ is fixed, finding the triples $(A, B, \lambda)$ which satisfy the property that $AB^{d_2}$ is $n$-th power is a simple matter. Indeed, it suffices to partition the list of possible values for $A$ and $B$ in $n$ sublists depending on the discrete logarithms of $A$ (resp. $B$) modulo $n$. Since $n$ is small, these values are easily computed by comparing $A^{(q^n-1)/n}$ (resp. $B^{(q^n-1)/n}$) with the possible $n$-th root of unity in $\mathbb{F}_Q$.

We can also remark that this form advanced pinpointing can be used for some extension fields which are not Kummer extension. Indeed, when $d_1d_2 - 1$ does not divides the order of $\mathbb{F}_q$, choosing $X = Y^{d_1}/\alpha$ and $Y = X^{d_2}$ cannot define an extension of degree $d_1d_2 - 1$ because the polynomial $X^{d_1d_2} - \alpha X$ has two roots in $\mathbb{F}_q$. However, depending on $q$, it may yield a extension of degree $d_1d_2 - 2$. The main drawback compared to the Kummer extensions is that we cannot use the action of Frobenius to reduce the size of the smoothness basis.

**Cost considerations** For each value of $\lambda$, creating the list of $A$-values costs $O(q)$ operations and the list contains about $(q-1)/(d_2+1)!$ elements. Similarly, the list of $B$-values costs $O(q)$ operations and contains approximately $(q-1)/(d_1+1)!$ elements. For a fixed $\lambda$, the total number of $(A, B)$ pairs that yields a good triple $(A, B, \lambda)$ is approximately:

$$\frac{(q-1)^2}{n(d_1+1)!(d_2+1)!}.$$

As a consequence, the average cost of constructing one relation is:

$$1 + O\left(\frac{n(d_1+1)!(d_2+1)!}{(q-1)}\right). \tag{3}$$

If we remember that the factor $n$ in the second term is compensated by the fact that we only need $q/n$ relations instead of $q$, we see that the other term is reduced from $(d_1+1)!$ to 1. As a consequence, the gain compared to sieving is at least $(q-1)/2$.

An interesting side-effect of this advanced pinpointing is that once the list of $A$ and $B$ values have been stored, the equations can be regenerated for a constant cost. This is interesting, because these lists are smaller than the list of equations. As a consequence, rather that storing the equations, it becomes preferable to recompute them on the fly whenever they are needed, thus saving disk space (and disk access time).

## 3.4 Complexity of relation construction using pinpointing

We first recall that the cost of sieving from [13]:

$$L_Q\left(1/3, \alpha + \frac{2}{3\sqrt{\alpha}}\right).$$

Moreover it is only applicable for $\alpha \geq 3^{-2/3}$.

**Using one-sided pinpointing** As in [13], we now assume consider the complexity of computing discrete logarithms in a field $\mathbb{F}_Q$, with $Q = q^n$, assuming that the parameter $\alpha$ defined as:

$$\alpha = \frac{1}{n}\left(\frac{\log Q}{\log\log Q}\right)^{2/3}$$

is fixed. In this setting, we have $q = L_Q(1/3, \alpha)$. Since the smoothness basis has size $2q$, the cost of the linear algebra is the same as in [13], i.e., it is $L_Q(1/3, c_2)$ with $c_2 = 2\alpha$.

However, the complexity of collecting the relations is reduced compared to sieving. Indeed, the cost of collecting approximately $2q$ relations becomes:

$$2(d_1 + 1)!(q + (d_2 + 1)!).$$

Using the usual choice for $d_1$ and $d_2$, this can be written as:

$$L_Q\left(1/3, \frac{1}{3\sqrt{\alpha}} + \max(\alpha, \frac{1}{3\sqrt{\alpha}})\right)$$

Note that this can be further improved by choosing the degrees $d_1$ and $d_2$ as follows:

$$d_1 \approx \frac{1}{3\alpha^2}\left(\frac{\log(Q)}{\log\log(Q)}\right)^{1/3} \quad \text{and}$$
$$d_2 \approx 3\alpha\left(\frac{\log(Q)}{\log\log(Q)}\right)^{1/3}.$$

For $\alpha \geq 3^{-2/3}$, this reduces the complexity to

$$L_Q\left(1/3, \alpha + \frac{1}{9\alpha^2}\right)$$

**Using advanced pinpointing** To determine the asymptotic complexity of the advanced pinpointing method, we can ignore the action of Frobenius. Indeed, despite offering a very useful practical improvement, it does not provide an asympotic gain. The cost of collecting enough relations in this case is:

$$2(q + (d_1 + 1)!(d_2 + 1)!).$$

We choose:

$$d_1 \approx d_2 \approx \alpha^{-1/2}\left(\frac{\log(Q)}{\log\log(Q)}\right)^{1/3}$$

As a consequence, the cost of building the relations becomes:

$$L_Q\left(1/3, \max\left(\alpha, \frac{2}{3\sqrt{\alpha}}\right)\right).$$

*Direct access to relations.* When $\alpha \geq \frac{2}{3\sqrt{\alpha}}$, i.e. $\alpha \geq (2/3)^{2/3}$, the cost of building relations becomes equal to the number of relations. In other words, the right summand in equation (3) becomes negligible and each relation can be built in constant time. In this context, the pinpointing technique gives direct access to multiplicative relations. It is weird to note that, in this best case for pinpointing, there is no improvement on the full complexity, as shown in the next paragraph.

**Impact on the full discrete logarithm complexity** In order to define the asymptotic complexity of the discrete logarithm computation for the algorithm with parameter $D = 1$, we also need to take into account the complexity of the linear algebra $L_Q(1/3, 2\alpha)$. For $\alpha \geq 3^{-2/3}$, this cost is higher than the cost of pinpointing in either version. As a consequence, in this range, the full complexity of discrete logarithm computation becomes $L_Q(1/3, 2\alpha)$. When $\alpha$ is in the interval $[3^{-2/3}; (2/3)^{2/3}[$, this is better than the algorithm of [13] whose cost is dominated by sieving. In particular, for $\alpha = 3^{-2/3}$, the cost is reduced from $L_Q(1/3, 3^{1/3}) \approx L_Q(1/3, 1.44)$ to $L_Q(1/3, 2/3^{2/3}) \approx L_Q(1/3, 0.96)$.

## 4 Generalization to $D > 1$

The one-sided pinpointing technique presented above can easily be generalized to the case where $D > 1$ in a straightforward way. More precisely, it suffices to remark that a polynomial:

$$X^d + \sum_{i=0}^{d-1} a_i X^i,$$

can be decomposed into a product of polynomials of degree at most $D$, if and only if, the polynomial:

$$U^d + U^{d_1} + \sum_{i=0}^{d-2} a_i\, a_{d-1}^{d-i} U^i$$

can be decomposed into a product of polynomials of degree at most $D$.

As a consequence, we can essentially save a factor $q-1$ compared to a sieving approach if we use a pinpointing approach in this general case.

*Resulting complexity.* As in [13], we consider the case where Equation (1) is satisfied. The amortized cost of constructing one relation is:

$$\frac{S_D(d_1 + D) + (q-1)}{(q-1)/S_D(Dd_2 + 1)} = \frac{S_D(d_1 + D)S_D(Dd_2 + 1)}{q-1} + S_D(Dd_2 + 1),$$

where $S_D(T)$ denotes the inverse of the probability for a degree $T$ polynomial to decompose as a product of polynomials of degree at most $D$. We recall that $S_D(T) \approx \exp((T/D)\log T/D)$ (see [13, 16]). As a consequence, the runtime of the relation collection is approximated by:

$$S_D(d_1 + D)S_D(Dd_2 + 1)q^{D-1} + S_D(Dd_2 + 1)q^D.$$

For the usual choice, $d_1 \approx \sqrt{Dn}$ and $d_2 \approx \sqrt{n/D}$ and writing $q = L_Q(1/3, \alpha\, D)$ this becomes:

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

Depending on the exact value of $\alpha$, it can be re-optimized by changing the value of $d_1$ and $d_2$. When possible, the complexity becomes:

$$L_Q\left(1/3, D^2\alpha + \frac{1}{9D^2\alpha^2}\right).$$

To test whether re-optimization is possible, it suffices to compare the two complexities and keep the smaller.

## 4.1 Kummer extensions with $D > 1$.

In the case where $D > 1$, it is clear that using Kummer extensions allows to account for the action of Frobenius, as in the $D = 1$ case. However, it is less clear that a dual-sided approach is also possible in this case. It turns out that the method used for $D = 1$ remains applicable. More precisely, define the relation between $X$ and $Y$ as in equation 2 and consider the space of candidates $\mathcal{A}(Y)\,X + \mathcal{B}(Y)$, where $\mathcal{A}$ and $\mathcal{B}$ are polynomials of degree $D$ and $\mathcal{A}$ is unitary. We write $\mathcal{A}(Y) = Y^D + aY^{D-1} + \cdots$ and $\mathcal{B}(Y) = bY^D + cY^{D-1} + \cdots$.

The $X$-side is:

$$X^{Dd_2+1} + bX^{Dd_2} + aX^{(D-1)d_2+1} + cX^{(D-1)d_2} + \cdots$$

It splits, if and only if:

$$U^{Dd_2+1} + U^{Dd_2} + \frac{a}{b^{d_2}}\left(U^{(D-1)d_2+1} + \frac{c}{ab}U^{(D-1)d_2}\right) + \cdots$$

also splits. Similarly, the $Y$-side is:

$$Y^{d_1+D}/\alpha + aY^{d_1+D-1}/\alpha + \cdots + bY^D + cY^{(D-1)} + \cdots$$

It splits, if and only if:

$$V^{d_1+D}/\alpha + V^{d_1+D-1}/\alpha + \cdots + \frac{b}{a^{d_1}}\left(V^D + \frac{c}{ab}V^{(D-1)}\right) + \cdots$$

also splits. As a consequence, given $\lambda = c/(ab)$, $A = b/a^{d_1}$ and $B = a/b^{d_2}$ such that $AB^{d_1}$ is a $n$-th power, we can transform all smooth polynomials in $U$ and $V$ into smooth polynomials in $X$ and $Y$ form with matching values for $a$, $b$ and $c$. If the other coefficients also match, we obtain a relation.

However, due to the cost of matching extra coefficients, this is not as favorable as in the case $D = 1$.

## 5 Application: a record on 1175 bits

In order to demonstrate the practicality of our algorithm, we give a new record for discrete logarithms in finite fields in a particularly favorable case. More precisely, we decided to improve on the discrete logarithm record in $\mathbb{F}_{370\,801^{30}}$ presented in [12] and to choose a much larger field $\mathbb{F}_{p^{47}}$, with $p = 33\,553\,771$.

To the best of our knowledge, the previous discrete logarithm record in a finite field concerned $\mathbb{F}_{3^{582}}$, a 923-bit field (see [11]). Our result thus increases the size of the previous record by more than 250 bits. In order to illustrate the running time improvements gained from our new technique, we compare in the sequel our running times and the running times from [11]. However, we wish to warn the reader than this comparison should be analyzed with care. Indeed, the finite field we have chosen is especially well-suited to our new techniques.

Concerning the chosen finite field, we first remark that $p \equiv 1 \pmod{47}$. As a consequence, we define the extension field using the relations $Y = X^6$ and $Y^8 = 2X$. This allows us to use advanced pinpointing and take advantage of the action of Frobenius. We obtain a smoothness basis of 1.43M elements. The cardinality of the finite field is:

$$p^{47} - 1 = 47 \cdot 2069 \cdot 12409 \cdot (p-1) \cdot 132103049403319 \cdot C,$$

where $C$ is a 1073-bit composite cofactor of unknow factorization[2].

By construction, $X$ has order $47(p-1)$ and thus cannot serve as a base for discrete logarithm. However, $X-3$ is very likely to have order $p^{47}-1$. Indeed, none of the values $(X-3)^{(p^{47}-1)/f}$ is equal to 1, when $f$ is chosen as one of the known factors of $p^{47}-1$. This choice is validated by our computation since we can find logarithms of random elements in basis $X-3$.

As expected, the construction of the multiplicative relations is extremely efficient. For this reason, it was performed on a single laptop, using one CPU. We used advanced pinpointing. The preparatory construction of smooth-polynomials, for 1000 different values of $\lambda$, took a little more than 3 hours on the laptop. Once this is done, we performed the computation of the relations together with the structured Gaussian elimination, in 2 minutes. The resulting linear system contains 829 405 unknowns.

As expected, the computation is dominated by the linear algebra step. We performed this step using a block Wiedemann approach (as in [17]), based on 32 independent series of matrix-vector evaluation. Each run in the series was performed on a 16-core[3] node of Genci's Curie computer, using OMP threads, thus using a total of 512 processors. The initial matrix-vector products required almost 37 hours. Due to memory requirement, the computation of a relation using block Wiedemann was done on 64 cores of a larger node[4] of Curie: it took

[2] We have not made any special effort to factor $C$, so it may well have some factors of moderate size.

[3] More precisely, it was on Curie's thin nodes: each node contains two octocore Intel Sandy Bridge EP (E5-2680) processors at 2.7 GHz.

[4] Here, we used half of a Curie's xlarge node, i.e. eight octocore Intel Nehalem-EX X7560 processors at 2.26 GHz .

9h30min. Finally the recovery of the solution took 32 additional matrix-vectors products of half length compared to the initial runs. Due to the extra cost of combining the intermediate values using the coefficients in the relation, this required almost 25 hours. The grand total amounts to about 32 000 CPU-hours. We give a comparison of the timings with the previous record in Table 1.

| | Bitsize | Total time | Relation construction | Linear algebra | Indiv. Log. |
|---|---|---|---|---|---|
| [11] | 923 bits | 813 000 CPU.hours | 270 000 CPU.h | 483 000 CPU.h | 60 000 CPU.h |
| This paper | 1175 bits | 32 000 CPU.hours | 3 CPU.h | 32 000 CPU.h | 4 CPU.h |
| This paper (sect. 6) | 1425 bits | 32 000 CPU.hours | 6 CPU.h | 32 000 CPU.h | $< 12$ CPU.h |

**Table 1.** Comparison between the computation and the previous record

To illustrate the results, we first give some discrete logarithms of factor base elements in base $x - 3$ modulo $C$. We note that, as expected, the logarithms of $x$ and $y$ are both zero modulo $C$.
$\log(x - 1) =$
9582795541737278896121111975072992426840673441812986512432729094303174813963307585
608132513690904548623048761915897361884626391990991915021252815743485371232521487
550419478952338539115508679394371566375230894207985911491080519272692791611988011
3567931724180868901724860373253630442936157531712330314891742481658155699597613
$\log(x - 2) =$
7090528473238095951689601257810214792022939168707127234747139128938017062973928799
784834427240863265771865333150182888786736466918234480207904443446176205810343860
258003225338333745643272436768815409667195040018407508999451787767277920160142155
4905236313618024147453867620526117100039277887621947094730742958277464818390465
$\log(x + 1) =$
2219197295801479208732758775937299916529838073890577586159555175186695936299329612
589321195206435255500444661214369102176664640590748154282384455232560365916466867
272653054497391873491553872115181752368292317397490449499747232720490530799293430
6560207700268824803542475788420305869403355275251219067807853032669466128535944
$\log(y - 1) =$
1000100548647187305396022314428175424995339319701459165990666283575439057474882300
678137375770156614456332808260867233779884537627368421534661910250604091137008522
810858342436029823671141625278463012489160339759341675107012806807122267180940999
3310027600177764254738174863646977658739191171546816174310362854136358144609763
$\log(y - 2) =$
6152769021272071427164382240109409711556872058173440596607139124811906977612264783
113197227671043771028486405816181851329919992761090054750149949658217142821761304
072023066379634003883005238853684703190601033833558745506007806517134746112096494
5190720689474052263809706188924679921054683415735227928798768439809316649275349

The logarithms can be easily checked by computing the values

$$(X - i)^{(p^{47}-1)/C} - (X - 3)^{\log(X-i)\,(p^{47}-1)/C} \quad \text{and}$$
$$(X^5 - i)^{(p^{47}-1)/C} - (X - 1)^{\log(Y-i)\,(p^{47}-1)/C}$$

in $\mathbb{F}_{p^{47}}$: these values are all equal to 0.

## 5.1 Individual discrete logarithm

The computation of individual discrete logarithms is unchanged from [13] and requires a moderate amount of computing power. We illustrate this by computing the logarithm of:

$$Z = \sum_{i=0}^{46} \left( \lfloor \pi\, p^{i+1} \rfloor \bmod p \right) X^i .$$

The first step is to find a value related to $Z$ which can be expressed using polynomials in $X$ of relatively low degree. Here, we find that:

$$Z \cdot (X+1)^{359} = \frac{N}{D},$$

where:

$$\begin{aligned}
N = {} & (X + 14210538) \cdot (X^3 + 11391523\,X^2 + 11557966\,X + 4735070) \cdot \\
& (X^5 + 19077323\,X^4 + 33518441\,X^3 + 10948280\,X^2 + 22585133\,X + 10781284) \cdot \\
& (X^6 + 23779790\,X^5 + 19694794\,X^4 + 3592889\,X^3 + 22256672\,X^2 + 26459485\,X + 17252086) \cdot \\
& (X^8 + 30752676\,X^7 + 23235735\,X^6 + 9985255\,X^5 + 12021197\,X^4 + \\
& 24717929\,X^3 + 32859134\,X^2 + 9427686\,X + 30050174) \\
D = {} & (X^2 + 1349926\,X + 27933009) \cdot (X^2 + 6270430\,X + 128544) \cdot \\
& (X^2 + 19943007\,X + 2446135) \cdot (X^4 + 11569768\,X^3 + 16137291\,X^2 + 16134928\,X + 18008733) \cdot \\
& (X^5 + 23551399\,X^4 + 20101733\,X^3 + 18928216\,X^2 + 459685\,X + 10057276) \cdot \\
& (X^8 + 29709444\,X^7 + 18269267\,X^6 + 8636523\,X^5 + 16115038\,X^4 + \\
& 422786\,X^3 + 30511605\,X^2 + 9551655\,X + 13032796).
\end{aligned}$$

Once, this is done, we use the descent procedure to express each factor using polynomials of lower degree in $X$ and $Y$. The slowest step in the descent is the final step that expresses polynomials of degree 2 using linear polynomials. After the earlier steps of the descent, we have a total of 278 degree polynomials whose logarithms are required (156 in $X$ and 122 in $Y$). In the final step, we consider all polynomials of the form $XY + aY + bX + c$ that are multiples of the target polynomial and use sieving to find a relation between this target and linear polynomials. When not possible, we use a relation that also include another degree 2 polynomial and restart from that polynomial. The total time to obtain all these logarithms on the laptop used for computing the relations is under 4 hours.

Finally, back-substituting all the logarithms of the linear polynomials, we derive the logarithm of $Z$ modulo $C$. To ease verification, we have also computed this logarithm modulo the small factors and thus give its complete value. We have:

$$\begin{aligned}\log(Z) = {} & 356633127146494066263281134740949440571780807878239530830992112523140 49\\ & 4277589347504555481509115749560473147631864963745877949210252568865 7986\\ & 4264903904703346205062752281331793708466214722799475637645216460889 8303\\ & 6872873337915243309378992279523113002528828381737389659610454461801 4057\\ & 3240231646914447899262099152488534480737568049333712088197470913054182\end{aligned}$$

## 6 Details of a larger record on 1425 bits

Since the above record only required a moderate amount of computing power, we further investigated the potential of the technique by choosing a new target of a similar form. We considered $\mathbb{F}_{p^{57}}$, with $p = 33\,341\,353$.

We have $p \equiv 1 \pmod{57}$ and we defined the extension field using the relations: $Y = X^7$ and $X = 2/Y^8$. The advanced pinpointing technique can easily be adapted to deal with this case and take advantage of the action of Frobenius. The initial smoothness basis thus contains1.17M elements. The cardinality of the finite field is:

$$\begin{aligned}p^{57} - 1 &= (p-1)\cdot(p^2+p+1)\cdot 19 \cdot p_1 \cdot p_2 \quad \text{where}\\ p_1 &= (\sum_{i=0}^{18} p^i)/19 \quad \text{and}\\ p_2 &= \sum_{i=0}^{12} p^{3i} - (p+p^{20})\sum_{i=0}^{5} p^{3i}.\end{aligned}$$

The two primes $p_1$ and $p_2$ respectively have 446 and 900 bits.

Since, $X$ has order $57(p-1)$, we use $X-11$ as our basis for discrete logarithms. The construction of the multiplicative relations was performed on the same laptop as previously. For the preparatory construction of smooth-polynomials, we used 2000 different values of $\lambda$, which took 6 hours on the laptop. Once this is done, we performed the computation of the relations together with the structured Gaussian elimination, in 2 minutes. The resulting linear system contains 714 931 unknowns.

Once again, the computation is dominated by the linear algebra step. This time, we split the computation into two independent parts, adressing $p_1$ and $p_2$ separately. For each of the two primes, the initial matrix-vector products required 18 hours and 30 minutes, using a 16-core node for each prime. The block Wiedemann step required 2h30m for $p_1$ and 6h10m for $p_2$, using 64 cores for each computation. The final run of matrix-vectors products took 12 hours for each prime. Once again, the grand total amounts to about 32 000 CPU-hours.

### 6.1 Individual discrete logarithm

The computation of individual discrete logarithms works as previously. However, for performance reasons, we reimplemented the descent procedure in C, instead of using a mix of PARI/GP scripts and C code as before. The computing power required for an individual logarithm remains moderate and could be parallelized if required. We illustrate this by computing the logarithm of:

$$Z = \sum_{i=0}^{56} \left( \lfloor \pi \, p^{i+1} \rfloor \bmod p \right) X^i.$$

We have:

$$Z \cdot (X - 11)^{2859} = \frac{N}{D},$$

where:

$$\begin{aligned}
N = {} & (X + 453778) \cdot (X^2 + 69543\, X + 24883037) \cdot \\
& (X^7 + 14652112\, X^6 + 23050491\, X^5 + 16035316\, X^4 + \\
& 16777829\, X^3 + 20432920\, X^2 + 10070364\, X + 1446274) \cdot \\
& (X^8 + 88887\, X^7 + 13267655\, X^6 + 14700063\, X^5 + 25848282\, X^4 + \\
& 15732085\, X^3 + 32131117\, X^2 + 21277564\, X + 78289) \cdot \\
& (X^{10} + 32382849\, X^9 + 21713401\, X^8 + 6783494\, X^7 + 33190934\, X^6 + \\
& 23972988\, X^5 + 31889065\, X^4 + 24039922\, X^3 + 29856003\, X^2 + 157980\, X + 27237978) \\
D = {} & (X + 17846191) \cdot \\
& (X^8 + 9691592\, X^7 + 14485138\, X^6 + 4075965\, X^5 + 1961487\, X^4 + \\
& 24681391\, X^3 + 6776957\, X^2 + 1299019\, X + 11459785) \\
& (X^9 + 32278483\, X^8 + 30661126\, X^7 + 15184481\, X^6 + 20998396\, X^5 + \\
& 30331331\, X^4 + 27365616\, X^3 + 25918194\, X^2 + 6196764\, X + 9404017) \\
& (X^{10} + 15736682\, X^9 + 24209918\, X^8 + 30399492\, X^7 + 13166125\, X^6 + \\
& 26074972\, X^5 + 1017876\, X^4 + 30123586\, X^3 + 24364998\, X^2 + 1124788\, X + 29766474)
\end{aligned}$$

Once, this is done, we use the descent procedure to express each factor using polynomials of lower degree in $X$ and $Y$. The slowest step in the descent is again the final step that expresses polynomials of degree 2 using linear polynomials. The total time to obtain all the logarithms on the laptop used for computing the relations is 11h20m hours.

Finally, back-substituting all the logarithms of the linear polynomials, we derive the logarithm of $Z$ modulo $p_1\, p_2$. To ease verification, we have also computed this logarithm modulo the small factors and thus give its complete value. We have:

$$\begin{aligned}
\log(Z) = {} & 386967279548486723402519963435606166899215654120310832592175430644903 14 \\
& 4740888395412686847662351430377499473537441208379213189393975471631517 4 \\
& 2484402992712936576072418509912503645350441229949735760120052465348429 7 \\
& 5781768790479781940290633966729576526948305287896083304119396966202700 0 \\
& 5822826745522861468256786676456002493610548297529063200082205245659542 2 \\
& 7246144528633360702659845991018671162540834330782804384739924956552212 0202
\end{aligned}$$

## 7 Conclusion and Open problem

In this paper, we have shown a new technique to replace sieving in some index calculus algorithms. Since we only know how to do this in the medium prime case of the function field size, it leaves open the problem of generalizing the approach to other index calculus algorithms. The natural targets are the function field size without a medium-size subfield and the number field sieve, either for factoring or computing discrete logarithms. It should be noted that the cubic sieve introduced in [3] (see [4] for more details) can be seen as a precursor of the method described in this paper and offers a partial answer to this question. However, generalizing it to index calculus with higher degree polynomials seems to be a difficult problem.

**Acknowledgements**

The author would like to thank the organizers of WMC and SCC 2012 for setting up such a nice and inspiring workshop.

We acknowledge that the results in this paper have been achieved using the PRACE Research Infrastructure resource Curie based in France at TGCC, Bruyères-le-Chatel (project number 2011050868).

## References

1. Leonard M. Adleman and Ming-Deh A. Huang. Function field sieve method for discrete logarithms over finite fields. In *Information and Computation*, volume 151, pages 5–16. Academic Press, 1999.
2. Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE transactions on information theory*, IT-30(4):587–594, July 1984.
3. Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.
4. Abhijit Das and C. E. Veni Madhavan. On the cubic sieve method for computing discrete logarithms over prime fields. *Int. J. Comput. Math.*, 82(12):1481–1495, 2005.
5. Claus Diem. The GHS attack in odd characteristic. *J. Ramanujan Math. Soc.*, 18(1):1–32, 2003.
6. Pierrick Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In *Advances in cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Comput. Sci.*, pages 19–34. Springer, 2000.
7. Pierrick Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symbolic Computation*, 2008.
8. Pierrick Gaudry, Florian Hess, and Nigel P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.
9. Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault, and Claus Diem. A double large prime variation for small genus hyperelliptic index calculus. *Mathematics of Computation*, 76:475–492, 2007.
10. Daniel M. Gordon. Discrete logarithms in GF($p$) using the number field sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.
11. Takuya Hayashi, Takeshi Shimoyama, Naoyuki Shinohara, and Tsuyoshi Takagi. Breaking pairing-based cryptosystems using $\eta_T$ pairing over $\mathbb{F}_{3^{97}}$. In *ASIACRYPT'2012*, pages 43–60, 2012.
12. Antoine Joux and Reynald Lercier. Discrete logarithms in GF($370\,801^{30}$). NMBRTHRY list, November 2005.
13. Antoine Joux and Reynald Lercier. The function field sieve in the medium prime case. In Serge Vaudenay, editor, *EUROCRYPT'2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2006.

14. Arjen K. Lenstra and Hendrick W. Lenstra, Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer–Verlag, 1993.
15. Brian A. Murphy. *Polynomial selection for the number field sieve integer factorisation algorithm*. PhD thesis, Australian national university, 1999.
16. Daniel Panario, Xavier Gourdon, and Philippe 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.
17. Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block wiedemann algorithm. *J. Symb. Comput.*, 33(5):757–775, 2002.