On the Alpha Value of Polynomials in the Tower Number Field Sieve Algorithm

Aurore Guillevic[*] and Shashank Singh[**]

[*]Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
aurore.guillevic@inria.fr
[**]Indian Institute of Science Education and Research Bhopal, Bhopal, India
shashank@iiserb.ac.in

February 6, 2020

Abstract

In this paper, we provide a notable step towards filling the gap between theory (estimates of running-time) and practice (a discrete logarithm record computation) for the Tower Number Field Sieve (TNFS) algorithm. We propose a generalisation of ranking formula for selecting the polynomials used in the very first step of TNFS algorithm. For this we provide a definition and an exact implementation (Magma and SageMath) of the α-function. This function measures the bias in the smoothness probability of norms in number fields compared to random integers of the same size. We use it to estimate the yield of polynomials, that is the expected number of relations, as a generalisation of Murphy's E function, and finally the total amount of operations needed to compute a discrete logarithm with TNFS algorithm in the targeted fields. This is an improvement of the earlier work of Barbulescu and Duquesne on estimating the running-time of the algorithm. We apply our estimates to a wide size range of finite fields GF(p^n), for small composite $n = 12, 16, 18, 24$, that are target fields of pairing-friendly curves.
keywords: discrete logarithm, tower number field sieve.

1 Introduction

The hardness of discrete logarithm computation in finite fields is at the heart of many cryptosystems since the introduction of the Diffie-Hellman problem. Elliptic curves now replaced finite fields in Diffie-Hellman based protocols, however this setting is still widely used in pairing-based cryptography, where protocols use bilinear maps. These maps are available on a subset of well-chosen elliptic curves named pairing-friendly. A bilinear pairing in cryptography maps two points from two distinct subgroups of ℓ-torsion of an elliptic curve $E/\mathbb{F}_q$ to a finite field extension $\mathbb{F}_{q^k}$. The protocol relies on the hardness of computing discrete logarithms on the elliptic curve side and on the finite field side.

Usually, to improve the pairing efficiency, several implementation choices were made, the most popular until 2015 being a Barreto-Naehrig curve whose target field is $\mathbb{F}_{p^{12}}$, and the characteristic has the special property of satisfying $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$, where u is some integer. One widely used sparse seed was $u = -2^{62} - 2^{55} - 1$, so that p is a 254-bit prime, and the elliptic curve $E/\mathbb{F}_p \colon y^2 = x^3 + 2$ has prime order.

In recent years, new advances have been made in computing discrete logarithms in non-prime finite fields, the peak of improvements being a quasi-polynomial time algorithm in small characteristic, in 2014. In medium characteristic, the major improvements are variants of the tower number field sieve (TNFS) that promise to be very efficient for finite fields having non-prime subfields of appropriate size. These new theoretical developments in medium-characteristic fields are of first importance for pairing-based cryptography. The most popular implementations used to include a 3072-bit finite field $\mathrm{GF}(p^{12})$, particularly subject to the new special TNFS algorithms.

In 2016, Kim and Barbulescu published a new variant of the Tower Number Field Sieve algorithm [30], that improved on the previous TNFS algorithm [9] in the case where the extension degree was composite. Combined with Joux and Pierrot Special-NFS variant [27], this reduced considerably the asymptotic complexity of a discrete logarithm computation in finite fields of composite extension degree and special characteristic, typically a target field of Barreto-Naehrig curve.

In this paper, we study the cost of the STNFS algorithm in the cases of pairing-friendly fields. For obtaining an accurate estimate, we generate optimised parameters as for running a real-world computation, and then simulate the STNFS algorithm with these parameters. Our aim is to provide a better estimate of the running-time of these new algorithms, since an implementation is not available for now (it would require a tremendous effort, and first, many algorithmic number theory issues need to be fixed).

Our contribution is twofold: first we provide algorithms to compute better parameters (alpha-value), together with Magma and SageMath implementations. This is the first step towards a complete implementation of the Tower-NFS algorithm. Murphy's E value can also be used to estimate the yield of polynomials and as a byproduct of our work, for some chosen popular pairing-friendly curves, we run the estimates for a large interval of parameter sizes (p^k), to obtain an overview of the way the algorithm scales for fixed extension degree k and increasing characteristic p. While we were polishing the present work, Barbulescu, El Mrabet and Ghammam posted a preprint [11] on security estimates. We have a different approach here.

Organisation of the paper. In Sections 2 and 3 we recall the NFS and TNFS algorithms. Section 4 presents the Murphy-α function for NFS and our generalisation to the TNFS setting. The implementation is explained in Section 5. Section 6 generalises Murphy E ranking function of polynomials to the TNFS setting and explains our simulation algorithm. Sections 7 present our results for finite fields of pairing-friendly curves.

2 Number Field Sieve

The number field sieve (NFS) is an index calculus algorithm for computing discrete logarithm problem in the finite field $\mathbb{F}_Q$ (say), where $Q = p^n$ for some prime p. It consists of following four main phases:

1. Polynomial Selection and Initial Setup
2. Relation Collection (Sieving)
3. Linear Algebra
4. Individual Discrete Logarithm (Descent) and Final Value

2.1 Polynomial Selection and Initial Setup

This is an important step of the NFS algorithm. It determines the overall cost of the algorithm. The basic aim of this step is to select two irreducible integer polynomials $f(X)$ and $g(X)$ having a common irreducible factor $\phi(X)$ of degree n modulo p i.e., $\gcd(f(X), g(X)) = \phi(X) \pmod{p}$. Additionally, we expect the polynomials to have some nice properties, the details of which will be discussed in the Section 4. The choice of these polynomials, i.e. (f, g), provides with us the following commutative diagram, given in the Figure 1.

$$\begin{tikzcd}
 & \mathbb{Z}[X] \arrow[dl] \arrow[dr] & \\
\mathbb{Z}[X]/(f(X)) \arrow[dr, "\text{mod } (p,\phi(X))"'] & & \mathbb{Z}[X]/(g(X)) \arrow[dl, "\text{mod } (p,\phi(X))"] \\
 & \mathbb{F}_{p^n} = \mathbb{F}_p[X]/(\phi(X)) = \mathbb{F}_p(\omega) &
\end{tikzcd}$$

Figure 1: Commutative diagram for NFS

In the diagram given in Figure 1, if f, g are monic, $\mathbb{Z}[X]/(f(X))$ (respectively $\mathbb{Z}[X]/(g(X)))$ is an order in the number field K_f (respectively K_g), generated by $f(X)$ (respectively $g(X)$). The actual computations are carried over to the maximal orders $\mathcal{O}_f$ and $\mathcal{O}_g$ of K_f and K_g respectively. Let $\alpha_f := X \pmod{f(X)}$, $\alpha_g := X \pmod{g(X)}$, $\omega := X \pmod{\phi(X)}$ and $\ell d(f)$ represents the leading coefficient of the polynomial $f(X)$. For a chosen bound B, we set the factor base $\mathcal{F} = \mathcal{F}_f \cup \mathcal{F}_g$, where

$$\mathcal{F}_f = \left\{ \mathfrak{a} : \begin{array}{l} \mathfrak{a} \text{ is a prime ideal of } \mathcal{O}_f \text{ with norm} \leq B \\ \text{or a prime ideal above a prime factor of } \ell d(f) \\ \text{or a prime ideal above index ideal } [O_f : \mathbb{Z}[\alpha_f]] \end{array} \right\}$$

and $\mathcal{F}_g$ is similarly defined with respect to the polynomial $g(X)$.

Remark 1. By the abuse of notation, we have used X for the polynomials defined over integers as well as over a finite field $\mathbb{F}_p$.

Remark 2. A factor base, in an index calculus algorithm, is usually defined as a small size subset of the cyclic group. In the above, this subset is identified by the prime ideals of $\mathcal{F}$ through the mapping, given in the commutative diagram. We provide more detail later in the Section 2.2.

2.2 Relation Collection

In this step linear relations between the logarithms of field elements, corresponding to the factor base, are generated. A set of random polynomials $T(X) = a + b\,X \in \mathbb{Z}[X]$ with coefficients $a, b \in [-A, A]$, for some chosen bound A, are considered. We call a polynomial $a + b\,X$ smooth, if the principal ideals $(a + b\alpha_f)\,\mathcal{O}_f$ and $(a + b\alpha_g)\,\mathcal{O}_g$ can be factored into the elements of $\mathcal{F}_f$ and $\mathcal{F}_g$ respectively i.e.,

$$(a + b\alpha_f)\,\mathcal{O}_f = \prod_{\mathfrak{a} \in \mathcal{F}_f} \mathfrak{a}^{e(\mathfrak{a})} \quad \text{and} \quad (a + b\alpha_g)\,\mathcal{O}_g = \prod_{\mathfrak{b} \in \mathcal{F}_g} \mathfrak{b}^{e(\mathfrak{b})}.$$

Let h_f and h_g be the class number of the number fields K_f and K_g respectively. We have

$$\begin{aligned} ((a + b\alpha_f)\,\mathcal{O}_f)^{h_f} &= \left(\prod_{\mathfrak{a} \in \mathcal{F}_f} \mathfrak{a}^{e(\mathfrak{a})} \right)^{h_f}, \\ \text{i.e. } ((a + b\alpha_f))^{h_f}\,\mathcal{O}_f &= \prod_{\mathfrak{a} \in \mathcal{F}_f} \left(\mathfrak{a}^{h_f}\right)^{e(\mathfrak{a})}. \end{aligned} \tag{2.1}$$

The ideal $\mathfrak{a}^{h_f}$ is a principal ideal, let $\delta_{\mathfrak{a}}$ be a generator i.e., $\mathfrak{a}^{h_f} = \delta_{\mathfrak{a}}\mathcal{O}_f$. Thus Equation (2.1) can be written as

$$((a + b\alpha_f))^{h_f}\,\mathcal{O}_f = \left(\prod_{\mathfrak{a} \in \mathcal{F}_f} (\delta_{\mathfrak{a}})^{e(\mathfrak{a})} \right) \mathcal{O}_f.$$

Converting the equality of principal ideals into the elements, we get

$$((a + b\alpha_f))^{h_f} = u \cdot \left(\prod_{\mathfrak{a} \in \mathcal{F}_f} (\delta_{\mathfrak{a}})^{e(\mathfrak{a})} \right) \tag{2.2}$$

for some unit $u \in \mathcal{O}_f^\star$. By the Dirichlet's Unit Theorem, we know that

$$\mathcal{O}_f^\star = U \times \mathbb{Z}^{r_\mathbb{R} + r_\mathbb{C} - 1}$$

where $U = \langle u_0 \rangle$ is a finite cyclic group consisting of all the roots of unity in K_f, $r_\mathbb{R}$ is the number of real embeddings of K_f in $\mathbb{R}$, and $r_\mathbb{C}$ is the number of complex embeddings of K_f in $\mathbb{C}$. Let $r = r_\mathbb{R} + r_\mathbb{C} - 1$ and $u_1, u_2, \cdots, u_r \in \mathcal{O}_f$ such that,

$$u = u_0^{e(u_0)} \prod_{i=1}^{r} u_i^{e(u_i)}. \tag{2.3}$$

The $\{u_i\}$ are called a fundamental system of units for K_f.

Substituting the value from Equation (2.3) to Equation (2.2), we get

$$((a+b\alpha_f))^{h_f} = u_0^{e(u_0)} \prod_{i=1}^{r} u_i^{e(u_i)} \cdot \left(\prod_{\mathfrak{a}\in\mathcal{F}_f} (\delta_{\mathfrak{a}})^{e(\mathfrak{a})} \right) \tag{2.4}$$

Let us denote the mapping from $\mathbb{Z}[X]/(f(X))$ $(\subset \mathcal{O}_f)$ to $\mathbb{F}_{p^n}$ by $\psi_f\colon \mathcal{O}_f \mapsto \mathbb{F}_{p^n}$ (more precisely to $\mathbb{F}_{p^n}^*/(\mathbb{F}_{p^n}^*)^{\ell}$). We have $\psi_f(a+b\alpha_f) = a+b\omega$. Applying ψ_f to Equation (2.4), we get

$$((a+b\omega))^{h_f} \quad = \quad \psi_f(u_0)^{e(u_0)} \prod_{i=1}^{r} \psi_f(u_i)^{e(u_i)} \cdot \left(\prod_{\mathfrak{a}\in\mathcal{F}_f} (\psi_f(\delta_{\mathfrak{a}}))^{e(\mathfrak{a})} \right). \tag{2.5}$$

Equation (2.5) is a multiplicative relation involving the elements of finite field only. The field element identified with the factor base element $\mathfrak{a} \in \mathcal{F}_f$ is $\psi_f(\delta_{\mathfrak{a}})$ and this is what we tried to explain in the Remark 2 above.

Taking logarithm on both sides of Equation (2.5) and dividing by h_f (we assume that $\gcd(h_f, \ell) = 1$, where $\ell \mid p^n - 1$ is the order of the subgroup considered to compute discrete logarithms) we get,

$$\log\left((a+b\omega)\right) \quad = \quad \sum_{i=0}^{r} e(u_i) \frac{\log\left(\psi_f(u_i)\right)}{h_f} + \sum_{\mathfrak{a}\in\mathcal{F}_f} e(\mathfrak{a}) \frac{\log\left(\psi_f(\delta_{\mathfrak{a}})\right)}{h_f} \pmod{\ell} \tag{2.6}$$

Note that the quantities $\frac{\log(\psi_f(u_i))}{h_f}$ are some fixed constants and they will appear in every equations, we should not bother about what they are. On the other hand each quantity $\frac{\log(\psi_f(\delta_{\mathfrak{a}}))}{h_f}$ is related to the factor base element $\mathfrak{a}$, we call it virtual logarithm corresponding to $\mathfrak{a}$.

Similarly, for the field K_g, let $\psi_g\colon \mathcal{O}_g \mapsto \mathbb{F}_{p^n}$ with $\psi_g(a+b\alpha_g) = a+b\omega$ and $\mathcal{O}_g = V \times \mathbb{Z}^s$, where $V = \langle v_0 \rangle$. Let $\{v_i\}_{i=1}^{s}$ be a fundamental system of units and h_g be the class number of K_g. We will have (assuming $\gcd(h_g, \ell) = 1$ where $\ell \mid p^n - 1$),

$$\log\left((a+b\omega)\right) \quad = \quad \sum_{i=0}^{s} e(v_i) \frac{\log\left(\psi_g(v_i)\right)}{h_g} + \sum_{\mathfrak{b}\in\mathcal{F}_g} e(\mathfrak{b}) \frac{\log\left(\psi_g(\delta_{\mathfrak{b}})\right)}{h_g} \pmod{\ell} \tag{2.7}$$

From Equation (2.6) and Equation (2.7), we get

$$\begin{aligned} &\sum_{i=0}^{r} e(u_i) \frac{\log\left(\psi_f(u_i)\right)}{h_f} + \sum_{\mathfrak{a}\in\mathcal{F}_f} e(\mathfrak{a}) \frac{\log\left(\psi_f(\delta_{\mathfrak{a}})\right)}{h_f} \\ &\qquad = \sum_{i=0}^{s} e(v_i) \frac{\log\left(\psi_g(v_i)\right)}{h_g} + \sum_{\mathfrak{a}\in\mathcal{F}_g} e(\mathfrak{b}) \frac{\log\left(\psi_g(\delta_{\mathfrak{b}})\right)}{h_g} \pmod{\ell}. \end{aligned} \tag{2.8}$$

The above Equation (2.8) is a valid relation, modulo ℓ where $\gcd(h_f h_g, \ell) = 1$ and $\ell \mid p^n - 1$, involving the virtual logarithms of factor base elements and $(r+s+2)$ extra fixed elements. The availability of such a relation is due to the assumptions that the chosen polynomial $T(X) = a + bX$ is smooth (in the sense defined earlier) and a fundamental system of units for both the number fields are available to us. The smoothness of $T(X)$ is ensured over the random choices of its coefficients. In practice, a fundamental system of units, i.e., the generators of unit groups of each number field is often not available. This difficulty is overcome by using the concept called Schirokauer's maps. The logarithm in finite field $\mathbb{F}_{p^n}$ is defined modulo $p^n - 1$ and $p^n - 1$ is composite. Thanks to the Chinese Remainder Theorem, it would be enough to compute logarithms modulo co-prime factors of $p^n - 1$. The discrete logarithm modulo small factors of $p^n - 1$ can be computed using Pollard's rho algorithm. Let ℓ be a (large) prime factor of $p^n - 1$. Schirokauer observed that if we are concerned about the discrete logarithm modulo ℓ, then it is enough to write a relation which is valid modulo ℓ. For this case he has proposed a way to circumvent the computation of generators of units and the contribution of units in the relations (the u_i and $e(u_i)$ and v_i and $e(v_i)$ in (2.8)). Instead, a *Schirokauer map* computes some data in terms of the coefficients of $T(X)$ i.e., a and b, to make the equations valid. We skip the details of Schirokauer's maps and refer to the paper [40] for the interested ones.

Using the Schirokauer's maps denoted λ_f, λ_g, Equation (2.8) can be written as

$$\begin{aligned}\sum_{i=0}^{r} \lambda_{i,f}(a + b\alpha_f)\log(\psi_f(u_i^*)) + \sum_{\mathfrak{a}\in\mathcal{F}_f} e(\mathfrak{a})\frac{\log(\psi_f(\delta_{\mathfrak{a}}))}{h_f} \\ = \sum_{j=0}^{s} \lambda_{j,g}(a + b\alpha_g)\log(\psi_g(v_i^*)) + \sum_{\mathfrak{a}\in\mathcal{F}_g} e(\mathfrak{b})\frac{\log(\psi_g(\delta_{\mathfrak{b}}))}{h_g} \pmod{\ell}, \quad (2.9)\end{aligned}$$

where $\{u_i^*\}$ and $\{v_j^*\}$ are some units in K_f and K_g respectively and $\lambda_{i,f}$ and $\lambda_{j,g}$ are easily computable functions called Schirokauer maps modulo ℓ for K_f and K_g respectively. We compute more than $\#\mathcal{F} + r + s + 2$ such independent relations by randomly trying with different polynomials $T(X)$. Note that we have taken $T(X)$ to be linear here, but nothing stops us from making it quadratic or cubic, such a variant of NFS is termed as NFS-HD.

Recall that a random polynomial $T(X) = a + bX$ will give a relation if the principal ideals $(a+b\alpha_f)\mathcal{O}_f$ and $(a+b\alpha_g)\mathcal{O}_g$ are respectively $\mathcal{F}_f$ and $\mathcal{F}_g$-smooth. The ideal $(a+b\alpha_f)\mathcal{O}_f$ is $\mathcal{F}_f$-smooth provided the quantity $\mathrm{Res}(f(X), a+bX) \in \mathbb{Z}$ is B-smooth i.e., all its prime divisors are less than or equal to B. Similar things hold for $(a + b\alpha_g)\mathcal{O}_g$. In other words, the probability that a randomly chosen $T(X)$, with coefficients in $[-A, A]$, gives us a relation, is the same as the probability that the quantities

$$|\mathrm{Res}(f(X), a + bX)| \text{ and } |\mathrm{Res}(g(X), a + bX)| \quad (2.10)$$

are B-smooth. Heuristically, B-smoothness behaviour of the quantities given in Equation (2.10) is assumed to be similar to that of a random integer of same size. This heuristic assumption is not precise and the imprecision is captured by the quantity called Murphy α-value, which we explain later in the Section 4 but for now we will go by the assumption. The quantity in Equation (2.10) is approximated by [28, 12]

$$|\mathrm{Res}(f(X), a + bX)| \times |\mathrm{Res}(g(X), a + bX)| \approx \|f\|_\infty \|g\|_\infty A^{\deg(f)+\deg(g)}, \quad (2.11)$$

where Res represents the resultant, and $\|f\|_\infty$ and $\|g\|_\infty$ represent the maximum of the absolute values of the coefficients of $f(X)$ and $g(X)$ respectively. The lower the value of product of resultants, given in Equation (2.11), the higher the chance of its B-smoothness and hence the chance of getting a relation. This is the reason why in the polynomial selection phase, we try to minimise the degrees and the absolute values of polynomial coefficients. In practice, sieving techniques [14, 20, 18, 21] are used for getting the polynomials $T(X)$'s which are most likely to be smooth and then the actual factorisation is used for constructing the relations.

2.3 Linear Algebra

The relation collection step provides with us a sparse system of linear equations with unknowns, the virtual logarithms of factor base elements and logarithms of $(r+s+2)$ field elements corresponding to units $\{u_i^*\}_{i=0}^{r}$ and $\{v_i^*\}_{i=0}^{s}$, which is valid modulo a prime ℓ. The system is solved modulo ℓ using the Lanczos or Block-Wiedemann algorithm. The cost of solving this system depends on the number of unknowns and hence roughly on the factor-base bound B.

2.4 Individual Discrete Logarithm and Final Value

The aim of this step is to compute the actual discrete logarithm of a target element $\tau(\omega)$ of the field $\mathbb{F}_{p^n}$. Suppose $\left(\mathbb{F}_{p^n}^{\star}, \cdot\right) = \langle \zeta(\omega) \rangle$. Let

$$p^n - 1 = \prod_{i=0}^{E} p_i^{e_i}, \text{ where } p_i\text{'s are prime and } p_0 < p_1 < \ldots < p_E \ (= \ell, \text{ say}).$$

If the discrete logarithms modulo $p_i^{e_i}$ for $i = 1, \ldots, E$ are known, the actual logarithm can be built using the Chinese Remainder Theorem. For a small factor $m = \prod_{i=0}^{j-1} p_i^{e_i}$ of $p^n - 1$, the Pollard's rho algorithm can be used for computing discrete logarithm modulo m. For the remaining prime factors $p_i, i = j, j+1, \ldots, E$, the discrete logarithm modulo p_i is computed first, and then using Hensel lifting, the logarithm modulo $p_i^{e_i}$ is computed. In most cases, the exponent e_j equals 1 for the larger prime divisors p_j of $p^n - 1$, so the Hensel lifting is rarely needed. Thus the computation of discrete logarithm finally boils down to the computation of discrete logarithm modulo a few larger prime factors of $p^n - 1$ and this is the basic aim of the individual discrete logarithm phase of the NFS algorithm. Let ℓ be one such prime factor. Moreover we assume that the prime ℓ divides $\Phi_n(p)$, that is, the subgroup cannot be embedded in a proper subfield of $\mathbb{F}_{p^n}$.

For a target element $\tau(\omega)$, we look for a field element of the form $\tau(\omega)^{m_1} \cdot \zeta(\omega)^{m_2}$ for $m_1, m_2 \in \mathbb{Z}^+$, such that $\tau(X)^{m_1} \cdot \zeta(X)^{m_2}$ is smooth either in K_f or in K_g. In contrast with the relation collection, we don't need it to be smooth in both the number fields and by the abuse of notation, we also call it smooth. If it is smooth in K_f, similar to Equation (2.6), we get

$$\log\left(\tau(\omega)^{m_1} \cdot \zeta(\omega)^{m_2}\right) = \sum_{i=0}^{r} \lambda_{i,f} \log\left(\psi_f(v_i')\right) + \sum_{\mathfrak{a} \in \mathcal{F}_f} e(\mathfrak{a}) \frac{\log\left(\psi_f(\delta_{\mathfrak{a}})\right)}{h_f} \pmod{\ell}.$$

On simplification, we get

$$\log\left(\zeta(\tau)\right) = \frac{1}{m_1}\left(-m_2 + \sum_{i=0}^{r} \lambda_{i,f}\log\left(\psi_f(v_i')\right) + \sum_{\mathfrak{a}\in\mathcal{F}_f} e(\mathfrak{a})\frac{\log\left(\psi_f(\delta_{\mathfrak{a}})\right)}{h_f}\right) \pmod{\ell}. \tag{2.12}$$

The target logarithm is obtained by substituting the values of virtual logarithms (which are already available from the linear algebra) on the right side of Equation (2.12). For more details on how an ideal $\left(\tau(\alpha_f)^{m_1}\cdot\zeta(\alpha_f)^{m_2}\right)\mathcal{O}_f$ is written in term of the elements of $\mathcal{F}_f$, we refer to the papers [25, 17]. This step is much less costly than Relation Collection and Linear Algebra.

2.5 Polynomial Selection Algorithms

The polynomial selection plays an important role in determining the cost of NFS algorithm. The basic aim of polynomial selection is to find two irreducible integer polynomials having a common irreducible factor of degree n modulo p. As seen in the relation collection step, in addition to the basic property, coefficient size (the infinity norm) and degrees of these polynomials should also be small. We broadly classify the polynomial selection algorithms into the following three types:

1. JLSV methods;
2. Joux-Pierrot (JP) method;
3. Sarkar-Singh (SS) method.

2.5.1 JLSV

There are three variants of it namely JLSV0, JLSV1 and JLSV2. In JLSV0, a random irreducible polynomial $f(X)$, with $\|f(X)\|_\infty = O(1)$ is chosen and $g(X) = f(X)+p$. In JLSV1, $f(X)$ is an irreducible polynomial randomly chosen as $f(X) = f_1(X) + u$, where $u \approx \sqrt{p}$ and $f_1(X)$ is a random polynomial of degree n with $\|f_1(X)\|_\infty = O(1)$, and $g(X) = u_2 f(X) + u_1$ where $u_1/u_2 \equiv u \pmod{p}$. We will skip the details of JLSV2, as it is never better than other existing ones, in performance.

2.5.2 Joux-Pierrot Method

This is the best polynomial selection algorithm for special primes. A prime p is said to be special if there exists a polynomial $\Gamma(X) \in \mathbb{Z}[X]$ of degree m with $\|\Gamma(X)\|_\infty = O(1)$ and a positive integer $u \approx p^{1/m}$ such that $\Gamma(u) \equiv 0 \pmod{p}$. The algorithm works as follows:

- Randomly select a monic polynomial $f_1(X) \in \mathbb{Z}[X]$ of degree n with $\|f_1(X)\|_\infty = O(1)$ such that $f(X) = f_1(X) - u \in \mathbb{Z}[x]$ is irreducible.
- Set $g(X) = \mathrm{Res}_U\left(\Gamma(U), U - f_1(X)\right)$.

The above process is repeated until both $f(X)$ and $g(X)$ are irreducible. Also note that $g(X) = \Gamma(f(X)+u) \equiv 0 \pmod{p, f(X)}$ and thus the basic requirement is satisfied.

2.5.3 Sarkar-Singh Method

This method is parameterised on a divisor d of n. Let $k = n \div d$ and let $r \geq k$. The following is repeated until $f(X)$ and $g(X)$ are irreducible over $\mathbb{Z}$ and $\phi(X)$ is irreducible modulo p:

1. A random irreducible polynomial $A(X)$ of degree $(r+1)$ is selected such that $\|A(X)\|_\infty = O(\log p)$ and $A(X)$ has an irreducible factor $A_1(X)$ of degree k modulo p.
2. A lattice $L \subset \mathbb{R}^{r+1}$ is constructed from the coefficients of polynomials $\{p\,X^0, p\,X^1, \ldots, p\,X^{k-1}, X^0 A_1(X), X^1 A_1(X), X^2 A_1(X), \ldots, X^{r-k} A_1(X)\}$.

   Let $(b_0, b_1, \cdots, b_r)$ be the smallest vector in the LLL-reduced basis of this lattice and let $B(X) = \sum b_i X^i$.
3. Two monic polynomials $C_0(X)$ and $C_1(X)$ with small coefficients such that $\deg(C_0(X)) = d$ and $\deg(C_1(X)) < d$ are randomly chosen and

$$\begin{aligned} f(X) &= \mathrm{Res}_U\left(A(U), C_0(X) + UC_1(X)\right); \\ g(X) &= \mathrm{Res}_U\left(B(U), C_0(X) + UC_1(X)\right); \\ \phi(X) &= \mathrm{Res}_U\left(A_1(U), C_0(X) + UC_1(X)\right) \pmod{p}. \end{aligned}$$

The generalised Joux-Lercier (gJL) and the Conjugation methods [7] of polynomial selection are the special cases of Sarkar-Singh method corresponding to $d = 1$ and $d = n, r = n \div d$ respectively. Note that the Conjugation method gives the lowest asymptotic complexity for the specific values of p. However there are the values of p where Sarkar-Singh polynomial selection method turns out to be better than Conjugation and generalised Joux-Lercier methods. From now on we will not consider the gJL and Conjugation methods separately and instead include them into Sarkar-Singh type algorithms.

2.6 Asymptotic Complexities

The Number Field Sieve is a complex algorithm. It is very difficult to work out the concrete cost of this algorithm. However, its asymptotic complexity analysis is comparatively easier and is based on heuristics. It is customary to use the following sub-exponential expression in the asymptotic complexity analysis:

$$L_Q(a, c) = \exp\Big((c + o(1))\,(\ln Q)^a\,(\ln\ln Q)^{1-a}\Big).$$

For a finite field $\mathbb{F}_Q$ where $Q = p^n$, we write $p = L_Q(a, c_p)$. In the complexity analysis, we mainly focus on the most expensive steps of NFS namely relation collection and linear algebra steps. The bound B and A are chosen in such a way that the costs of these steps turn out to be same, and the sum of the costs is referred as the overall asymptotic cost of the algorithm.

The field $\mathbb{F}_Q$ is classified as small characteristic, if $a < 1/3$; medium characteristic, if $1/3 \leq a < 2/3$ and large characteristic, if $a > 2/3$. The case $a = 2/3$ is referred as boundary case. In the small characteristic case the FFS algorithm [1, 2] and its QPA variants [8, 19] are currently the state-of-the-art and we will not discuss them in this paper. For the remaining cases, the complexity analysis is classified into two types: special prime and general prime.

General Primes

For general primes, Sarkar-Singh type algorithms are the ones which provide the best heuristic asymptotic complexities for boundary to large characteristic cases. For boundary case the asymptotic cost of NFS, is given by

$$\mathrm{Cost}_{\texttt{NFSb}} = L_Q\left(1/3, c_b\right) \text{ where } c_b = 2\left(\frac{2r+1}{3c_pkt_s} + \sqrt{\left(\frac{2r+1}{3c_pkt_s}\right)^2 + \frac{kc_p(t_s-1)}{3(r+1)}}\right),$$

where the various parameters are the one used in the description of SS algorithm and t_s is taken to be the sieving dimension i.e., $(t_s - 1)$ is the degree of $T(X)$. Note that in the description of NFS, we have taken $T(X)$ to be linear i.e., $t_s = 2$. The minimum cost is obtained for $p = L_Q\left(2/3, (12)^{1/3}\right)$ and which is equal to $L_Q\left(1/3, (48/9)^{1/3}\right)$, corresponding to $t_s = 2$ and $r = 1$. The asymptotic cost for large prime case, i.e. for $p = L_Q(a, c_p)$ where $a > 2/3$, turns out to be

$$\mathrm{Cost}_{\texttt{NFSl}} = L_Q\left(1/3, (64/9)^{1/3}\right).$$

For the details of how they are obtained, we refer to the papers [40, 7, 9].

Special Primes

All the polynomial selection algorithms mentioned for general primes are also applicable to special primes but they do not provide the best complexity. It is the JP polynomial selection algorithm which is the state-of-art and the corresponding cost of NFS is given by [27]

$$\begin{aligned}\mathrm{Cost}_{\texttt{SNFSb}} &= L_Q\left(\frac{1}{3}, \left(\frac{32}{9}\right)^{1/3} \cdot \left(\frac{m+1}{m}\right)^{1/3}\right),\\ \mathrm{Cost}_{\texttt{SNFSl}} &= L_Q\left(\frac{1}{3}, \left(\frac{32}{9}\right)^{1/3}\right).\end{aligned}$$

We have not yet mentioned the cost of NFS algorithm for medium characteristic finite fields. This is because the Tower Number Field Sieve (TNFS), a generalisation of NFS, is the best algorithm for it. The TNFS algorithm is described in the Section 3 below.

3 Tower Number Field Sieve Algorithm

It is a generalisation of the NFS algorithm and works exactly in the same way. The crucial difference between the two is in the initial setup. Let $n = \eta \cdot \kappa$ and $h(Y) \in \mathbb{Z}[Y]$ be a monic irreducible polynomial of degree η (when n is prime, $\eta = n$ and $\kappa = 1$) with small coefficients and $h(Y)$ is also irreducible modulo p. Let K_h be a number field generated by $h(Y)$ and let $y := Y \bmod h(Y)$. Suppose $\mathcal{O}_h$ be the ring of algebraic integers of K_h and $\mathfrak{p}$ be a prime ideal above p in $\mathcal{O}_h$. Let $\mathbb{Z}_y := \frac{\mathbb{Z}[Y]}{\langle h(Y)\rangle}$ be a subring of $\mathcal{O}_h$.

With this initial setup, we work with the polynomial ring $\mathbb{Z}_y[X]$ in the same fashion as with $\mathbb{Z}[X]$ in NFS. Two polynomials $f_y(X)$ and $g_y(X)$ in

$\mathbb{Z}_y[X]$ are selected such that they have a common irreducible factor $\phi_y(X)$ of degree κ modulo $\mathfrak{p}$ and this gives the following commutative diagram given in Figure 2 page 11. This can be seen as if we were trying to apply NFS

$$\begin{tikzcd}
 & \mathbb{Z}_y[X] \arrow[dl] \arrow[dr] & \\
\mathbb{Z}_y[X]/(f_y(X)) \arrow[dr, "\text{mod } \mathfrak{p}"'] & & \mathbb{Z}_y[X]/(g_y(X)) \arrow[dl, "\text{mod } \mathfrak{p}"] \\
 & \mathbb{F}_{p^n} &
\end{tikzcd}$$

Figure 2: Commutative diagram for TNFS.

to $\mathbb{F}_{q^\kappa}$ where $q = p^\eta$. Let $K_{y,f}$ and $K_{y,g}$ be the number fields generated by $f_y(X)$ and $g_y(X)$ respectively and $\mathcal{O}_{y,f}$ and $\mathcal{O}_{y,g}$ be their corresponding rings of algebraic integers. Similar to NFS, let $\alpha_{y,f} := X \bmod f_y(X)$, $\alpha_{y,g} := X \bmod (g_y(X))$ and $\ell d(f_y(X))$(respectively $\ell d(g_y(X))$) be the leading coefficient of $f_y(X)$ (respectively $g_y(X)$). The commutative diagram provides us two different ways to map an element, say $T_y[X] = a(y) + b(y)X$, of $\mathbb{Z}_y[X]$ into an element of $\mathbb{F}_{p^n}$. For a given bound B, consider a set $\mathcal{B}_{y,f}$ consisting of degree 1 prime ideals of norm at most B, prime ideals above $\ell d(f_y(X))$ and index ideal $[O_{yf} : \mathbb{Z}_y[\alpha_{yf}]]$ in K_{yf}. The $\mathcal{B}_{y,g}$ is similarly defined and the factor base is set to be $\mathcal{B}_y = \mathcal{B}_{yf} \cup \mathcal{B}_{yg}$.

Next step is to construct about $\#\mathcal{B}_y$ relations and then proceed to the linear algebra. For getting a relation, it is sufficient to look for a value of $a(y)$ and $b(y)$ for which the principal ideals $(a(y) + b(y)\alpha_{y,f})\,\mathcal{O}_{y,f}$ and $(a(y) + b(y)\alpha_{y,g})\,\mathcal{O}_{y,g}$ can be factored into the elements of $\mathcal{B}_{y,f}$ and $\mathcal{B}_{y,g}$ respectively. We call such a polynomial $T_y(X) = a(y) + b(y)X$ to be smooth, in the sense similar to that of NFS. In terms of implementation, testing a polynomial $a(y) + b(y)X$ for smoothness is equivalent to testing the two resultants $\mathrm{Res}_Y(\mathrm{Res}_X(a(Y) + b(Y)X, f_Y(X)), h(Y)) \in \mathbb{Z}$ and $\mathrm{Res}_Y(\mathrm{Res}_X(a(Y) + b(Y)X, g_Y(X)), h(Y)) \in \mathbb{Z}$ for B-smoothness. Modulo the complications caused by units, sufficiently many relations are generated. The linear algebra step remains the same as that of NFS. The individual discrete logarithm phase can be described similar to that of NFS. For more details of it, we refer to the papers [23, 22, 42].

When $\eta = 1$, the TNFS turns out to be same as NFS and in this case $\mathbb{Z}_y = \mathbb{Z}$. It is in this sense that the TNFS is said to be a generalisation of NFS. On the other hand, in TNFS the fields $K_{y,f}$ and $K_{y,g}$ can be seen as tower field extensions i.e., $\mathbb{Q} \to K_h \to K_{y,f}$ and $\mathbb{Q} \to K_h \to K_{y,g}$ and hence the name.

3.1 Polynomial Selection

The polynomial selection step in the TNFS consists of first selecting $h(Y) \in \mathbb{Z}[Y]$ of degree η and then a pair of irreducible polynomials $(f_y(X), g_y(X)) \in \mathbb{Z}_y[X]^2$ such that they have a common irreducible factor $\phi_y(X)$ of degree κ modulo the prime ideal $\mathfrak{p} = \langle p, h(y)\rangle$. For efficiency purpose, the absolute value of coefficients and degree of bi-variate polynomials $f_Y(X)$ and $g_Y(X)$ should be small and this is why, we also need $\|h(Y)\|_\infty$ to be the smallest possible. We divide the polynomial selection algorithms in the following two broad categories:

1. The generalised Singh-Sarkar (GSS) Method (for all primes);
2. The generalised Joux-Pierrot (GJP) Method (for special primes only).

Note that the JLSV methods of polynomial selection can also be used in the TNFS setting but they are not cost effective for the fields relevant to cryptography.

3.1.1 The Generalised Singh-Sarkar (GSS) Method

It is a generalisation of Sarkar-Singh method in the setting of tower number field sieve algorithm. First an irreducible polynomial $h(Y) \in \mathbb{Z}[Y]$, of degree η, is randomly selected such that $\|h(Y)\|_\infty$ is the smallest possible and $h(Y)$ is irreducible modulo p. Let $y := Y \mod h(Y)$ and $\mathbb{F}_{p^\eta} = \mathbb{F}_p[Y]/(h(Y))$. Let d be a divisor of κ, $k = \kappa \div d$ and let $r \geq k$. The following is repeated until $f_y(X)$ and $g_y(X)$ are irreducible over $\mathbb{Z}_y$ and $\phi_y(X)$ is irreducible over $\mathbb{F}_{p^\eta}$:

1. A random irreducible polynomial $A_y(X) \in \mathbb{Z}_y$ of degree $(r+1)$ is selected such that $\|A_Y(X)\|_\infty = O(\log p)$ and over $\mathbb{F}_{p^\eta}$, $A_y(X)$ has an irreducible factor $A_{1y}(X)$ of degree k.
2. A lattice $L \subset \mathbb{R}^{\eta\cdot(r+1)}$ is constructed from the coefficients of bi-variate polynomials
$$\left\{p\, Y^i\, X^j\right\}_{i=0,\ldots,\eta-1}^{j=0,\ldots,k-1} \bigcup \left\{Y^i\, X^j\, A_{1Y}(X)\right\}_{i=0,\ldots,\eta-1}^{j=0,\ldots,r-k}$$
3. Let $(b_{ij})_{i=0,\ldots,\eta-1}^{j=0,\ldots,r}$ be the smallest vector in the LLL-reduced basis of this lattice and let $B_y(X) = \sum b_{ij}\, y^i\, X^j$.
4. Two monic polynomials $C_0(X)$ and $C_1(X)$ in $\mathbb{Z}_y$ having very small infinity norms such that $\deg(C_0(X)) = d$ and $\deg(C_1(X)) < d$ are randomly chosen and
$$\begin{aligned} f_y(X) &= \mathrm{Res}_U\left(A_y(U), C_0(X) + UC_1(X)\right); \\ g_y(X) &= \mathrm{Res}_U\left(B_y(U), C_0(X) + UC_1(X)\right); \\ \phi_y(X) &= \mathrm{Res}_U\left(A_{1y}(U), C_0(X) + UC_1(X)\right) \pmod{p} \end{aligned}$$

Note: The GSS method of polynomial selection presented above is the most general presentation of the algorithm. With different parameters, it gives rise to Sarkar-Singh algorithms $\mathcal{A}, \mathcal{C}, \mathcal{D}$ [37, 38, 39] and the generalised Conjugation method of Jeong and Kim [31]. We will not consider these methods separately.

3.1.2 The Generalised Joux-Pierrot (GJP) Method

This method is applicable to the special primes only. It is a direct application of JP method to the TNFS setting. We assume that there exist an integer $u \approx p^{1/m}$ and a polynomial $\Gamma(U)$ with $\|\Gamma(U)\|_\infty = O(1)$ such that $\Gamma(u) \equiv 0 \pmod{p}$. The algorithm works as follows:

1. A polynomial $f_{1y}(X) \in \mathbb{Z}_y[X]$ of degree κ, with $\|f_{1y}(X)\|_\infty = O(1)$, is randomly selected.
2. $f_y(X) = f_{1y}(X) - u$ and $g_y(X) = \Gamma\left(f_{1y}(X)\right)$.

The above process is repeated until both $f_y(X)$ and $g_y(X)$ are irreducible. The TNFS algorithm along with GJP polynomial selection method is termed as SexTNFS [30] algorithm in the literature but we will simply call it TNFS for notional convenience. On the other hand it is interesting to note that for $\kappa = 2$, two polynomial selection algorithms viz. GSS and GJP turn out to be the same.

3.2 Asymptotic Complexities for Medium-Characteristic Finite Fields

The TNFS algorithm is best suited for the *medium characteristic* finite fields. The asymptotic cost of TNFS for these fields turns out to be similar to what we get for boundary to large characteristic fields using classical NFS. And this holds for special and general primes both.

General Primes. In the setting of GSS method of polynomial selection, if we consider $p = L_Q(a, c_p)$ with $1/3 < a \leq 2/3$ and $\eta = c_\eta(\ln Q/\ln\ln Q)^{2/3-a}$, the run time of TNFS algorithm is given by

$$\text{Cost}_{\texttt{TNFSm}} = L_Q\left(1/3, c_b\right) \tag{3.1}$$

where $c_b = 2\left(\frac{2r+1}{3c_pc_\eta kt_s} + \sqrt{\left(\frac{2r+1}{3c_pc_\eta kt_s}\right)^2 + \frac{kc_pc_\eta(t_s-1)}{3(r+1)}}\right)$.

Special Primes. For $p = L_Q(a, c_p)$ with $1/3 < a \leq 2/3$, considering the parameters of GJP polynomial selection method, we get the run time of the TNFS algorithm as

$$\text{Cost}_{\texttt{STNFSm}} = L_Q\left(\frac{1}{3}, \left(\frac{32}{9}\right)^{1/3}\right).$$

For more details on how it is derived, we refer to the papers [30, 31].

3.3 Galois Automorphism

The polynomial selection algorithms presented in the Sections 2.5 are randomised in nature. For a fixed set of parameters, each run of them outputs a new polynomial pair with similar properties (degree and infinity norm) and there is no reason to believe their behaviour to be same. If we somehow ensure that the number field K_f, corresponding to an NFS polynomial $f(X)$, has non-trivial automorphisms $\text{Aut}_{\mathbb{Q}}(K_f)$, the associated factor base $\mathcal{F}_f$ can be made $\#\text{Aut}_{\mathbb{Q}}(K_f)$ times smaller. Moreover $\text{Aut}_{\mathbb{Q}}(K_f)$ is a cyclic group and $\#\text{Aut}_{\mathbb{Q}}(K_f)$ divides $\deg(f)$. For more detail, we refer to the Section 4.3 of the paper [26]. The same is true for $g(X)$ as well. Such choices of NFS polynomials are possible, thanks to the work of Foster [16] which provides a list of such polynomials with degrees equal to $2, 3, 4, 5$ or 6. In the polynomial selection methods, e.g. JLSV0 and JLSV1, where one of the polynomials is selected as a random polynomial of degree n and the other is derived from it, it is very easy to have automorphisms for the first polynomial. All we need is to select the first polynomial randomly from the list suggested by Foster. However, it is not possible to have Galois automorphisms with GJL-NFS. If K_f has cyclic Galois group, then f cannot have an irreducible factor of degree $\kappa > 1$. Either f is irreducible mod p or it splits completely into degree 1 factors. On the other hand, in the Sarkar-Singh polynomial selection method, automorphisms of order d can be obtained for both the polynomials.

The case of TNFS is tricky. We have three polynomials namely $h(Y)$, $f_y(X)$ and $g_y(X)$. It is possible to select $h(Y)$ from the list suggested by Foster and hence the automorphism set $\mathrm{Aut}_{\mathbb{Q}}(K_h)$ has size $\deg(h)$. At the same time, it is also possible to have non-trivial automorphisms $\mathrm{Aut}_{K_h}(K_f)$ and $\mathrm{Aut}_{K_h}(K_g)$ for $f_y(X)$ and $g_y(X)$ respectively, similar to what we get in classical NFS. The effect of automorphisms $\mathrm{Aut}_{\mathbb{Q}}(K_h)$ and $\mathrm{Aut}_{K_h}(K_f)$ can be combined and the factor base $\mathcal{F}_f$ can be reduced by a factor of $\#\mathrm{Aut}_{\mathbb{Q}}(K_h) \cdot \#\mathrm{Aut}_{K_h}(K_f)$. The same holds for $\mathcal{F}_g$ as well.

When $\gcd(\kappa, \eta) = 1$ this is easy to set Galois automorphisms for h and f, g because the coefficients of f, g can be in $\mathbb{Z}$ (no y coefficient). When $\gcd(\kappa, \eta) > 1$, the trick from Barbulescu-Duquesne is to choose the coefficients of f, g to be invariant under the automorphisms of K_h. For example, if $\sigma(y) = 1/y$ is an automorphism of K_h, then the trace $y + 1/y$ of σ is invariant by σ.

Polynomials with Galois Automorphism. From [16], these polynomials have a Galois automorphism. We use them when possible (with Sarkar-Singh and Joux-Pierrot) to obtain a speed-up in the relation collection.

- $c_t(X) = X^2 - tX + 1$, $X \mapsto 1/X$; $c_t(X) = X^2 + t$, $X \mapsto -X$;
- $c_t(X) = X^3 - tX^2 - (t+3)X - 1$, $X \mapsto -(X+1)/X$;
- $c_t(X) = X^4 - tX^3 - 6X^2 + tX + 1$, $X \mapsto -(X+1)/(X-1)$;
- $c_t(X) = X^6 - 2tX^5 - (5t+15)X^4 - 20X^3 + 5tX^2 + (2t+6)X + 1$, $X \mapsto -(2X+1)/(X-1)$.

Galois Automorphism for Special Primes. The parameters of pairing-friendly curves are special. For BN curves, the prime p has a polynomial form $\Gamma(U) = 36U^4 + 36U^3 + 24U^2 + 6U + 1$. To obtain a Galois automorphism with f_Y, g_Y, we define $f_Y(X) = \mathrm{Res}_t(c_{tY}(X), \Gamma(t))$ and $g_Y(X) = c_{uY}(X)$. Practical examples are provided in Table 5.

4 Murphy α-value

As pointed out in the Section 3.3, two similar-looking polynomial pairs may differ in their behaviour. In this section, we present a measure due to Murphy, which determines the smoothness behaviour of a polynomial pair suitable for TNFS. We will first present it for classical NFS and then propose its extension for TNFS.

Recall that, in the asymptotic cost analysis of NFS, we approximate the B-smoothness behaviour of $\mathrm{Res}(T(X), f(X))$ with a random integer of similar size. It is not precise and the behaviour differs from polynomial to polynomial. The Murphy's α-value is a measure to capture it. Murphy [34] has suggested to compare the behaviours of the two polynomials locally with respect to small primes. Below we outline Murphy's description of the α-value of the polynomial $f(X)$, which is based on the papers [34, 4, 5].

For a given prime ℓ, let $\mathbf{val}_\ell(z)$ denotes the highest power of ℓ dividing the integer z. Let $\mathbf{V}$ be a random variable representing $\mathbf{val}_\ell$. We note that $\mathbf{V}$ takes its values in $\mathbb{Z}^+ \cup \{0\}$. Expectation of $\mathbf{V}$ is given by

$$\begin{aligned}\nu_\ell(\mathbb{Z}) = \mathrm{Exp}(\mathbf{V}) &= \sum_{v=0}^{\infty} v \cdot \mathbf{Pr}(\mathbf{V} = v)\\ &= \sum_{v=1}^{\infty} \mathbf{Pr}(\mathbf{V} \geq v)\\ &= \mathbf{Pr}(\mathbf{V} \geq 1) + \mathbf{Pr}(\mathbf{V} \geq 2) + \mathbf{Pr}(\mathbf{V} \geq 3) \ldots\\ &= \frac{1}{\ell} + \frac{1}{\ell^2} + \frac{1}{\ell^3} + \ldots = \frac{1}{\ell - 1} \end{aligned} \tag{4.1}$$

where $\mathbf{Pr}$ is the asymptotic probability.

Note that in the computation of expectation above, we have tacitly used the notion of asymptotic probabilities. This is done to avoid the mathematical fallacy caused by countably infinite sample space $\mathbb{Z}$. Asymptotic probability is the limit of probabilities over the sequence of closed ball of radius r, as r tends to infinity. For more rigorous mathematical setup, we refer the readers to the paper [10] by Barbulescu and Lachand.

The value of ν_ℓ gives an idea of the expected prime power, ℓ^{ν_ℓ}, contained in a random integer. We next define similar such expectation for the integers coming from the resultant $\mathrm{Res}(T(X), f(X))$ i.e.,

$$\nu_\ell(f) = \sum_{i=1}^{\infty} \mathbf{Pr}(\mathbf{val}_\ell(\mathrm{Res}(T(X), f(X))) \geq i). \tag{4.2}$$

Note that in Equation (4.2), the domain of random variable $\mathbf{V}$ is the set of sieving polynomials i.e., $T(X)$, whereas in Equation (4.1), the domain is a set of integers. Based on these expectations, Murphy defined the local value of α, i.e., α_ℓ as follows:

$$\begin{aligned}\alpha_\ell &= \log(\ell) \cdot (\nu_\ell(\mathbb{Z}) - \nu_\ell(f))\\ &= \log(\ell) \cdot \left(\frac{1}{\ell - 1} - \nu_\ell(f)\right) \ . \end{aligned} \tag{4.3}$$

For a given bound B, the α-value for the polynomials $f(x)$ is defined as the sum of local values of α for primes ℓ less than B.

$$\alpha(f, B) = \sum_{\ell \text{ prime},\ \ell < B} \alpha_\ell. \tag{4.4}$$

The value of $\alpha(f)$ indicates the B-smoothness benefit of $\mathrm{Res}(T(X), f(X))$ over a random integer of similar size, in the logarithmic scale. It is clear from the definition of $\alpha(f)$ that for negative values of α, the resultants start behaving better than random integers of same size (they have more small prime factors).

The α-value of $g(X)$ is similarly computed and the final suitable choice of polynomials $f(X)$ and $g(X)$ is made based on the Murphy-E function defined below:

$$\begin{aligned} E(f, g, A, B, Q) = \int_{\mathrm{Coeff}(T)\in A} & \rho\left(\frac{\log|\mathrm{Res}(T(X), f(X))| - Q + \alpha(f)}{\log(B)}\right) \\ & \cdot\rho\left(\frac{\log|\mathrm{Res}(T(X), f(X))| + \alpha(g)}{\log(B)}\right) \end{aligned} \tag{4.5}$$

where ρ is Dickman's rho function, A is the sieving bound and Q is the average size of special-$\mathfrak{q}$s on the f-side. For more details of Murphy-E function, we refer to the papers [34, 4, 5]. The values of $\alpha(f)$ and $\alpha(g)$ are required for the estimation of Murphy-E function and the computation of α-values boils down to the valuation of $\nu_\ell(f)$ and $\nu_\ell(g)$ for all primes ℓ less than B.

4.1 Classical 2-dimension $\alpha(f)$

When the degree of the sieving polynomial is 1, i.e. $T(X) = a + bX$, the α-value defined for this case is called two dimensional alpha i.e., $\alpha_{\mathrm{dim}=2}$. The computation of two-dimensional α was studied by Bai, Brent and Thomé in [5] and was implemented in C in the `cado-nfs` software [41] by Bai, Thomé and Zimmermann. We will revisit the details of computing $\alpha_{\mathrm{dim}=2}$ below, as it provides with us a way to extend this concept for TNFS. Our presentation is based on the description given in the paper [5].

For the computation of $\alpha_{\mathrm{dim}=2}$, it is required to compute the probabilities $\mathbf{Pr}(\mathbf{val}_\ell(\mathrm{Res}(a + bX, f(X))) \geq i)\ \forall i$, for a given prime ℓ. Let $F(X, Y)$ be a homogenisation of $f(X)$, then the above probabilities are represented by

$$\mathbf{Pr}(\mathbf{val}_\ell(F(a, b))) \geq i). \tag{4.6}$$

Since,

$$\mathbf{val}_\ell(F(a, b)) = \mathbf{val}_\ell(F(at, bt)),$$

for any t, coprime to ℓ, the suitable pairs of coprime integers satisfying Equation (4.6), correspond to the elements of the zero-dimensional variety on the projective line $\mathbb{P}^1(\mathbb{F}_{\ell^i})$, defined by $F(X, Y)$. The projective line $\mathbb{P}^1(\mathbb{F}_{\ell^i})$ consists of ℓ^i affine points $(a : 1)$ where $a \in \mathbb{F}_{\ell^i}$ and ℓ^{i-1} points at infinity i.e., $(1 : \ell y)$, $y \in [0, \ell^{i-1}]$. Thus we have,

$$\mathbf{Pr}(\mathbf{val}_\ell(F(a, b))) \geq i) = \frac{\#\text{ affine roots } + \#\text{ projective roots}}{\ell^i + \ell^{i-1}} \tag{4.7}$$

Now, it remains to compute the number of affine and the number of projective roots for a given prime ℓ. The affine roots of $F(X, Y)$ are the roots of $f(X)$ modulo ℓ^i and the projective roots are the root 0 of $f_{\mathrm{pro}}(X) := X^{\deg f} \cdot f\left(\frac{1}{X}\right)$ if it exists and its lifts modulo ℓ^i.

Proposition 3. *Let $f(X) \in \mathbb{Z}[X]$ and ℓ be a prime integer. Let $\iota = \mathbf{val}_\ell\,(\mathrm{Disc}(f))$.*

1. **Simple Roots:** *If $f(a) \equiv 0 \pmod{\ell^j}$ and $f'(a) \not\equiv 0 \pmod{\ell}$, then a can be uniquely lifted to a root of $f(X)$ modulo ℓ^{j+1} for $j \geq 1$.*

2. **Multiple Roots:** *If $f(a) \equiv 0 \pmod{\ell^j}$ and $f'(a) \equiv 0 \pmod{\ell}$, then a root a of $f(X)$ modulo ℓ^j, either lifts to ℓ roots $\{(a + t\,\ell^j) \colon t \in [0, \ell)\}$ modulo ℓ^{i+1} or does not lift modulo ℓ^{i+1}, depending on whether $f(a)$ is 0 modulo ℓ^{j+1} or not. Moreover whenever $j > \iota$, a collection of ℓ^τ solutions modulo ℓ^j give rise to ℓ^τ solutions modulo ℓ^{j+1}.*

Proof. For proof, we refer to the Section 2.6 of the book [35]. □

The Proposition 3 gives a way to compute the number of affine and projective roots modulo increasing powers of ℓ. For example, for a prime ℓ for which $\mathbf{val}_\ell\,(\mathrm{Disc}(f)) = 0$, there will not be any multiple roots. Let n_ℓ^{aff} (respectively n_ℓ^{pro}) be the number of distinct roots of $f(X)$(respectively $f_{\mathrm{pro}}(X)$) modulo ℓ. Then,

$$\begin{aligned}
\nu_\ell(f) &= \sum_{i=1}^{\infty} \frac{\left(n_\ell^{\mathrm{aff}} + n_\ell^{\mathrm{pro}}\right)}{\ell^i + \ell^{i-1}} \\
&= \frac{\left(n_\ell^{\mathrm{aff}} + n_\ell^{\mathrm{pro}}\right)}{\ell + 1} \sum_{i=1}^{\infty} \frac{1}{\ell^{i-1}} \\
&= \frac{\left(n_\ell^{\mathrm{aff}} + n_\ell^{\mathrm{pro}}\right)}{\ell + 1} \left(\frac{\ell}{\ell - 1}\right)
\end{aligned}$$

If ℓ is a bad prime, i.e., the prime for which $\mathbf{val}_\ell\,(\mathrm{Disc}(f)) = \iota\ (\neq 0)$, in this case in addition to some simple roots (as above), there will be multiple roots as well. The behaviour of the multiple roots are not very coherent but thanks to the Proposition 3, we have only to compute the number of roots modulo $(\iota+1)^{\mathrm{th}}$ power of ℓ. Let n_ℓ^{sim} be the sum of number of affine and projective simple roots, and $m_{\ell,i}^{\mathrm{aff}}$ (respectively $m_{\ell,i}^{\mathrm{pro}}$) represents the number of affine (respectively projective) multiple roots of $f(X)$ modulo ℓ^i, then

$$\begin{aligned}
\nu_\ell(f) &= \left(\frac{n_\ell^{\mathrm{sim}}}{\ell+1}\right)\left(\frac{\ell}{\ell-1}\right) + \sum_{i=1}^{\iota} \frac{\left(m_{\ell,i}^{\mathrm{aff}} + m_{\ell,i}^{\mathrm{pro}}\right)}{\ell^i + \ell^{i-1}} + \sum_{j=1}^{\infty} \frac{\left(m_{\ell,\iota+1}^{\mathrm{aff}} + m_{\ell,\iota+1}^{\mathrm{pro}}\right)}{\ell^{\iota+j} + \ell^{\iota+j-1}} \\
&= \left(\frac{n_\ell^{\mathrm{sim}}}{\ell+1}\right)\left(\frac{\ell}{\ell-1}\right) + \sum_{i=1}^{\iota} \frac{\left(m_{\ell,i}^{\mathrm{aff}} + m_{\ell,i}^{\mathrm{pro}}\right)}{(\ell+1)\ell^{i-1}} + \frac{\left(m_{\ell,\iota+1}^{\mathrm{aff}} + m_{\ell,\iota+1}^{\mathrm{pro}}\right)}{(\ell+1)\ell^{\iota}}\left(\frac{\ell}{\ell-1}\right)
\end{aligned} \tag{4.8}$$

The software cado-nfs provides a SageMath and a C implementation of the α function for NFS. It uses a recursive lifting process of the roots modulo bad primes that we describe in Algorithms A.1 and A.2 in Appendix A. We will use the same strategy but with prime ideals instead of prime numbers for lifting roots modulo bad prime ideals in the TNFS setting.

4.2 Extension of the Murphy-α Value to the TNFS

In this section, we will propose an extension of the concept of the Murphy α value to the TNFS polynomials and this is very much needed to make TNFS practical. Consider the tower $\mathbb{Q} \to K_h \to K_f$ in the TNFS set-up. Recall that K_h is a number field generated by $h(Y) \in \mathbb{Z}[Y]$, $\mathbb{Z}_y := \mathbb{Z}[Y]/(h(Y))$ is a subring of $\mathcal{O}_h$, the ring of algebraic integers of K_h, and $f(X) \in \mathbb{Z}_y[X]$ is an irreducible polynomial. We aim to formulate the concept of Murphy α-value for a TNFS polynomial $f_y(X)$.

Similar to the classical case, it would be enough to define the local value of α i.e., $\alpha_\ell(f_y)$ for a given prime integer ℓ. Given a prime integer ℓ, it would be logical to consider the prime ideals above ℓ in $\mathcal{O}_h$ and work with them. Let $\mathfrak{l}$ be a prime ideal above ℓ and $\mathfrak{f}(\mathfrak{l}, \ell)$ be its relative degree. Our plan is the following. We will first find the expected valuation of the resultant $\mathrm{Res}((a(y) + b(y)\,X, f_y(X)))$, with respect to the prime ideal $\mathfrak{l}$, and then we will bring down the result with respect to the prime integer ℓ.

Similar to Equation (4.2), we can define

$$\nu_{\mathfrak{l}}(f) = \sum_{i=1}^{\infty} \mathbf{Pr}(\mathbf{val}_{\mathfrak{l}}(\mathrm{Res}((a(y) + b(y)\,X, f_y(X))) \geq i)\ . \tag{4.9}$$

Let $F_y(X, Z)$ be the homogeneous equation corresponding to $f_y(X)$. The Equation (4.9) can now be written as

$$\nu_{\mathfrak{l}}(f_y) = \sum_{i=1}^{\infty} \mathbf{Pr}(\mathbf{val}_{\mathfrak{l}}(F_y(a(y), b(y))) \geq i)\ . \tag{4.10}$$

We can now define the expected valuation with respect to the prime integer ℓ as follows:

$$\nu_\ell(f_y) = \sum_{\mathfrak{l}|\ell} \mathfrak{f}(\mathfrak{l}, l) \times \nu_{\mathfrak{l}}(f_y)\ . \tag{4.11}$$

Thus a suitable Murphy-α value for a TNFS polynomial $f_y(X)$ can be defined as follows:

$$\begin{aligned} \alpha_\ell(f_y) &= \log(\ell) \cdot (\nu_\ell(\mathbb{Z}) - \nu_\ell(f_y)) && (4.12) \\ &= \log(\ell) \cdot \left(\frac{1}{\ell - 1} - \nu_\ell(f_y)\right) && (4.13) \end{aligned}$$

It still remains to compute the probabilities $\mathbf{Pr}(\mathbf{val}_{\mathfrak{l}}(F_y(a(y), b(y))) \geq i)$ for each $\mathfrak{l}$ above ℓ. The suitable pairs $(a(y), b(y))$ of coprime algebraic integers, for which $\mathbf{val}_{\mathfrak{l}}(F_y(a(y), b(y))) \geq i$, correspond to the roots of $F_y(a(y), b(y))$ on the projective line $\mathbb{P}^1\left(\frac{\mathcal{O}_h}{\mathfrak{l}^i}\right)$. The projective line $\mathbb{P}^1\left(\frac{\mathcal{O}_h}{\mathfrak{l}^i}\right)$ consists of $\mathrm{N}\left(\mathfrak{l}^i\right) := \left|\frac{\mathcal{O}_h}{\mathfrak{l}^i}\right|$ affine points and $\mathrm{N}\left(\mathfrak{l}^{i-1}\right)$ projective points. Affine roots of $F_y(a(y), b(y))$ are the zeros of $f_y(X)$ modulo $\mathfrak{l}^i$ and if ℓ divides the leading coefficient of F_y, the projective roots of $F_y(a(y), b(y))$ are the lifts of the root 0 of $f_{y_{\mathrm{pro}}}(X) := X^{\deg f_y(X)} \cdot f_y\left(\frac{1}{X}\right)$ modulo $\mathfrak{l}^i$. Thus,

$$\mathbf{Pr}(\mathbf{val}_{\mathfrak{l}}(F_y(a(y), b(y))) \geq i) = \frac{\#\text{ affine roots of } F_y\ + \#\text{ projective roots of } F_y}{\mathrm{N}\left(\mathfrak{l}^i\right) + \mathrm{N}\left(\mathfrak{l}^{i-1}\right)}$$

The number of affine and projective roots of F_y can be obtained using the Proposition 4.

Proposition 4. *Let $f_y(X) \in \mathcal{O}_h[X]$ and $\mathfrak{l}$ be a prime ideal in $\mathcal{O}_h$. Let $\iota_y = \mathbf{val}_{\mathfrak{l}}\left(\mathrm{Disc}_y(f_y(X))\right)$.*

Simple Roots: *If $f_y(\mathfrak{a}) \equiv 0 \pmod{\mathfrak{l}^j}$ and $f'_y(\mathfrak{a}) \not\equiv 0 \pmod{\mathfrak{l}}$, then $\mathfrak{a}$ can be uniquely lifted to a root of $f_y(X)$ modulo $\mathfrak{l}^{j+1}$ for $j \geq 1$.*

Multiple Roots: *If $f_y(\mathfrak{a}) \equiv 0 \pmod{\mathfrak{l}^j}$ and $f'_y(\mathfrak{a}) \equiv 0 \pmod{\mathfrak{l}}$, then the root $\mathfrak{a}$ of $f_y(X)$ modulo $\mathfrak{l}^j$, either lifts to $\left|\frac{\mathcal{O}_h}{\mathfrak{l}}\right|$ roots modulo $\mathfrak{l}^{j+1}$ or does not lift modulo $\mathfrak{l}^{j+1}$, depending on whether $f_y(\mathfrak{a})$ is 0 modulo $\mathfrak{l}^{j+1}$ or not. Moreover there exists ι such that whenever $j > \iota$, a collection of m solutions modulo $\mathfrak{l}^j$ give rise to m solutions modulo $\mathfrak{l}^{j+1}$.*

Proof. The proof is exactly the same as that of Proposition 3 when adopted to the Algebraic Number Theory setting. $\square$

The Proposition 4 is crucial. It provides us a way to compute the number of affine and projective roots of F_y modulo a power of prime ideal $\mathfrak{l}$. We have two cases:

Case 1: For a prime ideal $\mathfrak{l}$ for which $\mathbf{val}_{\mathfrak{l}}\left(\mathrm{Disc}(f_y)\right) = 0$, there are only simple roots and for each i, the number of simple roots (affine and projective) of $f_y(X)$ modulo $\mathfrak{l}^i$ is same as number of roots modulo $\mathfrak{l}$. Let $n_{\mathfrak{l}}^{\mathrm{aff}}$ (respectively $n_{\mathfrak{l}}^{\mathrm{pro}}$) be the number of distinct roots of $f_y(X)$(respectively $f_{y_{\mathrm{pro}}}(X)$) modulo $\mathfrak{l}$, then

$$\begin{aligned} \nu_{\mathfrak{l}}(f_y) &= \sum_{i=1}^{\infty} \frac{n_{\mathfrak{l}}^{\mathrm{aff}} + n_{\mathfrak{l}}^{\mathrm{pro}}}{\mathrm{N}\left(\mathfrak{l}^i\right) + \mathrm{N}\left(\mathfrak{l}^{i-1}\right)} \\ &= \left(n_{\mathfrak{l}}^{\mathrm{aff}} + n_{\mathfrak{l}}^{\mathrm{pro}}\right) \cdot \sum_{i=1}^{\infty} \frac{1}{\mathrm{N}\left(\mathfrak{l}\right)^i + \mathrm{N}\left(\mathfrak{l}\right)^{i-1}} \\ &= \frac{n_{\mathfrak{l}}^{\mathrm{aff}} + n_{\mathfrak{l}}^{\mathrm{pro}}}{\mathrm{N}\left(\mathfrak{l}\right) + 1} \cdot \frac{\mathrm{N}\left(\mathfrak{l}\right)}{\mathrm{N}\left(\mathfrak{l}\right) - 1}. \end{aligned}$$

Case 2: If $\mathbf{val}_{\mathfrak{l}}\left(\mathrm{Disc}(f_y)\right) = \iota \ (\neq 0)$, we call such prime ideals a bad prime. In this case, the roots of $f_y(X)$ modulo $\mathfrak{l}^i$ could be simple as well as with multiplicity. It turns out that the number of multiple roots modulo $\mathfrak{l}^i$ get fixed for $i \geq \iota + 1$, thanks to the Proposition 4.

Let $n_{\mathfrak{l}}^{\mathrm{sim}}$ be the sum of number of affine and projective simple roots modulo $\mathfrak{l}$, and $m_{\mathfrak{l},i}^{\mathrm{aff}}$ (respectively $m_{\mathfrak{l},i}^{\mathrm{pro}}$) represents the number of affine (respectively projective) multiple roots of $f(X)$ modulo $\mathfrak{l}^i$, then

$$\begin{aligned}\nu_{\mathfrak{l}}(f_y) &= \left(\frac{n_{\mathfrak{l}}^{\mathrm{sim}}}{\mathrm{N}(\mathfrak{l})+1}\right)\left(\frac{\mathrm{N}(\mathfrak{l})}{\mathrm{N}(\mathfrak{l})-1}\right) + \sum_{i=1}^{\iota} \frac{\left(m_{\mathfrak{l},i}^{\mathrm{aff}} + m_{\mathfrak{l},i}^{\mathrm{pro}}\right)}{\mathrm{N}(\mathfrak{l})^{i} + \mathrm{N}(\mathfrak{l})^{i-1}} \\ &\quad + \sum_{j=1}^{\infty} \frac{\left(m_{\mathfrak{l},\iota+1}^{\mathrm{aff}} + m_{\mathfrak{l},\iota+1}^{\mathrm{pro}}\right)}{\mathrm{N}(\mathfrak{l})^{\iota+j} + \mathrm{N}(\mathfrak{l})^{\iota+j-1}} \\ &= \left(\frac{n_{\mathfrak{l}}^{\mathrm{sim}}}{\mathrm{N}(\mathfrak{l})+1}\right)\left(\frac{\mathrm{N}(\mathfrak{l})}{\mathrm{N}(\mathfrak{l})-1}\right) + \sum_{i=1}^{\iota} \frac{\left(m_{\mathfrak{l},i}^{\mathrm{aff}} + m_{\mathfrak{l},i}^{\mathrm{pro}}\right)}{\mathrm{N}(\mathfrak{l})^{i} + \mathrm{N}(\mathfrak{l})^{i-1}} \\ &\quad + \frac{\left(m_{\mathfrak{l},\iota+1}^{\mathrm{aff}} + m_{\mathfrak{l},\iota+1}^{\mathrm{pro}}\right)}{(\mathrm{N}(\mathfrak{l})+1)\,\mathrm{N}(\mathfrak{l})^{\iota}}\left(\frac{\mathrm{N}(\mathfrak{l})}{\mathrm{N}(\mathfrak{l})-1}\right)\end{aligned}$$

Once we have $\nu_{\mathfrak{l}}(f_y)$, we can compute $\nu_\ell(f_y)$.

$$\nu_\ell(f_y) = \sum_{\mathfrak{l}|\ell} \mathfrak{f}(\mathfrak{l}, l) \times \nu_{\mathfrak{l}}(f_y) \tag{4.14}$$

Thus, we can finally compute the value of $\alpha_\ell(f_y)$ using Equation (4.13) and hence the value of $\alpha(f_y, B)$ for a given bound B.

5 Exact Implementation of α

An *exact* implementation of α means an exact algorithm to compute the number of roots modulo bad primes (resp. bad ideals) (the exact number of roots modulo good primes is achieved with classical root computation algorithms over finite fields). The software cado-nfs [41] provides an exact implementation of α for NFS in SageMath and in C from the paper [5]. The other strategy is a Monte-Carlo approximation of bad prime valuation, this is explained in [18], and implemented for NFS-HD in cado-nfs. The function `MurphyAlphaApproximation` in Magma applies this technique. The exact implementation has two advantages: it is exact, and it is much faster (when comparing the Magma approximation and the exact cado-nfs function in SageMath). We refer to [4, § 3.2.3] and [5] for the computation of α in dimension two for NFS. To have a fast ranking of polynomials for TNFS, we first need an exact and fast implementation of α. We take the same approach as in cado-nfs: a recursive lifting process of the roots modulo bad prime ideals. The algorithms and technical details of the cado-nfs implementation are provided in Appendix A. We present here the adaptation to the TNFS setting, and some experimental data. The source code is available at

`https://gitlab.inria.fr/tnfs-alpha/alpha`

5.1 Recursive Lifting Process Modulo Principal Ideals

To compute an exact value of α, we need a lifting process of the multiple roots of f_y modulo bad ideals $\mathfrak{l}$. For principal ideals $\mathfrak{l}$ (above a prime ℓ), we build on the algorithms of the cado-nfs implementation explained in Appendix A. This is our Algorithm 5.1. We now sketch the process, and we provide an explanation in Appendix A. We use the term *root* to denote the set of zeros of a polynomial, but note that modulo bad prime powers, it may happen that a polynomial has more zeros than its degree.

Let $r \in \mathcal{O}_h$ be a multiple affine root of f_y modulo $\mathfrak{l}$. We need to lift r modulo $\mathfrak{l}, \mathfrak{l}^2, \mathfrak{l}^3, \ldots, \mathfrak{l}^\iota$ and determine the minimal ι needed to obtain a simple root. Assume that $\mathfrak{l}$ is principal, and $\gamma \in \mathcal{O}_h$ is a generator of $\mathfrak{l}$. Since $f_y(r) = 0 \bmod \mathfrak{l}$, then $f_y(r + \gamma X) = 0 \bmod \mathfrak{l}$. If we assume that $f_y(r + \gamma X) \neq 0 \bmod \mathfrak{l}^2$, then we solve $f_y(r + \gamma X)/\gamma = 0 \bmod \mathfrak{l}$ (the roots are in $\mathcal{O}_h/\mathfrak{l}$). A solution s gives a lift $r + \gamma s$ modulo $\mathfrak{l}^2$ of r. Now let v be the valuation at $\mathfrak{l}$ of the content of the polynomial $f_y(r + \gamma X)$. It means that $\operatorname{cont}(f_y(r + \gamma X)) = 0 \bmod \mathfrak{l}^v$ and v is the maximum. We can lift r to many roots modulo $\mathfrak{l}^v$:

$$r + c_1\gamma + c_2\gamma^2 + c_3\gamma^3 + \ldots + c_{v-1}\gamma^{v-1} \pmod{\mathfrak{l}^v}$$

and $c_i \in \mathcal{O}_h/\mathfrak{l}$, that is, c_i can take $\#\mathcal{O}_h/\mathfrak{l} = N(\mathfrak{l})$ values: there are $N(\mathfrak{l})^{v-1}$ roots above r. Algorithm 5.2 line 5 adds v to the contribution of roots modulo $\mathfrak{l}$ and proceeds with $f_v = f_y(r + \gamma X)/\gamma^v$. At this point we know that f_y has one root r modulo $\mathfrak{l}$, in other words $m^{\text{aff}}_{\mathfrak{l},1} = \#\{r\} = 1$, and this root lifts to $|\mathcal{O}_h/\mathfrak{l}|^{k-1} = N(\mathfrak{l})^{k-1}$ roots modulo $\mathfrak{l}^k$ for all $2 \leq k \leq v$, in other words, $m^{\text{aff}}_{\mathfrak{l},k} = |\mathfrak{l}/\mathcal{O}_h|^{k-1}$. We need to count the number of roots modulo $\mathfrak{l}^{v+1}$, this is the number of roots s of $f_v = f_y(r + \gamma X)/\gamma^v$ modulo $\mathfrak{l}$. Each root s of f_v modulo $\mathfrak{l}$ gives a lift of the root r modulo $\mathfrak{l}^{v+1}$. Since

$$\gamma^v f_v(X) = f(r + \gamma X)$$

then a root of f_v satisfies

$$\gamma^v f_v(s) = f(r + \gamma s)$$

and since $f_v(s) = 0 \bmod \mathfrak{l}$, then

$$f(r + \gamma s) = 0 \bmod \mathfrak{l}^{v+1}$$

and f has $N(\mathfrak{l})^{v-1}$ roots modulo $\mathfrak{l}^{v+1}$ of the form

$$r + s\gamma + c_2\gamma^2 + \ldots + c_v\gamma^v \in \mathfrak{l}^{v+1},\ \forall c_i \in \mathcal{O}_h/\mathfrak{l}\ .$$

If $f_v'(s) \neq 0 \bmod \mathfrak{l}$ then the lifting process is over: we have $\iota = v+1$, the algorithm accounts for one more root s_i modulo $\mathfrak{l}^{v+1}$ (that is, $m^{\text{aff}}_{\mathfrak{l},v+1} = N(\mathfrak{l})^{v-1}$) and terminates, with $\sum_{k=v+1}^{\infty} m^{\text{aff}}_{\mathfrak{l},k}/N(\mathfrak{l})^{k-1} = \sum_{k=v+1}^{\infty} N(\mathfrak{l})^{v-1}/N(\mathfrak{l})^{k-1} = 1/(N(\mathfrak{l}) - 1)$. The contributions of the roots modulo $\mathfrak{l}$, with $m^{\text{aff}}_{\mathfrak{l},1} = 1$, $m^{\text{aff}}_{\mathfrak{l},k} = N(\mathfrak{l})^{k-1}$ for

$1 \le k \le v$, and $m^{\mathrm{aff}}_{\mathfrak{l},v} = N(\mathfrak{l})^{v-1}$ finally is

$$\begin{aligned}
&\sum_{i=1}^{\iota} \frac{m^{\mathrm{aff}}_{\mathfrak{l},i}}{N(\mathfrak{l})^i + N(\mathfrak{l})^{i-1}} + \sum_{j=1}^{\infty} \frac{m^{\mathrm{aff}}_{\mathfrak{l},\iota+1}}{N(\mathfrak{l})^{\iota+j} + N(\mathfrak{l})^{\iota+j-1}} \\
&\qquad = \frac{1}{N(\mathfrak{l})+1} \left(\sum_{k=1}^{v} \frac{m^{\mathrm{aff}}_{\mathfrak{l},k}}{N(\mathfrak{l})^{k-1}} + \sum_{k=v+1}^{\infty} \frac{m^{\mathrm{aff}}_{\mathfrak{l},v}}{N(\mathfrak{l})^{k-1}} \right) \\
&= \frac{1}{N(\mathfrak{l})+1} \Bigg(\#\{r\} + \sum_{k=2}^{v} \frac{\#\{r + c_1\gamma + \ldots + c_{k-1}\gamma^{k-1} : \forall c_i \in \mathcal{O}_h/\mathfrak{l}\}}{N(\mathfrak{l})^{k-1}} \\
&+ \sum_{k=v+1}^{\infty} \frac{\#\{r + s_1\gamma + \ldots + s_{k-v}\gamma^{k-v} + \ldots + c_{k-1}\gamma^{k-1} : \forall c_i \in \mathcal{O}_h/\mathfrak{l},\ s_i \text{ fixed}\}}{N(\mathfrak{l})^{k-1}} \Bigg) \\
&= \frac{1}{N(\mathfrak{l})+1} \left(1 + \sum_{k=2}^{v} \frac{N(\mathfrak{l})^{k-1}}{N(\mathfrak{l})^{k-1}} + \sum_{k=v+1}^{\infty} \frac{N(\mathfrak{l})^{v-1}}{N(\mathfrak{l})^{k-1}} \right) = \frac{1}{N(\mathfrak{l})+1} \left(v + \frac{1}{N(\mathfrak{l})-1} \right)
\end{aligned} \tag{5.1}$$

```
Algorithm 5.1: average_val_homogeneous_coprime_TNFS ($f_y, \mathrm{Disc}_{f_y}, \mathfrak{l}, N_\mathfrak{l}, K_h, \mathcal{O}_h$)
Input: Irreducible polynomial $f_y \in \mathcal{O}_h[X]$, discriminant
       $\mathrm{Disc}_{f_y} = \mathrm{Disc}(f_y) \in \mathcal{O}_h$, prime ideal $\mathfrak{l} \in \mathcal{O}_h$, norm
       $N_\mathfrak{l} = N(\mathfrak{l}) = \#|\mathcal{O}_h/\mathfrak{l}|$, number field $K_h$, maximal order $\mathcal{O}_h$
Output: $\mathrm{val}_\mathfrak{l}(f_y)$
if ($\mathrm{Disc}_{f_y} + \mathfrak{l}) = \mathcal{O}_h$ then                          $\mathrm{Disc}_{f_y}$ and $\mathfrak{l}$ are coprime
    return number_of_roots($f_y, \mathfrak{l})/(N_\mathfrak{l} - 1) \cdot N_\mathfrak{l}/(N_\mathfrak{l} + 1) = n_{f_y,\mathfrak{l}} N_\mathfrak{l}/(N_\mathfrak{l}^2 - 1)$
else                                                                         bad prime ideal
    if IsPrincipal($\mathfrak{l}$) then                              there exists a generator $\gamma$ of $\mathfrak{l}$
        $\gamma \leftarrow$ Generator($\mathfrak{l}$)
        $v \leftarrow$ average_val_affine_TNFS_Pr($f_y, \mathfrak{l}, \gamma) \cdot N_\mathfrak{l}$                    affine roots
        if $\mathrm{val}_\mathfrak{l}$(LeadingCoefficient($f_y$)) $\ge 1$ then              projective roots
            $v \leftarrow v$+average_val_affine_TNFS_Pr(Reverse($f_y$)($\gamma X$), $\mathfrak{l}, \gamma$)
    else                             more complicated: two generators, $\mathfrak{l} = \langle \delta, \gamma \rangle$
        $(\delta, \gamma) \leftarrow$ Generators($\mathfrak{l}$)
        $v \leftarrow$ average_val_affine_TNFS($f_y, \mathfrak{l}, \delta, \gamma) \cdot N_\mathfrak{l}$
        if $\mathrm{val}_\mathfrak{l}$(LeadingCoefficient($f_y$)) $\ge 1$ then
            $v \leftarrow v$+average_val_affine_TNFS_Bivariate(Reverse($f_y$)($\delta X_1 +$
              $\gamma X_2$), $\mathfrak{l}, \delta, \gamma$)
    $v \leftarrow v/(N_\mathfrak{l} + 1)$
    return $v$
```

5.2 Recursive Lifting Modulo Non-Principal Ideals

Assume that r is a multiple affine root of f_y modulo $\mathfrak{l}$, and $\mathfrak{l}$ is not principal. We want to lift r to a root modulo $\mathfrak{l}^2$. In this case, there is no generator γ of $\mathfrak{l}$. However, we can easily obtain a pair of generators (δ, γ) of $\mathfrak{l}$. A lift of r modulo $\mathfrak{l}^2$ can be expressed as $r + s_1\delta + s_2\gamma$, where $s_1, s_2 \in \mathcal{O}_h$. We need to lift r up to $\mathfrak{l}^\iota$ to obtain simple roots, for a certain ι. First we compute

Algorithm 5.2: `average_val_affine_TNFS_Pr`$(f_y, \mathfrak{l}, \gamma)$

Input: Irreducible polynomial $f_y \in \mathcal{O}_y[X]$, bad principal prime ideal $\mathfrak{l}$ of generator γ

Output: Contribution of affine roots at bad prime ideal $\mathfrak{l}$

```
$v \leftarrow \mathrm{val}_\mathfrak{l}\,\mathrm{cont}(f_y)$
$f_v \leftarrow f_y/\gamma^v$                                        $\gamma$ generator: $\mathfrak{l} = \langle\gamma\rangle$
for $\overline{s}$ in Roots$(f_v \bmod \mathfrak{l})$ do
    if $(f_v' \bmod \mathfrak{l})(\overline{s}) \neq 0$ then                   simple root, end of lifting
        $v \leftarrow v + 1/(N_\mathfrak{l} - 1)$       the lifting pattern stabilises, as in eq. (A.4)
    else                                          multiple root, lifting one more step
        $s \leftarrow \mathtt{lift}_{\mathcal{O}_h}(\overline{s})$                        a lift in $\mathcal{O}_h$ s.t. $s = \overline{s} \bmod \mathfrak{l}$
        $f_2 \leftarrow f_v(s + \gamma X)$                  by construction, $\mathrm{val}_\mathfrak{l}(\mathrm{cont}(f_2)) \geq 1$
        $v \leftarrow v +$ average_val_affine_TNFS_Pr$(f_2, \mathfrak{l}, \gamma)/N_\mathfrak{l}$
return $v$
```

$v = \mathrm{val}_\mathfrak{l}\,\mathrm{cont}(f_y(r+\delta X_1+\gamma X_2))$. Instead of computing the roots of $f_y(r+\gamma X)/\gamma^v$, we compute the roots of the bivariate polynomial $f_1 = f_y(r + \delta X_1 + \gamma X_2)/(d\gamma^v)$ where $d \in \mathbb{Q}$ so that f_1 has integer coefficients (in $\mathcal{O}_h$) and $\mathrm{cont}(f_1) = 1$. A root of f_1 modulo $\mathfrak{l}$ is a pair $(\overline{s_1}, \overline{s_2})$ where $\overline{s_j} \in \mathcal{O}_h/\mathfrak{l}$. The solution $(\overline{s_1}, \overline{s_2})$ can be lifted to (s_1, s_2) where $s_i \in \mathcal{O}_h$, and one has $f_1(s_1\delta + s_2\gamma) \in \mathfrak{l}$, that is, $f_1(s_1\delta + s_2\gamma) = 0 \bmod \mathfrak{l}$. Since $d\gamma^v f_1(X_1, X_2) = f_y(r + \delta X_1 + \gamma X_2)$ where d is coprime to ℓ, we have

$$d\gamma^v f_1(s_1, s_2) = f_y(r + \delta s_1 + \gamma s_2)$$

and since $f_1(s_1, s_2) \in \mathfrak{l}$ and $\gamma^v \in \mathfrak{l}^v$, then $\gamma^v f_1(s_1, s_2) \in \mathfrak{l}^{v+1}$ and

$$f_y(r + \delta s_1 + \gamma s_2) \in \mathfrak{l}^{v+1} \ .$$

At this point, the lifted root can be written $r+(s_1\delta+s_2\gamma)+r_2\mathfrak{l}^2+\ldots+r_v\mathfrak{l}^v$ modulo $\mathfrak{l}^{v+1}$, for any r_i. If $\partial f_1/\partial X_1(s_1, s_2) \neq 0 \bmod \mathfrak{l}$ or $\partial f_1/\partial X_2(s_1, s_2) \neq 0 \bmod \mathfrak{l}$ then the lifting process ends and the contribution of roots is the same as in (5.1). The corresponding algorithms are 5.3 and 5.4.

5.3 Experimental Results

We present the results obtained when computing $\alpha(f_y, h, B)$ for two pairs of polynomials (f_y, h). The polynomials h are Y^2+5 and Y^3+15 of class number 2. The polynomials f in $\mathbb{Z}_y[X]$ were generated with random coefficients, such that their discriminant has many small prime factors. Following [4, § 3.2.3 Table 3.1], we generated 10^8 random vectors $\boldsymbol{a}, \boldsymbol{b}$ of coefficients in $[-A, A]$ and positive leading coefficient, of length the degree of h, such that the ideals made of $\boldsymbol{a}$ and $\boldsymbol{b}$ in the maximal order $\mathcal{O}_h$ are coprime. For each sample $(\boldsymbol{a}, \boldsymbol{b})$, we counted the valuation at all prime ideals $\mathfrak{l}$ above the primes $\ell \leq B = 2000$ of the pseudo-norm (resultants), in other words we computed $\boldsymbol{a}(y) \in \mathbb{Z}_y$, $\boldsymbol{b}(y) \in \mathbb{Z}_y$, and

$$\mathrm{val}_\mathfrak{l}(\mathrm{Res}(\boldsymbol{a}(y) + \boldsymbol{b}(y)x, f_y(x))$$

and obtained the average frequencies over 10^8 samples. When the theoretical valuation $\mathrm{val}_\mathfrak{l}(f_y)$ is smaller than $100/N = 10^{-6}$, sampling 10^8 pairs is not

```
**Algorithm 5.3:** average_val_affine_TNFS($f_y(X), \mathfrak{l}, \delta, \gamma$)
**Input:** Irreducible univariate polynomial $f_y(X) \in \mathbb{Z}_y[X]$, bad non-principal
        prime ideal $\mathfrak{l}$ of generators $(\delta, \gamma)$
**Output:** Contribution of affine roots at bad prime ideal $\mathfrak{l}$
$v \leftarrow \mathrm{val}_\mathfrak{l}\, \mathrm{cont}(f_y)$
$f_v \leftarrow f_y/\gamma^v$                          $\gamma$ is not "the" generator: $\mathfrak{l} = \langle \delta, \gamma \rangle$
$f_v \leftarrow f_v \cdot$ lcm([Denominator($f_{vi}$) : $f_{vi}$ in Coefficients($f_v$)])
                         lcm is coprime to $\ell$, and $\gamma$ is a uniformising parameter
                                   now $f_v \in \mathcal{O}_h[X]$ and $\mathrm{val}_\mathfrak{l}(\mathrm{cont}(f_v)) = 0$
**for** $\overline{s} \in \mathbb{F}_{\ell^{d_\mathfrak{l}}}$ **in** Roots($f_v \bmod \mathfrak{l}$) **do**
    **if** $(f_v' \bmod \mathfrak{l})(\overline{s}) \neq 0$ **then**          simple root, end of lifting
        $v \leftarrow v + 1/(N_\mathfrak{l} - 1)$        the lifting pattern stabilises, as in eq. (A.4)
    **else**                                   multiple root, lifting one more step
        $s \leftarrow$ lift$_{\mathcal{O}_h}(\overline{s})$                  a lift in $\mathcal{O}_h$ s.t. $s = \overline{s} \bmod \mathfrak{l}$
        $f_2 \leftarrow f_v(s + \delta X_1 + \gamma X_2)$        by construction, $\mathrm{val}_\mathfrak{l}(\mathrm{cont}(f_2)) \geq 1$
        $v \leftarrow v +$ average_val_affine_TNFS_Bivariate($f_2, \mathfrak{l}, \delta, \gamma$)$/N_\mathfrak{l}$
**return** $v$
```

```
**Algorithm 5.4:** average_val_affine_TNFS_Bivariate($f_y(X_1, X_2), \mathfrak{l}, \delta, \gamma$)
**Input:** Irreducible bivariate polynomial $f_y(X_1, X_2)$, bad non-principal prime
        ideal $\mathfrak{l}$ of generators $(\delta, \gamma)$
**Output:** Contribution of affine roots at bad prime ideal $\mathfrak{l}$
$v \leftarrow \mathrm{val}_\mathfrak{l}\, \mathrm{cont}(f_y)$
$f_v \leftarrow f_y/\gamma^v$                          $\gamma$ is not "the" generator: $\mathfrak{l} = \langle \delta, \gamma \rangle$
$f_v \leftarrow f_v \cdot$ lcm([Denominator($f_{vij}$) : $f_{vij}$ in Coefficients($f_v$)])
                         lcm is coprime to $\ell$, and $\gamma$ is a uniformising parameter
                              now $f_v \in \mathcal{O}_h[X_1, X_2]$ and $\mathrm{val}_\mathfrak{l}(\mathrm{cont}(f_v)) = 0$
$R \leftarrow \{\,\}$
**for** $(\overline{s_1}, \overline{s_2}) \in (\mathbb{F}_{\ell^{d_\mathfrak{l}}})^2$ **in** Roots($f_v \bmod \mathfrak{l}$) **do**
    **if** $\big(\frac{\partial f_v}{\partial X_1} \bmod \mathfrak{l}\big)(\overline{s_1}, \overline{s_2}) \neq 0$ ***or*** $\big(\frac{\partial f_v}{\partial X_2} \bmod \mathfrak{l}\big)(\overline{s_1}, \overline{s_2}) \neq 0$ **then**      simple root
        $v \leftarrow v + 1/(N_\mathfrak{l} - 1)$                  the lifting pattern stabilises
    **else**                                   multiple root, lifting one more step
        $(s_1, s_2) \leftarrow$ (lift$_{\mathcal{O}_h}(\overline{s_1})$, lift$_{\mathcal{O}_h}(\overline{s_2})$)      a lift in $\mathcal{O}_h$ s.t. $s_i = \overline{s_i} \bmod \mathfrak{l}$
        $R \leftarrow R \cup \{(s_1, s_2)\}$                          $f_v(s_1\delta + s_2\gamma) = 0 \bmod \mathfrak{l}$
Remove from $R$ the duplicate pairs $(s_1', s_2')$ where $s_1'\delta + s_2'\gamma$ generates the same
  ideal in $\mathcal{O}_h$ as another $(s_1, s_2)$
**for** $(s_1, s_2) \in R$ **do**
    $f_2 \leftarrow f_v(s_1 + \delta X_1, s_2 + \gamma X_2)$          by construction, $\mathrm{val}_\mathfrak{l}(\mathrm{cont}(f_2)) \geq 1$
    $v \leftarrow v +$ average_val_affine_TNFS_Bivariate($f_2, \mathfrak{l}, \delta, \gamma$)$/N_\mathfrak{l}$
**return** $v$
```

enough for a comparison. In the other cases, we obtain good confidence in our implementation: the ratio of experimental valuation and exact theoretical valuation is in $[0.99, 1.01]$, in particular for the bad ideals and the projective ideals. Moreover, the proportion of pairs $(\boldsymbol{a}, \boldsymbol{b})$ producing coprime ideals is very close to $\zeta_{K_h}(2)$ (precision 10^{-4}).

5.3.1 Quadratic h, Monic f

We use $h = Y^2 + 5$ of class number 2 and $f_y = X^4 - 3yX^3 - (6y + 1)X^2 - (y + 10)X - 10y$ where y is a root in $\mathbb{C}$ of h. We present the results in Table 1. The experimental ratio of pairs co-prime ideals is 0.53895969, and $\zeta_{K_h}(2) = 0.53892176$ (computed with PARI-GP). Finally we compute $\alpha(h, f_y, 2000) = -1.432$ (in base e), that is, the norms are $1.432/\log(2) = 2.066$ bits smaller compared to random integers of the same size.

$N(\mathfrak{l})$	$\mathfrak{l}$	$\mathrm{val}_\mathfrak{l}$ $\mathrm{Disc}(f_y)$	$\mathrm{val}_\mathfrak{l}(f_y)$	$1/N \sum_{\boldsymbol{a},\boldsymbol{b}}$ $\mathrm{val}_\mathfrak{l}(\mathrm{Res}($ $\boldsymbol{a} + \boldsymbol{b}x, f_y))$	ratio
	bad ideals				
2	$(2, y+1)$	4	$4/3 = 1.3333333$	1.3333729	1.0000297
3	$(3, y+1)$	2	$0 = 0.0000000$	0.0000000	1.0000000
3	$(3, y+5)$	1	$1/4 = 0.2500000$	0.2499631	0.9998523
5	$(5, y)$	2	$5/6 = 0.8333333$	0.8333601	1.0000321
29	$(29, y+13)$	1	$1/30 = 0.0333333$	0.0333405	1.0002147
263	$(263, y+28)$	1	$197/17292 = 0.0113926$	0.0113707	0.9980793
487	$(487, y+344)$	1	$1/488 = 0.0020492$	0.0020496	1.0002048
	good ideals				
7	$(7, y+3)$	0	$7/24 = 0.2916667$	0.2916396	0.9999071
7	$(7, y+4)$	0	$7/48 = 0.1458333$	0.1458752	1.0002874
19^2	(19)	0	$361/130320 = 0.0027701$	0.0027685	0.9994064
23	$(23, y+8)$	0	$23/528 = 0.0435606$	0.0435682	1.0001746
29	$(29, y+16)$	0	$29/420 = 0.0690476$	0.0690197	0.9995954
41	$(41, y+6)$	0	$41/420 = 0.0976190$	0.0976229	1.0000399
41	$(41, y+35)$	0	$41/420 = 0.0976190$	0.0976069	0.9998754
43	$(43, y+34)$	0	$43/1848 = 0.0232684$	0.0232764	1.0003426
47	$(47, y+29)$	0	$47/1104 = 0.0425725$	0.0426022	1.0006992

Table 1: $\mathrm{val}_\mathfrak{l}(f_y)$ for ideals above primes $\ell < 50$, and experimental value for a sampling of $N = 10^8$ pairs of coprime ideals $(\boldsymbol{a}, \boldsymbol{b})$, for $h = Y^2 + 5$ and $f_y = X^4 - 3yX^3 - (6y + 1)X^2 - (y + 10)X - 10y$.

5.3.2 Cubic h, Non-Monic f

We use $h = Y^3 + 15$ of class number 2 and $f = (8y^2 - 8y - 6)X^4 - (11y^2 + 11y - 1)X^3 - (8y^2 - 12y - 9)X^2 - (6y^2 - 10y - 9)X + 9y^2 + 6y + 11$ where y is a root in $\mathbb{C}$ of h. We present the results in Table 2. The experimental ratio of pairs co-prime ideals is 0.55132143, and $\zeta_{K_h}(2) = 0.55133622$ (computed with PARI-GP). Finally we compute $\alpha(h, f_y, 2000) = -2.861$ (in base e), that is, the norms are $2.861/\log(2) = 4.127$ bits smaller compared to random integers of the same size.

We are now equipped with all the values, needed to plug in for the computation of the Murphy-E value (Equation (4.5)). The computation of Murphy-E value

$N(\mathfrak{l})$	$\mathfrak{l}$	$\mathrm{val}_{\mathfrak{l}}$ Disc(f_y)	$\mathrm{val}_{\mathfrak{l}}(f_y)$	$1/N \sum_{\boldsymbol{a},\boldsymbol{b}}$ $\mathrm{val}_{\mathfrak{l}}(\mathrm{Res}(\boldsymbol{a}+\boldsymbol{b}x, f_y))$	ratio
bad ideals					
2^2	$(2, y^2+y+3)$	2	$1/5 = 0.2000000$	0.2000187	1.0000935
3	$(3, y)$	3	$1 = 1.0000000$	0.9998711	0.9998711
7	$(7, y+1)$	2	$1/4 = 0.2500000$	0.2499979	0.9999916
7	$(7, y+2)$	3	$3/8 = 0.3750000$	0.3751308	1.0003488
7	$(7, y+11)$	2	$13/48 = 0.2708333$	0.2708383	1.0000185
283	$(283, y+85)$	1	$1/284 = 0.0035211$	0.0035212	1.0000350
projective ideals					
2	$(2, y+1)$	0	$4/3 = 1.3333333$	1.3334333	1.0000750
17	$(17, y+9)$	0	$17/72 = 0.2361111$	0.2360732	0.9998393
good ideals					
5	$(5, y)$	0	$5/24 = 0.2083333$	0.2082244	0.9994771
11^2	$(11, y^2+6y+3)$	0	$121/14640 = 0.0082650$	0.0082561	0.9989187
13^3	(13)	0	$2197/4826808 = 0.0004552$	0.0004569	1.0038091
17^2	$(17, y^2+8y+13)$	0	$289/83520 = 0.0034602$	0.0034666	1.0018470
19^3	$(19,)$	0	$6859/47045880 = 0.0001458$	0.0001459	1.0007281
23	$(23, y+21)$	0	$23/528 = 0.0435606$	0.0435377	0.9994746
31	$(31, y+17)$	0	$31/240 = 0.1291667$	0.1291838	1.0001323
31	$(31, y+22)$	0	$31/240 = 0.1291667$	0.1291491	0.9998638
37^3	$(37,)$	0	$50653/641431602 = 0.0000790$	0.0000793	1.0041957
41	$(41, y+7)$	0	$41/1680 = 0.0244048$	0.0244063	1.0000643
41^2	$(41, y^2+34y+8)$	0	$1681/2825760 = 0.0005949$	0.0005914	0.9942103
47	$(47, y+11)$	0	$47/1104 = 0.0425725$	0.0425984	1.0006092
47^2	$(47, y^2+36y+27)$	0	$2209/2439840 = 0.0009054$	0.0009038	0.9982027

Table 2: $\mathrm{val}_{\mathfrak{l}}(f_y)$ for bad and good (including projective) ideals above $\ell < 50$, and experimental value for a sampling of $N = 10^8$ pairs of coprime ideals $(\boldsymbol{a}, \boldsymbol{b})$, for $h = Y^3 + 15$ and $f = (8y^2 - 8y - 6)X^4 - (11y^2 + 11y - 1)X^3 - (8y^2 - 12y - 9)X^2 - (6y^2 - 10y - 9)X + 9y^2 + 6y + 11$.

for a given tuple of polynomials (h, f_y, g_y) and a bound B, requires the evaluation of integral given in Equation (4.5), which is again an uphill task. In practice, we do not compute this integral, instead do simulations to compute the actual cost of TNFS for a given (h, f_y, g_y). The value of (h, f_y, g_y) which gives the minimum run time complexity is taken as the suitable tuple of polynomials for TNFS and corresponding cost is the estimated run time complexity of TNFS algorithm. The more details of how it is achieved is discussed in the Section 6 below.

6 Cost Estimation of TNFS through Simulations

In this section, we aim to estimate an accurate cost of solving the DLP using TNFS algorithm. We are given with the values of field characteristic p and extension degree n. As explored in the papers [30, 6], the TNFS algorithm works best for the minimum possible value of κ. We choose κ a nontrivial smallest factor of n and η as $n \div \kappa$. When n is even, κ is taken as 2. Below we provide the details of our approach to estimating the cost of TNFS:

1. For a given η, we first generate all the irreducible polynomial $h(Y)$'s of degree η in $\mathbb{Z}[Y]$ having coefficients in $\{-1, 0, +1\}$, degree equal to η and which are also irreducible modulo p. These polynomials are generated

in such a way that the $d_h := \#\mathrm{Aut}_{\mathbb{Q}}(K_f)$ is the largest possible. Since $d_h | \deg(h)$, we aim to have d_h equal to η in the best case. If the set of $h(Y)$'s is empty, we increase the coefficient size i.e., $\|h\|_\infty = 2$ and check again. In almost all the cases $\|h\|_\infty = 1$ is sufficient (exceptions are for degree 2 and 3).

2. Corresponding to each $h(Y)$, we generate the pairs $(f_y(X), g_y(X))$'s, using the best polynomial selection algorithms available. Thus we are left with many triplets $(h(Y), f_y(X), g_y(X))$. Here again, we aim to generate $f_y(X)$ and $g_y(X)$ in such a way that $d_f := \#\mathrm{Aut}_{K_h}(K_f)$ and $d_g := \#\mathrm{Aut}_{K_h}(K_g)$ are largest possible. Ideally, we should rank them based on the Murphy-E function and choose the one which is optimum. Since, it is very difficult to evaluate the Murphy-E function, we compute the values of $\alpha(f_y, 1000)$ and $\alpha(g_y, 1000)$ for each triplets and rank them based on the values of $\alpha(f_y, 1000) + \alpha(g_y, 1000)$ and consider a few of them as a possible suitable polynomials. In practice, we consider 20 to 50 polynomial triplets based on sum of α values from lowest to highest, call them as good ones. With the GJP method, the choice of f_y, g_y is very limited, only h can vary.

3. For each of these triplets, we estimate the cost of the TNFS algorithm and take the one corresponding to the lowest cost. Estimating the cost of TNFS for a given tuple (h, f_y, g_y) is again a complicated task. In the Section 6.1, we provide the details of how the cost of TNFS is estimated.

6.1 Cost Estimation

We assume the setup given in the Section 3. Suppose that we are given with the triplets $(h(Y), f_y(X), g_y(X))$ along with p, η and κ. The α-values of $f_y(X)$ and $g_y(X)$ are also available with us. Further assume that $d_h := \#\mathrm{Aut}_{\mathbb{Q}}(K_h)$, $d_f := \#\mathrm{Aut}_{K_h}(K_f)$ and $d_g := \#\mathrm{Aut}_{K_h}(K_g)$.

We now choose a factor base bound B and a sieving bound A (relative to B) and proceed as follows:

Size of Factor Base

As pointed out in the paper [9], the size of factor base is

$$\#\mathcal{B} = \frac{B}{\log B}\,(2 + o(1)) \tag{6.1}$$

and we have also observed the same in our simulations. If we consider the existence of non-trivial automorphisms, the effective size of factor base is reduced to

$$\#\mathcal{B} = \frac{B}{d_h \cdot d_f \cdot \log B} + \frac{B}{d_h \cdot d_g \cdot \log B}\ . \tag{6.2}$$

Cost of Relation Collection

This is the sum of cost of sieving and cost of doing factorisation using the ECM. For a given sieving bound A, we sieve all the pairs $(a(y), b(y))$ where $\|a(y)\|_\infty \le A$ and $\|b(y)\|_\infty \le A$, so the volume of sieving space is $(2\,A{+}1)^{2\cdot\eta}$. More precisely, to avoid duplicate relations because of the equality $a(y)+b(y)x = -(-a(y)-b(y)x)$, we restrict to positive leading coefficients $\mathrm{lc}(b) > 0$. This is a usual trick in sieving: in classical NFS in dimension 2, we have $a \in [-A, A]$ and $b \in [1, A]$. For more details on sieving we refer to the paper [21]. It is not very easy to estimate the exact cost of sieving, but with the practical experience on the record discrete logarithm computations, the community tends to believe that it is of the order which is equal to $\log(\log(B))$ times the volume of sieving space.

$$\begin{aligned}\text{Cost of relation collection} &= \text{Cost of sieving} + \text{Cost of ECM}\\ &= (2\,A+1)^{2\cdot\eta}/2 \cdot \log(\log(B)) + \text{Cost of ECM}\end{aligned}$$

The cost of ECM for a sieved tuple is approximately $L_B\left(\frac{1}{2}, \sqrt{2}\right)$ and we expect to get $O(2B/\log(B))$ sieved tuples. So the cost of doing ECM is $L_B\left(\frac{1}{2}, \sqrt{2}\right) \cdot O(2B/\log(B))$. Thus the cost of relation collection is

$$\begin{aligned}&\text{Cost of relation collection}\\ &= (2\,A+1)^{2\cdot\eta}/2 \cdot \log(\log(B)) + \underbrace{L_B\left(\frac{1}{2}, \sqrt{2}\right) \cdot O(2B/\log(B))}_{\text{negligible}}.\end{aligned}$$

The cost of relation collection can further be brought down in the presence of non-trivial automorphisms. This can be understood with the following example: Assume $\gcd(\deg h, n/\deg h) = \gcd(\kappa, \eta) = 1$. Assume there is an automorphism $\sigma : X \mapsto -X$ in K_f and K_g. We can obtain a factor two speed-up: applying σ to $a(y) + b(y)X$ gives $a(y) - b(y)X$ and we can obtain for free its factorisation into smooth ideals by applying σ to each factor of $a(y) + b(y)X$. To avoid processing $-a(y) + b(y)X$, we restrict the sieving to positive leading coefficients $\mathrm{lc}(a) > 0$. The sieving time is divided by 2 because we consider only $\mathrm{lc}(a) \in [1, A]$. We get for free the relation with $-a$.

In practice, with the suitable choice of polynomials, one can obtain a speedup by a factor of $(\#\operatorname{aut}(h)\gcd(\deg(f), \deg(g)))$ due to the automorphisms. Most of the time, we have $\#\operatorname{aut}(h) = 1$ and $\gcd(\deg(f), \deg(g)) = 1$ or 2. Thus the estimated cost of relation collection turns out to be

$$\text{Cost of relation collection } = \frac{(2A+1)^{2\cdot\eta} \cdot \log(\log(B))}{2 \cdot (\#\operatorname{aut}(h)\gcd(\deg(f), \deg(g)))} \tag{6.3}$$

Number of relations

To estimate the number of relations, we follow Murphy's approach to define the E value (4.5), but we replace the integral sign by a sum over a large sample (in practice from 10^5 to 10^6 samples are needed to obtain enough accuracy). The algorithm 6.1 we obtain is also a refinement of [6]. The inputs to determine the number of relations are polynomials f_y, g_y, h and α-values α_f, α_g computed in Section 4. The drawback is a slower computation time compared to Murphy's E value defined for NFS.

$$E(f_y, g_y, h, A, B, Q) = \sum_{\substack{\text{coprime } (a\mathcal{O}_h, b\mathcal{O}_h),\ \mathrm{Coeff}(a)\in\{-A,A\}^{\deg h} \\ \mathrm{Coeff}(b)\in\{-A,A\}^{\deg h-1}\times\{0,A\}}} \left[\rho\left(\frac{\log|\mathrm{Res}(a(y)-b(y)X, f_y(X))| - Q + \alpha(f_y,h)}{\log(B)}\right) \cdot \rho\left(\frac{\log|\mathrm{Res}(a(y)-b(y)X, g_y(X))| + \alpha(g_y,h)}{\log(B)}\right)\right] \quad (6.4)$$

Algorithm 6.1: Monte-Carlo approximation of Murphy's E for TNFS (computes an estimation of the number of relations)

Input: Valid polynomials f_y, g_y, h, α_f, α_g, parameter $A \in \mathbb{N}$, smoothness bounds B_f, B_g, average special-q size Q, $N \approx 10^6$

Output: Yield estimate (number of relations)

```
1  $P_{fg} \leftarrow 0$
2  for $n := 1$ to $N$ do
3      $\boldsymbol{a} \leftarrow$ random vector in $\{-A, A\}^{2\deg h}$
4      $\boldsymbol{b} \leftarrow$ random vector in $\{-A, A\}^{2\deg h-1} \times \{0, A\}$
5      if $\gcd(\boldsymbol{a}, \boldsymbol{b}) \neq 1$ then                gcd of an array of integers
6          continue
7      $\mathfrak{a} \leftarrow \boldsymbol{a}\mathcal{O}_h$, $\mathfrak{b} \leftarrow \boldsymbol{b}\mathcal{O}_h$
8      if the ideals $\mathfrak{a}, \mathfrak{b}$ are not coprime ($\mathfrak{a} + \mathfrak{b} \neq 1$) then
9          continue
10     $N_f \leftarrow |\mathrm{Res}(h, \mathrm{Res}(f_y, \boldsymbol{a} - \boldsymbol{b}x))|$
11     $N_g \leftarrow |\mathrm{Res}(h, \mathrm{Res}(g_y, \boldsymbol{a} - \boldsymbol{b}x))|$
12     $u_f \leftarrow (\ln N_f - Q + \alpha_f)/\ln B_f$ ; $p_f \leftarrow \rho(u_f) + (1-\gamma)\rho(u-1)/\ln N_f$
13     $u_g \leftarrow (\ln N_g + \alpha_g)/\ln B_g$ ; $p_g \leftarrow \rho(u_g) + (1-\gamma)\rho(u-1)/\ln N_g$
14     $P_{fg} \leftarrow P_{fg} + p_f p_g$
15 $P_{fg} \leftarrow P_{fg}/N$
16 $w \leftarrow$ index of group of torsion units of $\mathcal{O}_h$
17 $V \leftarrow (2A+1)^{2\deg h}/(2w\zeta_{K_h}(2))$
18 return $V \times P_{fg}$
```

The choice of A should be made in such a way that the number of relations should be greater than or equal to the size of factor base, given in the Equation 6.2 and this does ensure a successful linear algebra step.

Cost of Linear Algebra

The cost of linear algebra is estimated using the number of relations from Alg. 6.1 and the size of factor base from Equation 6.2. The number of relations is further adjusted due to filtering. The filtering is the process of reducing the size of sparse system of linear equations for faster linear algebra.

We refer to [24, §B] about modelling the filtering step. We assume that the weight `wt` of the matrix is of 200 non-zero entries per row, and the filtering step reduces the size of the matrix by a constant factor `flt`.We agree that this is not satisfying enough compared to the effort to define α and Murphy's E for TNFS, and more work is needed in the future on this topic.

Computing the right kernel of a sparse matrix of N rows can be efficiently performed with the block-Wiedemann algorithm. We refer to [29, Theorem 7] for results on the complexity of this algorithm. For a choice of parameters n and m, typically $n = 2$ and $m = 4$, the algorithm is made of n Krylov sequences of $(N/m + N/n)$ iterations (that is, smvp for products of a sparse matrix times a vector), and one sequence of Mksol of N/n iterations (smvp and vector additions). The total cost in terms of iterations of smvp is $n(N/m+N/n)+N/n = N(1 + n/m + 1/n)$. One multiplication of the sparse matrix times a vector costs the number of rows N times the weight per row `wt` multiplications modulo ℓ, that is, $N\texttt{wt}$. The total number of multiplications modulo a large prime ℓ is $N^2\texttt{wt}(1 + n/m + 1/n)$ that we can approximate by $N^2\texttt{wt}$.

Thus we end up having a sparse linear system of weight `wt` per row and the size (number of rows) equal to $(\#\mathcal{B} \div \texttt{flt})$ and hence the estimated cost of linear algebra step performed with block-Wiedemann algorithm is

$$\text{Cost of Linear Algebra} = \texttt{cnst} \cdot \texttt{wt} \cdot (\#\mathcal{B} \div \texttt{flt})^2, \qquad (6.5)$$

where `cnst` is a constant representing the cost of a multiplication modulo ℓ. To reflect the higher cost with larger ℓ, we let `cnst` represents the machine word size of the prime modulo which the linear algebra is carried out.

The bounds B and A are chosen in such a way that the estimated cost of linear algebra and the estimated cost of relation collection turn out to be almost the same. The cost of individual discrete logarithm phase is very small in comparison to the other two steps. Hence the sum of the costs of linear algebra and relation collection steps are taken as the estimated cost of TNFS algorithm for the given parameters.

7 Some Simulation Results

7.1 BN and BLS-12 curves

We present now the experiments for BLS and BN curves. These curves are popular pairing-friendly curves in pairing-based cryptography. The target group of the pairing is a multiplicative subgroup of a finite field extension $\mathbb{F}_{p^k}$, and $k = 12$ for these two families of curves. They are *special* because the prime p is parameterised by a polynomial of degree 4, resp., 6, and tiny coefficients (Table 3). We run our STNFS simulation algorithm for parameters of curves available in public implementations and papers and report the seeds in Table 3. When there is no seed, we use the code `enumerate_sparse_T.py` from [24] and look for prime p and r. The aim is to get machine-word aligned parameters p. We did not check if the curves were subgroup-secure and twist-secure except for BLS12-446. We detail the experiments for BN-382 and BLS12-381. The parameters for the other curves are summarised in Tables 4 (BN) and 6, 8 (BLS12) and the polynomials are reported in Tables 5, 7, 9.

For BN-382, the seed is $u = -(2^{94} + 2^{76} + 2^{72} + 1)$ from [36]. We choose h of degree 6 among the list of monic irreducible degree 6 polynomials of coefficients in $\{1, -1, 0\}$. We set $a_{U,y}(X) = X^2 - Uy$, $g_y(X) = \mathrm{Res}_U(a_{U,y}(X), U - u) = X^2 - uy$ and $f_y(X) = \mathrm{Res}_U(a_{U,y}(X), P_{\mathrm{BN}}(U)) = 36X^8 + 36yX^6 + 24y^2X^4 + 6y^3X^2 + y^4$.

Curve parameters	$\log_2 p$	seed for p, r, t	
Barreto-Naehrig, $k = 12$, $D = 3$, $p + 1 - t = r$, $t^2 - 4p = -Dy^2$			
$p = 36x^4+36x^3+24x^2+6x+1$	254	$-(2^{62} + 2^{55} + 1)$	[36]
$r = 36x^4+36x^3+18x^2+6x+1$	382	$-(2^{94} + 2^{76} + 2^{72} + 1)$	[36]
$t = 6x^2 + 1$	446	$2^{110} + 2^{36} + 1$	[36]
$y = 6x^2 + 4x + 1$	462	$2^{114} + 2^{101} - 2^{14} - 1$	[6]
$c = 1$	1022	$-2^{254} + 2^{33} + 2^6$	
Barreto-Lynn-Scott, $k = 12$, $D = 3$, $p + 1 - t = rc$, $t^2 - 4p = -Dy^2$			
$p = (x-1)^2(x^4-x^2+1)/3+x$	381	$-(2^{63}+2^{62}+2^{60}+2^{57}+2^{48}+2^{16})$	[13]
$r = x^4 - x^2 + 1$	440	$-(2^{73} + 2^{72} + 2^{50} + 2^{24})$	[6]
$t = x + 1$	442	$-(2^{73} + 2^{72} + 2^{71} - 2^{48} + 2^{12})$	[6]
$y = (x - 1)(2x^2 - 1)/3$	446	$-(2^{74}+2^{73}+2^{63}+2^{57}+2^{50}+2^{17}+1)$	
$c = (x - 1)^2/3$	455	$2^{76} + 2^{53} + 2^{31} + 2^{11}$	[3]
	461	$-2^{77} - 2^{59} + 2^9$, $-2^{77} + 2^{50} + 2^{33}$	[6]
	1150	$-2^{192}+2^{188}-2^{115}-2^{110}-2^{44}-1$	

Table 3: Parameters of families BN and BLS with $k = 12$ and $D = 3$.

For each possible h, we run Algorithm 6.1 with 10^5 samples to obtain an estimation of the total number of relations the polynomials (h, f_y, g_y) would produce. We keep the best pair. Finally, with $h = Y^6 + Y - 1$ (see Table 7), we have $\alpha(f_y, h, 1000) = 2.7086$, $\alpha(g_y, h, 1000) = 1.1285$, and $1/\zeta_{K_h}(2) = 0.9390$. With parameter $A = 577$ (inclusive bound on the coefficients of $\boldsymbol{a} = [a_0, \ldots, a_5]$, $\boldsymbol{b} = [b_0, \ldots, b_5]$) one has a total relation collection space of $V_0 = (2A + 1)^{12}/2 = 2^{121.08}$ and a core-space (removing non-coprime pairs of ideals) of $V = V_0/\zeta_{K_h}(2) = 2^{120.99}$. The smoothness bound $B = 2^{63.481}$ induces a factor base of $\#\mathcal{F}_f + \#\mathcal{F}_g = 2\text{LogIntegral(B)} = 2^{59.0555}$. With these parameters, Algorithm 6.1 outputs a smoothness probability average of $2^{-61.4109}$, when multiplied by V_1, one gets Murphy's E value to be $2^{59.5823}$ relations. We have slightly more relations than primes in the factor base. The time of relation collection is $V_0 \log\log B = 2^{122.0041}$ and the time of linear algebra is $\lceil \ell/2^{64} \rceil \times 200 \times ((\#\mathcal{F}_f + \#\mathcal{F}_g)/20)^2 = 2^{122.0001}$ according to eq. (6.5). Finally the total estimated cost is 2^{123}. The other parameters for BN curves are presented in Table 4 and for BLS-12 curves in Tables 6 and 8.

We also ran the simulation for increasing sizes of p without a particular sparse seed, to compare how STNFS scales for larger values. Since the prime p is given by a polynomial of degree 4 for BN curves and 6 for BLS-12 curves, the choice of the degree of h and the estimated costs differ. Figure 3 presents the data. For BN curves, h has degree 6 up to p of around 600 bits, and for larger values of p, h of degree 4 provides a lower cost estimate. For BLS-12 curves, h of degree 12 is the best for p up to 320 bits, then degree 6 is better. We also plot the function $L_p(1/3, 1.923)/2^{8.2}$ which is the theoretical cost of NFS, assuming $o(1) = 0.0$ which is of course false since actually $o(1)$ is unknown, and linearly re-scaled to match the latest record computation of 768 bits from [32]. In dashed line we plot the function $L_p(1/3, 1.526)/2^{4.5}$ for the theoretical cost of SNFS, with unknown $o(1)$ set to 0.0 and linearly re-scaled to fit the record computation for p of 1024 bits from [17]. The source code is available at

`https://gitlab.inria.fr/tnfs-alpha/alpha`

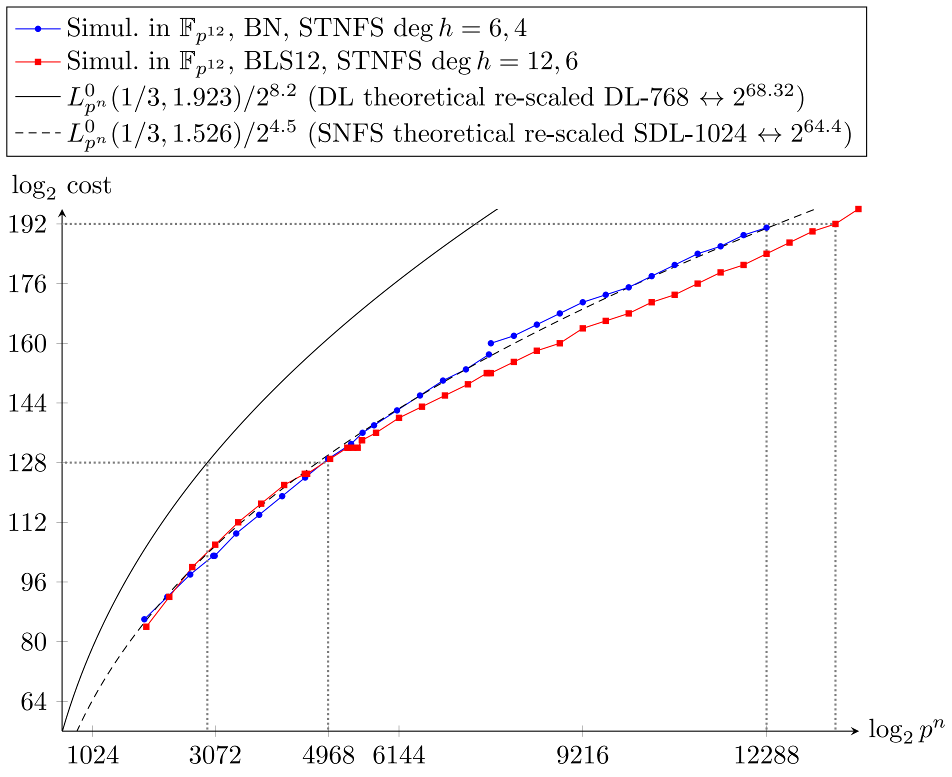

Figure 3: Simulation of STNFS for increasing p for BN and BLS12 curves using Algorithm 6.1 for 10^5 samples.

curve	Barreto-Naehrig				
p (bits)	254	382	446	462	1022
r (bits)	254	382	446	462	1022
p^k (bits)	3039	4575	5343	5535	12255
u (bits)	63	95	111	115	254
polynomials	STNFS	STNFS	STNFS	STNFS	STNFS
$\deg h$	6	6	6	6	4
$\deg f_y$	8	8	8	8	12
$\deg g_y$	2	2	2	2	3
$\lVert f_y \rVert_\infty$	36	36	36	36	3644
$\lVert g_y \rVert_\infty (\sim u)$	$2^{62.01}$	$2^{94.00}$	$2^{110.00}$	$2^{114.00}$	$2^{254.00}$
$1/\zeta_{K_h}(2)$	0.9530	0.9390	0.9334	0.8844	0.9461
$\alpha(f_y, h, 10^3)$	2.0239	2.7086	2.4156	-0.6489	0.5359
$\alpha(g_y, h, 10^3)$	2.4793	1.1285	1.8456	0.6647	2.3863
A	172	577	970	1152	7372857
B	$2^{53.006}$	$2^{63.481}$	$2^{67.971}$	$2^{69.405}$	$2^{97.403}$
av. N_f (bits)	407.49	489.46	526.06	542.85	1131.51
av. N_g (bits)	461.84	674.34	780.71	807.99	1287.55
av. $N_f \cdot N_g$ (bits)	869.33	1163.80	1306.77	1350.84	2419.06
av. B-smooth. Pr	$2^{-51.0592}$	$2^{-61.4109}$	$2^{-66.3912}$	$2^{-67.2135}$	$2^{-96.1975}$
rel. col. space	$2^{100.17}$	$2^{121.08}$	$2^{130.07}$	$2^{133.05}$	$2^{189.51}$
factor base	$2^{48.8480}$	$2^{59.0555}$	$2^{63.4444}$	$2^{64.8481}$	$2^{92.3481}$
rels. obtained	$2^{49.0368}$	$2^{59.5823}$	$2^{63.5804}$	$2^{65.6559}$	$2^{93.2330}$
total cost	$\mathbf{2^{102}}$	$\mathbf{2^{123}}$	$\mathbf{2^{132}}$	$\mathbf{2^{135}}$	$\mathbf{2^{191}}$

Table 4: Summary of parameters and estimated data for the simulation of STNFS (Alg. 6.1, average over 10^5 samples) for BN curves, $k = 12$ and $D = 3$.

curve	polynomials
BN-254	$h = Y^6 + Y^5 - Y^2 - Y - 1$ $f_y = 36X^8 + 36yX^6 + 24y^2X^4 + 6y^3X^2 + y^4$ $g_y = X^2 - uy = x^2 + 4647714815446351873y$
BN-382	$h = Y^6 + Y - 1$ $f_y = 36X^8 + 36yX^6 + 24y^2X^4 + 6y^3X^2 + y^4$ $g_y = X^2 - uy = x^2 + 19807120908796293182354620417y$
BN-446	$h = Y^6 - Y^4 + Y^3 - Y + 1$ $f_y = 36X^8 + 36yX^6 + 24y^2X^4 + 6y^3X^2 + y^4$ $g_y = X^2 - uy = x^2 - 1298074214633706907132692801781761y$
BN-462	$h = Y^6 + Y^5 + Y^3 + Y + 1$ $f_y = 36X^8 + 36yX^6 + 24y^2X^4 + 6y^3X^2 + y^4$ $g_y = X^2 - uy = X^2 - 20771722735339766972924978723274751y$
BN-1022	$h = Y^4 + Y - 1$ $f_y = 36X^{12} + 36X^{11} - 372X^{10} - 414X^9 + 1411X^8 + 1828X^7$ $-2124X^6 - 3644X^5 + 277X^4 + 2634X^3 + 1608X^2 + 396X + 36$ $g_y = X^3 - uX^2 - (u-3)X - 1,\ u = -2^{254} + 2^{33} + 2^6$

Table 5: Polynomials h, f_y, g_y chosen to minimise the total estimated cost of STNFS. The simulation of STNFS of Algorithm 6.1 with 10^5 samples produced the data of Table 4.

curve	Barreto-Lynn-Scott				
p (bits)	381	446	461	461	1150
r (bits)	255	299	309	308	768
p^k(bits)	4569	5352	5525	5525	13799
u (bits)	64	75	78	78	192
polynomials	STNFS	STNFS	STNFS	STNFS	STNFS
$\deg h$	6	6	6	6	6
$\deg f_y$	12	12	12	12	12
$\deg g_y$	2	2	2	2	2
$\|f_y\|_\infty$	2	2	2	2	2
$\|g_y\|_\infty (\sim u)$	$2^{63.71}$	$2^{74.59}$	$2^{77.00}$	$2^{77.00}$	$2^{191.91}$
$1/\zeta_{K_h}(2)$	0.9192	0.9333	0.9388	0.9390	0.9389
$\alpha(f_y, h, 10^3)$	2.1788	1.9147	1.0472	2.2555	2.2555
$\alpha(g_y, h, 10^3)$	0.9598	1.0647	2.1857	2.2899	2.3923
A	686	969	1152	1088	32619
B	$2^{65.316}$	$2^{68.219}$	$2^{69.752}$	$2^{69.241}$	$2^{98.629}$
av. N_f (bits)	724.89	760.82	787.52	771.10	1124.36
av. N_g (bits)	497.04	568.13	586.01	583.26	1331.61
av $N_f N_g$ (bits)	1221.93	1328.95	1373.53	1354.35	2455.97
av B-smooth Pr	$2^{-62.5660}$	$2^{-66.2127}$	$2^{-67.2722}$	$2^{-66.8158}$	$2^{-96.7408}$
rel. col. space	$2^{124.08}$	$2^{130.05}$	$2^{133.05}$	$2^{132.06}$	$2^{190.92}$
factor base	$2^{60.8480}$	$2^{63.6871}$	$2^{65.1871}$	$2^{64.6871}$	$2^{93.5556}$
rels. obtained	$2^{61.3898}$	$2^{63.7410}$	$2^{65.6833}$	$2^{65.1509}$	$2^{94.0898}$
total cost	$\mathbf{2^{126}}$	$\mathbf{2^{132}}$	$\mathbf{2^{135}}$	$\mathbf{2^{134}}$	$\mathbf{2^{193}}$

Table 6: Summary of parameters and estimated data for the simulation of STNFS (Alg. 6.1, average over 10^5 samples) for BLS-12 curves, $k = 12$ and $D = 3$.

curve	polynomials
BLS-381	$h = Y^6 - Y^2 + 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 + y^2 - 1$ $g_y = X^2 - uy = X^2 + 15132376222941642752y$
BLS-446	$h = Y^6 - Y^4 + Y^3 - Y + 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 + y^4 - y^3 + y - 1$ $g_y = X^2 - uy = X^2 + 28343567510342708887553y$
BLS-461a	$h = Y^6 + Y^5 + Y^2 - Y - 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y^5 - y^2 + y + 1$ $g_y = X^2 - uy = X^2 + 151116303912580950261248y$
BLS-461b	$h = Y^6 + Y - 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y + 1$ $g_y = X^2 - uy = X^2 + 151115726325920150061056y$
BLS-1150	$h = Y^6 + Y - 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y + 1$ $g_y = X^2 - uy$, $u = -2^{192} + 2^{188} - 2^{115} - 2^{110} - 2^{44} - 1$

Table 7: Polynomials h, f_y, g_y chosen to minimise the total estimated cost of STNFS. The simulation of STNFS of Algorithm 6.1 with 10^5 samples produced the data of Table 6.

curve	Barreto-Lynn-Scott				
u	u_1	u_2	u_3	u_4	u_5
p (bits)	440	442	440	443	455
r (bits)	295	296	295	297	305
p^k(bits)	5280	5296	5280	5309	5453
u (bits)	74	74	74	75	77
polynomials	STNFS	STNFS	STNFS	STNFS	STNFS
$\deg h$	6	6	6	6	6
$\deg f_y$	12	12	12	12	12
$\deg g_y$	2	2	2	2	2
$\lVert f_y \rVert_\infty$	2	2	2	2	2
$\lVert g_y \rVert_\infty (\sim u)$	$2^{73.58}$	$2^{73.81}$	$2^{73.58}$	$2^{74.00}$	$2^{76.00}$
$1/\zeta_{K_h}(2)$	0.8674	0.9370	0.8419	0.9650	0.9530
$\alpha(f_y, h, 10^3)$	0.9004	2.2555	1.5192	0.5307	2.1440
$\alpha(g_y, h, 10^3)$	0.9435	1.9999	2.1195	1.9002	1.9619
A	969	969	1027	969	1027
B	$2^{68.219}$	$2^{68.219}$	$2^{68.730}$	$2^{68.219}$	$2^{68.730}$
av. N_f (bits)	768.91	759.12	768.66	769.38	767.55
av. N_g (bits)	562.48	562.13	563.56	564.94	576.71
av $N_f N_g$ (bits)	1331.39	1321.24	1332.22	1334.32	1344.26
av B-smooth Pr	$2^{-66.0627}$	$2^{-65.8500}$	$2^{-66.1710}$	$2^{-66.0711}$	$2^{-66.5397}$
rel. col. space	$2^{130.05}$	$2^{130.05}$	$2^{131.06}$	$2^{130.05}$	$2^{131.06}$
factor base	$2^{63.6871}$	$2^{63.6871}$	$2^{64.1871}$	$2^{63.6871}$	$2^{64.1871}$
rels. obtained	$2^{63.7853}$	$2^{64.1123}$	$2^{64.6398}$	$2^{63.9306}$	$2^{64.4499}$
total cost	$\mathbf{2^{132}}$	$\mathbf{2^{132}}$	$\mathbf{2^{133}}$	$\mathbf{2^{132}}$	$\mathbf{2^{133}}$

Table 8: The seeds and polynomials are listed in Table 9. The 455-bit parameter is from RELIC [3], the other parameters communicated by Zhaohui Cheng.

curve	seed and polynomials
BLS-440a	$u_1 =$ `-0x3000004000001000000` $h = Y^6 + Y^4 + Y^3 + Y - 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y^4 - y^3 - y + 1$ $g_y = X^2 - uy = X^2 + 14167100574508859260928y$
BLS-442	$u_2 =$ `-0x37fffff000000001000` $h = Y^6 + Y - 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y + 1$ $g_y = X^2 - uy = X^2 + 16528282408568781541376y$
BLS-440b	$u_3 =$ `-0x300000001fffc010000` $h = Y^6 + Y + 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y - 1$ $g_y = X^2 - uy = X^2 + 14167099450807891853312y$
BLS-443	$u_4 =$ `0x4000000ffffffa80000` $h = Y^6 - Y^4 + 2Y^3 - Y^2 - Y + 1$ $f_y = X^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 + y^4 - 2y^3 + y^2 + y - 1$ $g_y = X^2 - uy = X^2 - 18889466212953551798272y$
BLS-455	$u_5 =$ `0x10000020000080000800` $h = Y^6 + Y^5 - Y^2 - Y - 1$ $f_y = x^{12} - 2yX^{10} + 2y^3X^6 + y^5X^2 - y^5 + y^2 + y + 1$ $g_y = X^2 - uy = X^2 - 75557872733115725645824y$

Table 9: Polynomials h, f_y, g_y chosen to minimise the total estimated cost of STNFS. The simulation of STNFS of Algorithm 6.1 with 10^5 samples produced the data of Table 8.

7.2 Other curves: KSS16, KSS18, BLS24 curves

For KSS16, KSS18 and BLS24 curves, we obtain roughly the same results as in [6]. We present in Figure 4 the estimated cost of running STNFS for these curves for increasing sizes of p. We observe a slight drift of the estimated cost above 192 compared to the theoretical $L_{p^k}(1/3, (32/9)^{1/3})$. It might be due to an underestimate in the cost of sieving or linear algebra.

In order to provide machine-word aligned parameters (p of bit-length $64w-2$), we run the code from [24] to generate a 766-bit p for KSS16 and a 638-bit p for KSS18 curves. For BLS24, the paper [15] contains seeds for parameters with p from 449 to 1119 bits, we took one of 509 bits.

Curve parameters	$\log_2 p$	seed for p, r, t	
KSS, $k = 16$, $D = 4$, $p + 1 - t = rc$, $t^2 - 4p = -Dy^2$			
$p = (x^{10} + 2x^9 + 5x^8 + 48x^6 + 152x^5 + 240x^4 + 625x^2 + 2398x + 3125)/980$			
$r = (x^8 + 48x^4 + 625)/61250$			
$t = (2x^5 + 41x + 35)/35$	330	$-2^{34} + 2^{27} - 2^{23} + 2^{20} - 2^{11} + 1$	[6]
$c = (125/2)(x^2 + 2x + 5)$	339	$2^{35} - 2^{32} - 2^{18} + 2^8 + 1$	[6]
$y = (x^5 + 5x^4 + 38x + 120)/70$	766	$2^{78} - 2^{76} - 2^{28} + 2^{14} + 2^7 + 1$	
KSS, $k = 18$, $D = 3$, $p + 1 - t = rc$, $t^2 - 4p = -Dy^2$			
$p = (x^8 + 5x^7 + 7x^6 + 37x^5 + 188x^4 + 259x^3 + 343x^2 + 1763x + 2401)/21$			
$r = (x^6 + 37x^3 + 343)/343$	348	$2^{44} + 2^{22} - 2^9 + 2$	[6]
$t = (x^4 + 16x + 7)/7$	638	$2^{80} + 2^{77} + 2^{76} - 2^{61} - 2^{53} - 2^{14}$	
$c = (49/3)(x^2 + 5x + 7)$	676	$-2^{85} - 2^{31} - 2^{26} + 2^6$	[6]
$y = (5x^4 + 14x^3 + 94x + 259)/21$	1484	$2^{186} - 2^{75} - 2^{22} + 2^4$	[6]
BLS, $k = 24$, $D = 3$, $p + 1 - t = rc$, $t^2 - 4p = -Dy^2$			
$p = (x - 1)^2(x^8 - x^4 + 1)/3 + x$	318	$-2^{32}+2^{28}-2^{23}+2^{21}+2^{18}+2^{12}-1$	
$r = x^8 - x^4 + 1$	509	$-2^{51} - 2^{28} + 2^{11} - 1$	[15]
$t = x + 1$	559	$-2^{56} - 2^{43} + 2^9 - 2^6$	[6]
$y = (x - 1)(2x^4 - 1)/3$	1022	$2^{102} + 2^{100} - 2^{10} + 2^7 + 2^2$	
$c = (x - 1)^2/3$	1032	$-2^{103} - 2^{101} + 2^{68} + 2^{50}$	[6]

Table 10: Parameters of families KSS16, KSS18 and BLS24, and seeds.

7.3 Recommendations

For efficiency reasons, it is better to choose parameter sizes that fit the machine-word size of the hardware (usually 32 or 64-bit words). We present in Table 15 the simulation results obtained for parameters that match the 128 and 192 bit security level and where the bit-length of p is a multiple of 64, minus 2 bits (for lazy-reduction compatibility).

For 128 bits of security, BN and BLS12 curves with p of 448 bits is a good option. For implementations using lazy modular reduction, one can prefer 446-bit parameters that offer the same security. KSS16 and KSS18 parameter sizes are constrained by the size of r that should be 256 bits to provide 128 bits of security on the curve. In this case p is 330-bit long for KSS16 and 348-bit long for KSS18 curves. For 192 bits of security, our error margin increases and the estimated cost should not be considered as a precise and exact cost. KSS16 curves of p of 768 bits, KSS18 curves where p is 640-bit long, and BLS24 curves where p is 512-bit

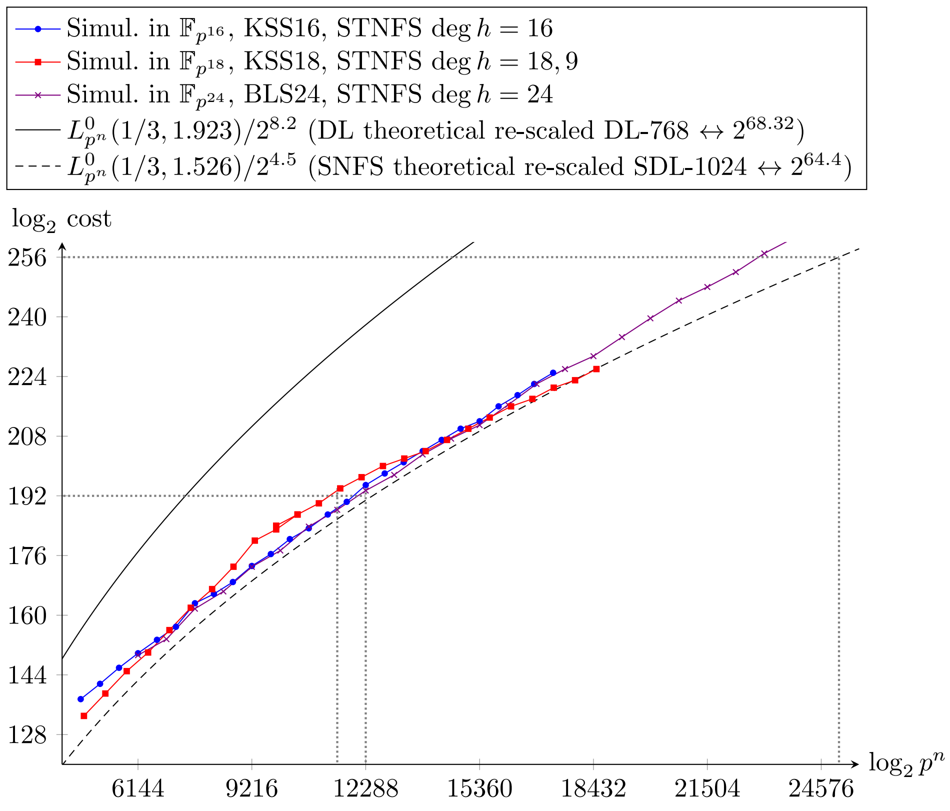

Figure 4: Simulation of STNFS for increasing p for KSS16, KSS18 and BLS24 curves using Algorithm 6.1 for 10^5 samples. As a comparison, the complexity of NFS-DL and SNFS-DL scaled with recent record computations (2017) are given, but we stress that the simulation *is not scaled.*

curve	KSS-16		KSS-18			BLS-24		
p (bits)	330	766	348	638	676	318	509	559
r (bits)	257	605	256	474	502	256	409	449
p^k(bits)	5280	12255	6257	11556	12161	7621	12202	13403
u (bits)	34	78	45	81	86	32	52	57
polynomials	STNFS	STNFS	STNFS	STNFS	STNFS	STNFS	STNFS	STNFS
$\deg h$	16	16	18	9	9	24	24	24
$\deg f_y$	10	10	8	16	16	10	10	10
$\deg g_y$	1	1	1	2	2	1	1	1
$\lVert f_y \rVert_\infty$	1492	1492	453	1767	1767	2	2	2
$\lVert g_y \rVert_\infty (\sim u)$	$2^{33.99}$	$2^{77.58}$	$2^{44.00}$	$2^{80.25}$	$2^{85.00}$	$2^{31.91}$	$2^{51.00}$	$2^{56.00}$
$1/\zeta_{K_h}(2)$	0.9906	0.9914	0.91	0.9427	0.9798	0.97	0.95	0.98
$\alpha(f_y, h, 10^3)$	2.0113	0.6609	2.1259	2.4426	2.6773	2.8417	2.9878	2.5452
$\alpha(g_y, h, 10^3)$	1.9487	2.1842	2.3048	1.4246	2.4999	2.1136	1.8235	2.1615
A	10	32	9	810	909	5	8	9
B	$2^{72.30}$	$2^{99.27}$	$2^{78.08}$	$2^{98.93}$	$2^{100.45}$	$2^{82.66}$	$2^{98.52}$	$2^{102.48}$
av. N_f (bits)	890	1156	782	1571	1597	1002	1154	1185
av. N_g (bits)	612	1336	868	904	950	853	1326	1450
av $N_f N_g$ (bits)	1503	2492	1650	2475	2547	1855	2480	2635
av B-smooth Pr	$2^{-71.69}$	$2^{-96.57}$	$2^{-74.69}$	$2^{-96.69}$	$2^{-98.18}$	$2^{-80.67}$	$2^{-96.83}$	$2^{-100.35}$
rel. col. space	$2^{139.55}$	$2^{191.72}$	$2^{151.92}$	$2^{190.93}$	$2^{193.92}$	$2^{165.05}$	$2^{195.20}$	$2^{202.90}$
factor base	$2^{67.687}$	$2^{94.19}$	$2^{73.35}$	$2^{93.85}$	$2^{95.35}$	$2^{77.85}$	$2^{93.44}$	$2^{97.35}$
rels. obtained	$2^{67.850}$	$2^{94.33}$	$2^{73.35}$	$2^{93.85}$	$2^{95.70}$	$2^{77.85}$	$2^{93.44}$	$2^{97.35}$
total cost	$\mathbf{2^{141}}$	$\mathbf{2^{194}}$	$\mathbf{2^{151}}$	$\mathbf{2^{193}}$	$\mathbf{2^{196}}$	$\mathbf{2^{161}}$	$\mathbf{2^{193}}$	$\mathbf{2^{201}}$

Table 11: Summary of parameters and estimated data for the simulation of STNFS (Alg. 6.1, average over 10^5 samples) for KSS-16, KSS-18 and BLS-24 curves.

curve	polynomials
KSS16 330	$h = Y^{16} - Y^{10} + Y^6 + Y^5 - 1$ $f = X^{10} - 8X^9 + 32X^8 - 88X^7 + 230X^6 - 416X^5 + 508X^4$ $-632X^3 + 1378X^2 + 628X + 1492$ $g_y = X - u = X + 17052993534$
KSS16 766	$h = Y^{16} + Y^{13} - Y^3 + Y - 1$ $f = X^{10} - 8X^9 + 32X^8 - 88X^7 + 230X^6 - 416X^5 + 508X^4$ $-632X^3 + 1378X^2 + 628X + 1492$ $g = X - 226673591177742701838466$
KSS18 348	$h = Y^{18} + Y^{10} + Y^8 + Y^2 + 1$ $f = X^8 - 11X^7 + 49X^6 - 75X^5 - 42X^4 + 123X^3 + 453X^2 + 315X + 63$ $g = X - 17592190238212$
KSS18 638	$h = Y^9 - Y^6 - Y^4 + Y^3 - 1$ $f = X^{16} - 11X^{15} + 57X^{14} - 152X^{13} + 280X^{12} - 483X^{11}$ $+1076X^{10} - 451X^9 + 1767X^8 - 451X^7 + 1076X^6 - 483X^5$ $+280X^4 - 152X^3 + 57X^2 - 11X + 1$ $g = X^2 - 1435597095942163676512258X + 1$
KSS18 676	$h = Y^9 + Y^4 - Y^2 - Y - 1$ $f = X^{16} - 11X^{15} + 57X^{14} - 152X^{13} + 280X^{12} - 483X^{11}$ $+1076X^{10} - 451X^9 + 1767X^8 - 451X^7 + 1076X^6 - 483X^5$ $+280X^4 - 152X^3 + 57X^2 - 11X + 1$ $g = x^2 + 38685626227668135805190078X + 1$
BLS24 318	$h = Y^{24} + Y^{16} - Y^4 - Y^2 - 1$ $f = X^{10} - 2X^9 + X^8 - X^6 + 2X^5 - X^4 + X^2 + X + 1$ $g = X + 4032557057$
BLS24 509	$h = Y^{24} + Y^{15} - Y^{11} - Y^2 - 1$ $f = X^{10} - 2X^9 + X^8 - X^6 + 2X^5 - X^4 + X^2 + X + 1$ $g = X + 2251800082118657$
BLS24 559	$h = Y^{24} + Y^{17} - Y^{12} - Y^5 + 1$ $f = X^{10} - 2X^9 + X^8 - X^6 + 2X^5 - X^4 + X^2 + X + 1$ $g = X + 72066390130949696$

Table 12: Polynomials h, f_y, g_y chosen to minimise the total estimated cost of STNFS. The simulation of STNFS of Algorithm 6.1 with 10^5 samples produced the data of Table 11.

field	curve	p bits	r bits	p^k bits	$\mathbb{F}_{p^k}$ security	$\deg h$	NFS variant
			targeted 128-bit security level				
$\mathbb{F}_{p^{12}}$	BN	311	311	3732	128	4	TNFS
$\mathbb{F}_{p^{12}}$	BN	383	383	4596	128	6	STNFS
$\mathbb{F}_{p^{12}}$	BLS12	384	256	4608	140	4	TNFS
$\mathbb{F}_{p^{12}}$	BLS12	384	256	4608	132	6	STNFS
$\mathbb{F}_{p^{18}}$	KSS18	342	256	6156	160	6	TNFS
$\mathbb{F}_{p^{18}}$	KSS18	342	256	6156	170	9	STNFS
$\mathbb{F}_{p^{24}}$	BLS24	320	256	7680	172	6	TNFS
$\mathbb{F}_{p^{24}}$	BLS24	320	256	7680	202	12	STNFS
			targeted 192-bit security level				
$\mathbb{F}_{p^{12}}$	BN	847	847	10164	192	3	TNFS
$\mathbb{F}_{p^{12}}$	BN	1031	1031	12372	192	6	STNFS
$\mathbb{F}_{p^{12}}$	BLS12	847	566	10164	192	3	TNFS
$\mathbb{F}_{p^{12}}$	BLS12	1147	766	13764	192	6	STNFS
$\mathbb{F}_{p^{18}}$	KSS18	512	384	9216	194	3	TNFS
$\mathbb{F}_{p^{18}}$	KSS18	597	443	10746	192	9	STNFS
$\mathbb{F}_{p^{24}}$	BLS24	480	386	11520	203	6	TNFS
$\mathbb{F}_{p^{24}}$	BLS24	480	386	11520	214	12	STNFS

Table 13: Menezes–Sarkar–Singh recommendations from [33, Table 5], without constants (conservative).

field	curve	p bits	r bits	p^k bits	$\mathbb{F}_{p^k}$ sec	h deg	A	B	$\log_2 N_f$	$\log_2 N_g$
$\mathbb{F}_{p^{12}}$	BN	462	462	5535	131.3	6	1098	$2^{74.2}$	557.0	808.9
$\mathbb{F}_{p^{12}}$	BLS12	461	309	5525	131.8	6	1169	$2^{73.5}$	791.2	584.8
$\mathbb{F}_{p^{16}}$	KSS16	330	257	5280	139.0	16	12	$2^{80.0}$	920.4	628.9
$\mathbb{F}_{p^{16}}$	KSS16	339	263	5411	140					
$\mathbb{F}_{p^{18}}$	KSS18	348	256	6257	152.4	18	11	$2^{82.5}$	*842.7*	*875.3*
$\mathbb{F}_{p^{18}}$	KSS18	676	502	12161	204.9	18	34	$2^{108.9}$	1114	1642
$\mathbb{F}_{p^{18}}$	KSS18	1484	1108	26705	257.13	9	11747	$2^{137.7}$	2185	1928
$\mathbb{F}_{p^{24}}$	BLS24	559	449	13403	203.72	24	9	$2^{109.8}$	1295	1460
$\mathbb{F}_{p^{24}}$	BLS24	1032	827	24760	260.9	24	23	$2^{138.5}$	1522	2619

Table 14: Barbulescu-Duquesne recommendations. The norms for KSS16-330 and KSS18-348 were the same (920.4 and 628.9), for KSS18 we deduced the exact value from $\rho(\log_2 N_f/82.5) = 2^{-36.21}$ and $\rho(\log_2 N_g/82.5) = 2^{-38.33}$.

field	curve	p	r	p^k	$\mathbb{F}_{p^k}$	h	A	B	$\log_2$	$\log_2$
		bits	bits	bits	sec	deg			N_f	N_g
$\mathbb{F}_{p^{12}}$	BN	446	446	5343	132	6	970	2^{68}	489.46	674.34
$\mathbb{F}_{p^{12}}$	BN	1022	1022	12255	191	4	7372857	$2^{97.4}$	1132	1288
$\mathbb{F}_{p^{12}}$	BLS12	446	299	5352	132	6	968	$2^{68.2}$	760.75	568.25
$\mathbb{F}_{p^{12}}$	BLS12	1150	768	13799	193	6	32619	$2^{98.6}$	1124	1332
$\mathbb{F}_{p^{16}}$	KSS16	330	*257*	5280	141	16	10	$2^{72.3}$	890	612
$\mathbb{F}_{p^{16}}$	KSS16	766	605	12255	194	16	32	$2^{99.3}$	1156	1336
$\mathbb{F}_{p^{18}}$	KSS18	348	*256*	6257	152	18	9	$2^{78.1}$	786	868
$\mathbb{F}_{p^{18}}$	KSS18	638	474	11556	193	9	810	$2^{98.9}$	1571	904
$\mathbb{F}_{p^{24}}$	BLS24	318	*256*	7621	162	24	5	$2^{82.6}$	1007	854
$\mathbb{F}_{p^{24}}$	BLS24	509	409	12202	193	24	8	$2^{98.5}$	1151	1326

Table 15: Our simulation results.

long offer a 192-bit security level (we obtained an estimation between 2^{191} and 2^{196}). BN curves of 1024-bit p and BLS12 curves of 1152-bit p also offer 192 bits of security according to our experiments, and this matches [33] (Table 13). Our estimation is not precise enough to be confident for a recommendation at the 256-bit security level. In particular, a model of the filtering step that matches the practical experiments of records computations is needed. Moreover, a model of the matrix density would be required.

8 Summary

In this paper, we have proposed an extension for the concept of Murphy's α function to the case of TNFS algorithm and which have helped us to refine the work of Barbulescu and Duquesne and to provide a better way to estimate the runtime of the algorithm. We have further provided an open source implementation of our approach for estimating the runtime of the TNFS algorithm for a range of finite fields coming from the elliptic curves suggested to be used in the pairing based cryptography.

Acknowledgements. We warmly thank Pierrick Gaudry and Emmanuel Thomé for the fruitful technical discussions on alpha.

References

[1] Leonard Adleman. The function field sieve. In Leonard M. Adleman and Ming-Deh Huang, editors, *Algorithmic Number Theory (ANTS-I)*, volume 877 of *LNCS*, pages 108–121. Springer, 1994. `https://doi.org/10.1007/3-540-58691-1_48`.

[2] Leonard M. Adleman and Ming-Deh A. Huang. Function field sieve method for discrete logarithms over finite fields. *Information and Computation*, 151(1/2):5–16, 1999. `https://doi.org/10.1006/inco.1998.2761`.

[3] Diego F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[4] Shi Bai. *Polynomial Selection for the Number Field Sieve.* Phd thesis, Australian National University, Australia, September 2011. `http://maths.anu.edu.au/~brent/pd/Bai-thesis.pdf`.

[5] Shi Bai, Richard P. Brent, and Emmanuel Thomé. Root optimization of polynomials in the number field sieve. *Math. Comp.*, 84(295):2447–2457, 2015. https://hal.inria.fr/hal-00919367, https://doi.org/10.1090/S0025-5718-2015-02926-3.

[6] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, 32(4):1298–1336, Oct 2019. https://doi.org/10.1007/s00145-018-9280-5, http://eprint.iacr.org/2017/334.

[7] Razvan Barbulescu, Pierrick Gaudry, Aurore Guillevic, and François Morain. Improving NFS for the discrete logarithm problem in non-prime finite fields. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 129–155. Springer, Heidelberg, April 2015. https://hal.inria.fr/hal-01112879v2.

[8] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 1–16. Springer, Heidelberg, May 2014. https://hal.inria.fr/hal-00835446v2.

[9] Razvan Barbulescu, Pierrick Gaudry, and Thorsten Kleinjung. The tower number field sieve. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 31–55. Springer, Heidelberg, November / December 2015. https://ia.cr/2015/505.

[10] Razvan Barbulescu and Armand Lachand. Some mathematical remarks on the polynomial selection in NFS. *Math. Comp.*, 86(303):397–418, 2017. https://hal.inria.fr/hal-00954365, https://doi.org/10.1090/mcom/3112.

[11] Razvan Barbulescu, Nadia El Mrabet, and Loubna Ghammam. A taxonomy of pairings, their security, their complexity. Cryptology ePrint Archive, Report 2019/485, 2019. https://eprint.iacr.org/2019/485.

[12] Yuval Bistritz and Alexander Lifshitz. Bounds for resultants of univariate and bivariate polynomials. *Linear Algebra and its Applications*, 432(8):1995 – 2005, 2010. Special issue devoted to the 15th ILAS Conference at Cancun, Mexico, June 16-20, 2008 http://dx.doi.org/10.1016/j.laa.2009.08.012.

[13] Sean Bowe. BLS12-381: New zk-SNARK elliptic curve construction. Zcash blog, March 11 2017. https://blog.z.cash/new-snark-curve/.

[14] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986. https://dl.acm.org/citation.cfm?id=6835, https://doi.org/10.1007/BF01840433.

[15] Craig Costello, Kristin Lauter, and Michael Naehrig. Attractive subfamilies of BLS curves for implementing high-security pairings. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT 2011*, volume 7107 of *LNCS*, pages 320–342. Springer, Heidelberg, December 2011. https://ia.cr/2011/465.

[16] Kurt Foster. HT90 and simplest number fields. *Illinois J. Math.*, 55(4):1621–1655, 2011. http://arxiv.org/abs/1207.6099.

[17] Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. A kilobit hidden SNFS discrete logarithm computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 202–231. Springer, Heidelberg, April / May 2017. https://eprint.iacr.org/2016/961.

[18] Pierrick Gaudry, Laurent Grémy, and Marion Videau. Collecting relations for the number field sieve in $GF(p^6)$. *LMS Journal of Computation and Mathematics*, 19:332 – 350, 2016. https://hal.inria.fr/hal-01273045.

[19] 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 Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 126–145. Springer, Heidelberg, August 2014. http://eprint.iacr.org/2014/119.

[20] Laurent Grémy. *Algorithmes de crible pour le logarithme discret dans les corps finis de moyenne caractéristique*. Doctorat, Université de Lorraine, Nancy, France, Septembre 2017. https://tel.archives-ouvertes.fr/tel-01647623.

[21] Laurent Grémy. Higher dimensional sieving for the number field sieve algorithms. In Renate Scheidler and Jonathan Sorenson, editors, *ANTS 2018 - Thirteenth Algorithmic Number Theory Symposium*, volume 2 of *The open book series*, pages 275–291, Madison, United States, February 2019. University of Wisconsin. `http://doi.org/10.2140/obs.2019.2.275`, `https://hal.inria.fr/hal-01890731`.

[22] Laurent Grémy, Aurore Guillevic, François Morain, and Emmanuel Thomé. Computing discrete logarithms in $\mathbb{F}_{p^6}$. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 85–105. Springer, Heidelberg, August 2017. `https://hal.inria.fr/hal-01624662`.

[23] Aurore Guillevic. Faster individual discrete logarithms in finite fields of composite extension degree. *Math. Comp.*, 88(317):1273–1301, February 2019. `https://dx.doi.org/10.1090/mcom/3376`, `https://hal.inria.fr/hal-01341849v3`.

[24] Aurore Guillevic, Simon Masson, and Emmanuel Thomé. Cocks-Pinch curves of embedding degrees five to eight and optimal ate pairing computation. Cryptology ePrint Archive, Report 2019/431, 2019. `https://eprint.iacr.org/2019/431`.

[25] Antoine Joux and Reynald Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Math. Comp.*, 72(242):953–967, 2003. `https://doi.org/10.1090/S0025-5718-02-01482-5`.

[26] Antoine Joux, Reynald Lercier, Nigel Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 326–344. Springer, Heidelberg, August 2006. `https://www.iacr.org/archive/crypto2006/41170323/41170323.pdf`.

[27] Antoine Joux and Cécile Pierrot. The special number field sieve in $\mathbb{F}_{p^n}$ - application to pairing-friendly constructions. In Zhenfu Cao and Fangguo Zhang, editors, *PAIRING 2013*, volume 8365 of *LNCS*, pages 45–61. Springer, Heidelberg, November 2014. `https://ia.cr/2013/582`.

[28] Michael Kalkbrener. An upper bound on the number of monomials in determinants of sparse matrices with symbolic entries. *Mathematica Pannonica*, 8:73–82, 1997. `http://kalkbrener.at/Selected_publications_files/Kalkbrener97b.pdf`.

[29] Erich Kaltofen. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Math. Comp.*, 64(210):777–806, 1995. `https://doi.org/10.1090/S0025-5718-1995-1270621-1`.

[30] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 543–571. Springer, Heidelberg, August 2016. `https://ia.cr/2015/1027`.

[31] Taechan Kim and Jinhyuck Jeong. Extended tower number field sieve with application to finite fields of arbitrary composite extension degree. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 388–408. Springer, Heidelberg, March 2017. `https://ia.cr/2016/526`.

[32] Thorsten Kleinjung, Claus Diem, Arjen K. Lenstra, Christine Priplata, and Colin Stahlke. Computation of a 768-bit prime field discrete logarithm. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 185–201. Springer, Heidelberg, April / May 2017. `https://eprint.iacr.org/2017/067`.

[33] Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In Raphael C.-W. Phan and Moti Yung, editors, *Mycrypt Conference, Revised Selected Papers*, volume 10311 of *LNCS*, pages 83–108, Kuala Lumpur, Malaysia, December 1-2 2016. Springer. `http://eprint.iacr.org/2016/1102`.

[34] B. A. Murphy. *Polynomial selection for the number field sieve integer factorisation algorithm.* Phd thesis, Australian National University, Australia, 1999. `http://maths-people.anu.edu.au/~brent/pd/Murphy-thesis.pdf`.

[35] Ivan Niven, Hugh L Montgomery, and Herbert S Zuckerman. *An introduction to the theory of numbers.* New York: Wiley, 5th edition, 1991.

[36] Geovandro C.C.F. Pereira, Marcos A. Simplício, Michael Naehrig, and Paulo S.L.M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319 – 1326, 2011. `https://doi.org/10.1016/j.jss.2011.03.083`, `https://eprint.iacr.org/2010/429`.

[37] Palash Sarkar and Shashank Singh. A general polynomial selection method and new asymptotic complexities for the tower number field sieve algorithm. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 37–62. Springer, Heidelberg, December 2016. `https://eprint.iacr.org/2016/485`.

[38] Palash Sarkar and Shashank Singh. New complexity trade-offs for the (multiple) number field sieve algorithm in non-prime fields. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 429–458. Springer, Heidelberg, May 2016. `https://eprint.iacr.org/2015/944`.

[39] Palash Sarkar and Shashank Singh. A unified polynomial selection method for the (tower) number field sieve algorithm. *Adv. in Math. of Comm.*, 13(3):435–455, 2019. `https://doi.org/10.3934/amc.2019028`.

[40] O. Schirokauer. Discrete logarithms and local units. *Philos. Trans. Roy. Soc. London Ser. A*, 345(1676):409–423, 1993. `http://doi.org/10.1098/rsta.1993.0139`.

[41] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm, 2019. GIT version at `http://cado-nfs.gforge.inria.fr/`.

[42] Yuqing Zhu, Jiejing Wen, Jincheng Zhuang, Chang Lv, and Dongdai Lin. Refined analysis to the extended tower number field sieve. *Theoretical Computer Science*, Jan 2020. In press `https://doi.org/10.1016/j.tcs.2020.01.010`.

A Implementation of α for NFS in cado-nfs

We briefly describe the implementation of α in cado-nfs [41]. The history (from July 2008) can be obtained with the command `git show 1deffd89` from the git repository. A SageMath code is written in `cado-nfs/polyselect/alpha.sage` and the C code in `cado-nfs/polyselect/auxiliary.c`. The files `makefb.sage` and `makefb.c` in `cado-nfs/sieve/` contain functions to compute explicitly roots of univariate polynomials modulo ℓ^k for a fixed k, while the `alpha` functions implicitly compute the number of roots modulo ℓ^k. According the cado-nfs team, the authors and contributors of this code are S. Bai, P. Gaudry, G. Hanrot, E. Thomé, and P. Zimmermann. The two main algorithms are A.1 and A.2. Algorithm A.1 returns $\mathrm{val}_\ell(f)$ as defined in Section 4, given by Equation 4.8:

$$\mathrm{val}_\ell(f) = \frac{n_\ell^{\mathrm{sim}}}{\ell+1}\,\frac{\ell}{\ell-1} + \sum_{i=1}^{\iota} \frac{m_{\ell,i}^{\mathrm{aff}} + m_{\ell,i}^{\mathrm{pro}}}{(\ell+1)\ell^{i-1}} + \frac{m_{\ell,\iota+1}^{\mathrm{aff}} + m_{\ell,\iota+1}^{\mathrm{pro}}}{(\ell+1)\ell^{\iota}}\,\frac{\ell}{\ell-1}$$

where n_ℓ^{sim} is the number of simple roots of $f \bmod \ell$, and $m_{\ell,i}^{\mathrm{aff}}$, $m_{\ell,i}^{\mathrm{pro}}$ are the number of multiple affine, resp., projective roots of f modulo ℓ^i. One needs to compute precisely the number of roots of f modulo $\ell, \ldots, \ell^{\iota+1}$. Note that we use the word *root* to denote the elements r of $\mathbb{Z}/\ell^k\mathbb{Z}$ s.t. $f(r) = 0 \bmod \ell^k$. When $\ell \mid \mathrm{Disc}(f)$, then it is possible to have more rs than the degree of f.

Here is a sketch of the lifting process. Let r be a root of f modulo ℓ, and $f'(r) = 0$, so that r is a multiple root. Assume the simplest case where there is only one multiple root r ($n_\ell^{\mathrm{sim}} = 0$, the number of multiple affine roots is $m_{\ell,1}^{\mathrm{aff}} = 1$, and there is no projective root, $m_{\ell,1}^{\mathrm{proj}} = 0$). We want to know ι and lift r modulo $\ell^2, \ell^3, \ldots, \ell^\iota$.

Since $\ell \mid f(r)$, then $\ell \mid f(r + \ell X)$. Solving $f(r+\ell X)/\ell = 0 \bmod \ell$ for $X \in [0, \ell-1]$ gives lifts $r+\ell s$ of r modulo ℓ^2. Since $f(r+\ell X) = f(r) + f'(r)\ell X \bmod \ell^2$ and $f'(r) = 0 \bmod \ell$, then $f(r+\ell X) = f(r) \bmod \ell^2$ and r lifts to roots modulo ℓ^2 if and only if $\ell^2 \mid f(r)$. To generalise this process, we need Lemma 5.

```
Algorithm A.1: average_valuation_homogeneous_coprime($f, \mathrm{Disc}_f, \ell$)
Input: Irreducible polynomial $f$, discriminant $\mathrm{Disc}_f = \mathrm{Disc}(f)$, prime $\ell$
Output: $\mathrm{val}_\ell(f)$
if ($\mathrm{Disc}_f \bmod \ell) \neq 0$ then
    return number_of_roots($f, \ell) \cdot \ell/(\ell^2 - 1) = n_{f,\ell}\ell/(\ell^2 - 1)$
else                                                              bad prime
    $v \leftarrow$ average_valuation_affine($f, \ell) \cdot \ell$            affine roots
    $v \leftarrow v+$ average_valuation_affine(Reverse($f$)($\ell X$), $\ell$)    proj. roots
    $v \leftarrow v/(\ell + 1)$
    return $v$
```

```
Algorithm A.2: average_valuation_affine($f, \ell$)
Input: Irreducible polynomial $f$, prime $\ell$
Output: Contribution of affine roots
$v \leftarrow \mathrm{val}_\ell \, \mathrm{cont}(f)$                  content of $f$: gcd of coefficients
$f_v \leftarrow f/\ell^v$
for $\overline{r} \in$ Roots($f_v \bmod \ell$) do
    if $(f_v' \bmod \ell)(\overline{r}) \neq 0$ then               simple root, end of lifting
        $v \leftarrow v + 1/(\ell - 1)$            the lifting pattern stabilises, eq. (A.4)
    else                                       multiple root, lifting one more step
        $r \leftarrow$ lift$_\mathbb{Z}(\overline{r})$                    a lift in $\mathbb{Z}$ s.t. $r = \overline{r} \bmod \ell$
        $f_2 \leftarrow f_v(r + \ell X)$                  by construction, $\ell \mid \mathrm{cont}(f_2)$
        $v \leftarrow v +$ average_valuation_affine($f_2, \ell)/\ell$
return $v$
```

Lemma 5. *Let $f(X)$ be a monic irreducible polynomial in $\mathbb{Z}[X]$ and let r be a multiple root of f modulo a prime ℓ, that is $f'(r) = 0 \bmod \ell$, where $f'(X)$ is the formal derivative of $f(X)$. Let $v = \mathrm{val}_\ell(\mathrm{cont}(f(r + \ell X)))$. We have $v \geq 1$. If $v \geq 2$, then r lifts to ℓ^{v-1} roots modulo ℓ^v.*

Proof of Lemma 5. First expand the formula

$$f(r + \ell X) = g_0 + g_1 X + g_2 X^2 + \ldots + g_d X^d \tag{A.1}$$

where $g_i \in \mathbb{Z}$. By definition, the content of $f(r+\ell X)$ is the gcd of the coefficients g_i and since we set $v = \mathrm{val}_\ell \, \mathrm{cont}(f(r + \ell X))$, then ℓ^v divides each $g_0, g_1, \ldots, g_d$ and $f(r + \ell X)$ is identically 0 modulo ℓ^v. Let us replace X by $a = a_1 + a_2\ell + a_3\ell^2 + \ldots + a_{v-1}\ell^{v-2}$ in Eq. (A.1):

$$f(r + \ell a) = g_0 + g_1 a + g_2 a^2 + \ldots + g_d a^d = 0 \bmod \ell^v \tag{A.2}$$

and this shows that the root r lifts to ℓ^{v-1} roots modulo ℓ^v. □

The initial call to algorithm A.2 with input f, ℓ in our setting has $\mathrm{cont}(f) = 1$ so $v = \mathrm{val}_\ell(\mathrm{cont}(f)) = 0$ and $f_v = f$, and since we assumed that there is only one multiple root r, then the execution arrives at line 8 where $f_2 = f(r+\ell X)$, then at line 9 and the algorithm is called (recursively) with the input $(f_2 = f(r + \ell X), \ell)$.

We now concentrate on this second run of Alg. A.2 with inputs f_2 and ℓ. Let v be the valuation at ℓ of the content of $f_2 = f(r + \ell X)$, in other words,

$\ell^v \mid \mathrm{cont}(f(r+\ell X))$. According to Lemma 5, we can lift $r \bmod \ell$ to ℓ^{v-1} roots modulo ℓ^v of the form

$$r + c_1\ell + c_2\ell^2 + c_3\ell^3 + \ldots + c_{v-1}\ell^{v-1} \pmod{\ell^v} \tag{A.3}$$

where $c_i \in [0, \ell-1]$ can take ℓ values, so there are ℓ^{v-1} roots above r. This means that the number of affine roots modulo ℓ^i is $m_{\ell,i}^{\mathrm{aff}} = \ell^{i-1}$ for i from 1 to v, and $\sum_{i=1}^{v} m_{\ell,i}^{\mathrm{aff}}/\ell^{i-1} = v$. Algorithm A.2 line 9 adds v to the contribution of roots modulo ℓ and calls itself with the new inputs $f_2 = f_v(r+\ell X), \ell$ (this is recursive).

Let us set a break-point at line 9. We know that f has one root modulo ℓ: $m_{\ell,1}^{\mathrm{aff}} = \#\{r\} = 1$, and this root lifts to ℓ^{k-1} roots modulo ℓ^k for all $2 \le k \le v$: $m_{\ell,k}^{\mathrm{aff}} = \ell^{k-1}$. We need to count the number of roots modulo ℓ^{v+1}, and this corresponds to the number of roots s of f_v. Here we need Lemma 6.

Lemma 6. *Let f be an irreducible polynomial in $\mathbb{Z}[X]$ and r a multiple root of f modulo a prime ℓ, that is, $f'(r) = 0 \bmod \ell$. Let $v = \mathrm{val}_\ell(\mathrm{cont}(f(r+\ell X)))$ and $f_v = f(r+\ell X)/\ell^v$. The root r lifts to ℓ^{v-1} roots modulo ℓ^{v+1} of the form $r + s\ell + a_2\ell^2 + a_3\ell^3 + \ldots + a_v\ell^v$ where $a_i \in [0, \ell-1]$ and s is a root of $f_v(X)$ modulo ℓ. If $f_v(X)$ has no root modulo ℓ then r does not lift modulo ℓ^{v+1}.*

Proof of Lemma 6. Write

$$\ell^v f_v(X) = f(r+\ell X)$$

hence by Lemma 5, for any $a = a_1 + a_2\ell + a_3\ell^2 + \ldots + a_{v-1}\ell^{v-2}$, we have $\ell^v f_v(a) = f(r+\ell a) = 0 \bmod \ell^v$. We want to lift this equation modulo ℓ^{v+1}. Since ℓ^v divides $\ell^v f_v(X)$, to lift r from ℓ^v to ℓ^{v+1}, we only need to solve $f_v(X) = 0 \bmod \ell$. Let s be a root of $f_v(X)$ modulo ℓ if any. Then

$$\ell^v \underbrace{f_v(s)}_{=0 \bmod \ell} = f(r+\ell s) = 0 \bmod \ell^{v+1}$$

but one can also replace s by any element $s + a\ell = s + \ell(a_2 + a_3\ell + \ldots + a_v\ell^{v-2})$ and obtain an equality modulo ℓ^{v+1}:

$$\begin{array}{rcl} \ell^v f_v(s+a\ell) & = & f(r+\ell(s+a\ell)) \\ & \Updownarrow & \\ \ell^v f_v(s + a_2\ell + a_3\ell^2 + \ldots + a_v\ell^{v-1}) & = & f(r + s\ell + a_2\ell^2 + a_3\ell^3 + \ldots + a_v\ell^v) \end{array}$$

and since $s + a\ell = s \bmod \ell$, then $f_v(s+a\ell) = 0 \bmod \ell$, and $f(r+\ell(s+a\ell)) = 0 \bmod \ell^{v+1}$. This shows that there are ℓ^{v-1} roots of f modulo ℓ^{v+1} of the form

$$r + s\ell + a_2\ell^2 + a_3\ell^3 + \ldots + a_v\ell^v$$

where $a_i \in [0, \ell-1]$ and $f_v(s) = 0 \bmod \ell$. If $f_v(X) = 0$ has no solution modulo ℓ, then r does not lift modulo ℓ^{v+1}. □

According to Lemma 6, each root s of f_v corresponds to a lift of the root r modulo ℓ^{v+1}, and f has ℓ^{v-1} roots modulo ℓ^{v+1} of the form

$$r + s\ell + c_2\ell^2 + \ldots + c_{v-1}\ell^{v-1} + c_v\ell^v \pmod{\ell^{v+1}} .$$

In other words, solving $f(r+\ell X) = 0 \bmod \ell$ fixed the variable c_1 in Eq. (A.3). If $f_v'(s) \neq 0 \bmod \ell$ then the lifting process is over: $\iota = v+1$, the algorithm accounts for the contribution of one more root s modulo ℓ^{v+1} ($m_{\ell,v+1} = \ell^{v-1}$) and terminates, with $\sum_{k=v+1}^{\infty} m_{\ell,k}/\ell^{k-1} = \sum_{k=v+1}^{\infty} \ell^{v-1}/\ell^{k-1} = 1/(\ell-1)$. The contributions of the roots modulo ℓ, with $n_\ell = 0$, $m_{\ell,1} = 1$, $m_{\ell,k} = \ell^{k-1}$ for $1 \le k \le v$, and $m_{\ell,v} = \ell^{v-1}$ is finally

$$\mathrm{val}_\ell(f) = \frac{1}{\ell+1}\left(\sum_{k=1}^{v} \frac{m_{\ell,k}}{\ell^{k-1}} + \sum_{k=v+1}^{\infty} \frac{m_{\ell,v+1}}{\ell^{k-1}}\right) =$$

$$\frac{1}{\ell+1}\Bigg(\#\{r\} + \sum_{k=2}^{v} \frac{\#\{r+c_1\ell+\ldots+c_{k-1}\ell^{k-1} \colon c_i \in [0,\ell-1]\}}{\ell^{k-1}}$$

$$+ \sum_{k=v+1}^{\infty} \frac{\#\{r+s_1\ell+\ldots+s_{k-v}\ell^{k-v}+\ldots+c_{k-1}\ell^{k-1} \colon c_i \in [0,\ell-1],\ s_i \text{ fixed}\}}{\ell^{k-1}}\Bigg)$$

$$= \frac{1}{\ell+1}\left(1 + \sum_{k=2}^{v} \frac{\ell^{k-1}}{\ell^{k-1}} + \sum_{k=v+1}^{\infty} \frac{\ell^{v-1}}{\ell^{k-1}}\right) = \frac{1}{\ell+1}\left(v + \frac{1}{\ell-1}\right) \quad \text{(A.4)}$$

This explains line 5 of Algorithm A.2, and $\iota = v+1$.

Finally we have the following Lemma 7.

Lemma 7. *Let $f(X)$ and r as above, and $v = \mathrm{val}_\ell(\mathrm{cont}(f(r+\ell X)))$, $f_v = f(r+\ell X)/\ell^v$. Let s be a root of f_v modulo ℓ. Then*

1. *if $f_v'(s) \neq 0 \bmod \ell$ then the lifting process stabilises, and the number of roots of f modulo ℓ^k for $k > v$ is constant and equals ℓ^{v-1}.*
2. *if $f_v'(s) = 0 \bmod \ell$ then the lifting process of Lemma 5 and Lemma 6 can be applied recursively with f replaced by f_v.*

Numerical example. Let $f = X^5 + 12X^3 + 12X^2 - 11X + 8$ be an irreducible monic polynomial of $\mathbb{Z}[x]$, $\mathrm{Disc}(f) = 2^9 \cdot 3^5 \cdot 5^3 \cdot 19 \cdot 23$, and $\alpha(f, 2000) = 0.511$. We compute the number of (affine) roots of f modulo $\ell \in \{2,3,5\}$.

Let $\ell = 2$. Then $f = X^5 + X = X(X+1)^4 \pmod 2$ and $f'(X) = X^4 + 1 = (X+1)^4 \bmod 2$. The polynomial f has one simple root $r = 0$ and one multiple root $r = 1$ of multiplicity 4, modulo 2: $n_{2,1} = 1$, $m_{2,1} = 1$. The simple root $r = 0$ will lift to one root modulo 2^k for any k, and $n_{2,k} = 1$. The recursive formula $f_{i+1} = f_i(r+2X)/2$ can be used to obtain a lift. For instance, $f_1 = f(0+2X)/2 = X \pmod 2$ has root 0. Then $f_2 = f_1(0+2X)/2 = X \pmod 2$ has again root 0; $f_3 = f_2(0+2X)/2 = X+1 \pmod 2$ has root 1. We deduce that $f(0 + 0\cdot 2 + 0 \cdot 2^2 + 1 \cdot 2^3) = 0 \pmod{2^4}$.

The root $r = 1$ requires more care. We have $f(1) = 22 = 0 \bmod 2$. Let us compute $f(1+2X) = 2(16X^5 + 40X^4 + 88X^3 + 116X^2 + 54X + 11)$, $v = \mathrm{val}_2 \mathrm{cont}(f(1+2X)) = 1$ and set $f_1(x) = f(1+2X)/2$. Now $f_1(X) = 1 \bmod 2$ has no root modulo 2, and the lifting process ends. It means that $f(X)$ has no root modulo 4 above the root 1. Finally $n_{2,k} = 1$ and $m_{2,k} = 0$ for any $k \ge 2$. We apply the formula with $\ell = 2$ and $v = 1$:

$$\mathrm{val}_2(f) = \frac{n_\ell}{\ell+1}\,\frac{\ell}{\ell-1} + \frac{1}{\ell+1}\left(\sum_{k=1}^{v} \frac{m_{\ell,k}}{\ell^{k-1}} + \sum_{k=v+1}^{\infty} \frac{m_{\ell,v+1}}{\ell^{k-1}}\right) = \frac{2}{3} + \frac{1}{3} = 1 \ .$$

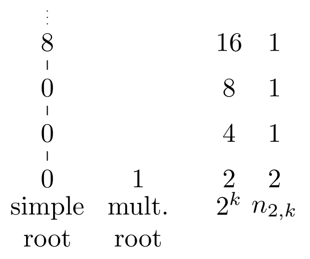

Figure 5: Lifting pattern modulo 2^k.

The lifting pattern is sketched in Fig. 5.

Let $\ell = 3$. Here is the pattern of roots mod 3^i, for any c_i in $[0, \ell - 1]$:

$$\begin{array}{ll}
3 \mid f(2) & n_3 = 1 \\
3^2 \mid f(2 + 3c_1) & n_{3^2} = 3 \\
3^3 \mid f(2 + 0 \cdot 3 + 3^2 c_2) & n_{3^3} = 3 \\
3^4 \mid f(2 + 0 \cdot 3 + 3^2 c_2 + 3^3 c_3) & n_{3^4} = 9 \\
3^5 \mid f(2 + 0 \cdot 3 + 1 \cdot 3^2 + 3^3 c_3 + 3^4 c_4) & n_{3^5} = 9 \\
3^6 \nmid f & n_{3^6} = 0
\end{array}$$

More precisely, $f(X) = X^5 + X + 2 \bmod 3$ has one root $f(2) = 0 \bmod 3$ with multiplicity 2, and $f'(X) = 2X^4 + 1 = -(X^2+1)(X+1)(X+2) \bmod 3$. We have $f(2) = 162 = 2 \cdot 3^4$ and $3^2 \mid f(2+3X)$. It means that the root $r = 2 \bmod \ell$ lifts to any root $r = 2 + 3s \bmod \ell^2$. Since $3^2 \mid f(2+3X)$, we set $f_2(X) = f(2+3X)/3^2$, and $f_2(x) = 2X^2 \bmod 3$ has one root $f_2(0) = 0 \bmod 3$ with multiplicity 2. Again $3^2 \mid f_2(0+3X)$, wet set $f_3(X) = f_2(0+3X)/3^2$ and $f_3(X) = 2X^2+2X+2 \bmod 3$ has one root $f_3(1) = 0 \bmod 3$ with multiplicity 2. Finally $3 \mid f_3(1+3X)$, we set $f_4 = f_3(1+3X)/3$ and $f_4 = 2 \bmod 3$ has no root modulo 3. We apply the formula with $\ell = 3$:

$$\mathrm{val}_3(f) = \frac{1}{3+1} \sum_{k=1}^{5} \frac{m_{3,k}}{3^{k-1}} = \frac{1}{4}\left(\frac{1}{1} + \frac{3}{3} + \frac{3}{3^2} + \frac{9}{3^3} + \frac{9}{3^4}\right) = 25/36 \; .$$

The lifting pattern is sketched in Fig. 6.

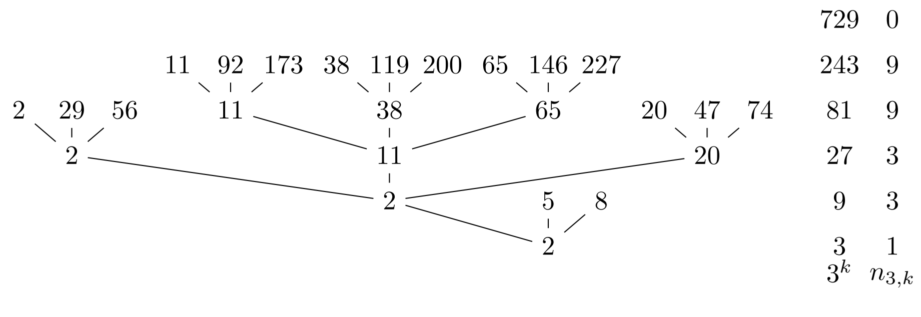

Figure 6: Lifting pattern modulo 3^k.

Let $\ell = 5$ and compute the roots of f modulo 5^i. First $f(X) = X^5 + 2X^3 + 2X^2 + 4X + 3 \bmod 5$ has one root $f(3) = 0 \bmod 5$ with multiplicity 3. Since

$5^2 \mid f(3+5X)$, we set $f_2(X) = f(3+5X)/5^2$ and $f_2(X) = 3X+1 \bmod 5$ has one root $f_2(3) = 0 \bmod 5$ with multiplicity 1, the lifting process ends (Fig. 7). We count the roots as follows, where $c_i, s_i \in [0,4]$, c_i takes any value and s_i is fixed:

$$\begin{array}{ll}
5 \mid f(3) & n_5 = 1 \\
5^2 \mid f(3+5c_1) & n_{5^2} = 5 \\
5^3 \mid f(3+3\cdot 5+5^2 c_2) & n_{5^3} = 5 \\
\vdots & \vdots \\
5^k \mid f(3+3\cdot 5+s_2\cdot 5^2+\ldots+s_{k-2}\cdot 5^{k-2}+c_{k-1}\cdot 5^{k-1}) & n_{5^k} = 5
\end{array}$$

We apply the formula with $\ell = 5$:

$$\mathrm{val}_5(f) = \frac{1}{5+1}\left(\sum_{k=1}^{3}\frac{m_{5,k}}{5^{k-1}} + m_{5,3}\sum_{k=4}^{\infty}\frac{1}{5^{k-1}}\right) = \frac{1}{6}\left(\frac{1}{1}+\frac{5}{5}+\frac{5}{5^2}+5\frac{1}{100}\right) = \frac{3}{8}$$

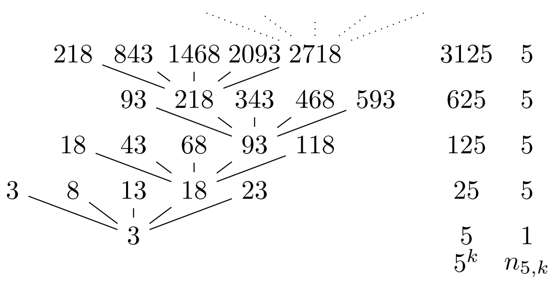

Figure 7: Lifting pattern modulo 5^k.

B Application: Counting the Number of Roots

Implicitly, the algorithms A.1 and A.2 count the number of roots of f modulo p^k until the pattern stabilises. We can easily modify the algorithms to count explicitly n_{p^k}: these are Algorithms B.1 and B.2. It is also possible to change the Algorithms of Section 5 to count the number of roots of polynomials modulo prime ideals.

Algorithm B.1: `no_roots_f_mod`$(f, \mathrm{Disc}_f, \ell, k)$
Input: Irreducible polynomial f, discriminant $\mathrm{Disc}(f)$, prime ℓ, integer $k > 0$
Output: n_{ℓ^k} number of roots of f modulo ℓ^k

```
if $(\mathrm{Disc}(f) \mod \ell) \neq 0$ then
    return number_of_roots$(f, \ell)$
else                                                                    bad prime
    $n^{\mathrm{aff}}_{\ell^k} \leftarrow$ no_roots_f_mod_rec$(f, \ell, k)$                      affine roots
    $n^{\mathrm{pro}}_{\ell^k} \leftarrow 0$
    if leading_coefficient$(f) = 0 \bmod \ell$ then
        $n^{\mathrm{pro}}_{\ell^k} \leftarrow 1$
        if $k \geq 2$ then
            $n^{\mathrm{pro}}_{\ell^k} \leftarrow$ no_roots_f_mod_rec(Reverse$(f)(\ell X)/\ell, \ell, k-1$)  proj. roots
    return $n^{\mathrm{aff}}_{\ell^k} + n^{\mathrm{pro}}_{\ell^k}$
```

Algorithm B.2: `no_roots_f_mod_rec`(f, ℓ, k)
Input: Irreducible polynomial f, prime ℓ, positive integer k
Output: $n_{\ell^k}(f)$

```
$v \leftarrow \mathrm{val}_\ell\, \mathrm{cont}(f)$ ; $f_v \leftarrow f/\ell^v$                 content of $f$: gcd of coefficients
if $v \geq k$ then  $n_{\ell^k} = \ell^k$
else if $k = 1$ then  $n_{\ell^k} = \#$Roots$(f_v \bmod \ell)$
else
    $n_{\ell^k} = 0$
    for $\overline{r} \in$ Roots$(f_v \bmod \ell)$ do
        if $(f_v' \bmod \ell)(\overline{r}) \neq 0$ then                  simple root, end of lifting
            $n_{\ell^k} \leftarrow n_{\ell^k} + \ell^v$                   the lifting pattern stabilises, eq. (A.4)
        else                                                  multiple root, lifting one more step
            $r \leftarrow$ lift$_{\mathbb{Z}}(\overline{r})$                             a lift in $\mathbb{Z}$ s.t. $r = \overline{r} \bmod \ell$
            $f_2 \leftarrow f_v(r + \ell X)/\ell$                            by construction, $\ell \mid \mathrm{cont}(f_2)$
            $n_{\ell^k} \leftarrow n_{\ell^k} + \ell^v *$ no_roots_f_mod_rec$(f_2, \ell, k - v - 1)$
return $n_{\ell^k}$
```