On the implementation of the new quasi-polynomial discrete logarithm solving algorithms

Fabrice Mouhartem[†] — **Supervisor:** Fréderik Vercauteren[‡]

[†]ÉNS de Lyon, France
[‡]Katholieke Universiteit Leuven, Belgium

August 29, 2014

Abstract

In this Master 1 internship report we analyse quasi-polynomial algorithms to solve the discrete logarithm problem in small characteristic. We will focus on the difficulties one encounters in the implementation of these algorithms and give some advices to make their use possible for practical implementation.

Keywords: Cryptology, Discrete Logarithm Problem, Implementation, Quasi-polynomial algorithms.

Contents

1 Introduction

In cryptology, the main idea is to use the intractability of a problem to prove the security of a scheme. For example the Diffie-Hellman key exchange [10] bases its security on the computational Diffie-Hellman (**CDH**) problem which is to compute g^{ab} given (g, g^a, g^b) in a group G. This problem is weaker than the discrete logarithm problem (**DLP**) which is the problem of computing x given (g, g^x) in an group G, because one can compute the discrete logarithm of g^a which is a and then exponentiate g^b to get g^{ab} and then solve CDH.

The DLP, as well as the factorisation problem, plays an important role in public key cryptography and the hardness of this problem is used as an assumption in many protocols like key exchange [10] or signature scheme [11]. A general purpose algorithm [4, 9, 14, 5, 20, 26] has been developed with complexity $L(1/3)$ where $L_N(\alpha) = \exp\left(\mathcal{O}\left((\log N)^{\alpha}(\log\log N)^{1-\alpha}\right)\right)$ which is subexponential in terms of N the finite field size ($L_N(1)$ denotes a function exponential in $\log N$ and $L_N(0)$ a polynomial function in $\log N$ the size of N).

Recently, some theoretical breakthroughs have been made, focused on small and medium characteristic finite fields [6, 17], resulting with quasi polynomial algorithms in those cases. These new algorithms are improvements over the function field sieve (**FFS**) algorithm [5] and are based on a special representation of the field called *sparse medium subfield representation* which we will define and explain later. The impact of these algorithms is a complete break of the type 1 pairings over elliptic or hyperelliptic curves [12].

However, far too little attention has been paid to practical implementation of these algorithms. Therefore we don't have a full understanding of the implication of those new techniques in real-world cryptography. The practical implementation of an algorithm gives us more information about its efficiency and a better understanding on how it works. In this report we will examine and explain some of the practical issues one will encounter during the implementation of these algorithms, and the influence of the parameter ranges onto these algorithms.

The implementation has been made with `Magma` computer algebra system [7]. The motivation of this choice is that it has many built-in function that allow us to implement the algorithm easily and the algorithms used are quite well-chosen, for instance the discrete logarithm in small characteristic is computed using the Coppersmith algorithm [9] which was the fastest algorithm for the DLP in small characteristic in 2001 according to Menezes, Van Oorschot, and Vanstone [22]. This allows us to compare the performances of the existing implementations with the state of the art results.

The rest of the report will be organized as follows: we will first define our notations and explain the FFS algorithm and the two new quasi-polynomial algorithms then an analysis of the parameter ranges will be performed. Then the results we get from our experiments will be shown. Finally we will conclude.

2 The different FFS algorithms

In this section we will focus on explaining the state of the art algorithms to solve the DLP, in particular the Barbulescu, Gaudry, Joux, and Thomé algorithm [6] and the Granger, Kleinjung, and Zumbrägel algorithm [17] respectively named **BGJT** and **GKZ** algorithms after the name of their authors.

The first subsection will present the FFS algorithm and introduce the notations that will be used, then the BGJT will be explained and finally the ideas developed in the GKZ algorithm

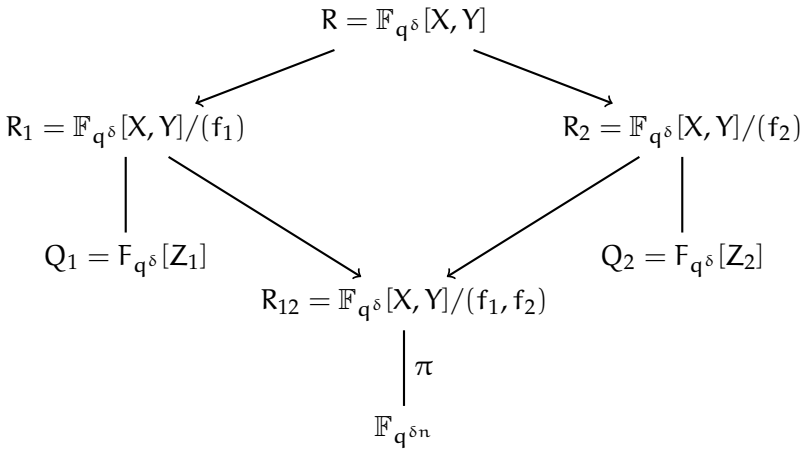


Figure 2: Setup for the FFS

will be described.

2.1 Overview of the function field sieve

In the following we denote by p the characteristic of the field F_{p^l} the finite field in which the DLP will be solved.

Let $\delta \geqslant 1$, and let $n \geqslant 1$ such that the DLP can be embedded into $\mathbb{F}_{q^{\delta n}}$, i.e. $p^l | q^{\delta n}$. We will represent elements in $\mathbb{F}_{q^{\delta n}}$ as polynomials in $R = \mathbb{F}_{q^\delta}[X,Y]$. In order to obtain a finite field $\mathbb{F}_{q^{\delta n}}$, we extend the field twice using the polynomials f_1 and f_2 and embed the resulting field R_{12} in $\mathbb{F}_{q^{\delta n}}$ through the surjective ring homomorphism π as shown in the commutative diagram in figure 2.

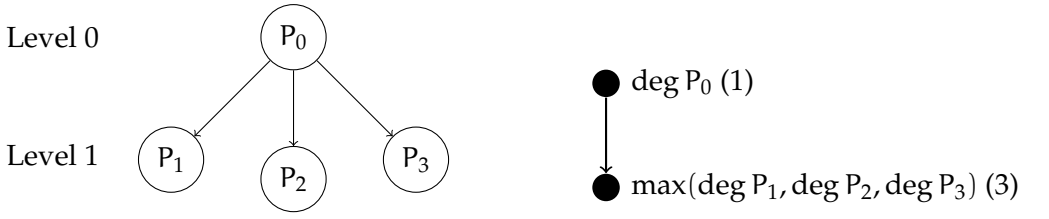
The main idea of the algorithm is to generate multiplicative relations between elements in $\mathbb{F}_{q^{\delta n}}$, in order to relate the logarithm of an element P in R in terms of "easier" elements. For instance considering the elements of $\mathbb{F}_{q^{\delta n}}$ as elements in $Q_1 = \mathbb{F}_{q^\delta}[Z_1]$ lead us to take for "easy" elements the low degree polynomials in Q_1, and our goal is then to relate the logarithm of a degree d polynomial in Z_1 into degree $d' < d$ polynomials in Z_1. This step is called the *descent phase*. The easy polynomials are called the *factor basis*, the logarithm of which is computed in a first phase. These are the two main phases of the function field sieve.

To illustrate this, let assume that we have the logarithm of linear polynomials in a given finite field $\mathcal{F}$, obtained from the factor basis solving phase. Let P be an univariate polynomial of degree 4 viewed as an element in $\mathcal{F}$ that can be expressed as follows: $P = (X^2 + aX + b)^\alpha (X + c)^\beta (X + d)^\gamma$ as the result of one descent step, and $(X^2 + aX + b) = (X + e)^\delta (X + f)^\varepsilon$ as the result of another descent step. From this two descents we can express $P = [(X + e)^\delta (X + f)^\varepsilon]^\alpha (X + c)^\beta (X + d)^\gamma$, which means that $\log P = \alpha[\delta \log(X+e) + \varepsilon \log(X+f)] + \beta \log(X+c) + \gamma \log(X+d)$. As all the $\{\log(X + a)\}_a$ are known, we can then compute the logarithm of P.

In the classical FFS [5], the relations are generated by factoring P in R_1 and in R_2 and then map this decomposition into $R_{12} \equiv \mathbb{F}_{q^{\delta n}}$. This leads to a heuristic running time of $L(1/3)$.

In the recent variation of the algorithm, we take $f_1 = Y - X^q$, which gives a linear relation between X and Y.

As a consequence of the above, a relation like $P(X) = \prod P_i^{\alpha_i}(Y)$ can be interpreted as a relation between $P(X)$ and $\bar{P}_i(X)$ where $\bar{P}_i$ is the polynomial P_i with its coefficients raised to the power $q^{\delta-1}$ using the linearity of the map $X \mapsto X^{1/q}$. We will keep this notation in the



(a) Classical descent tree example

(b) Adj. et al descent tree representation

Figure 3

following.

The other change is to take f_2 to be of degree 1 in X or Y. This means that $f_2 = h_1(Y)X - h_0(Y)$ or $f_2 = h_1(X)Y - h_0(X)$ respectively. Then a polynomial in $R_1 \equiv \mathbb{F}_{q^\delta}[X]$ can be expressed as a polynomial of low degree in $\frac{1}{h_1}$ and X or Y. Using Joux's special-q method, we end up with an heuristic running time of $L(1/4 + o(1))$ [19].

Remark 1. The choice of f_2 to be linear in X is common in the implementation of such algorithms, because extension degree of (f_1, f_2) is then at most $q \max(\deg h_0, 1 + \deg h_1)$ [16] when the choice of f_2 to be linear in Y leave us with a degree at most $q + \deg h_1$ for the extension. However we will use both, because using f_2 to be linear in Y allows us to represent all our polynomial in X, which is more convenient for our calculus (otherwise we have to use the map $X \mapsto X^q$ on the polynomials to express them in term of the right variable).

Definition 1 (Sparse medium representation)**.** A field admitting a representation as an extension of F_{q^δ} through $f_1 = Y - X^q$ and $f_2 | h_1(X)Y - h_0(X)$ (respectively $f_2 | h_1(Y)X - h_0(Y)$) with bounded degree for h_0 and h_1 (usually $\deg h_0, \deg h_1 \leqslant 2$) is said to have a *sparse medium representation*.

Remark 2. The sparse matrix representation is less restrictive than the definition we gave above. But it involves some problems called "traps" [8, 17] because of the use of relations living in $h_1(X)Y - h_0(X)$ (respectively $h_1(Y)X - h_0(Y)$) expressed modulo f_2. We will explain these problems later when we meet then.

A final remark is that the descent phase is an important one, and one can evaluate the efficiency of a descent method by counting the number of nodes the descent tree has. For instance the polynomial we obtain from the relation $P_0 = P_1P_2P_3$ will be the one displayed in figure 3a. In their paper, Adj et al. use another representation [1, 2] to synthesize the drawing of the tree, as depicted in figure 3b. As the important information is the degree of the polynomial and the number of nodes at one level, it is what the representation gives us.

2.2 The BGJT algorithm

In this subsection we take $f_1 = Y - X^q$ and $f_2 = h_1(X)Y - h_0(X)$ which allow us to represent elements of $\mathbb{F}_{q^{\delta n}}$ as polynomials in $\mathbb{F}_{q^\delta}[X]$.

The main idea of the algorithm is to create relations between polynomials of degree $\leqslant \lceil \frac{\deg P}{2} \rceil$ and the polynomials $\{P - a\}_{a \in \mathbb{F}_{q^\delta}}$ where P is the polynomial we want to descend. Once we have enough relations between the $\{P - a\}_{a \in \mathbb{F}_{q^\delta}}$, we can use linear algebra to rewrite $\log P = \log(P - 0)$ in terms of the degree $\leqslant \lceil \frac{\deg P}{2} \rceil$ polynomials, then we recurse in those lower degree polynomials. The older algorithm does not achieve a descent from degree d to $\lceil \frac{d}{2} \rceil$

polynomials, this in the main improvement of the BGJT QPA algorithm which allow the breakthrough of becoming a quasi polynomial algorithm.

In order to make this possible we use the so-called "systematic equation" to generate the relation:

$$X^q - X = \prod_{\alpha \in \mathbb{F}_q} (X - \alpha) \tag{1}$$

To make the translates of X appears in the equation (1), we do the following transformation: $H_m : X \mapsto \frac{aP+b}{cP+d}$ with $m = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and then multiply the relation by $(cP+d)^{q+1}$ gives us:

$$(aP+b)^q(cP+d) - (aP+b)(cP+b)^q = (cP+d) \prod_{\alpha \in \mathbb{F}_q} (aP + b - \alpha(cP+d)) \tag{2}$$

Considering the couple $(\alpha, \beta) \in \mathbb{F}_q \times \mathbb{F}_q$ to represent the elements in the right hand side of the previous equation as $\beta(aP+b) - \alpha(cP+d)$ makes us notice that we can assimilate $(cP+d)$ to $(1,0)$, and the elements under the product are assimilated to $(\alpha, 1)$. The set of such couples is known to be a *representative set* of the *projective line* $\mathbb{P}^1(\mathbb{F}_q)$ and the point $(1,0)$ is named the point at infinity.

From (2) we end up with the following equality for every $m = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathcal{P}_q$ by substitute α by $(\alpha, \beta) \in \mathbb{P}^1(\mathbb{F}_q)$ as described above:

$$(aP+b)^q(cP+d) - (aP+b)(cP+d)^q = \overbrace{\lambda}^{\in \mathbb{F}_{q^\delta}} \prod_{(\alpha,\beta) \in \mathbb{P}^1(\mathbb{F}_q)} P - x(m^{-1} \cdot (\alpha, \beta)) \tag{3}$$

Where $x(m^{-1} \cdot (\alpha, \beta))$ denotes $P - u$ when $m^{-1} \cdot (\alpha, \beta) = (u, 1)$, or 1 if it $m^{-1} \cdot (\alpha, \beta) = \infty$, in this case the coefficient in front of P vanishes and the remaining term is put into the λ. x can be viewed as the extraction of the first coordinate of a point of the projective line.

We can notice that the action of H_m becomes trivial when $m \in \mathbb{F}_q$: in this case we are just reordering the elements in the right hand side of (2) and, as $a^q = a$ in $\mathbb{F}_q$, the left hand side of (2) are all equal. This is why we can restrict the set of matrices m to $\mathcal{P}_q = \mathrm{PGL}(\mathbb{F}_{q^\delta})/\mathrm{PGL}(\mathbb{F}_q)$.

We finally rewrite the left hand side of (3) in term of low degree polynomials:

$$\frac{1}{h_1^{\deg P}} \left[\left(\tilde{a}\tilde{P}\left(\frac{h_0}{h_1}\right) h_1^{\deg P} + \tilde{b} h_1^{\deg P} \right)(cP+d) - (aP+b)\left(\tilde{c}\tilde{P}\left(\frac{h_0}{h_1}\right) h_1^{\deg P} + \tilde{d} h_1^{\deg P} \right) \right] = \lambda \prod_{(\alpha,\beta) \in \mathbb{P}^1(\mathbb{F}_q)} P - x(m^{-1} \cdot (\alpha, \beta)) \tag{4}$$

Where $\tilde{e} = e^q$ and $\tilde{P}$ is the polynomial P with its coefficients raised to the power q.

In the case where the left-hand side of the equation (4) is $\lceil \frac{\deg P}{2} \rceil$-smooth, we have a relation as wanted. Once we have $q^\delta + 1$ such independent relations, we can express the $\log P$ as a linear combination of degree $\leqslant \lceil \frac{\deg P}{2} \rceil$.

A final remark is that the degree 1 elimination step of most of the state-of-the-art art DLP solving algorithms are using the Joux's method [19]. The BGJT algorithm has been derived from this technique for higher degree polynomials. To do this, we take the relations in (4) and use X as the polynomial P.

2.3 The GKZ degree $2d$ descent

In this subsection we take $f_1 = Y - X^q$ and $f_2 = h_1(Y)X - h_0(Y)$. Then we keep the polynomials in X and Y to represent $\mathbb{F}_{q^{\delta n}}$.

This algorithm is based on an observation made in [13] where Göloğlu et al. described a method to generate a family of polynomials that always completely splits like $X^q - X$. They are derived from a polynomial relation, namely the image of $\mathbb{F}_{q^\delta}/\mathbb{F}_{q^2}$ under the map:

$$u \mapsto \frac{\left(u - u^{q^2}\right)^{q+1}}{(u - u^q)^{q^2+1}} \tag{5}$$

This relation describes a set $\mathcal{B} \subseteq \mathbb{F}_{q^\delta}$ of elements B such that $X^{q+1} - BX + B$ splits completely, which can be transformed into $X^{q+1} + aX^q + bX + c$ after scaling and translating:

$$B = \frac{(b - a^q)^{q+1}}{(c - ab)^q} \tag{6}$$

Given an irreducible polynomial P we want to descend, we build the following lattice:

$$\mathcal{L}_P = \left\{(w_0, w_1) \in \mathbb{F}_{q^\delta}[Y] \mid w_0 h_0 + w_1 h_1 \equiv 0 \mod P\right\} \tag{7}$$

If P does not divide $w_0 h_0 + w_1 h_1$ which is mostly the case, then we have a basis of the form $(u_0, Y + u_1)$, $(v_0 + Y, v_1)$ with $u_i, v_i \in \mathbb{F}_{q^\delta}$. To compute this base, we start from the following base:

$$(1, (-h_0/h_1) \mod P), ((-h_1/h_0) \mod P, 1)$$

As P is of degree 2, we then have a base of the form $(1, aY + b)$, $(cY + d, 1)$, which is indeed $(1/a, Y + b/a)$, $(Y + d/c, 1/c)$, and we have our base.

Remark 3. If $a = 0$, this means that $b = h_0/h_1 \mod P$ then $\exists k,\ bh_1 - h_0 = kP$. This is a relation between two degree 2 polynomials, then P divides $(-h_0 + bh_1)/k = (-X + b)/k = P$, which is a relation between P and a linear polynomial.

Then we want to build a polynomial $Q \in \mathcal{L}_P$ such that Q splits completely. To do this, we study the necessary condition for Q to be of the form

$$XY + aY + bX + c = \frac{1}{h_1}((Y + b)h_0 + (aY + c)h_1)$$

and splits completely: we need that $(Y + b, aY + c) \in \mathcal{L}_P$ in order to have P dividing Q. This leads to the following relations: $b = au_0 + v_0$ and $c = au_1 + v_1$. This means that b and c are fully determined by the choice of a and we only have one unknown left. Substitute the value of b and c by their expression it term of a in the condition (6) gives us a as the root of a polynomial relation:

$$(a^q + au_0 + v_0)^{q+1} - B(-u_0 a^2 + (-v_0 + u_1)a + v_1)^q = 0 \tag{8}$$

As $h_1(Y)Q(Y) = (Y + a)h_0(Y) + (bY + c)h_1(Y)$ is at most of degree 3 (under the assumption that h_0 and h_1 are at most of degree 2) and having P of degree 2 dividing it, we end up with a polynomial of degree at most 1, which means a linear one. Then we indeed have a relation between a degree 2 polynomial and degree 1 polynomials.

Hence from this descent from degree 2 to 1, we can extrapolate the method for irreducible degree $2d$ polynomial to degree d polynomials by embedding the degree $2d$ polynomial in $\mathbb{F}_{q^{\delta d}}$ where it splits into irreducible quadratics, which can be related with degree 1 polynomials in $\mathbb{F}_{q^{\delta d}}$, and then we can recurse by embedding those new polynomial into $\mathbb{F}_{q^{\delta \frac{d}{2}}}$. The details about how to do these embeddings will be shown is section 3.3.

The overall cost of the algorithm is then $q^{\log_2 q + \mathcal{O}(1)}$ [17] which is asymptotically more than the BGJT algorithm, but the descent tree has less nodes — a step generates $\mathcal{O}(q)$ new nodes instead of $\mathcal{O}(q^2)$ — and there is no linear algebra step, one only has to find the roots of the polynomial (8) in $\mathbb{F}_{q^{\delta d}}$, hence this step is quasi instant which make it suitable for practical implementation.

Remark 4 (A word about the equivalence between this polynomials and the systematic equation). One can wonder if we can mix both the systematic equation and the sieving technique we saw in this section. But the splitting polynomials we can obtain are the same.
The GKZ polynomials are the polynomials where $X^q - BX + B$ which can be translate into $X^{q+1} + aX^q + bX + c$. The BGJT systematic relations are derived from:

$$(aX+b)^q(cX+d) - (aX+b)(cX+d)^q = (\tilde{a}c - a\tilde{c})X^{q+1} + (\tilde{a}d - b\tilde{c})X^q + (\tilde{b}c - a\tilde{d})X + \tilde{b}d - b\tilde{d}$$

Which can easily be mapped to $X^{q+1} + aX^q + bX + c$ after a division by $\tilde{a}c - a\tilde{c}$.

3 Improvements and analysis of the parameter ranges

In this section we will present an analysis of the different parameter ranges based on the different problems we encountered during the practical implementation of the previous algorithms which will be explained in this section.

The first subsection will present some mathematical tools we used for this analysis, then an analysis of the BGJT algorithm will be made and finally an analysis of the GKZ descent will be presented.

3.1 Mathematical background

As we saw, the BGJT algorithms relies on the probability that the left hand side of (4) splits into degree $\lceil \frac{\deg P}{2} \rceil$ polynomial. To compute this probability we used the following results: for any prime power q and any integers $1 \leqslant m \leqslant n$ we denote by $N_q(m, n)$ the number of m-smooth monic polynomials of degree n. The exact formula we used in our analysis is given in [15, 2] and an approximation for asymptotic behaviour is given in [23]:

$$N_q(n, m) = q^n \rho\left(\frac{n}{m}\right)\left(1 + \mathcal{O}\left(\frac{\log n}{m}\right)\right) \tag{9}$$

Here ρ denotes the Dickman's function defined as the unique continuous function such that $\forall u \in [0,1], \rho(u) = 1$ and $\forall u > 1, u\rho'(u) = \rho(u-1)$.

If you are unfamiliar with Galois fields, I invite you to refer to appendix A.

3.2 The BGJT algorithm

In the original article, Barbulescu et al. chose 2 as the value of δ, the motivation of this choice is to make the field $\mathbb{F}_{q^{\delta n}}$ as big as possible with a reasonable size for the field $\mathbb{F}_{q^\delta}$.

Remark 5. The choice of the polynomial f_2 to be linear in X or Y will not change the factorisation probability as we can easily go from one representation to another through the transformations $X \mapsto Y^{q^{\delta-1}}$ or $Y \mapsto X^q$ which are both linear over $\mathbb{F}_{q^\delta}[X, Y]$.

Let $D = \deg P$ and $\Delta = \max(\deg h_0, \deg h_1)$. Then we need to have a degree $(\Delta+1)D$ polynomial to be $\lceil \frac{D}{2} \rceil$-smooth. The *smoothness condition* which is a necessary condition to have enough relation is then:

$$\underbrace{\frac{N_{q^\delta}((\Delta+1)D, \lceil \frac{D}{2} \rceil)}{q^{\delta\cdot(\Delta+1)D}}}_{\text{smoothness probability}} \overbrace{q^{3\delta-3}}^{\#\mathcal{P}_q} \gg \underbrace{q^\delta}_{\text{\#relations}} \tag{10}$$

We can also randomize the algorithm by taking as substitution $P \mapsto \frac{aP+bP_1}{cP+dP_1}$ with P_1 chosen uniformly at random such that $\deg P_1 < \deg P$ instead of H_m as described in section 2.2, this allows us to take parameters where the smoothness probability has the same order of magnitude as q^δ. This idea was originally proposed by Barbulescu et al.. They suggest the use of polynomial P_1 subject to a structure in order to be suitable for a sieving technique [6, Section 6.2].

As said before, it is common to select $(\Delta, \delta) = (2, 2)$ as parameters. Substituting this into (10) leads us to $q > 2^{16}$ Hence the matrix we will have to handle will be $2^{32} \times 2^{32}$ with 2^{48} non zero elements. As suggested by Adj et al. we can use the block Wiedemann algorithm for the linear factor descent step. This will already have a complexity of $2^{80}A_N$ where A_N denotes the cost of an addition in our medium sized field, hence the security of 80 bits is already reached. Therefore this algorithm cannot be used in this case.

A way to generate enough relations is to notice that

$$\#\mathcal{P}_q = \mathrm{Card}(\mathrm{PGL}(\mathbb{F}_{q^\delta})/\mathrm{PGL}(\mathbb{F}_q)) = \mathcal{O}\left(q^{3\delta-3}\right)$$

Hence taking $\delta = 3$ instead of $\delta = 2$ makes the condition (10) become $\frac{N_{q^3}(3D, \lceil \frac{D}{2} \rceil)}{q^{3\cdot 3D}} q^6 \gg q^3$, and the first possible q is then $38 < 2^6$. Then the matrices are of size $(2^{6\cdot 3})^2 = 2^{36}$ with 2^{24} non zero entries. And linear algebra is then possible even with cubic operations in the size of the matrix, we still have $2^{54}A_N$ operations, which is less than 2^{80}. And with a field of size 2^6 we can reach a field of size $2^{3\cdot 6\cdot 2\cdot 2^6} = 2^{2304}$ which has the same order of magnitude of the finite fields currently used for discrete logarithms.

Another way to improve this algorithm by working on the condition (10) is to notice that the coefficient $3 = \Delta + 1$ in front of D can be lowered to 2 by taking h_0 and h_1 of degree at most 1. As $N_{q^\delta}(\cdot, \cdot)$ behaves like the Dickman function, this modification allow us to make the size of the intermediate field dropping down by a factor $\approx \frac{\rho(4)}{\rho(6)} \approx 250$. The trade-off is that we hence need an intermediate field twice as big to reach the same field size, but the previous field size was way too big, it is already an improvement.

We can also notice that representatives of $\mathcal{P}_q$ are of two forms: $\begin{pmatrix} a & b \\ 1 & d \end{pmatrix}$ and $\begin{pmatrix} 1 & b \\ 0 & d \end{pmatrix}$. The matrices of the form $\begin{pmatrix} 1 & b \\ 0 & d \end{pmatrix}$ gives for (4) where $P \mapsto X$:

$$(h_0 + \tilde{b}h_1)d - (X + b)\tilde{d} = \lambda \prod_{(\alpha,\beta)\in\mathbb{P}^1(\mathbb{F}_q)} P - x(m^{-1} \cdot (\alpha, \beta)) \tag{11}$$

δ	Δ	q_{min}	q^{δ}_{min}
2	2	2^{16}	2^{32}
2	1	2^{8}	2^{16}
3	1	2^{3}	2^{9}
3	2	2^{6}	2^{18}

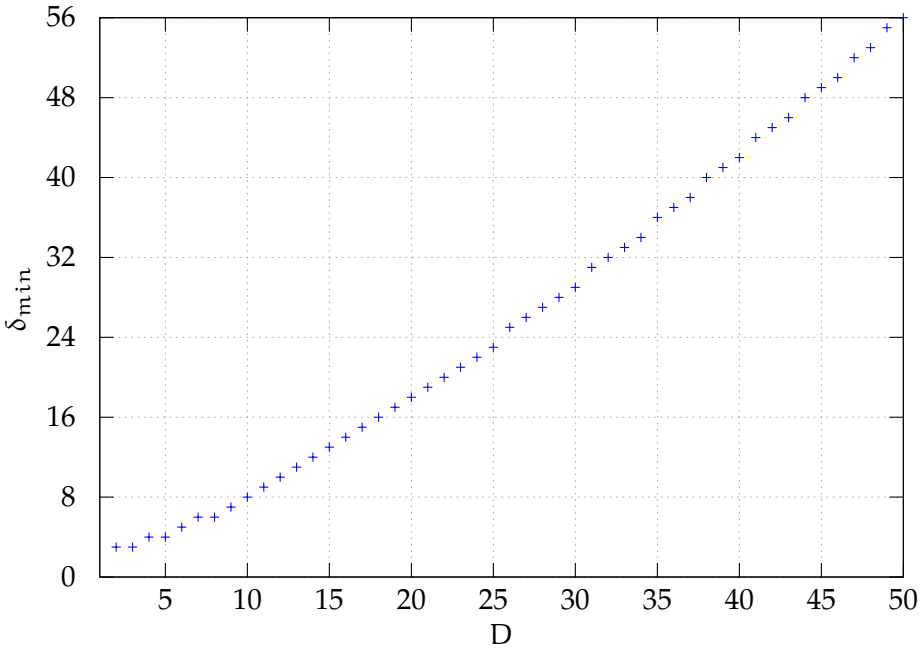
(a) First possible values of q for some parameters

δ	Δ	q	maximal descent	total tree size
2	1	2^{8}	$\frac{D}{2}$	$2^{16\cdot4}$
2	2	2^{16}	$\frac{D}{2}$	$2^{32\cdot5}$
2	1	2^{16}	$\frac{D}{3}$	$2^{32\cdot4}$
3	1	2^{6}	$\frac{D}{3}$	$2^{18\cdot4}$

(b) Some parameters quality analysis

Table 1

Figure 4: Minimum δ to descend from a degree D polynomial to a degree 1 polynomial in $\mathbb{F}_{16^{\delta}}[X]$



When $\Delta = 1$, this lead directly to a relation. And we have $\frac{q^{2\delta}-q^{\delta}}{q^2-q}$ such matrices. Then having $\delta > 2$ suffices to generate enough relation for the linear descent phase without smoothness test. However, the condition (10) then become $\frac{N_{q^{\delta}}(2D,\lceil\frac{D}{2}\rceil)}{q^{\delta\cdot 3D}} q^{2\delta-2} \gg q^{\delta}$. Then we must have $\delta \geqslant 3$ to have the hope of having enough relations raised from those matrices.

Another strategy is instead of filtering the relations by the condition "is the degree of the left hand side of (4) less than $\lceil\frac{D}{2}\rceil$", we can just keep the q^{δ} independent relations of lowest degree. This allow us to descend until the smoothness condition for going from a degree D polynomial to a degree $D-1$ does not hold, which is approximatively $1-\frac{1}{\rho^{-1}(q^{3-2\delta})}$. And then using other descent strategy for the low degree polynomials such as Gröbner basis. In combination with other descent methods, it is how Adj et al. were doing their security analysis [1].

The parameter ranges analysis is summarized in table 1. The table 1a presents some first value of q to make a $D \to \lceil\frac{D}{2}\rceil$ descent possible, and the table 1b shows the number of nodes in the descent tree we can expect from some parameters choice, using the "keep the best" strategy as presented in the previous paragraph.

To analyze the efficiency of this algorithm, another way could be to look at the extreme cases. In this case it can be the minimum δ we can take to make the descent in one step, we already know from our previous results that it will not be computable due to the linear algebra step which is costly in terms of time and memory, but it is a good way to look at the behaviour of the algorithm. Then we can see from figure 4 that δ grows linearly in D, as δ is in the exponent of q defining the size of our problem, this means that the one-step descent will be exponential in the size of the field (because the size of the field defines the maximal polynomial degree we will encounter), and then the behaviour of polynomial smoothness does not play in our favor.

At last but not least, we can speed up the factor basis elimination by considering the following relation for $a \in \mathbb{F}_{q^\delta}$:

$$(X + a)^q = X^q + a^q = \frac{h_0}{h_1} + \tilde{a} \tag{12}$$

In the case $\Delta = 1$, this leads to a relation with at most 3 non zero entry per line instead of $q + 1$ with the BGJT variant matrix. In fact the iteration of this ends with the relation $(X - a)^{q^{\delta n}} = (X - a)$ which is trivial and does not give us more information. There are at most $\frac{q^\delta - 1}{\delta n}$ trivial relations, which means that we need $\frac{q^\delta - 1}{\delta n}$ relation of the BGJT type. As in the *optimal* case $n \approx q$, we that there are $\mathcal{O}\left(q^{\delta-1}\right)$ missing relations. In the factor basis matrix there are then $\mathcal{O}\left(q^\delta\right)$ rows with $\mathcal{O}(1)$ elements and $\mathcal{O}\left(q^{\delta-1}\right)$ rows with $\mathcal{O}(q)$ relations, which leads to an overall cost of $\mathcal{O}\left(q^{2\delta}\right)$ instead of $\mathcal{O}\left(q^{2\delta+1}\right)$ using block Wiedemann's algorithm [25], we then have a speedup of the order of q.

3.3 The GKZ algorithm

In the following, we place ourselves in $R_1 = \mathbb{F}_{q^\delta}[X, Y]/(Y - X^q) = F_{q^\delta}[X]$ in order to manipulate univariate polynomials.

As we saw before, to be usable for a whole descent, the algorithm must start from an irreducible degree 2^{m+1} polynomial in $\mathbb{F}_{q^\delta}$ in X. The irreducible condition is necessary to have the roots of the polynomial to live in $\mathbb{F}_{q^{\delta d}} \backslash \mathbb{F}_{q^{\delta \frac{d}{2}}}$, otherwise we would not be able to descend.

In order to have a polynomial which is on this form, one way can be to write $P_{[\alpha]} = g^\alpha . P = \frac{A}{B}$ with A and B be two polynomials of degree 2^l in X and g the base of the discrete logarithm we want to compute. The coefficients of A and B can be computed through linear algebra for instance: $B \cdot P_{[\alpha]} = A \mod f_2$, then

$$\sum_{i=0}^{2n} B_i (P_{[\alpha]} X^i \mod f_2) = \sum_{i=0}^{2n} A_i X^i \mod f_2$$

leads to $\deg f_2 + 1$ relations for $2^{l+1} + 1$ variables: $\{A_i, B_i\}_i$ (the $+1$ instead of a $+2$ come from the fact that we can set A to be monic for instance, which removes one variable). Then in order to have enough relations to have at least one solution we need that $\deg f_2 \leqslant 2^{l+1} + 1$. Then we can notice that the *optimal* case $\deg f_2 = 2^{m+1} + 1$ allows us to skip one step of the descent, because we can go from a degree $q = 2^{m+1}$ polynomial in the general case to 2 degree 2^m polynomials.

As the probability for a random polynomial of degree q in $F_{q^\delta}[X]$ to be irreducible is $\mathcal{O}\left(\frac{1}{q}\right)$ [24, Lemma 2] we need approximatively q^2 tries before succeeding, assuming that A and B behave like two independent random polynomial.

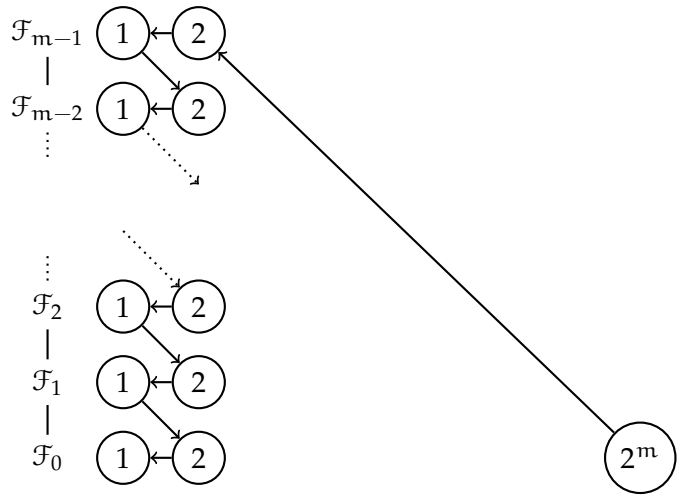


Figure 5: Elimination of a degree 2^m polynomial: $\leftarrow$ denotes a degree 2 elimination, $\searrow$ is taking the field norm with respect to the indicated fields and $\nwarrow$ represents the factorisation in the indicated field.

Once we have a degree 2^m polynomial P, we can extend the field by a random irreducible degree 2^{m-1} polynomial to end up with polynomial in the field: $\mathcal{F}_{m-1} = \mathbb{F}_{q^{\delta 2^{(m-1)}}}$. As P is irreducible, it splits entirely in $\mathcal{F}_m$ with roots in $\mathcal{F}_m \backslash \mathcal{F}_{m-1}$, which leads to:

$$P = \prod_{i=1}^{2^{m-1}} P_i \tag{13}$$

where P_i are degree 2 irreducible polynomials in $\mathcal{F}_{m-1}[X]$.

We can then apply the degree 2 elimination as described in section 2.3 to have:

$$P_1 = \bar{h_1} \prod_{i=1}^{q+2} Q_i \tag{14}$$

By noticing that all roots of P in $\mathcal{F}_m$ are conjugate under $\mathrm{Gal}(\mathcal{F}_m/\mathcal{F}_0)$ we can write $P_i = P_1^{[i(m-1)]}$ where $P^{[k]}$ denotes the polynomial P with its coefficients raised to the power $2^{\delta k}$. From the factorisation (14) we can derive a factorisation for each P_i using the fact that Galois conjugation is a ring homomorphism as follows:

$$P_i = \bar{h_1} \prod_{k=1}^{q+2} Q_k^{[i(m-1)]}$$

Therefore we only have to do the descent one time, and then multiply each factors by its conjugate under $\mathrm{Gal}(\mathcal{F}_{m-1}/\mathcal{F}_{m-2})$.

In order to obtain obtain irreducible polynomials of degree 2 in $\mathbb{F}_{m-2}$ we only have make the product of two conjugate polynomial under $\mathrm{Gal}(\mathcal{F}_{m-1}/\mathcal{F}_{m-2})$:

$$P_1 P_2 = \bar{h_1}^2 \prod_{i=1}^{q+2} Q_i Q_i^{[m-2]}$$

Then we can repeat the process by applying the 2-descent on $Q_i Q_i^{[m-2]}$ for $1 \leqslant i \leqslant q+2$ and iterate the Galois conjugate product which can be viewed as the norm of $P_1 \dots P_{2^j}$ under $\mathrm{Gal}(\mathcal{F}_{m-j}/\mathcal{F}_{m-j-1})$ for the j-th iteration of the algorithm. This is summarized in figure 5.

We finally end up with a relation between $P = P_1 \dots P_n$ and at most $(q+2)^{\log_2 q}$ linear polynomials which allow us to compute the discrete logarithm of the original polynomial.

We notice that as the computation is almost costless, we can afford to do it multiple times to save memory. Then instead of storing the whole descent, we store the element $B \in \mathcal{B}$ used to compute the splitting relation in (6) and the result of the different sub-logarithms in a depth-first way, and then throw the logarithms at level $j+1$ when we can compute the logarithm at level j in the descent tree. We then have a descent part which takes only $\mathcal{O}\left(\log_2 q \cdot q\right)$ space instead of $\mathcal{O}\left(q^{1.5\delta}\right)$ which is a nice improvement.

4 Results

At first we implemented the BGJT algorithm before noticing the first problem during the linear descent: the *trap polynomials*. Some research on the subject lead us to the article of Cheng, Wan, and Zhuang [8]. This article presents the problem: when we met a factor Q of $h_1(X)Y + h_0(X)$ during the descent, this lead to the relation:

$$X^q - X = \prod_{\alpha \in \mathbb{F}_q} (X - \alpha) = 0 \mod Q$$

Then Q is a factor of the left hand side of (4), and as the degree of the left hand side of (4) is 3, it is likely that Q will appear with power 1, and then simplify in the relation. Therefore we will not be able to make this logarithm appear through these relations.

This problem is solved by the introduction of the $(P+a)^q = \tilde{P} + \tilde{a}$ relations, which are independent of f_2 and $h_1(X)Y + h_0(X)$, and then relates the logarithm of Q with the logarithm of other elements of the factor basis.

However the use of this trick for the factor basis matrix as explained in the last paragraph of section 3.2 appears to have a great practical incidence, reducing the factor basis linear algebra part to go from 1 minute to 1 second in $\mathbb{F}_{2^{3\cdot 4\cdot (2^3-1)}}$, which is already a great improvement.

The relation generation is a quite fast phase as it is done in $\mathcal{O}\left(q^\delta \cdot \log(\deg P)\right)$ for a fixed (δ, Δ), the computational bottleneck is the linear algebra which has to be done for each sub-logarithm. This method gives us q^δ such sub-logarithms which makes the descent-tree width grow very fast.

If the factor base computation with $\Delta = 1$ is done quickly, the matrices to invert for the descent phase are denser, then it became hard to do more than one step in a reasonable time.

After that we worked on the implementation of the GKZ descent phase. But we met some problems with the matrix we used for the linear phase. Before noticing that our version of Magma: `v.2.15-12` has a bug in its irreducible test algorithm. As a consequence, it uses reducible matrices to extend the fields which was then no longer Galois.

To avoid it, we used the Magma online calculator to generates the polynomials h_0 and h_1 we used for the descent phase.

In order to do a big computation, we take the following parameters described in table 2. This gives us a representation for the field $\mathbb{F}_{2^{1300}}$

When I write my report, the computation is still running, we met a problem with the previous version due to a factorisation we did not take into account in our computation. The kernel

Parameter	Value
q	2^5
δ	4
h_0	$X + x^{10} \cdot y^3 + x^{26} \cdot y^2 + x^{13} \cdot y + x^{11}$
h_1	$X^2 + (y^3 + x^9 \cdot y^2 + x \cdot y + x^{18}) \cdot X + x^2 \cdot y^3 + x^8 \cdot y^2 + x^{10} \cdot y + x^{22}$

Table 2: Parameters we used. x is a primitive element of $\mathbb{F}_q$ and y a primitive element of $\mathbb{F}_{q^\delta}$. Respectively a root of $X^5 + X^3 + 1$ and a root of $Z^4 + Z^2 + (x+1)Z + 1$.

computation algorithm (namely the Lanczos block algorithm [21]) works over finite field, and then needs to work in $\mathbb{F}_{p^k}$. All our computations are done modulo $\ell | 2^{1300} - 1 = \left|\mathbb{F}_{q^{\delta n}}\right|$ with big enough factors. Then the result is build again using the Chinese remainder theorem. To do this, our first implementation was doing the factorisation of $2^{1300} - 1$ using the successive trial division until reaching a certain threshold. And then use ℓ as it is. But it was a bad idea because if Magma has built-in table to factorize $2^{1300} - 1$, it does not have one for every factors of $2^{1300} - 1$. And the computation aborted after 215h of calculus. Then I launched again the kernel computation on a GPU using the `gpulinalg` [18] tool developed in LORIA for this purpose implementing the Block Wiedemann algorithm we used in our analysis.

However, for a more modest example, we manage to solve the discrete logarithm in $\mathbb{F}_{2^{3\cdot 4\cdot (2\cdot 2^3+1)}}$ of 204 bits in 16.5 seconds to get the logarithms of the linear factor and 1.2 sec to descent the logarithm of an given element in $\mathbb{F}_{q^{\delta n}}$. For comparison, the built-in `Magma` algorithm [7, 9] takes 12 minutes to compute the same logarithm (including 1 minute for the individual logarithm descent).

5 Conclusion

We presented in this report the state-of-the-art algorithms to solve the discrete logarithm problems and analysed their efficiency from a practical point of view. We saw why, if it were a major theoretical breakthrough, the BGJT algorithm was not used excessively even for computational records [27], where the $L\left(\frac{1}{4} + o(1)\right)$-algorithm [19] is still privileged. As the GKZ algorithm is still very recent, not a lot of work has been done on it and we hope that we were the first ones to implement it and use it for a real world sized logarithm as we saw in section 4.

There is still work to do, as to find a way to choose the best algorithm to solve the discrete logarithm case by case in order to build a portfolio algorithm to solve the discrete logarithm. This has been the work of Adj et al. [1, 2, 3] in a case-by-case approach.

Another possible future work could be to find a way to enhance the smoothness basis resolution. As it were not the bottleneck of the computation and were already completely polynomial, it is rare to see the trace of the $(X + a)^q = X^q + \tilde{a}$ optimisation in the literature even if its existence is known.

The last and more natural way to improve the quality of discrete logarithm algorithm based on function field sieve could be to improve the quality of the descent (i.e. lowering the number of nodes for a complete descent). We can see than in the *optimal* case of the GKZ descent we can express an irreducible degree 2 polynomial in terms of only one degree 1 polynomial, as explained in remark 3. It could be a good idea to look further on the conditions to obtain such

a descent and see if it can be computed.

Another not exploited area of study are the polynomial h_0 and h_1 which can be chosen, and then give us some liberty. One interesting thing to do is to see how we can take advantage of this liberty, for example to choose h_0 and h_1 such that g and $h = g^x$ have the smaller possible degree.

To conclude, this internship has been the occasion to work on a completely new subject and to get familiar with the latest advances on it. It also has been the occasion to learn how to use a new computer algebra system that is Magma and developing a proof-of-concept implementation of the algorithms described in theoretical articles.

Acknowledgements

I want to thank the COSIC team for their warm and friendly welcome and their help, in particular Fre Vercauteren for taking time to help me with algebra and his reactivity, and also Filipe Beato, Tomer Ashur, Michael Herrmann, Kimmo Järvinen, Roel Peeters and Alan Szepieniec for the discussions at lunch time.

I also want to thank Pierrick Gaudry and Emmanuel Thomé for their help about understanding their algorithm and their advices for real world implementation of matrix algorithms.

Thanks to Christophe Mouilleron for taking his time to explain me the different matrix algorithms and the devastating effects of a bad choice of algorithm.

References

[1] G. Adj, A. Menezes, T. Oliveira, and F. Rodriguez-Henriquez. Weakness of $\mathbb{F}_{3^{6\cdot 1429}}$ and $\mathbb{F}_{2^{4\cdot 3041}}$ for discrete logarithm cryptography. *IACR Cryptology ePrint Archive*, 2013:737, 2013.

[2] G. Adj, A. Menezes, T. Oliveira, and F. Rodríguez-Henríquez. Weakness of $\mathbb{F}_{3^{6\cdot 509}}$ for discrete logarithm cryptography. In *Pairing-Based Cryptography–Pairing 2013*, pages 20–44. Springer, 2014.

[3] G. Adj, A. Menezes, T. Oliveira, and F. Rodríguez-Henríquez. Computing discrete logarithms in $\mathbb{F}_{3^{6*137}}$ and $\mathbb{F}_{3^{6*163}}$ using magma. Cryptology ePrint Archive, Report 2014/057, 2014. `http://eprint.iacr.org/`.

[4] L. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 55–60. IEEE, 1979.

[5] L. Adleman. The function field sieve. In *Algorithmic number theory*, pages 108–121. Springer, 1994.

[6] R. Barbulescu, P. Gaudry, A. Joux, and E. Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. Cryptology ePrint Archive, Report 2013/400, 2013. `http://eprint.iacr.org/`.

[7] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. ISSN 0747-7171. doi: 10.1006/jsco.1996.0125. URL `http://dx.doi.org/10.1006/jsco.1996.0125`. Computational algebra and number theory (London, 1993).

[8] Q. Cheng, D. Wan, and J. Zhuang. Traps to the BGJT-Algorithm for Discrete Logarithms. Cryptology ePrint Archive, Report 2013/673, 2013. http://eprint.iacr.org/.

[9] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *Information Theory, IEEE Transactions on*, 30(4):587–594, 1984.

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

[11] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[12] S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, 2006. http://eprint.iacr.org/.

[13] F. Göloğlu, R. Granger, G. McGuire, and J. Zumbrägel. On the function field sieve and the impact of higher splitting probabilities: Application to discrete logarithms in $\mathbb{F}_{2^{1971}}$ and $\mathbb{F}_{2^{3164}}$. Cryptology ePrint Archive, Report 2013/074, 2013. http://eprint.iacr.org/.

[14] D. M. Gordon. Discrete logarithms in GF(P) using the Number Field Sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.

[15] R. Granger. Estimates for discrete logarithm computations in finite fields of small characteristic. In *Cryptography and coding*, pages 190–206. Springer, 2003.

[16] R. Granger, T. Kleinjung, and J. Zumbrägel. Breaking '128-bit secure' supersingular binary curves (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). Cryptology ePrint Archive, Report 2014/119, 2014. http://eprint.iacr.org/.

[17] R. Granger, T. Kleinjung, and J. Zumbrägel. On the powers of 2. Cryptology ePrint Archive, Report 2014/300, 2014. http://eprint.iacr.org/.

[18] H. Jeljeli. Resolution of linear algebra for the discrete logarithm problem using GPU and multi-core architectures. *arXiv preprint arXiv:1402.3661*, 2014.

[19] A. Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic. Cryptology ePrint Archive, Report 2013/095, 2013. http://eprint.iacr.org/.

[20] A. Joux and R. Lercier. The function field sieve in the medium prime case. In *Advances in Cryptology-EUROCRYPT 2006*, pages 254–270. Springer, 2006.

[21] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In *Advances in Cryptology-CRYPT0'90*, pages 109–133. Springer, 1991.

[22] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 2010.

[23] D. Panario, X. Gourdon, and P. Flajolet. An analytic approach to smooth polynomials over finite fields. In *Algorithmic number theory*, pages 226–236. Springer, 1998.

[24] M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM Journal on Computing*, 9(2): 273–280, 1980.

[25] E. Thomé. Subquadratic computation of vector generating polynomials and improvement of the block wiedemann algorithm. *Journal of symbolic computation*, 33(5):757–775, 2002.

[26] F. Vercauteren. The number field sieve in the medium prime case. *status: published*, 2006.

[27] J. Zumbrägel. Discrete Logarithms in GF(2^{9234}). NMBRTHRY List, 2014. `https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;9aa2b043.1401`.

A A word about Galois fields

The goal of this section is not to rebuild all the Galois theory, but just to explain some properties of the Galois fields to help the reader to understand the tricks that are used in this report.

Let $\mathbb{F}_q$ be a finite field, and $P(X) = \sum_{i=0}^{n} p_i X^i$ a monic irreducible degree n polynomial in $\mathbb{F}_q[X]$. As we saw in section 2.1, a field extension can be interpreted as doing the calculations in $\mathbb{F}_q[X]$ modulo P which means basically replacing X^n by $-\sum_{i=0}^{n-1} p_i X^i$. Then taking a multiplicative invert in $\mathbb{F}_{q^n}$ can be viewed as doing a polynomial extended GCD.

Then if P is not irreducible, the invert modulo P does not exist for any polynomial Q dividing P. Therefore the structure we built is not a field.

Another interpretation of that is to see the extension as adding a primitive root of P to $\mathbb{F}_q$ and taking the minimal set that makes the new set a field. This is because for ω a root of P we have $P(\omega) = 0$ which means $\omega^n = -\sum_{i=0}^{n-1} p_i \omega^i$ which is the same operation as above.

And all other roots are of the form ω^{q^i} because:

$$P(\omega^{q^i}) = \sum_{k=0}^{n} p_k \omega^{kq^i} = \sum_{k=0}^{n} (p_k \omega^k)^{q^i} = P(\omega)^{q^i} = 0$$

Because $p_k \in \mathbb{F}_q$ which implies $p_k^q = p_k$ and in the additive group we have the relation: $(a+b)^q = a^q + b^q$. And there are $n-1$ of them.

We then can express P in $\mathbb{F}_{q^n}[Y]$ as the product of all the $(Y - \omega^{q^i})_{0 \leqslant i \leqslant n}$. What we can notice is that we can also express P in term of polynomials over the intermediate field $\left\{\mathbb{F}_{q^d}\right\}_{d|n}$ as the product of n/d degree d polynomials. The idea is to multiply together the $Y - \omega^{q^i}$ elements in order to have a polynomial of coefficients in $\mathbb{F}_{q^d}$. To do this we takes the elements of the form $Y - \omega^{q^i \frac{n}{d}}$, then:

$$\left[\prod_{i=0}^{d} (Y - \omega^{q^i \frac{n}{d}})\right]^{q^d} = \prod_{i=0}^{d} (Y - \omega^{q^i \frac{n}{d}}) \tag{15}$$

Which means that the elements lives in $\mathbb{F}_{q^d}$. There are n/d different polynomials (n/d choice of ω ending with different products) and the product of all of them is indeed P.

The product (15) is a degree d polynomials in $\mathbb{F}_{q^d}$, and we can add that it is irreducible. It is also called the *norm* over $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_{q^d})$.

Taking the element $\prod_{i=1}^{d} (Y - \omega^{q^i \frac{n}{d}})$ is called taking the *conjugate* of $Y - \omega$ over $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_{q^d})$