Technical history of discrete logarithms in small characteristic finite fields

The road from subexponential to quasi-polynomial complexity

Antoine Joux[1,2,3] and Cécile Pierrot[1,4]

[1]Sorbonne Universités, UPMC Université Paris 6, UMR 7606, LIP6, F-75005, Paris, France
[2]CryptoExperts
[3]Chaire de Cryptologie, Fondation UPMC
[4]CNRS and DGA

Abstract

Due to its use in cryptographic protocols such as the Diffie–Hellman key exchange, the discrete logarithm problem attracted a considerable amount of attention in the past 40 years. In this paper, we summarize the key technical ideas and their evolution for the case of discrete logarithms in small characteristic finite fields. This road leads from the original belief that this problem was hard enough for cryptographic purpose to the current state of the art where the algorithms are so efficient and practical that the problem can no longer be considered for cryptographic use.

Keywords: Cryptography, Discrete logarithms, Finite fields.

1 Introduction

The discrete logarithm is, together with integer factorization, one of the main candidate hard problems used to construct public key cryptosystems. Its cryptographic use was started by Diffie and Hellman in their seminal paper [10] where they introduced the notion of public-key cryptography. This problem can be stated in the following way: let G be a finite cyclic group (usually denoted multiplicatively) of order N and g a generator of G. Since we have computations in mind, we assume the availability of efficient algorithms to compute in G. In particular, multiplication of elements should be efficient. We now consider the map that sends an integer n to g^n in G. This map can be efficiently computed with $O(\log n)$ multiplications using the *square-and-multiply* algorithm. In the sequel, we call it *discrete exponentiation*. Discrete exponentiation is a periodic map of period N, surjective onto G. In fact, it defines an isomorphism Φ between G and the additive group $(\mathbb{Z}/N\mathbb{Z}, +)$. The inverse isomorphism Φ^{-1} is called the *discrete logarithm* in basis g. It maps g^n to the class of n modulo N.

Depending on the algorithmic description of the group G, computing discrete logarithms might be very easy or much more difficult. On the one hand, if we choose for G the group $(\mathbb{Z}/N\mathbb{Z}, +)$ itself, computing discrete logarithms is a trivial matter. On the other hand, there exist hardness results in the generic group model [30] which show that to compute discrete logarithms efficiently, it is essential to understand and make use of the algorithmic description of G. In generic groups, computing a discrete logarithm costs at

The final publication is available at Springer via http://dx.doi.org/10.1007/s10623-015-0147-6

Figure 1: History of discrete logarithm records in small characteristic finite fields

Date	Field	Bitsize	Algorithm	Authors
1984	2^{127}	127	[9]	Coppersmith
1991	2^{401}	401	[9, 12]	Gordon, McCurley
2001/09	2^{521}	521	[20]	Joux, Lercier
2002/02	2^{607}	607	[9]	Thomé
2005/09	2^{613}	613	[20]	Joux, Lercier
2012/06	$3^{6\cdot 97}$	923	[21]	Hayashi, Shimoyama, Shinohara, Takagi [29]
2013/02	2^{1778}	1778	[19]	Joux
2013/02	2^{1778}	1991	[11]	Göloglu, Granger, McGuire, Zumbrägel
2013/03	2^{4080}	4080	[19]	Joux
2013/04	2^{809}	809	[4, 21]	The Caramel Group [5]
2013/04	2^{6120}	6120	[11, 19]	Göloglu, Granger, McGuire, Zumbrägel
2013/05	2^{6168}	6168	[19]	Joux
2014/01	$3^{6\cdot 137}$	1303	[2, 11, 19]	Adj, Menezes, Oliveira, Rodríguez-Henríquez
2014/01	2^{9234}	9234	[11, 13, 14, 19]	Granger, Kleinjung, Zumbrägel
2014/09	$3^{5\cdot 479}$	3796	[19, 23]	Joux, Pierrot
2014/10	2^{1279}	1279	[11, 13, 14, 19]	Kleinjung

least $\Omega(\sqrt{p})$ operations, where p is the largest prime factor of N. This is much faster that exhaustively trying all possible values from 0 to $N-1$, but remains completely infeasible even for moderately large groups.

For cryptographic use, the two main types of groups that are usually considered are subgroups of either the multiplicative group of a finite field or the group of points of an elliptic curve defined over a finite field. For elliptic curves, there are no known algorithms that outperform generic algorithms, except in a few specific cases. However, for finite fields, much faster algorithms do exist. For a group of size N, the complexity of these faster algorithms is often of the form:

$$L_N(\alpha, c) = \exp\left((c + o(1))(\log N)^{\alpha}(\log\log N)^{1-\alpha}\right),$$

where α and c are two parameters such that $0 < \alpha < 1$ and $c > 0$. We also encounter the notation $L_N(\alpha)$ when specifying c is undesired. This shortcut is used because the most important parameter is the first one, which governs the transition from an exponential time algorithm to a polynomial time one. Indeed, when α becomes 1, $L_N(\alpha)$ is exponential in $\log N$ and on the other hand, if α tends to 0, $L_N(\alpha)$ becomes polynomial in $\log N$.

The first subexponential algorithms for discrete logarithms in finite fields had complexity $L_N(1/2)$. Initially, they only applied to prime fields. They were generalized to fixed characteristic fields, then improved to $L_N(1/3)$ for prime and fixed characteristic fields. By 2006, $L_N(1/3)$ complexity was achieved for all finite fields. In 2013, the situation changed for small characteristic finite fields and much faster algorithms appeared for this case. This change removed the small finite field discrete logarithm problem from the list of cryptographically usable problems. Note that most of these index calculus have been designed with practical efficiency in mind[1] and that the stated performances can be achieved only under a variety of heuristic assumptions. Thus, unless the contrary is explicitly stated, the reader should assumed that all the algorithms mentioned in the present paper rely on such heuristic assumptions.

In this paper, we survey the key technical ideas behind discrete logarithms in small characteristic finite fields and show the main evolutions that led to the state-of-the-art algorithms. It would be beyond the scope of the present paper to present all the techniques and refinements that have recently appeared for small characteristic fields [1, 2, 13, 14, 15]. Alternatively, the reader interested in a more general survey about

[1] This is illustrated by large-sized computations performed by many research groups in the past decades (see Figure 1).

discrete logarithms can refer to [22]. For simplicity of exposition, we focus on algorithms that work using polynomials over finite fields. In particular, we avoid all descriptions of the function field sieve algorithm of Adleman–Huang [4]. Instead, we present a variation on this algorithm based on bivariate polynomials [21].

2 Preliminaries and notations

Throughout the article, q denotes a prime power. For simplicity of exposition, we consider that q is fixed and that we wish to solve the discrete logarithm in $\mathbb{F}_{q^k}$ for some large extension degree k. As a consequence, all complexities are expressed as functions of k and operations in the basefield $\mathbb{F}_q$ are considered to have unit cost.

Furthermore, $\mathbb{F}_q[X]$ denotes the univariate polynomials in X with coefficients in $\mathbb{F}_q$. We recall that the finite field $\mathbb{F}_{q^k}^*$ can be represented as $\mathbb{F}_q[X]/(I_k(X))$ for any irreducible polynomial I_k of degree k, that is, elements of the finite field can be represented as polynomials reduced modulo $I_k(X)$. Since there is a unique finite field with q^k elements, all these representations are isomorphic. However, some of them are much more convenient for discrete logarithm computations and making good choices for I_k is an important consideration in this context.

Factoring polynomials over finite fields. It is well-known that any monic polynomial $F(X)$ in $\mathbb{F}_q[X]$ can be written as a product:

$$F(X) = \prod_{i=1}^{t} F_i(X)^{e_i},$$

where each F_i is a monic irreducible polynomial and $e_i \geq 1$ is its multiplicity. Moreover, up to the order of factors, this factorization into irreducibles is unique. For non-necessarily monic polynomials, we obtain a similar decomposition:

$$F(X) = \beta \cdot \prod_{i=1}^{t} F_i(X)^{e_i}$$

by factoring out the leading coefficient β of F.

Decomposing polynomials into a product of irreducibles is one of the key tools in discrete logarithm computations. Such a decomposition is especially useful when all the irreducibles that appear have low degree. To quantify this notion, we say that a polynomial $F(X)$ is ℓ-smooth when all the irreducibles in its decomposition have degree at most ℓ. This property is easy to test, due to the existence of fast algorithms for factoring univariate polynomials over finite fields [31].

Number of smooth polynomials and irreducibles in a finite field. It is also useful to consider a combinatorial point of view and count the number of irreducible and smooth polynomials. To be more precise, let us recall a major theorem used in order to estimate the probability of smoothness of polynomials. This result is due to Panario, Gourdon and Flajolet who generalized in 1998 the probability of smoothness of integers given in [8] to the probability of smoothness of polynomials in finite fields.

Estimate 1 (Panario, Gourdon and Flajolet [27])**.** *The probability for a random polynomial of degree less than n to be m-smooth is:*

$$(n/m)^{-(n/m)+o(1)}.$$

We do not give the range of validity of the ratio n/m as in the theorem, yet it covers all our cases. Besides, the number $N_q(n)$ of irreducible polynomials of degree n in a finite field of cardinality q is such that $N_q(n) = n^{-1} \sum_{d|n} \mu(d) q^{n/d}$ where μ is the Möbius function [25]. Hence we have the following result:

Lemma 2. *If $N_q(n)$ denotes the number of irreducible polynomials of degree n in a finite field of cardinality q then $q^n/n - \alpha \leq N_q(n) \leq q^n/n$, where $\alpha = q^{\lfloor n/2 \rfloor + 1}/n$.*

Proof. We know from [25] that:

$$n\,N_q(n) + \sum_{d|n, d\neq n} d\,N_q(d) = q^n. \tag{1}$$

This directly yields the upper bound $N_q(n) \leq q^n/n$.

Back-substituting in (1), we deduce that:

$$q^n \leq n\,N_q(n) + \sum_{d|n, d\neq n} q^d \leq n\,N_q(n) + \sum_{i=1}^{\lfloor n/2 \rfloor} q^d.$$

Thus, we find the lower bound $N_q(n) \geq n^{-1}(q^n - q^{\lfloor n/2 \rfloor + 1}/(q-1))$ and the result follows. □

Moving between different representations of the same field. Concrete discrete logarithm challenges in finite fields start from an explicit description of $\mathbb{F}_{q^k}$ as an extension of the base field $\mathbb{F}_q$ using a fixed irreducible polynomial I of degree k over $\mathbb{F}_q[X]$. Then, the problem is to find the discrete logarithm of an element h, given as a polynomial reduced modulo I, in the basis defined by g, another polynomial of the same form. Yet, most discrete logarithm algorithms start by choosing their own, usually different representation of $\mathbb{F}_{q^k}$. For this purpose, they select another irreducible polynomial J of degree k over $\mathbb{F}_q[X]$.

To shift the initial discrete logarithm problem from the original representation to the working representation we proceed as follows. Let β denote a root of J and consider I as a polynomial with coefficients in $\mathbb{F}_{q^k} = \mathbb{F}_q(\beta)$ and factorize it as a polynomial over this larger field. We know that I completely splits into linear terms and we thus express all of its roots as polynomials in β. Arbitrarily choose one of them and call it α. We are now able to write $\alpha = f(\beta)$ where f is a polynomial expression with coefficients in $\mathbb{F}_q$. In the initial problem, h and g are given as polynomials in α, say $H(\alpha)$ and $G(\alpha)$. In the new representation, it now suffices to find the discrete logarithm of $H(f(\beta))$ in basis $G(f(\beta))$.

On the multiplicative generator. One side issue related to the computation of discrete logarithms is to find a primitive element in a finite field $\mathbb{F}_{q^k}$, i.e. a multiplicative generator of the group $\mathbb{F}_{q^k}^*$. This might be unessential when computing discrete logarithms since we may assume that a generator g is given. Yet, it is important when choosing a group for cryptographic purposes. A commonly encountered method is to take random elements and verify whether their orders are strict divisors of $q^k - 1$ or equal to $q^k - 1$. This is easy to check assuming that the factorization of $q^k - 1$ is known. However, this factorization is hard to compute when q^k is large. Interestingly, the techniques to solve the discrete logarithm problem that we survey in the present paper have also been used in [17] to compute a multiplicative generator of $\mathbb{F}_{q^k}$ without knowledge of the factorization of $q^k - 1$.

3 Hellman–Reyneri algorithm

The algorithm of Hellman and Reyneri [16] is an adaptation to $\mathbb{F}_{q^k}$ of Adleman's index calculus algorithm [3] for computing discrete logarithms in prime fields. It allows us to give a simple introduction to the concepts used in index calculus method. In its original form, it is a heuristic algorithm, like all algorithms described in the present article. However, there exists a rigorous variation due to Pomerance [28] with essentially the same asymptotic complexity.

To use the Hellman–Reyneri algorithm, we first choose I_k, an arbitrary monic irreducible polynomial of degree k, in order to represent the finite field $\mathbb{F}_{q^k}$ by $\mathbb{F}_q[\alpha]$ where α denotes a (fixed) root of I_k. We also choose a multiplicative generator g of the multiplicative group $\mathbb{F}_{q^k}^*$. In addition, we fix an integer parameter ℓ with $1 < \ell < k$.

For an integer r selected uniformly at random in $[0, q^k - 1]$, we see that g^r is a uniformly random element of $\mathbb{F}_{q^k}^*$. This element can be represented by a polynomial $G_r(X)$ of degree at most $k-1$ which can be

efficiently computed by using a square-and-multiply exponentiation method. When this polynomial G_r is ℓ-smooth, we can write:

$$g^r = \beta_r \cdot \prod_{i=1}^{t_r} F_i^{(r)}(\alpha)^{e_i^{(r)}}, \tag{2}$$

where the $F_i^{(r)}$ are the irreducible monic polynomials of degree at most ℓ that appear in the factorization of G_r and β_r the leading coefficient of G_r. Taking logarithms in $\mathbb{F}_{q^k}$, we obtain an affine equation:

$$r = \log_g(\beta_r) + \sum_{i=1}^{t_r} e_i^{(r)} \log_g\left(F_i^{(r)}(\alpha)\right) \pmod{q^k - 1},$$

whose unknowns are formal logarithms of elements of the set:

$$\mathcal{F} = \{\beta | \beta \in \mathbb{F}_q\} \cup \{F(\alpha) | F \text{ monic irred. of degree at most } \ell \text{ in } \mathbb{F}_q[X]\}.$$

This set is usually called the *factor base*. Since $\mathcal{F}$ has cardinality smaller than $q^{\ell+1}$ and since the equations are randomly generated, we expect that after generating $q^{\ell+1}$ such equations, we obtain a system with a unique solution formed of the logarithms in basis g of the elements of $\mathcal{F}$. This expectation is precisely the reason why Hellman–Reyneri algorithm is heuristic.

Individual logarithms. Once these logarithms of elements of $\mathcal{F}$ are known, in order to find the logarithm of an arbitrary element h in $\mathbb{F}_{q^k}^*$, it suffices to find one extra equation of a form similar to Equation (2) but involving h. Typically, one can find an element $h \cdot g^r$ that factors into elements of $\mathcal{F}$ and deduce the logarithm of h by subtracting r from the sum of logarithms of the elements in the factorization. Thus, computing one individual logarithm in the finite field once the precomputation of logarithms of factor base elements has been done costs much less than this precomputation. For practical purposes, this is an extremely important property. For this reason, most discrete logarithm algorithms have a separate individual logarithm phase. Rigorous algorithms form a notable exception to this rule since these algorithms don't aim at being practically competitive and because having a separate individual logarithm phase would make a rigorous analysis much more complicated.

Note. In order to simplify exposition, the above presentation differs from the more traditional way of performing index calculus. Normally, the elements of $\mathbb{F}_q$ are not included in the factor base $\mathcal{F}$. Instead, they are simply disregarded in the equations. This is possible because these constant elements have order dividing $q-1$ which implies that their logarithms are multiples of $(q^k-1)/(q-1)$. Thus, removing them is possible if we restrict ourselves to computing logarithms modulo $(q^k-1)/(q-1)$. More generally, all small factors of $q^k - 1$ are usually removed from consideration when doing the linear algebra. There are two main reasons for this. First, small factors might cause problem due to the occurrence of non invertible elements during the linear algebra process. Second, thanks to the use of generic algorithms such as Pohlig–Hellman, Shanks' baby-step–giant-step and Pollard's Rho, recovering the missing part of the logarithm is an easy addition to the individual logarithm computation.

Complexity analysis. In order to analyze the complexity algorithm, since the individual logarithm phase is negligible compared to the precomputation, it suffices to express the cost of the collection of relations and linear algebra as a function of the parameter ℓ. Using our upper bound of $q^{\ell+1}$ on the cardinality of $\mathcal{F}$ and denoting by p_ℓ the probability for a random polynomial G_r to be ℓ-smooth, the expected running time of the collection of relations is upper bounded by $q^{\ell+1}/p$. To analyze the linear algebra, we first remark that the number of factor base elements in any multiplicative relation we form is upper bounded by k. This comes from the simple fact that the number of irreducible polynomials that appear when factor a polynomial is smaller than its degree. Thus the linear algebra is performed on a very sparse matrix containing at most k entries per line, and the cost of this phase is essentially quadratic in the size of $\mathcal{F}$ if we use sparse

linear algebra techniques such as Lanczos [24] or Wiedemann [32] algorithms. More precisely, the running time of the linear algebra is upper bounded by $k\,q^{2\ell+2}$.

From Estimate 1 we know that:

$$-\log_q p \leq \frac{k}{\ell} \cdot \log_q(k/\ell).$$

Thus, choosing our parameter as follows:

$$\ell = \left\lceil \sqrt{\frac{k \, \log_q k}{2}} \right\rceil,$$

yields an total runtime complexity of:

$$q^{\sqrt{(2+o(1))\, k \, \log_q k}} = L_{q^k}(1/2, \sqrt{2}).$$

4 Coppersmith's algorithm

In 1984, Coppersmith's algorithm [9] was the first to achieve complexity below $L_N(1/2)$ for discrete logarithms in finite fields. This new algorithm was illustrated with a record on the finite field $\mathbb{F}_{2^{127}}$. In its basic form, it works for fields of characteristic 2 but can easily be generalized [7] to any characteristic. The key idea compared to the Hellman–Reyneri algorithm is to use the freedom we have when representing $\mathbb{F}_{q^k}$ and to choose a representation that helps the construction of multiplicative relations.

In the original form proposed by Coppersmith, the irreducible that defines $\mathbb{F}_{q^k}$ is chosen to be of the form $x^k - S(x)$, where S is a polynomial of degree d_S as low as possible. Here, we replace this choice by a more recent construction from [19], which simplifies the presentation of the construction of multiplicative relations in Coppersmith's algorithm. In this section, we let n be the unique integer such that $q^{n-1} < k \leq q^n$ and choose a low degree polynomial $S(x)$ such that $x^{q^n} - S(x)$ has a single irreducible factor of degree k. We denote this factor by $I_k(x)$ and use it as a defining polynomial of $\mathbb{F}_{q^k}$.

We further write $n = n_1 + n_2$ for values of n_1 and n_2 to be determined during the complexity analysis. The key to the construction of relations is to note that for any pair of polynomials A and B with coefficients in $\mathbb{F}_q$, we have:

$$\left(A(x) + x^{q^{n_1}} B(x)\right)^{q^{n_2}} = A(x^{q^{n_2}}) + S(x)B(x^{q^{n_2}}) \pmod{I_k(x)}, \tag{3}$$

thanks to the linearity of raising to the power q and to the fact that $x^{q^n} = S(x) \pmod{I_k(x)}$.

Defining the factor basis $\mathcal{F}$ as in Section 3, we see that Equation (3) yields a multiplicative relation when both $A(x)+x^{q^{n_1}}B(x)$ and $A(x^{q^{n_2}})+S(x)B(x^{q^{n_2}})$ are ℓ-smooth. Here, it is traditional to say that this occurs whenever the product

$$\left(A(x) + x^{q^{n_1}} B(x)\right) \cdot \left(A(x^{q^{n_2}}) + S(x)B(x^{q^{n_2}})\right)$$

is ℓ-smooth and make the heuristic argument that this event happens which probability comparable to the smoothness of a random polynomial of the same degree.

To continue the analysis, we look at all the pairs of non-zero polynomials A and B of degree at most ℓ, with A being chosen monic. The reason for this last restriction is that multiplying A and B by any constant of $\mathbb{F}_q$ in fact yields the same equation. Under this condition on the degrees of A and B, we choose (n_1, n_2) in the way that minimizes the degree of the above product. Since the total degree is:

$$\ell + q^{n_1} + d_S + \frac{\ell \cdot q^n}{q^{n_1}},$$

it is minimized by making q^{n_1} as close as possible to $\sqrt{\ell\, q^n}$. With this choice, the degree asymptotically becomes $(2+o(1))\sqrt{\ell\, q^n}$. In terms of k, this varies between $(2+o(1))\sqrt{\ell\, k}$ and $(2\sqrt{q}+o(1))\sqrt{\ell\, k}$ depending on how close k is to the next power of q.

Finally, ℓ is chosen as in Section 3 by balancing the opposite of the logarithm of the probability of success with ℓ. This yields:

$$\ell = (4/3)^{1/3}\, q^{n/3}\, n^{2/3}.$$

Hence, the complexity becomes $L_{q^{q^n}}(1/3, (32/9)^{1/3})$. In terms of q^k, this varies between $L_{q^k}(1/3, (32/9)^{1/3})$ and $L_{q^k}(1/3, (32\,q/9)^{1/3})$ depending on how close k is to the next power of q. Note that, due to the simplified presentation we have chosen, the worst case is less good than with Coppersmith's original algorithm.

Notes on the linear algebra. Compared to the algorithm of Hellman and Reyneri, there are two main differences that occur during the computation of the linear algebra. More precisely, the type of equations that arise from relations as in (3) no longer contain a constant term and they include some large coefficients q^{n_2}. The consequence of the first difference is that the system has many solutions. Thankfully, any of these solutions (except the zero vector) gives logarithms, up to multiplication by a constant. Thus, if the multiplicative generator g appears in the factor base $\mathcal{F}$, it is easy to renormalize the vector we obtain to get logarithms in basis g. In the case where g is outside of $\mathcal{F}$, we need to use the individual logarithm process on g to obtain the correct renormalization constant. The large coefficients q^{n_2} impact the performance of matrix-vector multiplication and thus accordingly slow down the linear algebra process. However, since there is a single large constant, one multiplication per line of the matrix is enough instead of one multiplication per large entry. Moreover, in the frequently considered case where $q = 2$, the multiplication simply becomes a left shift by n_2 bits and does not really slow down the process.

Individual logarithms – Enters the descent. One difficulty with Coppersmith's algorithm is that the construction of an additional relation involving an arbitrary finite field element is no longer simple. Thus, an additional, extremely important idea becomes necessary: the notion of descent. In a nutshell, this process is a way to write a relation between a polynomial in α and a small number of other polynomials of lower degree. Iterating the method many times, it becomes possible to express everything in terms of the elements of $\mathcal{F}$. As a consequence, the computation of an individual logarithm is now represented as a tree where the descendants of a node (labeled by a polynomial) are the nodes labeled by the lower degree polynomials occurring in the corresponding relation. All the leaves of the tree should belong to $\mathcal{F}$. In order to determine the cost of the computation, we need to bound the number of nodes in the tree and the cost of searching for one relation between a node and its sons. This analysis is beyond the scope of the present survey, the important result being that this whole descent process has negligible cost compared to the initial computation of the logarithms of elements of $\mathcal{F}$.

To find a relation involving a polynomial $G(\alpha)$, which can be assumed to be irreducible without loss of generality, we proceed as follows :

- Choose values of n_1 and n_2, possibly different from the ones used initially and adapted to the degree of G, such that $n = n_1 + n_2$.
- Note that the set of polynomials (A, B) such that G divides $A(x) + x^{q^{n_1}} B(x)$ form a lattice. Compute a basis (A_1, B_1) and (A_2, B_2) of this lattice.
- For a pair of low degree polynomials (λ_1, λ_2), let $A = \lambda_1 A_1 + \lambda_2 A_2$ and $B = \lambda_1 B_1 + \lambda_2 B_2$ and consider the candidate relation:

$$\Big(A(x) + x^{q^{n_1}} B(x)\Big)^{q^{n_2}} = A(x^{q^{n_2}}) + S(x)B(x^{q^{n_2}}) \pmod{I_k(x)}.$$

- Among these candidates relations which all contain G, keep one such that the degrees of all the other irreducible polynomials involved are smaller that $\kappa \deg G$ for a constant $\kappa < 1$ (a typical choice would be $\kappa = 3/4$).

5 Bivariate polynomials

In order to improve on Coppersmith's algorithm, the next step is the function field sieve algorithm [4] proposed by Adleman and Huang in 1999. This algorithm was used for several record breaking computations as shown in Figure 1. We present a variation of this method that appeared in [21] and only makes use of polynomials. In particular, this variation was used in 2012 to compute discrete logarithms in a 923-bit field of characteristic 3 using about 900 000 CPU·hours [29]. The starting point of the algorithm is to define the extension field implicitly using two polynomial bivariate relations over $\mathbb{F}_q$:

$$y = f(x) \quad \text{and} \quad x = g(y).$$

Putting the two relations together, we obtain $x = g(f(x))$. Thus, if the polynomial $g(f(x)) - x$ has an irreducible factor $I_k(x)$ of degree k, we obtain a representation of $\mathbb{F}_{q^k}$. Under the constraint that the product of the degree of f and g is greater than k, we heuristically expect to easily find a satisfying pair of polynomials f and g. Once this is done, we let α denote a root in $\mathbb{F}_{q^k}$ of $I_k(x)$ and $\beta = f(\alpha)$. By construction, we also have $\alpha = g(\beta)$ in $\mathbb{F}_{q^k}$.

From this, given a pair of polynomials A and B, we can write:

$$\begin{aligned} A(\alpha) + \beta\, B(\alpha) &= A(\alpha) + f(\alpha)\, B(\alpha) \quad \text{and} \\ A(\alpha) + \beta\, B(\alpha) &= A(g(\beta)) + \beta\, B(g(\beta)). \end{aligned}$$

Putting together the right-hand sides, we have equality in $\mathbb{F}_{q^k}$ between a polynomial in α and a polynomial in β. Moreover, both polynomials have relatively low-degree if parameters are properly selected. As a consequence, if both polynomials factor into irreducible factors of degree at most ℓ, we obtain a multiplicative relation. Note that we have essentially doubled the size of the factor base since it now contains the evaluation of each irreducible polynomial both at α and at β.

To analyze the complexity, it suffices to optimize on a single parameter ℓ. This parameter is the degree bound on the irreducible of $\mathcal{F}$. The degrees of A and B are also fixed to be at most ℓ. In addition, the degrees of f and g are chosen to satisfy $\deg f \cdot \deg g \approx k$ and $\deg f / \deg g \approx \ell$. Once optimized, this yields a heuristic complexity of $L_{q^k}\left(1/3, (32/9)^{1/3}\right)$.

With this bivariate representation, the computation of individual logarithms also involves a descent process as in Section 4, whose cost is negligible compared to the precomputation of logarithms of elements of the factor base.

6 The paradigm shift: Constructed smooth polynomials

Despite their obvious differences, the algorithms of Sections 4 and 5 share a common principle. Both create medium degree polynomials that respectively appear on the two sides of a multiplicative relation and hope that these two polynomials simultaneously factor into low-degree irreducibles. The fundamental obstruction that arises here is that the degrees of the two medium-sized polynomials cannot be made arbitrarily small. The reason for this is the fact that the rule that relates these two polynomials encodes the representation of the finite field that we are using. Due to this, lowering the degree of one of the two medium-sized polynomial increases the degree of the other. As long as we rely on chance to obtain smooth polynomials, the best strategy is to balance the two degrees. In this setting, the bivariate algorithm is essentially the best we can do.

In order to escape this obstruction, a different approach is needed. A recently discovered option [19] that allows to break the $L_N(1/3)$ barrier works by writing an equation between a high-degree polynomial and a low-degree polynomial, where the high-degree polynomial is constructed to guarantee its smoothness. First announced in 2013, this yielded a $L_N(1/4)$ complexity, which was later improved to quasi-polynomial complexity in [6]. This idea of constructing smooth polynomials first appeared in a less efficient version in [18]. Independently of the $L_N(1/4)$ algorithm from [19] which directly construct smooth high degree polynomials,

[11] considers a family of high-degree polynomials containing many smooth polynomials. However, this option only leads to an $L_N(1/3)$ algorithm. At the present time, the largest field of small characteristic that was broken with this family of algorithms, called Frobenius Representation algorithms, is a Kummer extension field with a 9234-bit cardinality.

In order to construct smooth polynomials of high degree, one starts from the identity:

$$X^q - X = \prod_{\gamma \in \mathbb{F}_q} (X - \gamma). \tag{4}$$

For simplicity, we follow the approach of [23] and consider two non-zero polynomials A and B with coefficients in $\mathbb{F}_q$. We can write:

$$\begin{aligned} A(X)^q\, B(X) - A(X)\, B(X)^q &= B(X)^{q+1} \left(\left(\frac{A(X)}{B(X)} \right)^q - \frac{A(X)}{B(X)} \right) \\ &= B(X)^{q+1} \prod_{\gamma \in \mathbb{F}_q} \left(\frac{A(X)}{B(X)} - \gamma \right) \\ &= B(X) \prod_{\gamma \in \mathbb{F}_q} \left(A(X) - \gamma\, B(X) \right). \end{aligned}$$

We see that $A(X)^q\, B(X) - A(X)\, B(X)^q$ is constructed to be a product of polynomials of degree at most equal to the maximum of the degrees of A and B. In order to use this polynomial in a multiplicative relation, we need to relate it to a low degree polynomial in the finite field. For this purpose, we first use the linearity of the Frobenius map and write:

$$A(X)^q\, B(X) - A(X)\, B(X)^q = A(X^q)\, B(X) - A(X)\, B(X^q).$$

Remembering Coppersmith's algorithm, it is natural to construct the finite field $\mathbb{F}_{q^k}$ by searching for a low-degree polynomial S such that $X^q - S(X)$ has an irreducible factor I_k of degree k. Once again, this requires $q \geq k$. Letting α denote a root of I_k and putting everything together, we can now write:

$$B(\alpha) \prod_{\gamma \in \mathbb{F}_q} (A(\alpha) - \gamma\, B(\alpha)) = A(S(\alpha))\, B(\alpha) - A(\alpha)\, B(S(\alpha)), \tag{5}$$

with the constructed smooth polynomial on the left and a low-degree polynomial on the right. More precisely, if A and B have degree at most ℓ, the degree of the polynomial on the right can be upper bounded by $\ell\,(\deg S + 1)$.

As usual, the parameter ℓ dictates the size of the factor base. The fundamental difference is that, in this new context, ℓ can be chosen to be a small constant. As a consequence, the precomputation of the logarithms of the elements of the factor base becomes polynomial time.

By contrast with the previous algorithm, the computation of individual discrete logarithms becomes the asymptotically limiting phase. Its complexity is quasi-polynomial, thanks to [6] or [14], more precisely of the form $q^{O(\log(\max(q,k)))}$ for an arbitrary finite field $\mathbb{F}_{q^k}$. This is worse than any polynomial but incredibly better than subexponential $L_N(1/3)$ complexity.

Note. To add flexibility, in most papers, instead of choosing I_k as a divisor of $X^q - S(X)$, it is chosen as a divisor of $S_1(X)X^q - S_0(X)$. This means that the right-hand side in Equation (5) becomes:

$$A\left(\frac{S_0(\alpha)}{S_1(\alpha)} \right) B(\alpha) - A(\alpha)\, B\left(\frac{S_0(\alpha)}{S_1(\alpha)} \right),$$

a rational fraction, whose denominator is fixed and whose numerator has low-degree.

7 Current state and open problems

After a few very active years, the state of discrete logarithms in small characteristic finite fields has been more or less stabilized. From a practical point of view, the complexity of the polynomial time precomputation of the logarithms of factor base elements has been reduced [23] to $O(q^6)$ for finite fields of the form $\mathbb{F}_{q^k}$ with $k < 2q$. From a theoretical point of view, the most recent progress appears in [15] and proposes a method to reduce the heuristic hypotheses required : it would now suffice to prove the existence of a pair of polynomials (S_0, S_1) such that $S_1(X)X^q - S_0(X)$ has an irreducible factor of degree k. This final, challenging question remains open, despite the fact that in practical computations, finding (S_0, S_1) is easy and efficient.

Another problem would be to find a truly polynomial time algorithm rather than a quasi-polynomial one. With the current techniques for the individual logarithm part, this does not seem possible. This is due to the fact that each inner node of the tree involved during the descent process has $O(q)$ sons. Since the height of the tree is logarithmic in k, the total number of leaves is already more that polynomial in the size of the field. More precisely, for $k \approx q$, the number of leaves is $O(q^{O(\log q)})$, thus dominating any polynomial in q.

Finally, the hardest and possibly most interesting open problem would be to somehow generalize these algorithms to the case of larger characteristic fields, the goal being to beat the apparent $L_N(1/3)$ barrier for large characteristic. Unfortunately, the ideas based on the use of polynomials and change of variables seem to be unusable in this context and different techniques would be required. Related issues are the possibility of finding improved algorithms for the factorization of large integers or for discrete logarithms on elliptic curve. This last question is discussed in [26].

References

[1] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodriguez-Henriquez. Weakness of $\mathbb{F}_{3^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ 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. Computing discrete logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ using Magma. In *Arithmetic of Finite Fields: WAIFI'2014*, pages 3–22, 2014.

[3] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science: FOCS'79*, pages 55–60, 1979.

[4] 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.

[5] Razvan Barbulescu, Cyril Bouvier, Jérémie Detrey, Pierrick Gaudry, Hamza Jeljeli, Emmanuel Thomé, Marion Videau, and Paul Zimmermann. Discrete logarithm in $GF(2^{809})$ with FFS. In *Public-Key Cryptography, PKC 2014*, pages 221–238, 2014.

[6] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasipolynomial algorithm for discrete logarithm in finite fields of small characteristic. In *Advances in Cryptology: EUROCRYPT 2014*, pages 1–16, 2014.

[7] Ian F. Blake, Ronald C. Mullin, and Scott A. Vanstone. Computing logarithms in GF(2^n). In *Advances in Cryptology, CRYPTO'84*, pages 73–82, 1984.

[8] E.R Canfield, Paul Erdös, and Carl Pomerance. On a problem of Oppenheim concerning factorisatio numerorum. In *Journal of Number Theory*, volume 17, pages 1–28, 1983.

[9] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–593, 1984.

[10] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[11] Faruk Göloglu, Robert Granger, Gary McGuire, and Jens Zumbrägel. On the function field sieve and the impact of higher splitting probabilities. In *Advances in Cryptology: CRYPTO'2013*, pages 109–128, 2013.

[12] Daniel M. Gordon and Kevin S. McCurley. Massively parallel computation of discrete logarithms. In *Advances in Cryptology: CRYPTO'92*, pages 312–323, 1992.

[13] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. Breaking "128-bit secure" supersingular binary curves (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In *Advances in Cryptology: CRYPTO'2014 (Part II)*, pages 126–145, 2014.

[14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. On the powers of 2. Cryptology ePrint Archive, Report 2014/300, 2014. `http://eprint.iacr.org/`.

[15] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. On the discrete logarithm problem in finite fields of fixed characteristic. Cryptology ePrint Archive, Report 2015/685, 2015. `http://eprint.iacr.org/`.

[16] Martin E. Hellman and Justin M. Reyneri. Fast computation of discrete logarithms in GF(q). In *Advances in Cryptology: CRYPTO'82*, pages 3–13, 1982.

[17] Ming-Deh Huang and Anand Kumar Narayanan. Finding primitive elements in finite fields of small characteristic. *CoRR*, abs/1304.1206, 2013.

[18] Antoine Joux. Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields. In *Advances in Cryptology: EUROCRYPT'2013*, pages 177–193, 2013.

[19] Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in small characteristic. In *Selected Areas in Cryptography, SAC 2013*, pages 355–379, 2013.

[20] Antoine Joux and Reynald Lercier. The function field sieve is quite special. In *ANTS*, pages 431–445, 2002.

[21] Antoine Joux and Reynald Lercier. The function field sieve in the medium prime case. In *Advances in Cryptology: EUROCRYPT'2006*, pages 254–270, 2006.

[22] Antoine Joux, Andrew Odlyzko, and Cécile Pierrot. The past, evolving present, and future of the discrete logarithm. In Çetin Kaya Koç, editor, *Open Problems in Mathematics and Computational Science*, pages 5–36. Springer International Publishing, 2014.

[23] Antoine Joux and Cécile Pierrot. Improving the polynomial time precomputation of frobenius representation discrete logarithm algorithms. In *Advances in Cryptology: ASIACRYPT'2014*, pages 378–397, 2014.

[24] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of National Bureau of Standards*, 45(4), 1950.

[25] Rudolf Lidl and Harald Niederreiter. *Finite fields*. Encyclopaedia of mathematics and its applications. Cambridge University Press, New York, 1997.

[26] Maike Massierer. Some experiments investigating a possible $L(1/4)$ algorithm for the discrete logarithm problem in algebraic curves. Cryptology ePrint Archive, Report 2014/996, 2014. `http://eprint.iacr.org/`.

[27] Daniel Panario, Xavier Gourdon, and Philippe Flajolet. An analytic approach to smooth polynomials over finite fields. In *ANTS*, pages 226–236, 1998.

[28] Carl Pomerance. Fast, rigorous factorization and discrete logarithm algorithms. In *Discrete algorithms and complexity*, pages 119–143, 1987.

[29] Naoyuki Shinohara, Takeshi Shimoyama, Takuya Hayashi, and Tsuyoshi Takagi. Key length estimation of pairing-based cryptosystems using eta pairing over $GF(3^n)$. *IEICE Transactions*, 97-A(1):236–244, 2014.

[30] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[31] Joachim von zur Gathen and Daniel Panario. Factoring polynomials over finite fields: A survey. *Journal of Symbolic Computation*, 31(12):3 – 17, 2001.

[32] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986.